Lecture Notes\textsuperscript{1} 
on 
Operations Research 3OR 

Song Wang 
School of Mathematics \& Statistics 
The University of Western Australia 
song.wang@uwa.edu.au

\textsuperscript{1}Some parts of these notes are copied from Prof L. Jennings’ 3OR notes.
## Contents

1 Introduction and examples 3
   1.1 Optimization techniques ........................................... 3
   1.2 Various examples .................................................. 4
   1.3 Resource allocation ................................................. 5
   1.4 Review of linear programs ......................................... 6

2 Scheduling and Project Management 11
   2.1 Scheduling .......................................................... 11
   2.2 Batch Processing ................................................... 13
   2.3 Traffic example ..................................................... 15
   2.4 CPM — Critical Path Method ....................................... 16
   2.5 Generalize the notion of resource allocation ...................... 18
   2.6 PERT — Programme Evaluation and Review Technique ............ 19

3 Integer and Mixed Integer Programming (IP or MIP) 22
   3.1 Various examples ................................................... 22
   3.2 Tricks with binary variables ...................................... 25
   3.3 Branch-and-Bound Method for IPs .................................. 28
      3.3.1 Branch and Bound Algorithm ................................ 28
      3.3.2 Branch and Bound for general IP ............................ 32
      3.3.3 Branch and Bound for MIP ................................... 34
      3.3.4 Some notes .................................................... 36
   3.4 The Cutting Plane Method (Gomory 1958) ......................... 37
      3.4.1 Motivation ................................................... 37
      3.4.2 Properties of the cutting plane ............................. 38
      3.4.3 Example ..................................................... 39

4 Dynamic Programming (Bellman 1950) 42
   4.1 Basic concepts ..................................................... 42
   4.2 Recursive relationship ............................................. 43
   4.3 Dynamic programming algorithm .................................. 44
   4.4 Various examples in management .................................. 47
Chapter 1

Introduction and examples

1.1 Optimization techniques

- Scalar functions, \( f : \mathbb{R} \to \mathbb{R} \),
  \[
  \max_{x \in [a,b]} f(x).
  \]
  If \( f \) is continuous and differentiable on closed interval, then max and min at critical points or boundary (end points). If \( f \) is only continuous, have to include points of non-differentiability.

- Scalar functions \( f : \mathbb{R}^2 \to \mathbb{R} \), if differentiable, critical points where \( \frac{\partial f}{\partial x} = 0 \) and \( \frac{\partial f}{\partial y} = 0 \). Boundaries can be more complicated, but generally given by \( h_i(x, y) = 0 \) and the feasible region for the \( i \)-th constraint is one of \( h_i(x, y) \leq 0 \), \( h_i(x, y) = 0 \) or \( h_i(x, y) \geq 0 \). The feasible region for the problem will be the intersection over all \( i \) of these constraints, equivalent to saying AND between constraints.

- The general statement of an optimization problem takes the form
  \[
  \min_{x \in \mathbb{R}^n} f(x)
  \]
  subject to \( h(x) \geq 0 \)

- Special linear case (LP)
  \[
  \min_{x \in \mathbb{R}^n} c^T x
  \]
  subject to \( Ax \geq b, \quad x \geq 0 \).

Types of problems and methods

- Linear programs with integer variables. Branch-and-bound and cutting plan methods, dynamic programming.

- Project planning and scheduling, minimum completion time, critical path analysis.

- Nonlinear optimisation, numerical techniques, classification of problems and methods.

- Heuristic methods: simulated annealing and genetic algorithms.
1.2 Various examples

Example 1. Suppose that you have an amount of free cash, $B$, and would like to invest it. In the market there are $n$ projects $P_i$, $i = 1, 2, ..., n$ available and assume that the amount (fixed) and expected return for $P_j$ are respectively $a_j$ and $c_j$ for $j = 1, ..., n$. There are three constraints: (i) if $P_1$ is chosen, $P_2$ must be chosen; (ii) at least one of $P_3$ and $P_4$ has to be chosen; (iii) two out of $P_5, P_6$ and $P_7$ have to be chosen.

**Question:** how to invest your money so as to maximize the total profit?

**Solution.** Let $x_j$ be a binary variable so that $x_j = 1$ if $P_j$ is chosen and $x_j = 0$ otherwise. The problem can be formulated as

$$\max \sum_{j=1}^n c_jx_j$$

subject to

$$\sum_{j=1}^n a_jx_j \leq B,$$

$$x_2 \geq x_1,$$

$$x_3 + x_4 \geq 1,$$

$$x_5 + x_6 + x_7 = 2,$$

$$x_j \in \{0, 1\}, j = 1, 2, ..., n.$$

Clearly this is an integer (0-1) programming problem.

Example 2. In addition to the conditions in the previous example, we assume that the probabilities of making a profit $c_j$ and loosing your capital $a_j$ are respectively $p_j(1)$ and $p_j(2)$ with $p_j(1) + p_j(2) = 1$ for $j = 1, 2, ..., n$. We also assume that $a_j$ is not fixed and $c_j = \alpha(a_j)$. The question is to find an investment combination so that the expected total return is maximized.

**Solution.** We use the decision variables $x_j$, $j = 1, 2, ..., n$ as defined in the previous example. Then, the problem becomes

$$\max_{x,a} \sum_{j=1}^n [c_jp_j^{(1)} - a_jp_j^{(2)}]x_j$$

subject to

$$\sum_{j=1}^n a_jx_j \leq B,$$

$$x_2 \geq x_1,$$

$$x_3 + x_4 \geq 1,$$

$$x_5 + x_6 + x_7 = 2,$$

$$x_j \in \{0, 1\}, a_j \geq 0, j = 1, 2, ..., n.$$

This is a mixed integer programming problem.

Example 3. A supermarket chain company would like to open 4 new stores at three
possible locations/suburbs. A survey gives the following monthly profit chart:

<table>
<thead>
<tr>
<th>location</th>
<th>number of stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16 25 30 32</td>
</tr>
<tr>
<td>2</td>
<td>12 17 20 22</td>
</tr>
<tr>
<td>3</td>
<td>10 14 16 17</td>
</tr>
</tbody>
</table>

Find the right number of stores at each location so that the profit is maximized.

**Solution.** The problem can be solved by Dynamic Programming. We formulate it as an IP.

Let $x_i$ be the number of stores at location $i$, $i = 1, 2, 3$. Then,

$$\max_x P(x_1) + P(x_2) + P(x_3)$$

subject to

- $x_1 + x_2 + x_3 = 4$,
- $x_i \in \{0, 1, 2, 3, 4\}, i = 1, 2, 3$,

where $P$ denotes the profits given in the table.

### 1.3 Resource allocation

$M$ resources of amounts $b_i$, $i = 1, \ldots, M$.

$N$ products to be made from resources, let amounts of each product be $x_j$, $j = 1, \ldots, N$.

One unit of product $j$ requires $a_{ij}$ units of resource $i$. So

$$\sum_j a_{ij} x_j \leq b_i, \quad i = 1, \ldots, M.$$

Profit of $p_j$ for each unit of product $j$, so goal is to maximise total profit $\sum_j p_j x_j$.

**Notation** We write

- $x = (x_1, x_2, \cdots, x_N)^T \in \mathbb{R}^N$,
- $b = (b_1, b_2, \cdots, b_M)^T \in \mathbb{R}^M$,
- $p = (p_1, p_2, \cdots, p_N)^T \in \mathbb{R}^N$,

where they are all column vectors by default. For vector inequalities $x \geq 0$, for all $i = 1, \ldots, N$, so $Ax \leq b$ means $\sum_j a_{ij} x_j \leq b_i$, $i = 1, \ldots, M$. It is implied that $A$ is the matrix with elements $a_{ij} = [A]_{ij}$. Note that $\sum p_j x_j = p^T x$, so the resource allocation problem is

$$\max_x p^T x$$

on the region in $\mathbb{R}^N$ such that

$$x \geq 0, \quad \text{and} \quad Ax \leq b.$$
Terminology: Consider the general problem stated as $f : \mathbb{R}^N \rightarrow \mathbb{R}$, $g : \mathbb{R}^N \rightarrow \mathbb{R}^M$, $h : \mathbb{R}^N \rightarrow \mathbb{R}^L$ and $X \subset \mathbb{R}^N$, (for example, $X$ may be an integer point set),

$$
\min_{x \in X} f(x)
$$

subject to

$$
h(x) = 0, \quad \text{and} \quad g(x) \geq 0.
$$

The function $f$ is known as the objective function, or the cost or the profit or the utility (function). The functions $h$ are the equality constraints while $g$ are the inequality constraints. $X$ is sometimes called the domain. Constraints define a feasible region, as the region of $X$ where all constraints are satisfied (true). If an inequality constraint is zero ($g_i(x^*) = 0$) at a solution $x^*$, we say the constraint is active, otherwise inactive.

1.4 Review of linear programs

The following is called the standard form of LP problems:

minimize $z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$

subject to $a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n = b_1$

$a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n = b_2$

$\vdots$

$a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = b_m$

$x_1, x_2, \ldots, x_n \geq 0$

$b_1, b_2, \ldots, b_m \geq 0$

In matrix form this is

minimize $z = c^T x$

subject to $Ax = b$

$x \geq 0, b \geq 0$

where

$c = (c_1, c_2, \ldots, c_n)^T, \quad x = (x_1, x_2, \ldots, x_n)^T, \quad A$ an $m \times n$ matrix and $b = (b_1, b_2, \ldots, b_m)^T$

Notes:

1. $c_1, c_2, \ldots, c_n$ — cost coefficients,
2. $b_1, b_2, \ldots, b_m$ — requirement coefficients,
3. $a_{ij}, i = 1, \ldots, m, j = 1, \ldots, n$ — activity/constraint coefficients,
4. $z$ — objective function.

2. We assume that $m < n$. Otherwise $Ax = b$ is determined or over-determined.

Now, we present some methods for reducing a general LP problem to the standard form.
1. Slack variables for inequality constraints

\[ Ax \leq b, \quad x \geq 0. \]

Introducing slack variables \( y = (y_1, y_2, ..., y_m)^T \), we have

\[ Ax + y = b; \quad x, y \geq 0. \]

or

\[
\begin{align*}
  a_{11} x_1 + \cdots + a_{1n} + y_1 &= b_1 \\
  &\vdots & \vdots \\
  a_{m1} x_1 + \cdots + a_{mn} + y_m &= b_m \\
  \quad x \geq 0, \quad y \geq 0
\end{align*}
\]

2. Surplus variables for inequality constraints.

\[ Ax \geq b, \quad x \geq 0. \]

We introduce \( y = (y_1, y_2, ..., y_m)^T \) such that

\[ Ax - y = b, \quad x, y \geq 0. \]

3. Some \( b_i \) negative.

Multiply the equation by \( -1 \).

4. A constant in the objective function:

\[
\text{minimize} \quad z = c_0 + c^T x \\
\text{subject to} \quad Ax = b \\
\quad x \geq 0.
\]

Since \( c_0 \) is constant, we have that

\[
\text{minimize} \{ c_0 + c^T x \} \iff \text{minimize} c^T x.
\]

So, we just drop the constant \( c_0 \).

5. The objective is maximized, i.e. \( \max c^T z \)

Multiplying by \( -1 \) we have

\[
\max c^T z \iff \min (-c^T z).
\]

6. \( x_k \) satisfies \( -\infty < x < \infty \).

(i) Set \( x_k = u_k - v_k \) with \( u_k, v_k \geq 0 \). Replace \( x_k \) by \( u_k - v_k \) everywhere. The problem becomes an \( (n+1) \)-dimensional with unknowns

\[ x_1, \ldots, x_{k-1}, u_k, v_k, x_{k+1}, \ldots, x_n \]
(ii) Elimination of $x_k$.

There is at least one $l \in \{1, 2, ..., m\}$ such that $a_{lk} \neq 0$. So, we have

$$a_{l1}x_1 + \cdots + a_{lk}x_k + \cdots + a_{ln} = b_l$$

Solving this gives

$$x_k = \frac{1}{a_{lk}} (b_l - \sum_{j=1,\ldots,m, j \neq k} a_{lj}x_j)$$

Then, replace all $x_k$ in the cost and the constraints by this expression.

Matlab LINPROG — LP solver. Here is the help page from Matlab.

LINPROG    Linear programming.
X=LINPROG(f,A,b) solves the linear programming problem:

$$\min f'*x \quad \text{subject to:} \quad A*x \leq b$$

X=LINPROG(f,A,b,Aeq,beq) solves the problem above while additionally satisfying the equality constraints $Aeq*x = beq$.

X=LINPROG(f,A,b,Aeq,beq,lb,ub) defines a set of lower and upper bounds on the design variables, $X$, so that the solution is in the range $lb <= X <= ub$. Use empty matrices for $lb$ and $ub$ if no bounds exist. Set $lb(i) = -\infty$ if $X(i)$ is unbounded below; set $ub(i) = \infty$ if $X(i)$ is unbounded above.

X=LINPROG(f,A,b,Aeq,beq,lb,ub,x0) sets the starting point to $x0$. This option is only available with the active-set algorithm. The default interior point algorithm will ignore any non-empty starting point.

X=LINPROG(f,A,b,Aeq,Beq,lb,ub,x0,options) minimizes with the default optimization parameters replaced by values in the structure $OPTIONS$, an argument created with the OPTIMSET function. See OPTIMSET for details. Use options are Display, Diagnostics, TolFun, LargeScale, MaxIter. Currently, only 'final' and 'off' are valid values for the parameter Display when LargeScale is 'off' (‘iter’ is valid when LargeScale is ‘on’).

[X,FVAL]=LINPROG(f,A,b) returns the value of the objective function at $X$: $FVAL = f'*X$.

[X,FVAL,EXITFLAG] = LINPROG(f,A,b) returns EXITFLAG that describes the exit condition of LINPROG.
If EXITFLAG is:
> 0 then LINPROG converged with a solution $X$. 8
0 then LINPROG reached the maximum number of iterations without converging.
< 0 then the problem was infeasible or LINPROG failed.

\[ X, FVAL, EXITFLAG, OUTPUT \] = LINPROG(f, A, b) returns a structure
OUTPUT with the number of iterations taken in OUTPUT.iterations, the type
of algorithm used in OUTPUT.algorithm, the number of conjugate gradient
iterations (if used) in OUTPUT.cgiterations.

\[ X, FVAL, EXITFLAG, OUTPUT, LAMBDA \] = LINPROG(f, A, b) returns the set of
Lagrangian multipliers LAMBDA, at the solution: LAMBDA.ineqlin for the
linear inequalities A, LAMBDA.eqlin for the linear equalities Aeq,
LAMBDA.lower for LB, and LAMBDA.upper for UB.

NOTE: the LargeScale (the default) version of LINPROG uses a primal-dual
method. Both the primal problem and the dual problem must be feasible
for convergence. Infeasibility messages of either the primal or dual,
or both, are given as appropriate. The primal problem in standard
form is
\[
\text{min } f^\prime x \text{ such that } A x = b, x \geq 0.
\]
The dual problem is
\[
\text{max } b^\prime y \text{ such that } A^\prime y + s = f, s \geq 0.
\]

Example. Farmer Furniture makes chairs, arm-chairs and sofas. The profits are $50 per
chair, $60 per arm-chair and $80 per sofa. The material used to manufacture these items
are fabric and wood. A supplier can provide a maximum of 300 meters of fabric and 350
units of wood each week. Each item requires a certain amount of wood and fabric as well
as a certain assembly time. These are given in the following table

<table>
<thead>
<tr>
<th>Item</th>
<th>Fabric</th>
<th>Wood</th>
<th>Ass. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>chair</td>
<td>2m</td>
<td>6 units</td>
<td>8hrs</td>
</tr>
<tr>
<td>armchair</td>
<td>5m</td>
<td>4 units</td>
<td>4hrs</td>
</tr>
<tr>
<td>sofa</td>
<td>8m</td>
<td>5 units</td>
<td>5hrs</td>
</tr>
<tr>
<td>Avail./Wk</td>
<td>300m</td>
<td>350 units</td>
<td>480 hrs</td>
</tr>
</tbody>
</table>

Question: How many chairs, arm-chairs and sofas that the company should make per
week so that the total profit is maximized?

Solution. Let \( x_1, x_2 \) and \( x_3 \) be respectively the numbers of chairs, armchairs, sofa made
per week. Then,

\[
\text{minimize } P = 50x_1 + 60x_2 + 80x_3 \\
\text{subject to } 2x_1 + 5x_2 + 8x_3 \leq 300 \\
6x_1 + 4x_2 + 5x_3 \leq 350 \\
8x_1 + 4x_2 + 5x_3 \leq 480 \\
x_1 \geq 0, x_2 \geq 0, x_3 \geq 0.
\]

(We omit the constraints that \( x_1, x_2 \) and \( x_3 \) are integers.) This LP can be solved by the
following Matlab code:

\[
c = [-50; -60; -80];
\]
A = [ 2 5 8; 6 4 5; 8 4 5; -1 0 0; 0 -1 0; 0 0 -1];
b = [300; 350; 480; 0; 0; 0];
[x, z, flag] = linprog(c,A,b)
Chapter 2

Scheduling and Project Management

2.1 Scheduling

There are $N$ tasks with $T_i$ being the minimum time required to complete the $i$-th task. There are $M$ processors. Aim is to schedule tasks for optimal completion, usually minimum overall time. There are different classes of scheduling problems. In the flow shop, there is an object which moves through a production line, tasks done on the object have to be ordered into compatible or incompatible tasks. Different processors may have specific tasks assigned to them, for example a robot which drills holes does not do welding. There may be capacity constraints, that is, the number of tasks are greater than the number of processors. There may also be compatibility constraints, between the order of the tasks and between tasks and processors. A complex job or project may be viewed as consisting of a number of smaller tasks, for example constructing large buildings, bridges, operating computers, traffic lights, etc. Suppose a job or project can be viewed as composed of $n$ tasks numbered $1, 2, \ldots, n$. If the task $i$ takes a time $T_i$, what is the minimum time to complete the job? Upper and lower bounds are easily computed from the following. If all tasks can be done simultaneously then the minimum time is

$$T_{min} = \max_i \{T_1, T_2, \ldots, T_n\} = \max_{1 \leq i \leq n} T_i.$$  

The longest time occurs if all tasks have to be done sequentially (assume no slack)

$$T_{max} = T_1 + T_2 + \cdots + T_n = \sum_{i=1}^{n} T_i.$$

Here we assume there is no delay between tasks and that $T_i$ is the minimum time to complete task $i$. We have that the actual time of completion, $T^*$ is bounded by

$$T_{min} \leq T^* \leq T_{max}.$$  

The calculation of $T^*$ will require consideration of other constraints.

(i) The number of staff, machines, processors available to complete tasks at the same time.

(ii) Some tasks must be completed before others or cannot be performed simultaneously.
Type (i) constraints are called capacity constraints, while type (ii) constraints are called compatibility constraints.

**Example:** Bridge building.

![Diagram showing incompatibility and compatibility graphs]

Here tasks are, 1, footing, 2 pylon, 3 North approach, 4 South approach, 5 North span and 6 South span. There are some obvious precedences.

**Definition 2.1** An \( m \)-clique of a graph \( G \) is a subgraph of \( G \) with no more than \( m \) mutually connected nodes of \( G \).

**Definition 2.2** A set of cliques covers the graph \( G \) provided every vertex is present in at least one of the cliques. (An alternative is to add that every edge of \( G \) should be in at least one clique. This makes the covering set larger usually.)

**Example:** The cliques of the incompatibility graph are

\[
\{\{3, 5\}, \{4, 6\}, \{1, 2, 5\}, \{1, 2, 6\}\}.
\]

The cliques of the compatibility graph are

\[
\{C_1 = \{1, 3, 4\}, C_2 = \{2, 3, 4\}, C_3 = \{5, 6\}, C_4 = \{3, 6\}, C_5 = \{4, 5\}\}.
\]

Note that cliques 4 and 5 are not really needed because the first three cover the set of tasks. Let \( T_i \) be time allocated to task \( i \). Let \( t_j \) be the time allocated to clique \( j \). Then for each task we have the constraints:

1: \( t_1 \geq T_1 \)

2: \( t_2 \geq T_2 \)

3: \( t_1 + t_2 + t_4 \geq T_3 \)

4: \( t_1 + t_2 + t_5 \geq T_4 \)

5: \( t_3 + t_5 \geq T_5 \)

6: \( t_3 + t_4 \geq T_6 \)

The objective is to minimize \( t_1 + t_2 + t_3 + t_4 + t_5 \) over non-negative \( t_j \).

The tasks and cliques could be plotted out against time to give a graphical indication of start and end times.
2.2 Batch Processing.

Also called queue or stream processing. For example:

1. Office processing licences. Cannot use equipment at same time. More efficient to process batches of same type of licences.

2. Workshop. Processing orders. Require setting up machines which takes time. Cannot have two or more people using same machine at same time. Batches better.

3. Computers. Takes time to load software or databases onto memory from disk. Better to process a batch of jobs requiring the same software or data. Applies also to the efficient use of cache memory.

Suppose there are $M$ processors. Tasks can be divided into $N$ categories. Assume that tasks in the $i$-th category arrive randomly but never-the-less constant rate $a_i$, the arrival rate. Assume that in processing a batch of $i$-th tasks there is an initial set up time $d_i$ after which tasks are processed at a rate $p_i$, a clearing or processing rate. Certain tasks might be incompatible (at the same time), that is, two processors cannot deal with batches of these two tasks at the same time. (Might require same resources or equipment.) Tasks will form $N$ queues. We want to allocate batches of tasks to processors so that every task is guaranteed to be processed within some time period $T$.

**Simple case first.**

- Assume infinite number of processors (compatibility is only determining factor).
- assume startup times are zero, $d_i = 0$, for all $i$.
- assume a task is compatible with itself.
- Consider a period of time $T$.
  - In this time, the number of tasks arriving on stream $i$ is $a_iT$.
  - Suppose we allocate to the equivalent of one processor, a time $T_i$ to process stream $i$. The number of tasks processed is $p_iT_i$.
  - A fundamental requirement is that the system as a whole should be able to process all the tasks coming in. Hence

$$a_iT \leq p_iT_i \quad \text{or} \quad \frac{a_i}{p_i} \leq \frac{T_i}{T} \leq 1.$$  

We call the ratio $s_i = a_i/p_i$ the saturation rate and might describe that stream $i$ is saturated if $s_i = 1$.

We need to take into account the cliques of compatible tasks as we have lots of processors.

- Allocate processors to cliques of compatible tasks. Let $C_j, j = 1, \ldots, K$ be the compatibility cliques. Allocate a proportion of the total time $q_j$ to $C_j$. These $q_j$ are the variables to be determined.
• Fundamental inequality is
  \[ \sum_{i \in C_j} q_{ij} \geq s_i. \]
  Constraints are that \( q_{ij} \geq 0 \), and we might also consider \( q_{ij} \leq 1 \) and \( \sum_j q_{ij} = 1 \) in some cases.

• The objective is to minimize \( Q = \sum_j q_j \). That is, allocate the minimum proportion of time to process compatible batches. \( Q \) is called the operating capacity.
  - If \( Q > 1 \) then goal cannot be achieved, the queue will grow and the waiting time will exceed \( T \).
  - If \( Q = 1 \) then just achieve completion time of \( T \). At least one queue grows to have waiting time \( T \).
  - If \( Q < 1 \) then \( 1 - Q \) is the slack proportion of time. That is, a proportion \( 1 - Q \) of the time processors can remain idle and yet still achieve maximum waiting time \( T \).

• How to use the slack time? Do something else compatible with all tasks \( \cdots \). Re-allocate extra time, allocating \( q'_{ij} = q_{ij}/Q \), so that \( \sum_j q'_{ij} = 1 \).

• Observe that the solution is independent of \( T \), because there are no delays.

• Observe that the size of the \( i \)-th batch is \( q_{ij} T p_i \).

Case of start up delays.

• Simple case, semi-manual solution. First solve with \( d_i = 0 \), \( \forall i \).

• Now observe that we can use the slack time \( (1 - Q)T \) for set up time.
  - The set up time for clique \( C_j \) is \( \delta_j = \max_{i \in C_j} d_i \).
  - Hence set up time is less than \( \sum_j \delta_j \).
  - Note, since \( \delta_j \) are fixed times, puts a lower bound on \( T \). That is, \( T \) must be big enough to have enough slack time.

Case of limited processors.

An example: A covering set of cliques for the graph is

\[ C_1 = \{1, 3\} \]
\[ C_2 = \{2, 3, 4\} \]
\[ C_3 = \{3, 4, 5, 6\} \]

But we might only have three processors so the clique \( C_3 \) cannot be processed in parallel. Hence we use only 3-cliques which cover the graph, namely
\[ C'_1 = \{1, 3\} \]
\[ C'_2 = \{2, 3, 4\} \]
\[ C'_3 = \{3, 4, 5\} \]
\[ C'_4 = \{3, 4, 6\} \text{ enough?} \]
\[ C'_5 = \{3, 5, 6\} \]
\[ C'_6 = \{4, 5, 6\} \]

Proceed as above allocating time to the covering set of 3-cliques.

**What if \( s_i > 1 \)?**

- If task \( i \) is compatible with itself then split stream \( i \) into two streams and carry on if enough processors.
- Otherwise do something to make compatible.

**Summary comment** Basic assumption is that \( a_i T \) is relatively large, approximately 10-100.

### 2.3 Traffic example.

![Traffic example diagram](image_url)

We have an intersection with four streams of traffic as shown. Note the incompatibility and compatibility graphs. The covering cliques are \( C_1 = \{1, 2, 4\} \) and \( C_2 = \{2, 3\} \). Let \( p_i \) be the clearing rate of stream \( i \), \( a_i \) be the arrival rate of stream \( i \), \( t_i \) the total amount of green light time allocated to stream \( i \) and let \( T \) be the total time of a cycle through all streams. The fundamental inequality is that

\[ p_i t_i \geq a_i T \]

and note that \( f_i = t_i / T \geq a_i / p_i = s_i \) where \( f_i \) is the proportion of the cycle time that stream \( i \) has green and \( s_i \) is the saturation rate of stream \( i \).
Let $q_j$ be the time allocated to clique $C_j$ of compatibility graph. The objective is to minimize $Q = q_1 + q_2$ subject to stream constraints,

1: $q_1 \geq s_1$
2: $q_1 + q_2 \geq s_2$
3: $q_2 \geq s_3$
4: $q_1 \geq s_4$

with all $q_j$ non-negative. To re-allocate the spare time ($Q < 1$) we could redefine $q_j$ using $q_j \rightarrow q_j/Q$. This means that we effectively redefine $T$ so that at least one $f_i = s_i$, so that at least one stream is saturated ($s_i = 1$).

### 2.4 CPM — Critical Path Method

Project planning graph.

1. Each node represents an event, generally when the task is completed, (but sometimes when it begins).

2. Each directed arc (edge) represents an activity or task. Always have a begin and end node for the whole project.

3. Occasionally require arcs that do not represent tasks but simply precedence, can think of these as a waiting task or activity.

The bridge example: Tasks are, $A$ — footing for centre pylon, $B$ — centre pylon, $C$ — North approach, $D$ — South approach, $E$ — North span, $F$ — south span. Hence nodes are, 0 — start, 1 — completed A, 2 — completed B, 3 — completed C, 4 — completed D, 5 — completed whole job. (We could have nodes 5 and 6 for completing tasks E and F and then a node 7 for whole project.) The edges or arcs are labelled with the tasks. Note that we need edges (2, 3) and (2, 4) as tasks B must be completed before tasks E and F begin.

![Diagram of project planning graph](image)

As a Linear Program. Let $T_j$ be the known minimum time to complete task $j = A, B, \ldots, F$. Let $t_i$ be the minimum time to node $i = 1, 2, \ldots, 5$. Constraints are

$$t_1 \geq T_A$$
$$t_2 - t_1 \geq T_B$$
\[ t_3 \geq T_C \quad \text{and} \quad t_3 - t_2 \geq 0 \]
\[ t_4 \geq T_D \quad \text{and} \quad t_4 - t_2 \geq 0 \]
\[ t_5 - t_3 \geq T_E \quad \text{and} \quad t_5 - t_4 \geq T_F \]

The objective is to minimise \( t_5 \). Note this is easier to set up and solve than using cliques. Finding cliques is hard. Ordering implies sparse constraints. At a solution some of the above constraints are active. These constraints correspond to the so called critical path through the di-graph. The critical path is that path where there is no waiting (slack) for tasks to finish. Finding this path is important as putting more resources on tasks on this critical path may mean the project can be completed in shorter time. This path is the longest path from start to finish if time of each job is put on the arcs.

**Example:** We will demonstrate a so called labelling algorithm, which solves the LP without having to use the standard Simplex Algorithm. For problems represented by graphs, having a LP formulation, there is usually a graphical algorithm which makes use of the existence of links between only some of the vertices. This is the same as saying we are taking special regard to the type of constraints occurring in the LP.

<table>
<thead>
<tr>
<th>Task</th>
<th>Task arc</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>footing</td>
<td>A (0,1)</td>
<td>4</td>
</tr>
<tr>
<td>pylon</td>
<td>B (1,2)</td>
<td>4</td>
</tr>
<tr>
<td>N. approach</td>
<td>C (0,3)</td>
<td>10</td>
</tr>
<tr>
<td>S. approach</td>
<td>D (0,4)</td>
<td>6</td>
</tr>
<tr>
<td>N. span</td>
<td>E (3,5)</td>
<td>3</td>
</tr>
<tr>
<td>S. span</td>
<td>F (4,5)</td>
<td>4</td>
</tr>
<tr>
<td>waiting</td>
<td>G (2,3)</td>
<td>0</td>
</tr>
<tr>
<td>waiting</td>
<td>H (2,4)</td>
<td>0</td>
</tr>
</tbody>
</table>

1. Construct graph of project, putting in time of all tasks on arcs. We will be labelling the vertices. Let \( D_{i,j} \) be the time of the task on arc \((i, j)\). Arcs that do not exist should be given a value of \(-\infty\). Label vertex zero with \( E_0 = 0 \), and put it into a set called the finished set, \( F \).

2. Forward pass. (Finding the earliest event time, that is, time to complete tasks.) In turn, look at each vertex \( j \) adjacent to \( F \), and label it with

\[ E_j = \max_{i \in F} \{ E_i + D_{ij} \} \]

and add vertex \( j \) to \( F \). This can be refined somewhat by considering a 'frontier' set of \( F \) over which to take the maximum. With the correct values on arcs which do not exist, a simple search over all vertices could also be done.
3. Backward pass. (Finding latest event times.) Let $L_i$ be the latest event time and label the finish vertex with $L_6 = E_6$. Follow a similar algorithm but label now

$$L_i = \min_{j \in G} \{L_j - D_{ij}\},$$

where $G$ is a similarly defined ‘finished’ set of vertices which starts with just the finish vertex, 6 in this case.

4. The slack at each vertex is now $S_i = L_i - E_i$ and can be used to find which jobs can be delayed starting at that vertex. Slacks for individual tasks are best looked at by plotting tasks on a time line.

5. The critical path(s) are along those paths which have vertices with zero slack. In this case $\{0, 3, 6\}$, or $\{C, E\}$.

To reduce the time of the whole project we put more resources on selected tasks on the critical path. For example, if task C is reduced to 7 weeks from 10, the new critical path is $\{A, B, F\}$, or $\{0, 1, 2, 4, 6\}$.

2.5 Generalize the notion of resource allocation

Allocating extra resources to reduce time of a task usually costs more. We might assume there is a linear relationship (to keep the problem as a LP), between task completion time and cost.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{resource_allocation}
\caption{Resource Allocation Graph}
\end{figure}

\[(\text{Actually more likely to be hyperbolic } cost = a + b/time, \text{ but assume there is an operating band}) \text{ There is usually a lower bound to minimum completion time for a task.} \]

Let $D_{ij}$, $D_{ij}^*$ represent the normal and ‘crash’ completion times, and let $C_{ij}$ and $C_{ij}^*$ represent the corresponding costs. ‘Crash means the ‘must do, no expense spared’ job, or the fastest job within reason. Now define a decision variable $x_{ij}$ representing the (min) time to do task $(i,j)$, where $D_{ij}^* \leq x_{ij} \leq D_{ij}$. Observe that task $(i,j)$ now has cost

$$C_{ij} + m_{ij}(x_{ij} - D_{ij})$$

where $m_{ij} = (C_{ij} - C_{ij}^*)/(D_{ij} - D_{ij}^*) < 0$, because of the assumed linear relationship.

Given an allocation to all $x_{ij}$, we have a total cost

$$\sum_{(i,j)} (C_{ij} + m_{ij}(x_{ij} - D_{ij}))$$

Now recall previous LP formulation of minimum completion time problem:
(a) Let $t_i$ be the actual completion time of event $i$.

(b) Note that if there is a task $(i, j)$ then a constraint is

$$t_i + x_{ij} \leq t_j.$$  

The aim is to complete all tasks before a time $T^*$, assume we start at time zero. The LP is

$$\min \sum_{(i,j)} m_{ij} x_{ij},$$

with $D_{ij}^* \leq x_{ij} \leq D_{ij}$ and such that

$$t_i + x_{ij} \leq t_j, \quad \forall \ (i, j)$$  

and

$$0 \leq t_i \leq T^*, \quad \forall \ i.$$  

This last set of constraints could be replaced by one, namely, $t_{final} \leq T^*$. If $T^*$ is set too small there will be no feasible solution.

In practice there are also costs which vary positively with time, some are indirect (capital appreciation, bank charges on loans, wages, penalties for non-completion on time, etc). These could be included in the objective function.

\[ \blacksquare \]

### 2.6 PERT — Programme Evaluation and Review Technique

This is an alternative to CPM, useful when estimates of completion times are uncertain. Typically establish optimistic, most likely and pessimistic estimates of time of a task. Note that in these methods the graph would be updated as the project proceeds. This means that the critical paths may change from day to day.

**COMMENTS**

1. Minimum cost occurs when $T^* \geq$ minimum completion time, $T$ got from original ‘normal’ (no crash) formulation. In this case (i.e., large enough $T$) it is clear that $x_{ij} = d_{ij}$, because $m_{ij} < 0$, and so decreasing $x_{ij}$ will increase total cost.

2. It is not immediately obvious why solving the LP with $T^* < T$ should give completion time $t_i$, because they do not appear in the objective function. The reason they do is the constraint $t_{final} \leq T^*$ coupled with the constraints $t_i + x_{ij} \leq t_j$ forms a sequence of values for the values of $t_i$ given $t_j$ backward through the project planning graph.

3. Also note that the constraints satisfied by equality (active constraints) at the optimal solution for $x_{ij}$, will correspond to the critical path(s). Curious fact, it is usually the case when $T^* < T$ that every path is a critical path. The reason for this is that if $t_i + x_{ij} \leq t_j$ is not satisfied by equality, then $x_{ij}$ can be increased to reduce the cost. The occasion when a path is not critical is when completion time of a path is $< T^*$, even when operating at the normal point.
4. As $T^*$ is decreased the total cost increases in piecewise linear fashion. The kinks occur when a task hits its crash point. The LP finds a task with cheapest cost slope and squeezes it to its crash point. For example:

<table>
<thead>
<tr>
<th>Task</th>
<th>$D_{ij}$</th>
<th>$D_{ij}^*$</th>
<th>$-m_{ij}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Using the data above consider what happens as $T^*$ is squeezed. See lecture diagrams.

**Observations.**

- Squeezing $T^*$ creates multiple critical paths.
- Squeezing stops when a critical path has all tasks at crash limits.
- When multiple critical paths, have to squeeze a task in each path. Always squeeze one of least cost first.
- Multiple critical paths implies that we have increased risk of delays causing problems. That is, there are more critical tasks to cause delay if things go wrong.
- Furthermore, with $T^*$ squeezed, delays are more expensive, spent extra money to finish early only have to wait for delayed tasks.
- Have to balance importance of $T^*$ against risk (weather) and cost of delay. Game theory?
- Penalties on delays (past a certain date) create Mixed Integer Programming (MIP) as we need a variable which is zero or one to turn penalty off or on.

**General principle** Over optimizing can be to one’s detriment when risks are not taken into account, e.g., Californian power crisis, South East Aust. power crisis.

**Final comment on scheduling.** There are many scheduling algorithms tailored to specific situations. For example, minimise the total processing time of $N$ jobs on two machines. Suppose there are six jobs which require time on both of two machines A and B. The times each job requires on each machine are different. Assume machine A must be used before machine B on each job. Make a table of the tasks.

<table>
<thead>
<tr>
<th>Job</th>
<th>Time A</th>
<th>Time B</th>
<th>$order$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

**Algorithm**
1. Find the smallest time in table, ties broken arbitrarily.

2. If this time is for machine A, then schedule first for A.

   If this time is for machine B, then schedule last for B (within previous lasts).

3. Cross this line off the table and start again on remaining table, building schedule towards the centre of table below.

\[
\begin{array}{c|ccc}
A & 6, 5, 4 \\
B & 3, 2, 1 \\
\end{array}
\]

Basically what happens is that A gets started with smallest job, so that delay to start on B is smallest. The smallest A jobs go through first, the smaller B jobs last, de-facto large B jobs first. In this way there is no waiting on machine B.
Chapter 3

Integer and Mixed Integer Programming (IP or MIP)

Consider

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Ax \leq b, \\
& \quad x \geq 0.
\end{align*}
\]

- Pure integer LP: all \( x_i \) are integers.
- Mixed integer LP: some \( x_i \) are integers.
- Binary (0-1) LP (BIP/BLP): \( x_i \in \{0, 1\} \) for all \( i \).

3.1 Various examples

**Examples 1.** A company has two products A and B. Each of them needs three operations: molding, assembly and polishing. The times in hours, maximum resources/week and profit per unit are given in the following table:

<table>
<thead>
<tr>
<th></th>
<th>molding</th>
<th>assembly</th>
<th>polishing</th>
<th>profit/unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td>25</td>
</tr>
<tr>
<td>B</td>
<td>0.7</td>
<td>0.1</td>
<td>0.5</td>
<td>40</td>
</tr>
<tr>
<td>capacity</td>
<td>250 hrs</td>
<td>100 hrs</td>
<td>150 hrs</td>
<td></td>
</tr>
</tbody>
</table>

**Question:** How many A and B the company needs to produce per week in order to maximize the total profit.

**Solution.** Let \( x_1 \) and \( x_2 \) be respectively the numbers of units for A and B. Then,

\[
\begin{align*}
\text{maximize} & \quad z = 25x_1 + 40x_2 \\
\text{subject to} & \quad 0.3x_1 + 0.7x_2 \leq 250 \\
& \quad 0.2x_1 + 0.1x_2 \leq 100 \\
& \quad 0.3x_1 + 0.5x_2 \leq 150 \\
& \quad x_1, x_2 \text{ are non-negative integers.}
\end{align*}
\]
Solution of this problem is $x_1 = 500$, $x_2 = 0$ and $z = 12,500$.

**Examples 2.** A hotline service divides one day (16 hours) into 8 periods. One staff member is required to work for 4 periods per day. The minimum number of staff in each period is given in the following table:

<table>
<thead>
<tr>
<th>period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>min no.</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>8</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

**Question:** find the minimum number of total staff members so that the requirements in the table are met.

**Solution.** Let $x_i$ be the number of staff in period $i$, $i = 1, 2, ..., 5$, since each staff is required to work for 4 periods $i, i+1, i+2, i+3$. So, we have

$$
\text{maximize } \quad z = \sum_{i=1}^{5} x_i \\
\text{subject to } \quad x_1 \geq 10, \\
\quad \quad \quad \quad \quad \quad x_1 + x_2 \geq 8, \\
\quad \quad \quad \quad \quad \quad x_1 + x_2 + x_3 \geq 9, \\
\quad \quad \quad \quad \quad \quad x_1 + x_2 + x_3 + x_4 \geq 11, \\
\quad \quad \quad \quad \quad \quad x_2 + x_3 + x_4 + x_5 \geq 13, \\
\quad \quad \quad \quad \quad \quad x_3 + x_4 + x_5 \geq 8, \\
\quad \quad \quad \quad \quad \quad x_4 + x_5 \geq 5, \\
\quad \quad \quad \quad \quad \quad x_5 \geq 3, \\
\quad \quad \quad \quad \quad \quad x_i \text{ are non-negative integers.}
$$

Solution to this problem is $(10, 5, 3, 2, 3)$ and $z = 23$.

**Examples 3.** A company has 2 factories A1 and A2 producing a product. This product is shipped to 4 stores B1, B2, B3 and B4. The company plans to set up another factory and has two choices A3 and A4. The shipping cost per KT from one factory to a store is given in the table.

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>capacity (KT/yr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>20</td>
<td>90</td>
<td>30</td>
<td>40</td>
<td>400</td>
</tr>
<tr>
<td>A2</td>
<td>80</td>
<td>30</td>
<td>50</td>
<td>70</td>
<td>600</td>
</tr>
<tr>
<td>A3</td>
<td>70</td>
<td>60</td>
<td>10</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>A4</td>
<td>40</td>
<td>50</td>
<td>20</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>demand (KT)</td>
<td>350</td>
<td>400</td>
<td>300</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

The set-up costs for A3 and A4 are $12M and $15M respectively.

**Question:** which of A3 and A4 is to be chosen and how many KTs are to be shipped from Ai to Bj for $i, j = 1,2,3,4$.

**Solution.** Let $x_{ij}$ be the number of KTs shipped from Ai to Bj and $y \in \{0, 1\}$ such that
\( y = 1 \) if A3 is chosen or 0 if A4 is chosen. We denote the unit shipping cost by \( c_{ij} \). Then

\[
\text{maximize } \quad z = \sum_{i,j=1}^{4} c_{ij}x_{ij} + [12,000y + 15000(1 - y)]
\]

subject to
\[
\begin{align*}
\sum_{i=1}^{4} x_{i1} & \geq 350, \\
\sum_{i=1}^{4} x_{i2} & \geq 400, \\
\sum_{i=1}^{4} x_{i3} & \geq 300 \\
\sum_{i=1}^{4} x_{i4} & \geq 150, \\
\sum_{j=1}^{4} x_{1j} & \leq 400, \\
\sum_{j=1}^{4} x_{2j} & \leq 600, \\
\sum_{j=1}^{4} x_{3j} & \leq 200y, \\
\sum_{j=1}^{4} x_{4j} & \leq 200(1 - y), \\
x_5 & \geq 3, \\
x_i & \geq 0 \text{ and are integers}, \\
y & \in \{0, 1\}.
\end{align*}
\]

Alternatively, we may also introduce two binary variables \( y_1 \) and \( y_2 \) such that \( y_1 = 1 \) (\( y_2 = 1 \)) if A3 (A4) is chosen and 0 otherwise.

**Job allocation.**

Suppose we have \( N \) jobs to fill using \( N \) people, or aircraft to fit routes, etc. Let

\[
x_{ij} = \begin{cases} 1 & \text{person } i \text{ takes job } j \\ 0 & \text{otherwise} \end{cases}, \quad i, j = 1, 2, \ldots, N
\]

Suppose the fitness of person \( i \) to job \( j \) is \( f_{ij} \) for \( i, j = 1, \ldots, N \). Then, the problem to optimize the total fitness is

\[
\text{maximize } \quad \sum_{i,j} f_{ij}x_{ij},
\]

subject to
\[
\begin{align*}
\sum_{i} x_{ij} & = 1, \quad j = 1, 2, \ldots, N, \\
\sum_{j} x_{ij} & = 1, \quad i = 1, 2, \ldots, N.
\end{align*}
\]
This is a BLP, but there are some special algorithms for it, because of the special constraints.

**Airline schedule for aircraft.**

Suppose we have a fixed number of planes. We have a number of possible routes that the planes could fly to cover getting passengers from place to place on the routes. Let the places on the possible routes be called $A$, $B$, $C$, $D$. All routes start at $A$ and finish at $A$, the home port. There are 8 possible routes to consider:

1. $A \to B \to A$
2. $A \to B \to C \to A$
3. $A \to B \to C \to B \to A$
4. $A \to C \to A$
5. $A \to C \to B \to A$
6. $A \to C \to B \to C \to A$
7. $A \to D \to C \to A$
8. $A \to D \to C \to B \to A$

Suppose the cost of running a route is the number of takeoff and landings. Hence the cost of getting passengers from place to place, depends on what route they fly. Hence the following table showing place to place leg number on routes and of costs for routes.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \to B$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_1 + x_2 + x_3 \geq 1$</td>
</tr>
<tr>
<td>$A \to C$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_4 + x_5 + x_6 \geq 1$</td>
</tr>
<tr>
<td>$A \to D$</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_7 + x_8 \geq 1$</td>
</tr>
<tr>
<td>$B \to A$</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_1 + x_3 + x_5 + x_8 \geq 1$</td>
</tr>
<tr>
<td>$B \to C$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_2 + x_3 + x_6 \geq 1$</td>
</tr>
<tr>
<td>$C \to A$</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_2 + x_4 + x_6 + x_7 \geq 1$</td>
</tr>
<tr>
<td>$C \to B$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_3 + x_5 + x_6 + x_8 \geq 1$</td>
</tr>
<tr>
<td>$D \to C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$x_7 + x_8 \geq 1$</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Let $x_k$, $k = 1, \ldots, 8$ be decision variables as to whether route $k$ is flown. We want to cover the place to place combinations in the table, hence the constraints on the right of the table. We want to do it with only three planes, so $\sum_{i=1}^8 x_i = 3$ is an equality constraint. The objective is to minimize the cost given by the total number of takeoff and landings,

$$\min_x 2x_1 + 3x_2 + 4x_3 + 2x_4 + 3x_5 + 4x_6 + 3x_7 + 4x_8.$$ 

Obviously different cost or profit criteria could be used if likely passenger numbers are known. This is good for testing feasibility on the number of aircraft to use, that is, can the place to place combinations be covered with 3 planes?

**3.2 Tricks with binary variables**

**Simple either/or constraints.**

The standard constraint set up for optimization problems has an implicit AND between each constraint. If each constraint by itself forms a convex region then the AND of any
number of these form a convex region, making it relatively easy to find optimal points. Consider the feasible region defined by \((A \cup B \cup C \text{ in diagram})

\[ x_1 \geq 0 \text{ AND } x_2 \geq 0 \text{ AND (EITHER } 3x_1 + 2x_2 \leq 18 \text{ OR } x_1 + 4x_2 \leq 16\).\]

If \(M\) is a large enough number then this is equivalent to

\[ x_1 \geq 0 \text{ AND } x_2 \geq 0 \text{ AND (EITHER } [3x_1 + 2x_2 \leq 18 \text{ AND } x_1 + 4x_2 \leq 16 + M] \text{ OR } [3x_1 + 2x_2 \leq 18 + M \text{ AND } x_1 + 4x_2 \leq 16]\).

Hence we get

\[ x_1 \geq 0 \text{ AND } x_2 \geq 0 \text{ AND } 3x_1 + 2x_2 \leq 18 + My \text{ AND } x_1 + 4x_2 \leq 16 + M(1 - y) \quad y \in \{0, 1\}.\]

Or creating two binary variables

\[ x_1 \geq 0 \text{ AND } x_2 \geq 0 \text{ AND } 3x_1 + 2x_2 \leq 18 + My_1 \text{ AND } x_1 + 4x_2 \leq 16 + My_2, \quad y_1 + y_2 = 1.\]

**K out of N inequality constraints must be satisfied.**

Given \(N\) possible constraints

\[ f_i(x) \leq 0, \quad i = 1, 2, \ldots, N, \]

we need to make sure that \(K(\leq N)\) of them hold. Introduce \(N\) binary variables, \(y_i, i = 1, \ldots, N\). Replace the \(N\) constraints with

\[ f_i(x) \leq y_i M, \quad \sum_{i=1}^{N} y_i = N - K, \quad y_i \in \{0, 1\}, \quad \forall i = 1, 2, \ldots, N,\]

where \(M \gg 0\) is a positive constant. Clearly, when \(y_i = 0\), the \(i\)th constraint is 'ON'. Otherwise, it is 'OFF'.

**Functions with \(N\) possible values**

\[ f(x) \in \{d_1, d_2, \ldots, d_N\}. \]
Introduce \( y_i \in \{0, 1\}, \ i = 1, 2, ..., N \). The above is equivalent to

\[
f(x) = \sum_{i=1}^{N} d_i y_i, \quad \sum_{i=1}^{N} y_i = 1, \quad y_i \in \{0, 1\}, \ i = 1, 2, ..., N.
\]

**Fixed setup charges on positive production.**

In a production problem it often happens that if you choose to make product \( j \) it incurs a fixed setup cost \( K_j \) and unit cost \( C_j \). So the total cost of product \( j \) is \( K_j + C_j x_j \) if \( x_j > 0 \) and zero if \( x_j = 0 \). Total cost over all items is then

\[
\sum_{j=1}^{N} (C_j x_j + K_j y_j)
\]

where

\[
y_j = \begin{cases} 
1, & x_j > 0, \\
0, & x_j = 0.
\end{cases}
\]

To handle the relationship between \( x_j \) and \( y_j \), let \( M \) be large (greater than maximum that \( x_j \) could be), and use the constraint \( x_j \leq M y_j \). Note that \( y_j \leq x_j \) does not work.

**Binary representations of general integer variables**

Consider and integer variable \( x \) satisfying

\[
0 \leq x \leq u \text{ with } 2^N \leq u \leq 2^{N+1}
\]

for some integer \( N \). We introduce \( y_i \in \{0, 1\}, \ i = 0, 1, ..., N \) and put

\[
x = \sum_{i=0}^{N} 2^i y_i.
\]

**Example.** Consider the constraints \( x_1 \leq 5, \ 2x_1 + 3x_2 \leq 30, \ x_1, x_2 \geq 0 \) and \( x_1 \) and \( x_2 \) are integers.

**Solution.** From \( x_1 \leq 5 \) we see that \( u_1 = 5 \). From \( 2x_1 + 3x_2 \leq 30 \) we have

\[
x_2 \leq \frac{1}{3}(30 - 2x_1) \leq 10 \text{ since } x_1 \geq 0.
\]

Therefore, \( u_2 = 10 \). Since \( 2^2 < 5 < 2^3 \) and \( 2^3 < 10 < 2^4 \), we introduce binary variables \( y_0, y_1, y_2 \) for \( x_1 \) and \( y_i, i = 3, ..., 6 \) for \( x_2 \) so that

\[
x_1 = y_0 + 2y_1 + 4y_2, \quad x_2 = \sum_{i=3}^{6} 2^{i-3} y_i.
\]

Therefore, the constraints become

\[
y_0 + 2y_1 + 4y_2 \leq 5, \\
y_3 + 2y_4 + 4y_5 + 8y_6 \leq 10, \\
y_i \in \{0, 1\}, \ i = 0, 1, ..., 6.
\]
3.3 Branch-and-Bound Method for IPs

We may think of solving an IP by the following simple ideas.

- Exhaustive search: For BIP, there is a finite number of points. For example, The number of feasible point for \( n \) variables equals \( 2^n \). OK if \( n \) is small, but the number grows exponentially as \( n \) grows. The problem is NP-hard.

- Rounding the LP solution to the nearest integer values. This often gives infeasible solutions or solution far from the true ones. See examples below.

Example 1. maximize \( z = x_1 + 5x_2 \), subject to \( x_1 \geq 0, x_2 \geq 0, x_1, x_2 \) integer, \(-x_1 + x_2 \leq 1/2 \) and \( x_1 + x_2 \leq 3/2 \). The real LP solution is \( (x_1, x_2) = (3/2, 2) \), with \( z = 11 \frac{1}{2} \). But neither \((1, 2)\) nor \((2, 2)\) are feasible. The IP solution is at \((2, 1)\) with \( z = 7 \). See left diagram in Figure 3.3.3.1.

Example 2. Same objective as in Example 1 but constraints are now \( x_1 + 10x_2 \leq 20 \) and \( x_1 \leq 2 \). The real LP solution is \( (x_1, x_2) = (2, 9/5) \) with \( z = 11 \). Rounded solutions are \((2, 2)\) not feasible, and \((2, 1)\) feasible, \( z = 7 \), but not optimal. The optimal solution is at \((0, 2)\), with \( z = 10 \). See right diagram in Figure 3.3.3.1.

Bounding: Whatever the LP optimal \( z \) value is, it is a bound on the integer solution. If a max problem, an upper bound on the true solution is the LP \( z \) value, rounded down if objective coefficients are integer. To see this look at the contours (dotted) of the second example, where the LP solution has higher value than the IP solution. If a min problem, a lower bound is the LP \( z \) value, rounded up if objective coefficients are integer.

3.3.1 Branch and Bound Algorithm

Suppose we have an optimization problem \( P \) which has an optimal objective value \( z \). If we add a further constraint to \( P \) to form \( P^* \), then the optimal value of \( P^* \), call it \( z^* \), is less than or equally optimal to \( z \).

We will explain the algorithm using the example of factory and warehouse placement.
Example BIP. Consider

$$\begin{align*}
\max \quad & z = 9x_1 + 5x_2 + 6x_3 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 6 & 3 & 5 & 2 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} x \leq \begin{pmatrix} 10 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \\
x_i \in \{0, 1\}, i = 1, 2, 3, 4.
\end{align*}$$

The solution to the LP with each \(x_i \in [0, 1]\) is \(x = (5/6, 1, 0, 1)\), \(z = 16\frac{1}{2}\). Hence a bound on the BP objective is 16 as all cost coefficients are integer.

Iteration 1.

1. Branching. As \(x_1\) is binary, either 0 or 1, consider the two LP’s with the extra constraint \(x_1 = 0\) or \(x_1 = 1\). We will denote each subproblem according to the values chosen for each variable, where \(X\) denotes a possible real value.

Subproblem 1, \(x_1 = 0\):

$$\begin{align*}
\max \quad & z = 5x_2 + 6x_3 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 0 & 3 & 5 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} x \leq \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \\
x_i \in \{0, 1\}, i = 2, 3, 4.
\end{align*}$$

Solving the LP gives a solution \((0, 1, 0, 1)\) with \(z = 9\).

Subproblem 2, \(x_1 = 1\):

$$\begin{align*}
\max \quad & z = 9 + 5x_2 + 6x_3 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 0 & 3 & 5 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} x \leq \begin{pmatrix} 4 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \\
x_i \in \{0, 1\}, i = 2, 3, 4.
\end{align*}$$

Solving the LP gives a solution \((1, 4/5, 0, 4/5)\) with \(z = 16\frac{1}{5}\).

2. Bounding. We use the LP solutions to bound how good the solution can be for each subproblem. This enables use to dispense with some branches because they cannot yield a better solution than what we have. For subproblem 1, all subproblems with \(x_1 = 0\) have a equal or lower value than 9. For subproblem 2 all subproblems with \(x_1 = 1\) must have value less or equal to 16, as the cost function has integer coefficients.

3. Fathoming. Observe that subproblem 1 has binary optimal solution, so there is no need to branch from this point. We say this branch of the tree is fathomed with depth 1. We also know that the BP solution to the original problem is bounded by
9 and 16. Any further branching must refine these bounds, and any values falling outside these bounds may be ignored. This represents another way to fathom a branch, that is, the upper bound on \( z \) is less than the value of \( z \) obtained on another fathomed branch.

4. Incumbent. The solution to 1 is called the incumbent solution, that is, the best so far. We can do at least this good with further branching and bounding.

**Iteration 2.** As \( x_1 = 0 \) is fathomed, we proceed keeping \( x_1 = 1 \).

**Subproblem 3, \( x_1 = 1, x_2 = 0 \):**

\[
\begin{align*}
\max \quad & z = 9 + 0 + 6x_3 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 0 & 0 & 5 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 4 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \\
x_i \in \{0, 1\}, i = 3, 4.
\end{align*}
\]

Solving the LP gives a solution \((1, 0, 4/5, 0)\) with \( z = 13\frac{4}{5} \).

**Subproblem 3, \( x_1 = 1, x_2 = 1 \):**

\[
\begin{align*}
\max \quad & z = 9 + 5 + 6x_3 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \\
x_i \in \{0, 1\}, i = 3, 4.
\end{align*}
\]

Solving the LP gives a solution \((1, 1, 0, 1/2)\) with \( z = 16 \).

Bounding. We know that the best we can do with \( x_1 = 1 \) and \( x_2 = 0 \) is 13, while with \( x_1 = 1 \) and \( x_2 = 1 \) the best is 16.

Fathoming. No fathoming is possible at this step as neither subproblem is binary valued with bound less than 9.

Incumbent. No Change.

**Iteration 3.**

1. Branching.

**Subproblem 5, \( x_1 = 1, x_2 = 1, x_3 = 0 \):**

\[
\begin{align*}
\max \quad & z = 14 + 4x_4 \\
\text{such that} \quad & \begin{pmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_4 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \\
x_4 \in \{0, 1\}.
\end{align*}
\]
Subproblem 6, \( x_1 = 1, x_2 = 1, x_3 = 1 \):

\[
\begin{align*}
\text{max} & \quad z = 20 + 4x_4 \\
\text{such that} & \quad \begin{pmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} x \leq \begin{pmatrix} -4 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \\
& \quad x_4 \in \{0, 1\}.
\end{align*}
\]

Solving the LP relaxation of Subproblem 5 gives \((1, 1, 0, 1/2), z = 16\). and subproblem 6 does not have a solution.

2. Bounding. No changes.

3. Fathoming. Subproblem 6 is fathomed as all problems with \( x_1 = x_2 = x_3 = 1 \) are infeasible.

4. Incumbent, no changes.

**Iteration 4.**

1. Branching. Only \( x_4 \) remains un-branched.
   - \( x_4 = 0 \), we have a feasible solution \((1, 1, 0, 0)\) with \( z = 14 \).
   - \( x_4 = 1 \), infeasible solution \((1,1,0,1)\) since \(6x_1 + 3x_2 + 5x_3 + 2x_4 = 11 > 10\).

2. The IP contender of 14 is better than the existing incumbent (with 9) so change the incumbent to solution of subproblem.

**Iteration 5.** As all subproblems have been fathomed, we can declare that \((1, 1, 0, 0)\) is the solution with \( z = 14 \).

The above process can be demonstrated by Figure 3.3.3.2 in which \( F(i) \) denotes fathoming according to rule i given below in the summary of B-a-B method.

**Summary of Branch and Bound for BP.**

Aim: To solve BP (BIP) \( \max_c, P^T x, Ax \leq b, x \text{ binary} \).

Definitions: let \( x^* \) and \( z^* = p^T x^* \) be the best feasible solution to BP at any time (incumbent).
0. Initiation: Let $x^* = \emptyset$ and $z^* = -\infty$. Solve the problem as a LP, replacing $x \in \{0, 1\}$ with $x \in [0, 1]$. If the solution is binary, STOP.

1. Branching. Select a non-binary valued variable to branch on. Normally these would be taken in some predetermined order. Construct two subproblems with extra constraint $x_i = 0$ and $x_i = 1$ respectively.

2. Bounding. Solve the two subproblems as LP’s and round down if $p$ is integer valued.

3. Fathoming. Apply the following tests to each of the currently unfathomed subproblems, including any previously generated branches.
   a. $z \leq z^*$.
   b. LP subproblem has no feasible solution.
   c. LP solution is binary.

If any test is true the subproblem branch is called fathomed and no further branching from this subproblem is necessary.

If case (c) occurs and $z \geq z^*$, then set $z^* = z$ and $x^*$ the subproblem solution. It is the new best solution.

If $z^*$ is increased should go back and check 3(a) again for previously checked branches.

4. Loop or Terminate. If there exist unfathomed subproblems (branches), then return to step 1. Selecting the subproblem with largest bound $z$ is usually best. If all branches are fathomed, STOP.

### 3.3.2 Branch and Bound for general IP.

The problem is $\max_x p^T x$, $Ax \leq b$, $x$ integer valued, usually restricted to a (very) finite set for each component. We can change this problem into a BP problem if all components of $x$ are bounded by say $2^{m+1} - 1$ and non-negative. (Can always change a finite interval to be positive using a translation of variable.) We can represent each component of $x$ by $m + 1$ new binary variables using the binary representation where the digits are either zero or one.

$$x_i = \sum_{j=0}^{m} y_{ij} 2^j .$$

Sorting this out gives an equivalent BP

$$\max_y \hat{p}^T y, \quad \hat{A} y \leq \hat{b}, \quad y \text{ binary}.$$ 

An alternative is to use the B&B algorithm with appropriate changes. For example we cannot just take the neighbouring integers to a non-integer real. The branching process instead of being $x_i = 0$ and $x_i = 1$ is now $x_i \leq [v_i]$ and $x_i \geq [v_i] + 1$. We still add constraints but they are now inequality not equality.
Example IP.

$$\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & \begin{pmatrix} 1 & 9/14 \\ -2 & 1 \end{pmatrix} x \leq \begin{pmatrix} 51/14 \\ 1/3 \end{pmatrix}, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}$$

Solution. Solving the LP relaxation gives $(3/2, 10/3)$ with $z = 29/6$. Clearly, it is not an integer solution.
Branching on $x_1 = 3/2$: $x_1 \leq 1$ or $x_1 \geq 2$.

Subproblem S1:

$$\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \geq 2, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}$$

Solving the LP gives $(2, 23/9)$ with $z = 41/9$.

Subproblem S2:

$$\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \leq 1, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}$$

Solving the LP gives $(1, 7/2)$ with $z = 10/3$.
Neither is an integer solution, but $41/9 > 10/3$.

We now consider S1 and branch on $x_2$ by $x_2 \geq 3$ and $x_2 \leq 2$.

Subproblem S11:

$$\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \geq 2, \\
& x_2 \geq 3, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}$$

No feasible solution to S11. So, it is fathomed.
Subproblem S12:

\[
\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \geq 2, \\
& x_2 \leq 2, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}
\]

Solution is \((33/14, 2), z = 61/14\).

We have two branches left: S2 and S12. Since \(61/14 > 10/3\), we branch on the latter by \(x_1 \geq 3\) and \(x_1 \leq 2\).

**Subproblem S121:**

\[
\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \geq 2, \\
& x_2 \leq 2, \\
& x_1 \geq 3, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}
\]

Optimal solution: \((3,1), z = 4\).

**Subproblem S122:**

\[
\begin{align*}
\text{max} \quad & z = x_1 + x_2 \\
\text{subject to} \quad & x_1 + \frac{9}{14} x_2 \leq \frac{51}{14}, \\
& -2x_1 + x_2 \leq \frac{1}{3}, \\
& x_1 \geq 2, \\
& x_2 \leq 2, \\
& x_1 \leq 2, \\
& x_1, x_2 \text{ are non-negative integers.}
\end{align*}
\]

Optimal solution: \((2,2), z = 4\). Since \(4 > 10/3\), we fathom S2. So, two optimal solution given in S121 and S122.

Figure 3.3.3.3 shows the above process with bounds and branches.

### 3.3.3 Branch and Bound for MIP.

Problem is \(\max_{x,y} p^T x + q^T y, A[x; y] \leq b, x \text{ integer}, y \text{ real.}\)
Figure 3.3.3.3: Branch and bound process for solving Example IP.

This translates straight across to B&B algorithm, where only the integer variables are branched on. Of course can no longer round down to nearest integer value for $z$. Ditto the condition on getting integer solutions only applies to the integer variables. In effect the real variables are just carried through.

**Example MIP.**

$$\begin{align*}
\text{max} & \quad z = 4x_1 - 2x_2 + 7x_3 - x_4 \\
\text{subject to} & \quad x_1 + \frac{9}{14}x_2 \leq \frac{51}{14}, \\
& \quad x_1 + 5x_3 \leq 10, \\
& \quad x_1 + x_2 - x_3 \leq 1, \\
& \quad 6x_1 - 5x_2 \leq 0, \\
& \quad -x_1 + 2x_3 - 2x_4 \leq 3, \\
& \quad x_j \geq 0, \ j = 1, 2, 3, 4, \\
& \quad x_j \text{ are integers}, \ j = 1, 2, 3.
\end{align*}$$

**Solution.** Initialization: $z^* = -\infty$. The LP relaxation has the solution $(5/4, 3/2, 7/4, 0)$ with $z = 14\frac{1}{4}$.

**Iteration 1.** Branch on $x_1$ by $x_1 \leq 1$ and $x_1 \geq 2$.

S1: the original problem + $x_1 \leq 1$. Solution to the LP1: $(1, 6/5, 9/5, 0), z = 14\frac{1}{5}$.

S2: the original problem + $x_1 \geq 2$. Solution: no feasible solutions. Fathomed!

Bound for S1: $z = 14\frac{1}{5}$.

**Iteration 2.** Consider subproblems of S1:

Branch $x_2$ by $x_2 \leq 1$ and $x_2 \geq 2$.

S11: S1 + $x_2 \leq 1$; solution to LP11: $(5/6, 1, 11/6, 0), z = 14\frac{1}{5}$.

S12: S1 + $x_2 \geq 2$; solution to LP12: $(5/6, 2, 11/6, 0), z = 12\frac{1}{6}$.

**Iteration 3.** We consider S12 and branch on $x_1$

S111: S11 + $x_1 \leq 0$; solution to LP111: $(0, 0, 2, 1/2), z = 13\frac{1}{2}$. Feasible solution; fathomed!

S112: S11 + $x_1 \geq 1$; solution to LP112: no feasible solutions. Fathomed!

Now, we have a feasible solution and the incumbent solution is $z^* = z = 13\frac{1}{2}$ and $x^* = (0, 0, 2, 1/2)$.
The only one branch which is alive is S12. Since $\frac{121}{6} < z^*$, we fathom S12. Therefore, the problem is solved.

### 3.3.4 Some notes

Efficiency. BP B&B algorithm as described works well for up to 100 variables, after which it can become slow (impossible). Remember that at each branch two LPs have to be solved in up to the 100 variables. Since the 1980’s there have been a number of improvements (heuristics even) taking the number of variables up to 1000s.

One improvement is to carefully monitor the constraints and note the obvious (get computer code to recognise and act upon). For example in BP, $2x_2 \leq 1$ implies $x_2 = 0$, $x_2 \leq 2$ can be ignored. Even more complicated constraints like $2x_1 + 3x_2 + 4x_3 \leq 2$ means that $x_2 = x_3 = 0$.

**Bottleneck problem.** This is a variant of the assignment problem (people to jobs). Let $C_{ij}$ be the cost of assigning task $i$ to operator $j$, $i, j = 1, \ldots, N$. It is usual to think of $C_{ij}$ as completion time. Assign one task only to each operator, so if

$$x_{ij} = \begin{cases} 1, & \text{if } i \text{ assigned to } j, \\ 0, & \text{otherwise}, \end{cases}$$

we have constraints

$$\sum_i x_{ij} = 1 \quad \text{and} \quad \sum_j x_{ij} = 1.$$ 

The objective is to minimize $z = \max_{i,j} C_{ij}x_{ij}$, that is, minimize the longest time of any assigned task, or finish all the tasks in minimum time. Not obviously a BP, but if we add a variable $z$ which is real, add constraints

$$C_{ij}x_{ij} \leq \alpha \quad \forall i, j,$$

and now minimize $\alpha$ over variables $x_{ij}$ and $\alpha$, subject to all constraints.

**General Branch and Bound algorithm**

- Let $S$ be a finite set, e.g., $S \subset \mathbb{Z}^n$ of finite size.
- Let $f: S \to \{\mathbb{R}, \infty\}$.
- Objective is to $\min_{x \in S} z = f(x)$.
• Algorithm:

0. Set $z^* = \infty$ and $x^*$ is undefined. Let $A = S$.
1. Branch: Portion $A$ into at least two disjoint subsets so that

$$\bigcup_i A_i = A, \quad A_i \cap A_j = \emptyset, \quad i \neq j.$$ 

2. Bound: For each subset $A_i$, obtain a lower bound $z_i$, i.e., $z_i \leq f(x)$, for all $x \in A_i$. If $A_i = \emptyset$, then put $z_i = \infty$.

3. Fathom: Eliminate (fathom) a currently unfathomed subset $A_i$ (i.e., this includes subsets generated in previous branching steps and not previously fathomed and eliminated) if any of the following three conditions are satisfied.

   a. $z_i = \infty$.
   b. $z_i \geq z^*$.
   c. There exists $x_i \in A_i$ with $f(x_i) = z_i$.

   If test (c) is satisfied with $z_i < z^*$ then set $z^* = z_i$, $x^* = x_i$ and retest all currently unfathomed subsets.

4. If there exists any unfathomed subsets then select one $A_i$, set $A = A_i$ and goto 1,
   else optimum has been found and it is $x^*$.

The key step is the bound step 2, where all the work is done.

3.4 The Cutting Plane Method (Gomory 1958)

3.4.1 Motivation

Consider the IP:

$$\max c^T x, \quad \text{subject to} \quad Ax = b,$$

$$x \geq 0 \quad \text{and integer.}$$

Here $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. We assume that all $a_{ij}$ and $b_j$ are integers.

**Solution procedure:**

1. Solve the IP as an LP relaxation.

2. If one of the basic variables is non-integer, generate a new constraint to tighten the feasible region of the LP relaxation and solve the new relaxed problem.

3. Continue the above steps until all the basic variables are integers.
Question: How to construct a new constraint (called a cut)?

Suppose we have the optimal solution $x^*$ for the LP relaxation, i.e.,

$$x^* = \begin{cases} b'_j, & j \in Q, \\ 0, & j \in K, \end{cases}$$

where $Q$ and $K$ denote respectively the index sets of basic and non-basic variables. Let $B$ be the $m \times m$ matrix containing the columns of $A$ corresponding to $Q$. Then

$$A^{-1}Ax = B^{-1}b \Rightarrow (I + N)x = b',$$

where $I$ denotes the $m \times m$ identity and $N$ is an $m \times (n - m)$ matrix. Where $x = x^*$, we have the optimal feasible solution. Now, suppose one of the components of $x^*$ is non-integer, say $x_k$ for a $k \in Q$. Then, the $k$th equation in (3.3.4.1) is

$$x_k + \sum_{j \in K} a'_{kj}x_j = b'_k,$$  

(3.3.4.2)

where $a'_{kj}$ is the element in $N$.

Let $[p]$ denote the integer part of $p$ (i.e., the floor function). We decompose $a'_{kj}$ and $b'_k$ into

$$a'_{kj} = [a'_{kj}] + f_{kj}, \quad b'_k = [b'_k] + f_k,$$

(3.3.4.3)

so that $0 \leq f_{kj} < 1$ and $0 < f_k < 1$. Using these, (3.3.4.2) then becomes

$$x_k + \sum_{j \in K} [a'_{kj}]x_j - [b'_k] = f_k - \sum_{j \in K} f_{kj}x_j.$$  

(3.3.4.4)

Since $f_k \in (0,1)$ and $f_{kj} \in [0,1)$ for all $j \in K$ and $x_j \geq 0$, we have

$$f_k - \sum_{j \in K} f_{kj}x_j < 1.$$  

If we wish to find a new solution such that the LHS of (3.3.4.4) becomes an integer, then

$$f_k - \sum_{j \in K} f_{kj}x_j \leq 0.$$  

(3.3.4.5)

This is a new constraint, or a cutting plane.

### 3.4.2 Properties of the cutting plane

The cutting plane given in (3.3.4.5) has the following two properties.

**Theorem 3.1** The LP optimal feasible solution does not satisfy (3.3.4.5).

**Proof.** Since $x_j = 0$ for all $j \in K$, we have from (3.3.4.5) $f_k \leq 0$, contradicting the fact that $0 < f_k < 1$. \qed

**Theorem 3.2** Any feasible solution of the IP is in the new feasible region satisfying (3.3.4.5).
PROOF. Let \( y = (y_1, y_2, \ldots, y_n)^T \) be a feasible solution of the IP. Then, all \( y_i \)'s are integers and satisfy
\[
y_k + \sum_{j \in K} a'_{kj} y_j = b'_k, \quad \forall k \in Q.
\]
Using an argument similar to that for (3.3.4.4), we have
\[
y_k + \sum_{j \in K} \lfloor a'_{kj} \rfloor y_j - \lfloor b'_k \rfloor = f_k - \sum_{j \in K} f_{kj} y_j, \quad (3.3.4.6)
\]
where \( f_{kj} \) and \( f_k \) are defined in (3.3.4.3). Note that the LHS of (3.3.4.6) is an integer, so is the RHS of (3.3.4.6). This implies
\[
f_k - \sum_{j \in K} f_{kj} y_j \leq 0,
\]
since \( f_k \in (0, 1), f_{kj} \in [0, 1) \) and \( y_j \geq 0 \) for all feasible \( j \).

3.4.3 Example.

Solve
\[
\begin{align*}
\max & \quad z = 3x_1 - x_2, \\
\text{subject to} & \quad \begin{pmatrix} 3 & -2 \\ -5 & -4 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 3 \\ -10 \end{pmatrix}, \\
& \quad x \geq 0 \text{ and integer}.
\end{align*}
\]

Solution. Introducing slack variables \( x_3, x_4 \) and \( x_5 \), the standard form of the LP becomes
\[
\begin{align*}
\min & \quad z = -(3x_1 - x_2), \\
\text{subject to} & \quad A x = \begin{pmatrix} 3 & -2 & 1 & 0 & 0 \\ -5 & -4 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 3 \\ -10 \end{pmatrix}, \\
& \quad x \geq 0.
\end{align*}
\]

Step 1. Solving this by Matlab gives
\[
x = (1.85714285619198, 1.28571428560471, 0.00000000130913, 2.11024142448272, 0.00000000164709, 0)
\]
\[
= (13/7, 9/7, 0, 31/7, 0)^T.
\]
The basic variables are \( x_1, x_2 \) and \( x_4 \),
\[
B = \begin{pmatrix} 3 & -2 & 0 \\ -5 & -4 & 1 \\ 2 & 1 & 0 \end{pmatrix}, \quad A' = B^{-1} A = \begin{pmatrix} 1 & 0 & 0.14285714285714 & 0 & 0.28571428571429 \\ 0 & 1 & -0.28571428571429 & 0 & 0.42857142857143 \\ 0 & 0 & -0.42857142857143 & 1 & 3.14285714285714 \end{pmatrix}
\]

39
and \( b' = (1.85714285714286, 1.28571428571429, 4.42857142857143)^T \). Since \( x_1 \) is non-integer, we have

\[
b'_1 = 1 + 0.85714285619198, \quad a'_{13} = 0 + 0.14285714285714, \quad a'_{15} = 0 + 0.28571428571429.
\]

The first cut is defined by

\[
0.85714285619198 - (0.14285714285714x_3 + 0.28571428571429x_5) \leq 0.
\]

Multiplying this by 7 and rearranging give

\[
-x_3 - 2x_5 \leq -6.
\]

Note that from the constraints in the standard form we have

\[
x_3 = 3 - 3x_1 + 2x_2 \text{ and } x_5 = 5 - 2x_1 - x_2.
\]

Thus the cut becomes

\[
x_1 \leq 1. \tag{3.3.4.7}
\]

**Step 2.** Taking into account of (3.3.4.7), we define a new IP as follows.

\[
\begin{align*}
\text{max} & \quad z = 3x_1 - x_2, \\
\text{subject to} & \quad \begin{pmatrix} 3 & -2 & 1 & 0 & 0 & 0 \\ -5 & -4 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \leq \begin{pmatrix} 3 \\ -10 \\ 5 \\ 1 \end{pmatrix}, \\
x & \geq 0 \text{ and integer}.
\end{align*}
\]

The standard form of the LP relaxation is

\[
\begin{align*}
\text{min} & \quad z = -(3x_1 - x_2), \\
\text{subject to} & \quad Ax = \begin{pmatrix} 3 & -2 & 1 & 0 & 0 & 0 \\ -5 & -4 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 3 \\ -10 \\ 5 \\ 1 \end{pmatrix}, \\
x & \geq 0.
\end{align*}
\]

Solving by a programme (or by hand) gives \( x = (1.5/4, 5/2, 0, 7/4, 0)^T \), and the basic basis is

\[
B = \begin{pmatrix} 3 & -2 & 1 & 0 \\ -5 & -4 & 0 & 0 \\ 2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \Rightarrow A' = B^{-1}A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1/4 & 0 & 5/4 \\ 0 & 0 & 1 & -1/2 & 0 & -11/2 \\ 0 & 0 & 0 & 1/4 & 1 & -3/4 \end{pmatrix}.
\]

We choose \( x_5 \), then

\[
f_{54} = 1/4, \quad f_{56} = 1/4, \quad \text{and} \quad f_5 = 3/4.
\]
The 2nd cut is defined by
\[ \frac{1}{4}x_4 + \frac{1}{4}x_6 \geq \frac{3}{4}. \]

But \( x_4 = -10 + 5x_1 + 4x_2 \) and \( x_6 = 1 - x_1 \). Substituting these into the above and simplifying give
\[ -x_1 - x_2 \leq -3. \]  \hspace{1cm} (3.3.4.8)

**Step 3.** Combining the IP in Step 2 and (3.3.4.8) gives
\[
\begin{align*}
\text{max} & \quad z = 3x_1 - x_2, \\
\text{subject to} & \quad \begin{pmatrix} 3 & -2 \\ -5 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 3 \\ -10 \end{pmatrix}, \\
& \quad \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 5 \\ 1 \end{pmatrix}, \\
& \quad \begin{pmatrix} -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -3 \end{pmatrix}, \\
& \quad x \geq 0 \text{ and integer.}
\end{align*}
\]

Solving the corresponding LP relaxation of this gives \((x_1, x_2) = (1, 2)\). This is an integer solution.
Chapter 4

Dynamic Programming (Bellman 1950)

4.1 Basic concepts

Let us consider the example given in the following network.

From the graph we see the following three items.

1. The graph is divided into stages indicated by A, B, ..., E with a policy decision required for each stage.

2. Each stage has a number of states, denoted by the subscripts 1, 2, ....
   - The effect of the policy decision at each stage is to transform the current state to a state associated with the beginning of the next stage. Each node denotes a state and they are grouped as columns (stages).
   - The links form a node to nodes in the next column correspond to the possible policy decisions on which state to go to the next.
   - The value assigned to each link can be interpreted as the immediate contribution to the objective function from making that policy decision.

3. The solution procedure is designed to find an optimal policy for the overall problem.

**Proposition 4.1 (Principle of optimality for dynamic programming)** Given the current state, an optimal policy for the remaining stages is independent of the policy decisions adopted in previous stages. Therefore, the optimal immediate decision depends on only the current state and not on how you get there.
4.2 Recursive relationship

Based on the optimality principle of dynamic programming, we can define a recursive relationship between stages. We let

- \( N \) — number of stages,
- \( n \) — label for current stage, \( n = 1, 2, \ldots, N \),
- \( s_n \) — current state,
- \( x_n \) — decision variable for stage \( n \),
- \( x_n^* \) — optimal value of \( x_n \) (given \( s_n \)),
- \( f_n(s_n, x_n) \) — contribution of stages \( n, n + 1, \ldots, N \) to objective function if system starts in state \( s_n \) of stage \( n \), immediate decision is \( x_n \) and optimal decisions are made thereafter.
- \( f_n^*(s_n) = f_n(s_n, x_n^*) \).

Using these, the recursive relationship is

\[
 f_n^*(s_n) = \max_{x_n} \{f_n(s_n, x_n)\} \quad \text{or} \quad f_n^*(s_n) = \min_{x_n} \{f_n(s_n, x_n)\},
\]

where \( f_n(s_n, x_n) \) can be written in terms of \( s_n, x_n \) and \( f_{n+1}(s_{n+1}) \). Clearly, this is a backward procedure with \( f_N^*(s_N) = 0 \).

Let us solve the problem in the above figure by this method. From the figure we see \( N = 6 \). Therefore, we have the following steps.

**step 0** \( f_0(F) = 0 \).

**step 1** \( k = 5 \), \( f_5^*(E_1) = 4 \), \( f^*(E_2) = 3 \).

**step 2** \( k = 4 \), we have 3 states \( D_1, D_2, D_3 \).

\[
 f_4^*(D_1) = \min \{d(D_1, E_1) + f_5(E_1), d(D_1, E_2) + f_5(E_2)\} = \min \{3 + 4, 5 + 3\} = 7.
\]

So, \( D_1 \to E_1 \to F \), or \( x_4^*(D_1) = E_1 \). Similarly, we have

\[
 f_4^*(D_2) = \min \{d(D_2, E_1) + f_5(E_1), d(D_2, E_2) + f_5(E_2)\} = \min \{4 + 4, 2 + 3\} = 5.
\]

\( D_2 \to E_2 \to F \), and \( x_4^*(D_2) = E_2 \).

\[
 f_4^*(D_3) = \min \{d(D_3, E_1) + f_5(E_1), d(D_3, E_2) + f_5(E_2)\} = \min \{1 + 4, 3 + 3\} = 5.
\]

So, \( D_3 \to E_1 \to F \), and \( x_4^*(D_3) = E_1 \).

**step 3** \( k = 3 \), 4 states \( C_1, \ldots, C_4 \). We have

\[
 f_3^*(C_1) = 12, \quad x_3^*(C_1) = D_1, \quad C_1 \to D_1,
 f_3^*(C_2) = 10, \quad x_3^*(C_2) = D_2, \quad C_2 \to D_2,
 f_3^*(C_3) = 8, \quad x_3^*(C_3) = D_2, \quad C_3 \to D_2,
 f_3^*(C_4) = 9, \quad x_3^*(C_4) = D_3, \quad C_4 \to D_4.
\]
step 4 \( k = 2 \), 2 states \( B_1 \) and \( B_2 \).

\[
\begin{align*}
&f^*_2(B_1) = 13, \quad x^*_2(B_1) = C_2, \quad B_1 \rightarrow C_2, \\
&f^*_2(B_2) = 15, \quad x^*_2(B_2) = C_3, \quad B_2 \rightarrow C_3,
\end{align*}
\]

step 5 \( k = 1 \), one state \( A \).

\[
\begin{align*}
f_1(A) &= \min\{d(A, B_1) + f_2(B_1), d(A, B_2) + f_2(B_2)\} = \min\{4 + 13, 5 + 15\} = 17 \\
x^*_1(A) &= B_1.
\end{align*}
\]

Note the shortest distance is 17. Tracing back gives

\[ A \rightarrow B_1 \rightarrow C_2 \rightarrow D_2 \rightarrow E_2 \rightarrow F. \]

Altogether, the above contains 12 sub-steps or paths. This is in contrast to the total number of 22 possible paths.

### 4.3 Dynamic programming algorithm

\[
\begin{align*}
f^*_n(s_n) &= \text{opt}_{x_n \in D_n(s_n)} \left[v_n(s_n, x_n) + f^*_n(s_{n+1})\right], \quad n = N - 1, N - 2, \ldots, 1 \\
f^*_N(s_N) &= 0,
\end{align*}
\]

where \( x_n \) and \( x_n \) are state and decision variables respectively, and \( D_n \) is the set of feasible policies/decisions.

**Example 1.** An investor has $10K free capital and would like to invest it. There are 3 projects available and their expected return functions are respectively \( g_1(z) = 4z \), \( g_2(z) = 9z \) and \( g_3(z) = 2z^2 \). How much should the investor invest in each of these projects?

**Solution.** Let \( x_i \) denote the amount invested in the \( i \)th project for \( i = 1, 2, 3 \). Then the problem becomes

\[
\begin{align*}
\max & \quad g_1(x_1) + g_2(x) + g_3(x_3) = 4x_1 + 9x_2 + 2x_3^2, \\
\text{subj. to} & \quad x_1 + x_2 + x_3 = 10, \\
& \quad x_i \geq 0, \quad i = 1, 2, 3.
\end{align*}
\]

The problem does not have a natural 'time' order, but we can order it in terms of projects. We assume that the decision and state variables are respectively \( x_k \) and \( s_k \) for \( k = 1, 2, 3 \).

The transitions of the state and decision variables are

\[
\begin{align*}
s_1 &= 10, \quad s_2 = s_1 - x_1, \quad s_3 = s_2 - x_2.
\end{align*}
\]

Let \( f^*_k(s_k) \) be the optimal contribution of stages \( k, k + 1, \ldots, N \). Then,

\[
\begin{align*}
f^*_k(s_k) &= \max_{0 \leq x_k \leq s_k} \left[g_k(x_k) + f_{n+1}(s_{n+1})\right], \quad k = 3, 2, 1 \\
f^*_4(s_4) &= 0,
\end{align*}
\]
• $k = 3$.  
\[
 f_3^*(s_3) = \max_{0 \leq x_3 \leq s_3} \{2x_3^2 + 0\} = 2s_3^2, \quad (x_3^* = s_3).
\]

• $k = 2$.  
\[
 f_2^*(s_2) = \max_{0 \leq x_2 \leq s_2} \{9x_2 + f_3^*(s_3)\} = \max_{0 \leq x_2 \leq s_2} \{9x_2 + 2s_3^2\} = \max_{0 \leq x_2 \leq s_2} \left\{9x_2 + 2(s_2 - x_2)^2\right\}. \quad h(x_2)
\]

Differentiating, $h'(x_2) = 9 - 4(s_2 - x_2) = 0 \implies x_2 = s_2 - 9/4$. Also, $h(0) = 2s^2_2$ and $h(s_2) = 9s_2$. Note $h'(x_2) = 4 > 0$, the maximum value of $h$ is attained at either 0 or $s_2$, and the minimum is given by 
\[
h(s_2 - 9/4) = 9(s_2 - 9/4) + 2 \cdot 9^2/4^2 = 9s_2 - 81/8.
\]

1. If $h(0) = h(s_2)$, we have $2s^2_2 = 9s_2$, and so $s_2 = 9/2$ since $s_2 \neq 0$. In this case, $x_2^* = 0$ or $S_2$ and  
2. When $h(0) > h(s_2)$, we have $x_2^* = 0$, and $s_2 > 9/2$. In this case $f(x_2^*) = 2s^2_2$.  
3. When $h(0) < h(s_2)$, we have $x_2^* = s_2$, and $s_2 < 9/2$. In this case $f(x_2^*) = 9s_2$.

• $k = 1$. We have a few cases as follows.

1. When $f_2^*(s_2) = 9s_2$  
   \[
   f_1^*(s_1) = \max_{0 \leq x_1 \leq 10} \{4x_1 + f_2^*(s_2)\} = \max_{0 \leq x_1 \leq 10} \{4x_1 + 9s_2\} = \max_{0 \leq x_1 \leq 10} \{4x_1 + 9(10 - x_1)\} = 90
   \]
   when $x_1^* = 0$. Therefore, $s_2 - 10 - x_1^* = 10 > 9/2$, contradicting to the fact that $s_2 \leq 9/2$.  
2. When $f_2^*(s_2) = 2s^2_2$, we have  
   \[
   f_1^*(s_1) = \max_{0 \leq x_1 \leq 10} \left\{4x_1 + 2\left(10 - x_1\right)^2\right\}. \quad e(x_1)
   \]
   $e'(x_1) = 4 - 4(10 - x_1) = 0$ implies $x_1 = 9$. $e''(x_1) = 4 > 0$. So, this is a local minimum. For the two end-points we have when $x_1 = 0$, $f_1^*(s_1) = 200$ and when $x_1 = 10$, $f_1^*(s_1) = 40$.  

Therefore, we have  
\[
x_1^* = 0, \quad \text{and} \quad s_2 = s_1 - x_1^* = 10,
\]
\[
x_2^* = 0, \quad \text{since} \quad s_2 = 10 > 9/2,
\]
\[
x_3^* = s_2 = 10.
\]

**Example 2.** The above problem can also be solved in a forward way as follows. We let  
\[
s_4 = 10, \quad s_3 = s_4 - x_3, \quad s_2 = s_3 - x_2, \quad s_1 = s_2 - x_1.
\]
Then, the recursive relationship is

\[ f_k(s_{k+1}) = \max_{0 \leq x_k \leq s_{k+1}} \{ g_k(x_k) + f_{k-1}(s_k) \}, \]

\[ f_0(s_1) = 0, \]

where \( f_k(s_{k+1}) \) denotes the contribution/profit of stages 1 to \( k \) with the invested amount \( s_{k+1} \) at stage \( k \).

- \( k = 1 \). \( f_1(s_2) = \max_{0 \leq x_1 \leq s_2} \{ 4x_1 + f_0(s_1) \} = 4s_2 \) when \( x_1^* = s_2 \).

- \( k = 2 \).

\[ f_2(s_3) = \max_{0 \leq x_2 \leq s_3} \{ 9x_2 + f(s_2) \} = \max_{0 \leq x_1 \leq s_3} \{ 9x_2 + 4(s_3 - x_2) \} = \max_{0 \leq x_2 \leq s_3} \{ 5x_2 + 4s_3 \} = 9s_3 \]

when \( x_2 = s_3 \).

- \( k = 3 \).

\[ f_3(s_4) = \max_{0 \leq x_3 \leq s_4} \{ 2x_2^2 + f_2(s_3) \} = \max_{0 \leq x_2 \leq s_4} \{ 2x_2^2 + 9(s_4 - x_3) \}. \]

\[ h'(x_3) = 4x_3 - 9 = 0 \quad \Rightarrow \quad x_3 = 9/4. \] But \( h'(x_3) = 4 > 0 \). So, it is a local minimum and the max value is attained at one of the end-points.

- When \( x_3 = 0 \), \( f_3(s_4) = f_3(0) = 90 \);
- When \( x_3 = 10 \), \( f_3(s_4) = f_3(10) = 200 \).

Therefore, \( x_3^* = 10 \), and \( x_2^* = x_1^* = 0 \).

**Example 3.** Discretisation of the above problem. We divide \([0, 10]\) into 5 intervals with 6 points, 0, 2, 4, 6, 8, 10. Assume \( x_k \in [0, s_k] \) can only take the discrete values. So does \( s_k \). The (backward) dynamic programming equation is

\[ f_k(s_k) = \max_{0 \leq x_k \leq s_k} \{ g_k(x_k) + f_{k+1}(s_k - x_k) \}, \]

\[ f_4(s_4) = 0, \]

in which \( s_k, x_k \in \{0, 2, 4, 6, 8, 10\} \). This can be solved by the following tables.

- \( k = 3 \). \( f_3(s_3) = \max\{2x_2^2\} \).

<table>
<thead>
<tr>
<th>( s_3 )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_3(s_3) )</td>
<td>0</td>
<td>8</td>
<td>32</td>
<td>72</td>
<td>128</td>
<td>200</td>
</tr>
<tr>
<td>( x_3^* )</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

- \( k = 2 \). \( f_2(s_2) = \max_{0 \leq x_2 \leq s_2} [9x_2 + f_3(s_2 - x_2)] \).

<table>
<thead>
<tr>
<th>( s_2 )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0, 2, 4</td>
<td>0, 2, 4, 6, 8</td>
<td>0, 2, 4, 6, 8, 10</td>
</tr>
<tr>
<td>( g_2 + f_3 )</td>
<td>0</td>
<td>8, 18</td>
<td>32, 26, 36</td>
<td>72, 50, 44, 54</td>
<td>128, 90, 68, 62, 72</td>
<td>200, 146, 108, 86, 80, 90</td>
</tr>
<tr>
<td>( f_2^* )</td>
<td>0</td>
<td>18</td>
<td>36</td>
<td>72</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>( x_2^* )</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ k = 1. \quad f_1(s_1) = \max_{0 \leq x_1 \leq 10} [4x_1 + f_2(s_1 - x_1)]. \]

| \(s_1\) | 10 \\
|---|---|
| \(x_1\) | 0 4 6 8 10 \\
| \(g_1 + f_2\) | 200 130 88 60 50 40 \\
| \(f_1^*\) | 200 \\
| \(x_1^*\) | 0 \\

Therefore, \(x_1^* = 0 = x_2^*\) and \(x_3^* = 10\).

### 4.4 Various examples in management

**Example 4.** A truck of capacity 10t is used to transport 3 different products, 1, 2, 3. The unit weight and value of each product are listed in the table below. Use dynamic programming to find how many units of each product to be loaded so that the total value transported by the truck is maximised.

<table>
<thead>
<tr>
<th>Product</th>
<th>Unit weight</th>
<th>Unit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

**Solution.** Let \(x_i\) be the number of units of product \(i\) for \(i = 1, 2, 3\). Then, we have

\[
\begin{align*}
\text{max} & \quad z = 4x_1 + 5x_2 + 6x_3 \\
\text{subj. to} & \quad 3x_1 + 4x_2 + 5x_3 \leq 10, \\
& \quad x_i \geq 0, \quad x_i \text{ is an integer, } i = 1, 2, 3.
\end{align*}
\]

We use the forward approach to solve this problem. We now consider a more general problem with products 1, 2, ..., \(N\). Assume that we load the products in the order 1, 2, 3, ..., \(N\). Let

\(s_k\) — the total weight allowable for product \(k\) at the beginning of stage \(k\).
\(x_k\) — the number of units of product \(k\).

Then, we have \(s_k = s_{k-1} - a_k x_k\), where \(a_k\) denotes the unit weight of product \(k\). The feasible set is

\[ D_k(s_k) = \{x_k : 0 \leq x_k \leq \lfloor s_{k-1}/a_k \rfloor, x_k \text{ is integer} \}. \]

The recursive relationship then becomes

\[
\begin{align*}
f_k(s_k) &= \max_{x_k=0,1,\ldots,\lfloor s_k/a_k \rfloor} [c_k(x_k) + f_{k-1}(s_{k-1} - a_k x_k)], \\
f_0(s_0) &= 0,
\end{align*}
\]

where \(c_k\) denotes the unit value if product \(k\). Let us use this to solve the original problem.

\bullet \quad k = 1. \quad f_1(s_1) = \max_{x_1 = 0, 1, \ldots, \lfloor s_1/a_1 \rfloor} [4x_1] = 4\lfloor s_1/3 \rfloor.

| \(s_1\) | 1 2 3 4 5 6 7 8 9 10 |
|---|---|---|---|---|---|---|---|---|---|
| \(x_1\) | 0 0 0 0,1 0,1 0,1 0,1,2 0,1,2 0,1,2,3 |
| \(f_1^*\) | 0 0 0 0,4 0,4 0,4,8 0,4,8 0,4,8,12 0,4,8,12 |
| \(x_1^*\) | 0 0 0 1 1 2 2 2 3 3 |
\( k = 2. \) \( f_2(s_2) = \max_{2 \leq x_2 \leq \lfloor s_2/4 \rfloor} [5x_2 + f_1(s_2 - 4x_2)]. \)

<table>
<thead>
<tr>
<th>( s_2 )</th>
<th>0, 1, 2, 3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>0, 0, 0, 0</td>
<td>0, 1</td>
<td>1</td>
<td>0, 1, 2</td>
<td>0, 1, 2</td>
<td>0, 1, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( c_2 + f_2 )</td>
<td>0, 0, 0, 0</td>
<td>4, 5</td>
<td>4, 5</td>
<td>8, 5</td>
<td>8, 9</td>
<td>8, 9, 10</td>
<td>12, 9, 10</td>
<td>12, 13, 10</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>0, 0, 0, 0</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>( x^*_k )</td>
<td>0, 0, 0, 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\( k = 3. \) We have to take the capacity constraint and demands into consideration. Clearly, \( s_3 \in \{0, 1, 2, 3\} \), and

\[
\max\{0, 2 - s_3\} \leq x_3 \leq \min\{6, g_3 + g_4 - s_3, g_3 + 3 - s_3\} = 5 - s_3.
\]

Example 5. Production planning. A company has a product. A survey shows that in the next 4 months, the market demands of the product are respectively 2, 3, 2 and 4 units. The cost of manufacturing \( j \) units of the product in one month is \( c(0) = 0 \) and \( C(j) = 3 + j \) for \( j = 1, 2, \ldots, 6 \). The storage cost of \( j \) units of the product is \( E(j) = 0.5j \). Find the production plan for the company in the next 4 months, so that the cost is minimised, while satisfying the demands. Assume that initially everything is zero. We also assume that the production and storage capacities are 6 and 3 respectively.

Solution. Let

\( s_k \) — units in storage at the beginning of the \( k \)th month.
\( x_k \) — units produced in the \( k \)th month.

The state transfer equation is \( s_{k+1} = s_k + x_k - g_k \), where \( g_k \) denotes the units sold (or demands of the market) in month \( k \). The recursive relationship is

\[
f_k(s_k) = \min\{C(x_k) + E(s_k) + f_{k+1}(s_{k+1})\} \\
= \min\{C(x_k) + E(s_k) + f_{k+1}(s_k + x_k - g_k)\}
\]

\( k = 4, 3, 2, 1, \)

\[
f_5(s_5) = 0.
\]

\( k = 4. \) \( f_4(s_4) = \min[C(x_4) + E(s_4)] \), and \( x_4 = 4 - s_4. \)

\[
\begin{array}{c|cccc}
  s_4 & 0 & 1 & 2 & 3 \\
  \hline
  f^*_4 & 7 & 6.5 & 6 & 5.5 \\
  x^*_4 & 4 & 3 & 2 & 1 \\
\end{array}
\]

\( k = 3. \) We have to take the capacity constraint and demands into consideration. Clearly, \( s_3 \in \{0, 1, 2, 3\} \), and

\[
\max\{0, 2 - s_3\} \leq x_3 \leq \min\{6, g_3 + g_4 - s_3, g_3 + 3 - s_3\} = 5 - s_3.
\]
Here

\[ g_3 + g_4 - s_3 \] — difference between the total demand of months 3 and 4 and the storage at the beginning of month 3. (Assuming there are no units left at the end of month 4.)

\[ g_3 + 3 - s_3 \] — number of units left unsold at the end of month 3. (needs to be stored.)

<table>
<thead>
<tr>
<th>( s_3 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_3 )</td>
<td>2,3,4,5</td>
<td>1,2,3,4</td>
<td>0,1,2,3,4,8</td>
<td>0,1,2,3,8</td>
</tr>
<tr>
<td>( C + E + f_4 )</td>
<td>12,12.5,13,13.5</td>
<td>11.5,12,12.5,13</td>
<td>8,11.5,12,12.5</td>
<td>8,11.5,12</td>
</tr>
<tr>
<td>( f_3^* )</td>
<td>12</td>
<td>11.5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>( x_3^* )</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ k = 2. \quad f_2(s2) = \min[C(x_2) + E(s_2) + f_2(s_2 + x_2 - g_2)] \] for \( s_2 \in \{0,1,2,3\} \). The range for \( x_2 \) is

\[ \max\{0,g_3-s_2\} \leq x_2 \leq \min\{6,g_2+3-s_2,g_2+g_3+g_4-s_2\} = \min\{6,6-s_2,9-s_2\} = 6-s_2. \]

<table>
<thead>
<tr>
<th>( s_2 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>3,4,5,6</td>
<td>2,3,4,5</td>
<td>1,2,3,4</td>
<td>0,1,2,3</td>
</tr>
<tr>
<td>( C + E + f_3 )</td>
<td>18,18.5,16,17</td>
<td>17.5,18,15.5,16.5</td>
<td>17,17.5,15,16</td>
<td>13.5,17,14.5,15.5</td>
</tr>
<tr>
<td>( f_2^* )</td>
<td>16</td>
<td>15.5</td>
<td>15</td>
<td>13.5</td>
</tr>
<tr>
<td>( x_2^* )</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ k = 1. \quad s_1 = 0 \] and \( x_1 \in \{2,3,4,5\} \)

\[ f_1(0) = \min[C(x_1) + E(0) + f_2(s_1 + x_1 - g_1)]. \]

<table>
<thead>
<tr>
<th>( s_1 )</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>2,3,4,5</td>
</tr>
<tr>
<td>( C + f_2 )</td>
<td>21,21.5,22,21.5</td>
</tr>
<tr>
<td>( f_1^* )</td>
<td>21</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>2</td>
</tr>
</tbody>
</table>

So, \( f_1(0) = 21 \) — min cost.

Tracing back:

\[ x_1^* = 2 \quad \Rightarrow \quad s_2 = 0 \quad \Rightarrow \quad x_2^*(0) = 5 \quad \Rightarrow \quad s_3 = s_2 + x_2^* - g_2 = 0 + 5 - 3 = 2 \]

\[ \Rightarrow x_3^*(2) = 0 \quad \Rightarrow \quad s_4 = s_3 + x_3^* - g_3 = 2 + 0 - 2 = 0 \quad \Rightarrow \quad x_4^*(0) = 0. \]

**Example 6.** Equipment replacement problem. Consider a machine of age \( t \) years. The question is whether we shall keep the machine or replace it with a new one (or when is the best time to replace it).

**Solution.** For \( k = 1,2,...,n \), we let

- \( r_k(t) \) — the efficiency if the use of a machine in the \( k \)th year of age \( t \).
- \( u_k(t) \) — maintenance cost in year \( k \) for a machine of age \( t \).
- \( c_k(t) \) — net cost of selling a machine of age \( t \) and buying a new one in year \( k \).
- \( \alpha \) — discount factor. \( \alpha \in [0,1] \).

Introducing \( s_k \), age of the machine at the beginning of year \( k \), we have

\[ s_{k+1} = \begin{cases} 
  s_k + 1, & x_k = K \quad (\text{keep}), \\
  1, & x_k = R \quad (\text{replace}),
\end{cases} \]
where $x_k$ is the decision variable of either replacing or keeping the machine at the beginning of year $k$.

Profit in year $k$:

\[
v(s_k, x_k) = \begin{cases} 
  r_k(s_k) - u_k(s_k), & x_k = K, \\
  r_k(0) - u_k(0) - c_k(s_k), & x_k = R.
\end{cases}
\]

Objective function:

\[
f_k(s_k) = \max_{x_k} \{ v(s_k, x_k) + \alpha f_{k+1}(s_{k+1}) \} = \max \left\{ r_k(s_k) - u_k(s_k) + \alpha f_{k+1}(s_{k+1}), \right.
\]

\[
\left. r_k(0) - u_k(0) - c_k(s_k) + \alpha f_{k+1}(1) \right\}.
\]

To complete the question, we assume that the parameters are given in the following table.

<table>
<thead>
<tr>
<th>year</th>
<th>$r_k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_k$</td>
<td>0.5</td>
<td>1</td>
<td>1.5</td>
<td>2</td>
<td>2.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$c_k$</td>
<td>0.5</td>
<td>1.5</td>
<td>2.2</td>
<td>2.5</td>
<td>3</td>
<td>3.5</td>
<td></td>
</tr>
</tbody>
</table>

We now solve it by the dynamic programming principle.

- **$k = 5$.** $f_5(s_5) = \max \{ r_5(s_5) - u_5(s_5), r_5(0) - u_5(0) - c_5(s_5) \}$ with $s_5 = 1, 2, 4, 5$.
  
  Therefore,
  - $f_5(1) = \max \{ 3.5, 5 - 0.5 - 1.5 \} = 3.5, x^*_5(1) = K$.
  - $f_5(2) = \max \{ 4 - 1.5, 4.5 - 2.2 \} = 2.5, x^*_5(2) = K$.
  - $f_5(3) = \max \{ 3.75 - 2, 4.5 - 2.5 \} = 2, x^*_5(3) = R$.
  - $f_5(4) = \max \{ 3 - 2.5, 4.5 - 3 \} = 1.5, x^*_5(4) = R$.

- **$k = 4$.** $f_4(s_4) = \max \{ r_4(s_4) - u_4(s_4), r_4(0) - u_4(0) - c_4(s_4) + f_5(1) \}$ for $s_4 = 1, 2, 3$.
  - $f_4(1) = 6.5, x^*_4(1) = R$.
  - $f_4(2) = 5.8, x^*_4(2) = R$.
  - $f_4(3) = 5.5, x^*_4(3) = R$.

- **$k = 3$.** $f_3(s_3) = \max \{ r_3(s_3) - u_3(s_3), r_3(0) - u_3(0) - c_3(s_3) + f_4(1) \}$ for $s_3 = 1, 2$.
  - $f_3(1) = 9.5, x^*_3(1) = R$.
  - $f_3(2) = 8.8, x^*_3(2) = R$.

- **$k = 2$.** $s_2 = 1, f_2(1) = 12.5$ and $x^*_2(1) = R$.

- **$k = 1$.** $s_1 = 0, f_1(0) = 17$ and $x^*_1(0) = K$.

Therefore, we have, using the state transfer formula,

- $s_2 = s_1 + 1 = 1$ since $x^*_1(0) = K \implies x^*_2(1) = R$.
- $s_3 = 1$ since $x^*_2(1) = R \implies x^*_3(1) = R$.
- $s_4 = 1$ since $x^*_3(1) = R \implies x^*_4(1) = R$.
- $s_5 = 1$ since $x^*_4(1) = R \implies x^*_5(1) = K$. The decision sequence is $(K, R, R, R, K)$.  

50
Example 7. A company plans to buy a special metal in the next 5 weeks. The probabilities of the 3 different prices are listed in the following table. Find the best plan for the company so that the expected price is minimized.

<table>
<thead>
<tr>
<th>price $P_i$</th>
<th>500</th>
<th>600</th>
<th>700</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability $p_i$</td>
<td>0.3</td>
<td>0.3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Solution. For $k = 1, 2, 3, 4, 5$, let
- $s_k$ — actual price in week $k$,
- $x_k$ — decision variable, i.e. $x_k = 1$ if to buy, or 0 if not to buy,
- $E_k$ — expected price after week $k$. Note $x_k = \begin{cases} 1, & s_k < E_k; \\ 0, & s_k > E_k \end{cases}$
- $f_k(s_k)$ — expected minimum price from week $k$ to week 5 when the actual price in week $k$ is $s_k$.

Recursive relationship:

$$f_k(s_k) = \min\{s_k, E_k\}, \quad s_k \in D, \quad k = 4, 3, 2, 1,$$

$$f_5(s_5) = s_5, \quad s_5 \in D,$$

where $D = \{500, 600, 700\}$. We now solve this problem.

- $k = 5$. $f_5(s_5) = \begin{cases} 500, & s_5 = 500; \\ 600, & s_5 = 600; \\ 700, & s_5 = 700, \end{cases}$, and $x_5^* = 1$. This is because if the company has not bought any in the previous weeks, it has to buy in week 5.

- $k = 4$.

$$E_4 = 0.3f_5(500) + 0.3f_5(600) + 0.4f_5(700) = 610,$$

$$f_4(s_4) = \min\{s_4, E_4\} = \begin{cases} 500, & s_4 = 500, \quad x_4^* = 1; \\ 600, & s_4 = 600, \quad x_4^* = 1; \\ 610, & s_4 = 700, \quad x_4^* = 0. \end{cases}$$

- $k = 3$.

$$E_3 = 0.3f_4(500) + 0.3f_4(600) + 0.4f_4(700) = 574,$$

$$f_3(s_3) = \min\{s_3, E_3\} = \begin{cases} 500, & s_3 = 500, \quad x_3^* = 1; \\ 570, & s_3 = 600, \quad x_3^* = 0; \\ 570, & s_3 = 700, \quad x_3^* = 0. \end{cases}$$

- $k = 2$.

$$E_2 = 0.3f_3(500) + 0.3f_3(600) + 0.4f_3(700) = 551.8,$$

$$f_2(s_2) = \min\{s_2, E_2\} = \begin{cases} 500, & s_2 = 500, \quad x_2^* = 1; \\ 551.8, & s_3 = 600, 700, \quad x_2^* = 0. \end{cases}$$

- $k = 1$.

$$E_1 = 0.3f_2(500) + 0.3f_2(600) + 0.4f_2(700) = 536.26,$$

$$f_1(s_1) = \min\{s_1, E_1\} = \begin{cases} 500, & s_1 = 500, \quad x_1^* = 1; \\ 551.8, & s_3 = 600, 700, \quad x_1^* = 0. \end{cases}$$