An Example (based on the Phillips article)

- Suppose you're the hapless MBA, and you haven't been fired
- You decide to use IP to find the best N-product solution, for N = 21 to 56
  - Let y_i be 0 if you don't produce product ***i***, 1 if you do
  - Let M_i be the maximum of product ***i*** you could produce

$$x_i \le M_1 y_1 \text{ for all } i$$

$$\sum_i y_i = N$$

- Suppose there's a minimum production quantity L_i for each product you opt to produce. What constraints implement that?

Price Breaks/Quantity Discounts

- A typical situation:
  - The first N_1 items cost $\$D$ apiece
  - The next N_2 items cost, say $\$0.8D$ apiece
  - The next N_3 items cost, say, $\$0.6D$ apiece
- With no constraints, an optimization will try to buy the cheapest ones first
- So, how do we implement these conditions?
  - Let x_1, x_2, x_3 be the number bought at each price
  - Let y_1, y_2 be binary

$$x_1 \le N_1 \,,\; y_1 \le x_1 / N_1$$

$$x_2 \le N_2 * y_1$$

$$y_2 \le x_2 / N_2 \,,\; x_3 \le N_3 * y_2$$

A Slightly Different Scheme

- Suppose instead the price break is as follows:
  - Buy up to N_1 items: cost $\$D$ apiece
  - Buy up to N_2 items: cost, say $\$0.8D$ apiece
  - Buy up to N_3 items: cost, say, $\$0.6D$ apiece
- This defines a cost curve for the items
  - Let x_1, x_2, x_3 be the number bought
  - Let y_1, y_2, y_3 be binary; the constraints are:

$$x_1 \le N_1 * y_1$$

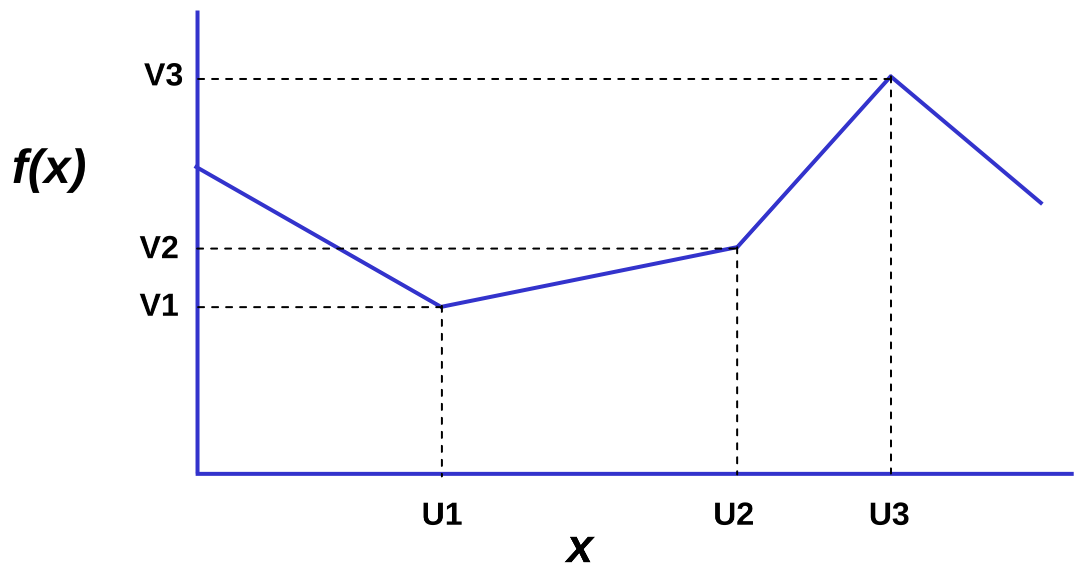
$$x_2 \le N_2 * y_2 \,,\; x_2 \ge N_1 * y_2$$

$$x_3 \le N_3 * y_3 \,,\; x_3 \ge N_2 * y_3$$

$$y_1 + y_2 + y_3 \le 1$$

Modeling Piecewise Linear Functions

- Suppose you have some variable x that has a “piece-wise linear” cost function:



- This appears to be a forbidding thing to model

But, We Can Handle It

- Replace every occurrence of the variable ***x*** in the model with:

$$U_1 y_1 + U_2 y_2 + U_3 y_3 + \ldots + U_n y_n$$

- Replace every occurrence of ***f(x)*** within the model with:

$$V_1 y_1 + V_2 y_2 + V_3 y_3 + \ldots + V_n y_n$$

- Add the constraints

$$y_1 + y_2 + y_3 + \ldots + y_n = 1$$

$$0 \le y_i \le 1 \text{ for all } i$$

Handling the Adjacency Condition

- Note that for this to work:
  - The ***y***'s give the weight on the ***i***th and ***i+1***st point (they are NOT binary!)
  - At most two ***y***'s can be nonzero, and they must be adjacent
- Winston shows how to do this with a bunch more constraints (p. 482)
- That is NOT what we're going to do
- Instead, tell your solver that these variables are type "SOS2"
- The solver will *automatically* enforce the adjacency condition

OK, So What is an SOS Variable?

- SOS stands for “special ordered set”
  - There are two general types
  - SOS type 1: a set of variables for which at most 1 may be nonzero
  - SOS type 2: a set of variables for which at most 2 may be nonzero; the two must be adjacent
- SOS variable processing is a special procedure inside the simplex algorithm
- Generally much more efficient to use SOS
- Warning: not all solvers implement SOS variables, nor are the definitions standard!

If You're Stuck Without SOS Variables

- The following set of constraints will enforce the adjacency condition:

$$
\begin{aligned}
&y_1 \le w_1 \\
&y_2 \le w_1 + w_2 \\
&y_3 \le w_2 + w_3 \\
&\vdots \\
&y_{n-1} \le w_{n-2} + w_{n-1} \\
&y_n \le w_{n-1} \\
&\sum_i w_i = 1 \\
&w_i \in \{0,1\} \text{for all } i
\end{aligned}
$$

Example: A Mining Problem (Williams, 1985)

- A company has 4 mines it can operate for the next 5 years
  - They can operate at most 3 mines a year
  - They pay a yearly royalty every year a mine is open
  - Once the mine is closed, it's closed permanently
- Each mine, if operating, has a max production each year, and a known "ore quality"
- There are yearly targets for overall ore quality, which is a weighted average of the quality of the outputs from the mines
- The selling price/ton of ore is known
- What mines should the company operate, and when?

Example: A Mining Problem (Williams, 1985)

- A company has 4 mines it can operate for the next 5 years
  - They can operate at most 3 mines a year
  - They pay a yearly royalty every year a mine is open
  - Once the mine is closed, it's closed permanently
- Each mine, if operating, has a max production each year, and a known "ore quality"
- There are yearly targets for overall ore quality, which is a weighted average of the quality of the outputs from the mines
- The selling price/ton of ore is known
- What mines should the company operate, and when?

Mining Problem (cont'd)

- Indicies
  - ***i*** = mine {1-4}
  - ***t*** = year {1-5}
- Data
  - $\boldsymbol{ROYALTY_i}$ = yearly royalty paid if mine ***i*** is open
  - $\boldsymbol{PROD_i}$ = limit on yearly production in mine ***i***
  - $\boldsymbol{QUAL_i}$ = quality of mine ***i*** ore
  - $\boldsymbol{PRICE_i}$ = selling price of blended ore
  - $\boldsymbol{D_t}$ = discount rate in year ***t*** (1, 0.9, 0.81, …)
  - $\boldsymbol{RQUAL_t}$ = required quality of blended ore in year ***t***
- Variables?

Mine Decisions

- Each year, we have to decide whether to keep a mine open, and if so, whether to produce
- So:
  - o_{it} = 1 if mine ***i*** is open in year ***t***, 0 otherwise
  - p_{it} = 1 if mine ***i*** produces in year ***t***, 0 otherwise
  - x_{it} = amount of ore produced by mine ***i*** in year ***t***
- Objective
  - $$\max z = \sum_{it} D_t * PRICE * x_{it} - \sum_{it} D_t * ROYALTY_i * o_{it}$$

Enforcing Opening, Closing, and Production

- If a mine's closed, it can't produce:

$$p_{it} \le o_{it} \text{ for all } i, t$$

- Once a mine's closed, it stays closed:

$$o_{it} \ge o_{i,t+1} \text{ for all } i, t < 5$$

- Limit production of open mines:

$$\sum_i p_{it} \le 3 \text{ for all } t$$

$$PROD_i * p_{it} \ge x_{it} \text{ for all } i, t$$

Finally, the Quality Requirements

- Blending constraints:

$$\frac{\sum_i QUAL_i * x_{it}}{\sum_i x_{it}} = RQUAL_t \text{ for all } t$$

- Linearize:

$$\sum_i QUAL_i * x_{it} = RQUAL_t * \sum_i x_{it} \text{ for all } t$$

- Nonnegativity and binary variable constraints:

$$x_{it} \geq 0 \text{ for all } i, t$$
$$o_{it}, p_{it} \in \{0,1\} \text{ for all } i, t$$

Covers, Partitions, Packs

- These are very common types of IP's
- General description of a cover:
  - Have some set of objects S = {1,2,3, … N}
  - Also have a collection of subsets of S, e.g.,
    - s1 = {1,2}
    - s2 = {1,3,5}
    - s3 = {2,6}
  - Each subset has a cost associated with it ($\boldsymbol{C_i}$)
- Objective is to "cover" S with some collection of the subsets at minimum cost
  - Each element of S must be in *one or more* of the chosen subsets
  - Want to choose the minimum-cost collection of subsets

Winston Ex. 5, p. 486-487

$$
\begin{aligned}
\min z = \; & c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5 + c_6x_6 \\
& x_1 + x_2 \geq 1 \\
& x_1 + x_2 + x_6 \geq 1 \\
& x_3 + x_4 \geq 1 \\
& x_3 + x_4 + x_5 \geq 1 \\
& x_4 + x_5 + x_6 \geq 1 \\
& x_2 + x_5 + x_6 \geq 1
\end{aligned}
$$

$x_i \in \{0,1\}$ for all i

[box] = subset; each constraint is an object

General Form

- The standard cover problem is:

$$\min z = \sum_i C_i * x_i$$

subject to

$$\sum_i A_{ij} * x_i \geq 1 \text{ for all } j$$

$$x_i \in \{0,1\} \text{ for all } i$$

- Data:
  - $\boldsymbol{A_{ij}}$ = 1 if subset ***i*** covers object ***j***, 0 otherwise

Partitions, Packs

- Partition: each object can only be covered by 1 subset

$$\min z = \sum_i C_i * x_i$$

subject to

$$\sum_i A_{ij} * x_i = 1 \text{ for all } j$$

$$x_i \in \{0,1\} \text{ for all } i$$

- Pack: each subset has value $\boldsymbol{V_i}$, and we want to maximize the value of the subsets "packed" in:

$$\max z = \sum_i V_i * x_i$$

subject to

$$\sum_i A_{ij} * x_i \leq 1 \text{ for all } j$$

$$x_i \in \{0,1\} \text{ for all } i$$

Pack Example (Winston p. 555, #21

- Indicies
  - ***d*** = districts { 1-8}
- Data
  - $\boldsymbol{POP_d}$ = population of district ***d*** in 1000's
  - $\boldsymbol{A_{d,d'}}$ = 1 if ambulance in ***d*** can respond to ***d'*** in time
- Variables, Objective and Constraints
  - $\boldsymbol{x_d}$ = 1 if ambulance assigned to ***d***, 0 otherwise

$$\max z = \sum_{d,d'} A_{d,d'} * POP_d * x_d$$

subject to

$$\sum_d A_{d,d'} * x_d \le 1 \text{ for all } d'$$

$$\sum_d x_d = 2$$

$$x_d \in \{0,1\} \text{ for all } d$$

Another Pack Example (from Schrage)

- A financial firm wants to package a set of mortgages
  - They want to maximize the number of packages
  - Each package must be worth at least $1M
- Mortgage values:

	A	B	C	D	E	F	G	H	I
Value (1000's)	910	870	810	640	550	250	120	95	55

- There are 270 packages that are worth more than $1M that contain 4 or less mortgages
- So:
  - Let $\boldsymbol{i}$ = package #, $\boldsymbol{j}$ = mortgage
  - $\boldsymbol{A_{ij}}$ = 1 if mortgage $\boldsymbol{j}$ is in package $\boldsymbol{i}$, 0 otherwise

Mortgage Packing (cont'd)

- The problem is then:

$$\max z = \sum_i x_i$$

subject to

$$\sum_i A_{ij} * x_i \leq 1 \text{ for all } j$$

$$x_i \in \{0,1\} \text{ for all } i$$

- Note that in these types of problems, you usually have to generate the subsets

MPL Code for Mortgage Packing

- The following MPL code does the mortgage packing problem, including *generating the subsets*

```
INDEX

  m  := (A,B,C,D,E,F,G,H,I,D1,D2);
  m1 := m;
  m2 := m;
  m3 := m;
  m4 := m;

DATA

  V1[m1] := (910,870,810,640,550,250,120,95,55,0,0);
  V2[m2] := (910,870,810,640,550,250,120,95,55,0,0);
  V3[m3] := (910,870,810,640,550,250,120,95,55,0,0);
  V4[m4] := (910,870,810,640,550,250,120,95,55,0,0);

BINARY VARIABLES

  x[m1,m2,m3,m4] WHERE ( (m1<m2) and (m2<m3) and (m3<m4) and
                       (V1[m1]+V2[m2]+V3[m3]+V4[m4] >= 1000) );
```

dummies (D1, D2)

Aliases of m (m1, m2, m3, m4)

MPL Code (cont'd)

```
MODEL

  max npack = sum(m1,m2,m3,m4: x[m1,m2,m3,m4]);

SUBJECT TO

  packcon[m] WHERE ( (m <> "D1") and (m <> "D2") ):

     sum( m1=m,m2,m3,m4: x[m1,m2,m3,m4]) +
     sum( m1,m2=m,m3,m4: x[m1,m2,m3,m4]) +
     sum( m1,m2,m3=m,m4: x[m1,m2,m3,m4]) +
     sum( m1,m2,m3,m4=m: x[m1,m2,m3,m4])  <= 1;

END
```

"Natural" Integer Solutions

- When solving integer or mixed-integer problems, first look at the "LP relaxation"
  - Allow integer variables to be fractional
  - Allow binary variables to be fractional, with bounds of 0 to 1
- If you solve the LP and get integral answers:
  - Quit! Answer is optimal
  - Why can't it get any better?
- We know some LP's will have integral solutions
  - If the LP is a network model
  - If the "A" matrix (all constraint coefficients) is "totally unimodular," and the constraint RHS's are integer
- But what if the LP relaxation has fractions?

Solving MIPs via Branch-and-Bound

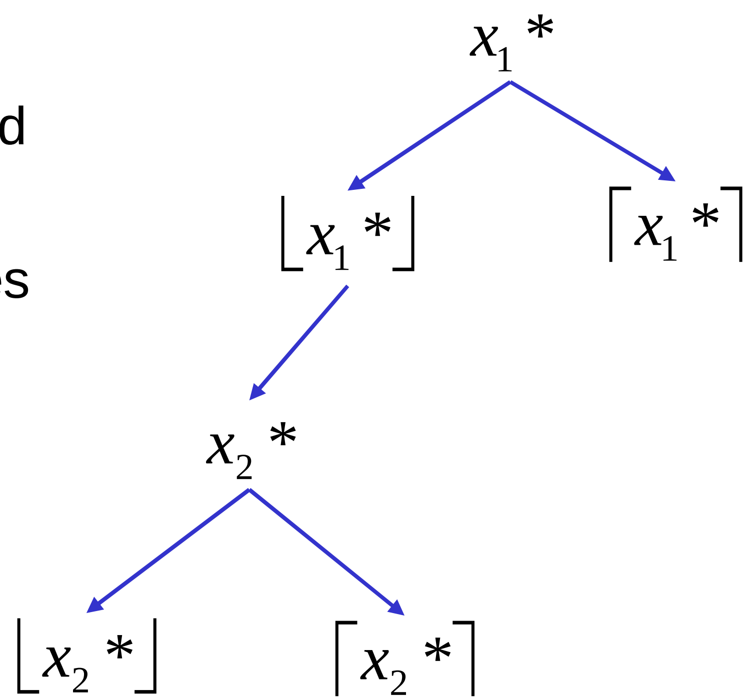
- Introduced by Land and Doig (1960)
- Ideas
  - Solve LP relaxation of problem
  - Choose a fractional variable, say x_1, with value x_1^*
  - Create two new LP's:

$$\begin{aligned} &\max z = cx \\ &\text{subject to} \\ &Ax \le b \\ &x_1 \le \lfloor x_1^* \rfloor \\ &x \ge 0 \end{aligned}$$

$$\begin{aligned} &\max z = cx \\ &\text{subject to} \\ &Ax \le b \\ &x_1 \ge \lceil x_1^* \rceil \\ &x \ge 0 \end{aligned}$$

Branch-and-Bound (cont'd)

- Adding these restrictions and resolving (via dual simplex) is very quick
- Leads to a "tree" of solutions:
  - Each branch tightens upper bound
  - Each branch adds a constraint
  - Each branch (hopefully) eliminates a fractional variable
- Some issues:
  - What do you branch on?
  - How do explore the tree?
  - How do you know when you're done?



Node Selection

- Commercial codes have lots of clever tricks
  - Look at the objective function coefficients/reduced costs of the fractional variables
  - Look at “degree of fractionation” (where .5 is the most fractional)
- Branch priorities
  - Supplied by users
  - Tells code which variables to branch on first
  - Example: $\boldsymbol{y_1}$ = build a factory, $\boldsymbol{y_{11} \ldots y_{1n}}$ produce products at that factory
  - Which variable should you branch on first?
  - NOTE: MPL doesn’t appear to support this, although CPLEX does

Branch-and-Bound: Probing

- Commercial codes look at the "implications" of a branch
- Suppose we have the following constraints:

$$
\begin{aligned}
x_1 + x_2 + x_3 \qquad\qquad &\le 1 \\
x_1 + x_2 \qquad + x_4 \qquad &\le 1 \\
x_2 + x_3 + x_4 + x_5 &\le 1 \\
x_i \in \{0,1\} \text{ for all } i &
\end{aligned}
$$

- Suppose we solve the relaxation, and $\boldsymbol{x_2} = 0.5$
  - What happens if we set $\boldsymbol{x_2} = 0$?
  - What happens if we set $\boldsymbol{x_2} = 1$?

Branch-and-Bound: Tree Traversal

- Tradeoff here is feasibility versus optimality
- Winston's "last-in-first-out"
  - Actually is "depth-first-search"
  - Technique is to dive as deep into the tree as necessary to get an integer feasible solution
  - Idea is to get integer feasible first, then search for improvement
  - Getting an integer feasible solution provides a lower bound, may cut off large parts of the tree later
- See Winston, figures 10-17, pp. 512-517

More Traversal

- If you already have a feasible solution, you may want to traverse the tree differently
  - Winston calls this “jumptracking”
  - Actually is “breadth-first search”
  - Instead of diving into the tree, you solve each node resulting from each branch
- Regardless of the traversal, note what happens at each node:
  - Problem is either infeasible, integer feasible, or “tightened”
  - First two cases: node is “fathomed” and no more search is necessary

Branch-and-Bound: Stopping Criteria

- Winston gives the impression you stop when everything is fathomed
- Not so - would be deadly for many big problems
  - To prove integer optimality, you need to fathom *every* node
  - Untenable for a big MIP
- Need to set an “integrality gap” (usually 0.01 - 0.05)
  - For a max problem, we have an upper bound at any stage of branch and bound
  - An integer feasible solution gives a lower bound
  - The integrality gap is usually defined as:

$$gap = \frac{upper\ bound - lower\ bound}{lower\ bound}$$

Prudence in Solving a Big MIP

- Set an iteration limit
  - Most solvers let you limit the number of iterations
  - Allows you to avoid long, useless runs
- Set a solve time limit (for same reasons as above)
- Set a loose integrality gap to start (say, 0.20)
- If you have an existing solution:
  - Compare it to the LP relaxation; use it to give advice on an integrality gap
  - Use the objective function value as a “cutoff” parameter; solver won’t explore branches worse than the cutoff
- Use branch priorities