Today we will look at

- Differences between Linear and Integer Programming
- Network Flow Problems
- Methods for Integer Programming
  - Network Simplex
  - Branch & Bound
  - Cutting Plane
- Modelling with 0-1 constraints
Simple Example, revisited

Reconsider our example for production plan optimization. This time we assume that B stands for “Book Cases”, C for “Chairs”, D for “Desks”.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finishing</td>
<td>2.0</td>
<td>2.0</td>
<td>1.0</td>
<td>30.0</td>
</tr>
<tr>
<td>Labor</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
<td>25.0</td>
</tr>
<tr>
<td>Machine Time</td>
<td>2.0</td>
<td>1.0</td>
<td>1.0</td>
<td>20.0</td>
</tr>
<tr>
<td>Profit</td>
<td>3.0</td>
<td>1.0</td>
<td>3.0</td>
<td></td>
</tr>
</tbody>
</table>

The optimum solution for this problem was to produce 7 bookcases and 6 desks.
Now assume some coefficients would have been slightly different. For example, each bookcase needs 1.5 units of labor.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finishing</td>
<td>2.0</td>
<td>2.0</td>
<td>1.0</td>
<td>30.0</td>
</tr>
<tr>
<td>Labor</td>
<td>1.5</td>
<td>2.0</td>
<td>3.0</td>
<td>25.0</td>
</tr>
<tr>
<td>Machine Time</td>
<td>2.0</td>
<td>1.0</td>
<td>1.0</td>
<td>20.0</td>
</tr>
<tr>
<td>Profit</td>
<td>3.0</td>
<td>1.0</td>
<td>3.0</td>
<td></td>
</tr>
</tbody>
</table>
Integer Problems

We apply Simplex and obtain the final tableau

\[ z = 36.6667 - 0.6667 s[l] - 1.0 s[m] - 1.333 x[c] \]

\[ s[f] = 10 + s[m] - x[c] \]
\[ x[d] = 4.4444 - 0.4444 s[l] + 0.3333 s[m] - 0.5556 x[c] \]
\[ x[b] = 7.7778 + 0.2222 s[l] - 0.6667 s[m] - 0.2222 x[c] \]

This makes little sense as a solution, since we can hardly manufacture 4.444 desks (maybe we can, but who would buy 0.444 of them).

The problem that we are facing is that we need the problem variables to assume integer values for a feasible solution. Such problems are called integer programming problems.
Classes of Optimization Problems

Optimization

unconstrained

constrained

linear

non-linear

real-valued integer

quadratic

... ...

\[
\max_{\vec{x}} f(\vec{x}) \quad \text{subject to} \quad C(\vec{x})
\]
max $f(\bar{x})$ subject to $C(\bar{x})$

where $f(\bar{x}) = f(x_1, \ldots, x_n)$

is a linear function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and

$C(\bar{x}) = c_1(x_1, \ldots, x_n) \land \ldots \land c_k(x_1, \ldots, x_n)$

is a conjunction of linear inequalities

and the $x_i$ are required to be integer.
Recall: Convex Problems

In general the feasible space for a linear integer problem is non-convex.


...the bad news

integer linear problems are significantly harder to solve than linear problems on reals.

No general algorithm is known that allows to optimize a solution by directly moving from a feasible solution to an improved feasible solution.

In fact, integer linear programming is known to be \textit{NP-complete}.

... It is easy to formulate a TSP as an IP:

minimize \[ \sum \sum c_{ij} x_{ij} \]

subject to \[ \forall i: \ 1 = \sum_{j=1}^{n} x_{ij} \land \forall j: \ 1 = \sum_{i=1}^{n} x_{ij} \]

where \[ x_{ij} = \begin{cases} 1 & \text{if tour leads from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases} \]
The first idea might be to try rounding of the optimal fractional solution. The corresponding problem in which the variables are not required to take integer values is called the linear relaxation.

As the above example proves, this cannot be guaranteed to work: 
(2/0) is infeasible 
(1/0) is feasible, but not optimal
A particular form of linear integer problems can be solved in elegantly (and efficiently) with drastic simplification of the simplex method.

Network Flow Optimization
Transshipment Problems

Intuitively, a transshipment problem (or network flow problem) consists in finding the cheapest way of shipping goods through a network of routes so that all given demands at all points of the network are satisfied. Given:

- a network of routes as a graph
- a set of nodes which act as sources (supplies)
- a set of nodes which act as sinks (demands)
- the amount of supply and demand at each node
- the cost of each transport route (edge)

Assumption: the total supply equals the total demand
A transshipment problem can concisely be represented in matrix form, with Incidence Matrix and Demand Vector.

\[
A = \begin{pmatrix}
-1 & -1 & -1 & 1 & 1 \\
-1 & -1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & -1 & 1 \\
-1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & 1 & 1 \\
\end{pmatrix}
\]

The incidence matrix A contains one column \(a=(a_1, \ldots, a_n)\) for each edge \(e=(n_i, n_j)\) in the network with

\[
ak := \begin{cases} 
-1 & \text{if } k=i \\
1 & \text{if } k=j \\
0 & \text{otherwise}
\end{cases}
\]
Network Flow Matrix Representation

the demand vector

\[
b = \begin{bmatrix}
0 \\
0 \\
6 \\
10 \\
8 \\
-9 \\
-15
\end{bmatrix}
\]

contains one element for each node given the total demand at this node. Supplies are indicated as negative demands.

The cost vector

\[
c = \begin{bmatrix}
c_{13} & c_{14} & c_{15} & c_{21} & c_{23} & c_{24} & c_{25} & c_{54} & c_{61} & c_{62} & c_{63} & c_{67} & c_{72} & c_{75}
\end{bmatrix}
= \begin{bmatrix}
53 & 18 & 29 & 8 & 60 & 28 & 37 & 5 & 44 & 38 & 98 & 14 & 23 & 59
\end{bmatrix}
\]

contains one entry for each edge \(e=(ni, nj)\) with the cost of shipping one unit along this arc.
the transshipment problem can now be formalized as:

\[
\text{minimize} \quad cx = \sum c_{ij}x_{ij}
\]

subject to

\[
Ax = b,
\]

\[
x_{ij} \geq 0
\]

such that A is an incidence matrix and

\[
\sum_{i=1}^{n} b_i = 0
\]

Every problem that can be expressed in this form is a transshipment problem and can be solved by the network simplex algorithm.
Truncated Matrix

Note that the equations expressed in $Ax = b$

are not independent. Since $\sum_{i=1...n} b_i = 0$

each of these equations is the sum of the remaining n-1 equations. We can therefore arbitrarily delete one of the equations from the matrices A and b obtaining A’ and b’.

The truncated problem has the same solution as the original problem.

$$\text{minimize } cx = \sum c_{ij}x_{ij}$$

subject to

$$A'x = b', \quad A'x = b', \quad x_{ij} \geq 0$$
Feasible Solutions = Spanning Trees

Like the Simplex method, the Network Simplex works by improving feasible solutions. For a transshipment problem, a feasible solution can be defined by a spanning tree $T$ and a solution vector $x$ such that the component $x_{ij}=0$ if $(ni,nj)$ is not in $T$.

Note that the feasible tree need not be a directed tree. It is defined on underlying undirected graph.
Uniqueness of Feasible Solutions

Let $B$ ($B'$) be the (truncated) incidence matrix of a feasible tree for a transshipment problem.

\[
\text{minimize} \quad cx = \sum c_{ij} x_{ij}
\]

subject to

\[
B' x = b',
\]

\[
x_{ij} \geq 0
\]

Note that $B'$ determines $x$ uniquely in $B'x=b'$.

$B'$ can easily be transformed into a triangular matrix with non-zero entries on the diagonals ($B$ is a tree!).

The solution to the original problem is uniquely determined by the associated feasible tree.
Integrality Theorem

Let \( b \) be a vector of integers.

The transshipment problem

\[
\text{minimize } \quad cx = \sum c_{ij} x_{ij} \\
\text{subject to } \\
Ax = b, \\
x_{ij} \geq 0
\]

(a) is feasible if and only if it has an integer-valued solution vector \( x \).

(b) has an optimal solution if and only if it has an integer-valued optimal solution.

This can be proved by analyzing the pivot operation of the network simplex:
Whenever it starts from an integer-valued solution the subsequent solution is also integer-valued.
Economic Interpretation: Fair Prices

A feasible tree determines a possible shipment plan. Assume the unit price at \( n_i \) is \( y_i \), then a fair price at \( n_j \) is \( y_j = y_i + c_{ij} \).

To find a better shipment option we can then check whether we can reasonably achieve a better fair price at some node by changing a single connection.

To do this we try to find a pair of nodes \((n_i, n_j)\) with fair prices \((y_i, y_j)\) such that there is an edge from \( n_i \) to \( n_j \) with associated cost \( c_{ij} \) such that

\[ y_j > y_i + c_{ij} \]
Entering Arcs

The arc with $y_j > y_i + c_{ij}$ is now added to the spanning tree. It is therefore called the “entering arc”.

Strategy: If there are several possible choices for entering arcs, choose the one which maximises the gain (i.e. $y_j - y_i - c_{ij}$).

Note: large scale computer implementations use different strategies.
The pivot step improves the solution by shipping some quantity $t$ along the entering arc. As in the Simplex we try to exploit the entering arc maximally by maximizing $t$ until constraints are tight.

The obvious adjustments are obtained by traversing the cycle introduced by the entering arc (in its direction) and labelling each arc $a$ along the cycle with $(x_{ij} + t)$ if $a$ has the same direction on the cycle as the entering arc $(x_{ij} - t)$ if $a$ has the opposite direction on the cycle.

With the resulting adjustments we have to satisfy:
$$t \geq 0, \quad 8-t \geq 0, \quad 9-t \geq 0, \quad 15-t \geq 0, \quad 1+t \geq 0$$

Obviously this leads to $t=8$ which removes all shipments from the dotted arc. This arc is referred to as the “leaving arc”.

Network Pivot
Summary: Network Simplex

(1) construct a basic feasible tree.
(2) solve the linear system for this tree.
(3a) find an entering arc that can improve the solution
(3b) if no such arc can be found, the solution is optimal
(4) adjust the quantities shipped
(5) re-iterate from step (1)

As the Simplex algorithm, network simplex only works with feasible solutions.

The obvious problem is how to find an initial basic feasible spanning tree!
In general, finding a basic feasible tree is not simple.

However, we can perform a “trick” similar to Simplex phase 0.

The basic idea is to add “artificial arcs” that guarantee a trivial basic feasible tree.

We then use a phase 0 optimization on the augmented graph that penalizes these arcs so that they are not used in the optimum solution to phase 0.

The result of phase 0 with the (unused) artificial arcs removed is then a basic feasible solution for the original problem.
Auxiliary Problem & Graph

- Select an arbitrary fixed node $w$
- Complete the arcs from $w$ to each sink and intermediate node
- Complete the arcs from each source to $w$
  - set $x_{wj}=b_j$ for all arcs to sinks
  - set $x_{iw}=-b_i$ for all arcs from sources (supply is negative demand)

This is a trivial feasible tree for the auxiliary graph

- set penalty $p_{ij}=1$ for each artificial arc
- set penalty $p_{ij}=0$ for each original arc

- Solve the following auxiliary problem

\[
\begin{align*}
\text{minimize} & \quad cx = \sum p_{ij} x_{ij} \\
\text{subject to} & \quad Ax = b, \\
& \quad x_{ij} \geq 0
\end{align*}
\]
Solving the Auxiliary Problem

The auxiliary problem has a trivial basic feasible tree: The artificial arcs

It can simply be optimized by network simplex. Three outcomes for the spanning tree of the solution of phase 0 are possible:

• T contains no artificial arc
  -> use T as a start for phase I optimization of original problem
• T contains an artificial arc aij with xij>0
  -> the original problem is infeasible
• T contains artificial arcs aij with xij=0
  -> the original problem may have a solution, but not a tree solution (eg network not sufficiently connected)

• Decompose the original problem into two networks.
  Let a=(ni, nj) be the artificial arc with xij=0
    - Network A contains all nodes that precede ni in T
    - Network B contains all nodes that follow ni in T

• A and B can now be solved independently
Degeneracy

It can happen that the pivot steps generates values of \( x_{ij} = 0 \) for more than one arc. However, only one arc can be chosen as the leaving arc, because otherwise the tree would be disconnected. After such a pivot the solution contains arcs with \( x_{ij} = 0 \).

Such a solution is called “degenerate”.

It can lead to subsequent pivots, in which only the tree changes, but not the actual shipment (because leaving arcs with \( x_{ij} = 0 \) are exchanged against entering arcs with \( x_{uv} = 0 \) after the pivot).

This can cause

- stalling (a phase of no visible improvement due to swapping arcs with zero shipment)
- cycling
Cycling

Cycling is extremely rare in practice.

It can easily be avoided by the following method (due to W.H. Cunnigham, 1976)

Let a strongly feasible tree be a feasible tree in which each arc is directed away from the (arbitrarily chosen, but fixed) root w.

(To check the direction of a=(u,v) in T, remove a from T. This partitions T in Tu (containing u) and Tv (containing v). The arc a is directed towards the root w if and only if w is in Tv)

1. Ensure that the initial solution is strongly feasible. This always holds for the initial tree of the auxiliary problem!
2. In each pivot generate a strongly feasible solution.

The resulting network simplex process will not cycle, though it may still go through degenerate immediate solutions.
“Strong” Pivot

The entering arc $e=(u,v)$ introduces a cycle into $(T+e)$. Let the “join” node $nj$ be the node at which the paths from $u$ to the root of $T$ and from $v$ to the root of $T$ meet.

*Strong feasiblity of a solution is maintained in each pivot, if the following refinement is introduced.*

If there is more than one candidate for the leaving arc, the choice is made in the following way:

- Traverse the cycle starting at the join node $nj$ in the direction of the entering edge $e=(u,v)$. Choose the first candidate encountered on this traversal as the leaving edge.

*Every tree generated by a strong pivot is strongly feasible -> no cycling*
Complexity

- Linear Programming: polynomial
  (by reduction to linear strict inequality)

  - Simplex algorithm
    worst case: exponential in size of LP
    in practice: well-behaved

- Integer Linear Programming: NP-complete

- Network Simplex
  worst case: exponential in node size
  (proven for largest merit selection ‘73)
  in practice: roughly linear in node size
Generalization to Inequalities

The assumption \( \sum_{i=1}^{n} b_i = 0 \) (supply matches demand) is often unrealistic.

We need to generalize to supplies that exceed the demand.

- Introduce “virtual dump” (new node) for exceeding supply.

The new “dummy node” must be connected to each source by a new “dummy arc” of zero cost.
Caterer Problem
(aircraft overhaul scheduling)

A caterer needs to plan (and optimize) his napkin provisions over n days
- \(d_j\): number of napkins needed on day \(j\) (known in advance)

Napkins can be
- bought: \(a\) cents per piece
- laundered:
  - \(b\) cents per napkin (fast service takes \(q\) days)
  - \(c\) cents per piece (slow service takes \(p\) days)

"buy"
"quick laundry"
"keep"
"slow"
"at end of day"  "at start of day"
"exceeding supply in shop"
"Dump"
"Shop"
Other Integer Programming Methods

What to do if the problem cannot be reduced to transshipment?

Problem: Linear Relaxation does not work

Observation: (1) All feasible points for IP are also feasible for the relaxation
(2) If the optimum of the relaxation is integer-valued, it also is the solution of the corresponding IP.

Idea: Enumerate the feasible points of the relaxed problem

Challenge: This must be done cleverly, since the number of feasible points may be huge.

Methods: based on repeatedly solving improved relaxations (with simplex)

• Branch & Bound: splits the search space into sub-problems
• Cutting Plane: removes non-integer optimaums
Let \( P \) be a linear integer program.
To solve \( P \):

1. solve \( P \)'s linear relaxation \( P' \).
2a. If the solution of \( P' \) is integer-valued, it is the optimum of \( P \).
2b. If the solution contains a fractional variable \( v \) create two subproblems:

\[
P' a: \quad P \land x_i \leq \lfloor v \rfloor
\]
\[
P' b: \quad P \land x_i \geq \lceil v \rceil
\]

3. solve the subproblems recursively.
4. The solution to \( P \) is the better solution of \( P' a, P' b \).
max $z = 8x + 5y$
such that $x + y \leq 6$
$9x + 5y \leq 45$
$x, y \geq 0$

optimum: $z = \frac{165}{4}$
$x = \frac{15}{4}$, $y = \frac{9}{4}$

max $z = 8x + 5y$
such that $x + y \leq 6$
$9x + 5y \leq 45$
$x \geq 4$ (*)
$x, y \geq 0$

optimum: $z = 4$
$x = 4$, $y = \frac{9}{5}$

max $z = 8x + 5y$
such that $x + y \leq 6$
$9x + 5y \leq 45$
$x \geq 4$
$y \leq 1$ (*)
$x, y \geq 0$

optimum: $z = \frac{365}{9}$
$x = \frac{40}{9}$, $y = 1$
Completely integer-valued solutions of a sub-problem are called **Candidate Solutions**
Fathomed Sub-Problems

In principle, every sub-problem (branch) needs to be explored.

In three cases, a sub-problem does not need to be explored further.

• it is infeasible
• its optimum solution is integer-valued
• the optimum of its relaxation is smaller than that of some candidate.

This is because

• the candidate solution provides a lower bound for the original problem
• the optimum of the relaxation provides an upper bound for the corresponding integer problem.

Such sub-problems are called fathomed.
Final Decision Tree

\[
\begin{align*}
\text{z} &= 165/4 \\
\text{x} &= 15/4, \text{y} = 9/4 \\
\text{z} &= 41 \\
\text{x} &= 4, \text{y} = 9/5 \\
\text{z} &= 39 \\
\text{x} &= 3, \text{y} = 4 \\
&\text{(integer solution)} \\
\text{z} &= 40 \\
\text{x} &= 5, \text{y} = 0 \\
&\text{(integer solution)} \\
\text{z} &= 365/9 \\
\text{x} &= 40/9, \text{y} = 1 \\
\text{infeasible} \\
\text{z} &= 37 \\
\text{x} &= 4, \text{y} = 1 \\
&\text{(integer solution)}
\end{align*}
\]
Backtracking vs. Jumptracking

The order in which sub-problems are expanded is not fixed.

Two strategies are common

• Backtracking (aka LIFO / last-in-first-out)
  • Expand the most recently created sub-problem first
  • This results in depth-first exploration.

• Jumptracking
  • Obtain upper bounds for all sub-problems through relaxation
  • Then branch on the best upper bound
An Alternative: Cutting Planes

Idea: If the solution of the LP relaxation is not integer-valued we can find an additional constraint (called “cut”) that
- removes the current optimum from the feasible region
- does not remove any feasible point for the original IP

Problem:
maximize \[ 8x[1] + 5x[2] \]
subject to \[ 9x[1] + 5x[2] \leq 45 \]
and \[ x[1] + x[2] \leq 6 \]
Finding the Cut (1)

Problem: maximize  
\[ 8x[1] + 5x[2] \]

Initial Tableau:  
\[ z == 8x[1] + 5x[2] \]

Solution:  
\[ z == \frac{165}{4} - \frac{5}{4}s[1] - \frac{3}{4}s[2] \]
\[ x[2] == \frac{9}{4} - \frac{9}{4}s[1] + \frac{1}{4}s[2] \]
\[ x[1] == \frac{15}{4} + \frac{5}{4}s[1] - \frac{1}{4}s[2] \]

• Find an equation for a fractional basic variable  
\[ x[1] - \frac{5}{4}s[1] + \frac{1}{4}s[2] == \frac{15}{4} \]
Finding the Cut (2)

(1) rearrange fractional constraint

\[ x[1] - \frac{5}{4} s[1] + \frac{1}{4} s[2] = \frac{15}{4} \]

(2) rewrite fractional coefficients into fractional part and integer part using \( \lfloor x \rfloor \)

\[ x[1] - 2s[1] + \frac{3}{4} s[1] + \frac{1}{4} s[2] = 3 + \frac{3}{4} \]

(3) move fractional parts to right hand side

\[ x[1] - 2s[1] - 3 = \frac{3}{4} - \frac{3}{4} s[1] - \frac{1}{4} s[2] \]

(4) Generate cut as (RHS <= 0)

\[ \frac{3}{4} - \frac{3}{4} s[1] - \frac{1}{4} s[2] \leq 0 \]

(5) Add slack to cut

\[ s[3] = -\frac{3}{4} + \frac{3}{4} s[1] + \frac{1}{4} s[2] \]

(6) Add cut to tableau & resolve

Note: The cut equation \( 3x[1] + 2x[2] = 15 \)

is obtained by substituting the definitions of the slack variables in \( s[3] = 0 \).
Cutting Plane Algorithm

(1) Solve the LP relaxation
(2a) If the solution to the relaxation is integer-valued optimum is found
(2b) Otherwise add new cut to tableau of LP relaxation
(3) goto step 1

Note: the row of the cut constraint (after adding slack) in the tableau has a negative intercept. Simplex as we know it is unsuitable for this problem.

\[
\begin{align*}
z &= \frac{165}{4} - \frac{5}{4} s[1] - \frac{3}{4} s[2] \\
x[2] &= \frac{9}{4} - \frac{9}{4} s[1] + \frac{s[2]}{4} \\
x[1] &= \frac{15}{4} + \frac{5}{4} s[1] - \frac{s[2]}{4} \\
s[3] &= -\frac{3}{4} + \frac{3}{4} s[1] + \frac{1}{4} s[2]
\end{align*}
\]
Dual Simplex

(1) If all intercepts are positive terminate.

(2) Choose the basic variable with the most negative intercept to leave the basis. The corresponding row is the pivot row.

(3) Compute the following ratio for each variable $x[j]$ with positive coefficient in the pivot row:

$$\frac{\text{coefficient}(x[j]) \text{ in objective}}{\text{coefficient}(x[j]) \text{ in pivot row}}$$

(4) Choose the variable with the smallest absolute ratio to enter the basis.

(5) Pivot the entering variable into the basis.

(6a) If there is a row with a negative intercept and only negative coefficients terminate with “infeasible”

(6b) Otherwise go to step (1).
Mixed Integer Problems

\[
\begin{align*}
\max_{\bar{x}} \ f(\bar{x}) & \quad \text{subject to} \quad C(\bar{x}) \\
\bar{x} & \\
\text{where} \quad f(\bar{x}) &= f(x_1, \ldots, x_n) \\
\text{is a linear function} \quad f : \mathbb{R}^n \to \mathbb{R} & \quad \text{and} \\
C(\bar{x}) &= c_1(x_1, \ldots, x_n) \land \ldots \land c_k(x_1, \ldots, x_n) \\
\text{is a conjunction of linear inequalities} \\
\text{and some} \ x_i \ \text{are required to be integer-valued.}
\end{align*}
\]

MIPs can be solved by branch & bound.
we only need to branch on the integer variables.
0-1 Integer Problems

\[
\begin{align*}
\text{max } f(\bar{x}) & \quad \text{subject to } C(\bar{x}) \\
\bar{x} & \\
\text{where } f(\bar{x}) &= f(x_1, \ldots, x_n) \\
\text{is a linear function } f: \mathbb{R}^n \to \mathbb{R} \text{ and } \\
C(\bar{x}) &= c_1(x_1, \ldots, x_n) \land \ldots \land c_k(x_1, \ldots, x_n) \\
\text{is a conjunction of linear inequalities} \\
\text{and } \forall i: x_i \in \{0, 1\}
\end{align*}
\]

It is obviously easier to solve 0-1 IPs, because branching on \( x \) only needs to consider two cases: \( x=0 \) and \( x=1 \).
Implicit Enumeration for 0-1 IP

Generate a search tree for the problem variables. At each node some variables have fixed values, others are “free” (value not yet known). Prune the search tree with the following checks:

(1) **Check Bounds**
Assign values to the free variables that maximize/minimize the objective function. This is an upper bound.

(1a) If this valuation is feasible, it must be the optimum (for the sub-tree with this root)
Keep as a candidate solution.

(1b) If the objective value for this valuation is less than some candidate,
do not consider this node further.
Implicit Enumeration for 0-1 IP

(2) **Check Feasibility**
For each constraint try to assign values to the free variables according the table. If some constraint cannot be satisfied, the node is **infeasible.**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Coefficient</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>≤</td>
<td>−</td>
<td>1</td>
</tr>
<tr>
<td>≥</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>≥</td>
<td>−</td>
<td>0</td>
</tr>
</tbody>
</table>
Modelling Or Constraints in 0-1 IP

To model two constraints of which at least one must be satisfied (\(f\) or \(g\))

\[
\begin{align*}
  f(\vec{x}) & \leq 0 \\
  g(\vec{x}) & \leq 0
\end{align*}
\]

Use a 0-1 variable \(y\) and reformulate as

\[
\begin{align*}
  f(\vec{x}) & \leq k \cdot y \\
  g(\vec{x}) & \leq k \cdot (1 - y)
\end{align*}
\]

where \(k\) is a sufficiently large number
To model an if-then relation between two constraints \((f \Rightarrow g)\)

\[
\begin{align*}
  f(\vec{x}) &> 0 \\
  g(\vec{x}) &\geq 0
\end{align*}
\]

Use a 0-1 variable \(y\) (\(y=0\) if \(f\) is true) and reformulate as

\[
\begin{align*}
  -g(\vec{x}) &\leq k \cdot y \\
  f(\vec{x}) &\leq k \cdot (1 - y)
\end{align*}
\]

Where \(k\) is a sufficiently large number
Piecewise Linear Functions

To model a piecewise linear function
- break it into linear segments,
- model the separate segments,
- “chain” the segments with an additional constraint

\[ f(x) = q \cdot f(b_1) + (1 - q) \cdot f(b_2) \]

for \( b_1 \leq x \leq b_2 \) with \( 0 \leq q \leq 1 \)
Modelling Piecewise Linear Functions

(1) replace $f$ throughout the problem by
$$f(x) = z_1 f(b_1) + ... + z_n f(b_n)$$

(2) add a constraint for $x$
$$x = z_1 \cdot b_1 + ... + z_n \cdot b_n$$

(3) add the following constraints

$z_1 \leq y_1$,  $z_2 \leq y_1 + y_2$,  $z_3 \leq y_2 + y_3$,  ...,  $z_{n-1} \leq y_{n-2} + y_{n-1}$,  $z_n \leq y_{n-1}$

$$\sum y_i = 1 \quad \sum z_i = 1$$

$\forall \ y_i \in \{0, 1\}$

$\forall \ z_i \geq 0$

These constraints ensure that only two neighbouring $z$’s can have $z_i > 0$ and must add up to 1 => this models the piecewise interpolation
Summary

Today we have looked at

• Differences between Linear and Integer Programming
• Network Flow Problems
• Methods for Integer Programming
  • Network Simplex
  • Branch & Bound
  • Cutting Plane
• Modelling with 0-1 constraints

Homework

• Study the modelling examples in Section 9.1-9.2 of Winston