Consider This Set Covering Problem

$$
\begin{array}{llllllll}
\min z = & 2x_1 & +x_2 & +5x_3 & +2x_4 & +x_5 & +x_6 & \\
\text{subject to} & x_1 & +x_2 & & +x_4 & +x_5 & & \geq 1 \\
& x_1 & & +x_3 & & & & \geq 1 \\
& & x_2 & & +x_4 & & & \geq 1 \\
& & & x_3 & & & +x_6 & \geq 1 \\
& & x_2 & +x_3 & & & +x_6 & \geq 1 \\
\multicolumn{8}{l}{x_i \in \{0,1\} \text{ for all } i}
\end{array}
$$

I claim I can solve this by inspection

Now I Start Throwing Things Away ...

$$\begin{array}{llllllll}
\min z = & 2x_1 & +x_2 & +5x_3 & +2x_4 & +x_5 & +x_6 & \\
\text{subject to} & \cancel{x_1} & \cancel{+x_2} & & & \cancel{+x_5} & & \cancel{\geq 1} \\
 & x_1 & & +x_3 & & & & \geq 1 \\
 & & x_2 & & +x_4 & & & \geq 1 \\
 & & & x_3 & & & +x_6 & \geq 1 \\
 & & \cancel{x_2} & \cancel{+x_3} & & & \cancel{+x_6} & \cancel{\geq 1} \\
\multicolumn{8}{l}{x_i \in \{0,1\} \text{ for all } i}
\end{array}$$

The first and last constraints are redundant – why?

$$\begin{array}{llllllll}
\min z = & 2x_1 & +x_2 & \cancel{+5x_3} & +2x_4 & +x_5 & +x_6 & \\
\text{subject to} & \cancel{x_1} & \cancel{+x_2} & & & \cancel{+x_5} & & \cancel{\geq 1} \\
 & x_1 & & \cancel{+x_3} & & & & \geq 1 \\
 & & x_2 & & +x_4 & & & \geq 1 \\
 & & & \cancel{x_3} & & & +x_6 & \geq 1 \\
 & & \cancel{x_2} & \cancel{+x_3} & & & \cancel{+x_6} & \cancel{\geq 1} \\
\multicolumn{8}{l}{x_i \in \{0,1\} \text{ for all } i}
\end{array}$$

x3 can be set to 0 - why?

Answer: $x_1 = 1$, $x_2 = 1$, $x_6 = 1$, $z = 4$

Presolve and Node Reductions

- An important feature of commercial codes is presolve
  - Looks at problem structure, particularly binary variables
  - Uses various techniques to reduce the problem
  - Can be applied at any node in a branch-and-bound tree
- These techniques are responsible for much recent improvement in MIP codes
- Following is a (partial) set of rules for cover (>=) and partition (=) problems
  - Note: can covert a pack to a partition by adding slack variables
  - Then, use the rules for a partition
  - These rules assume the $\boldsymbol{C_i}$'s are all > 0

Reduction Rules

- (1) (cover, partition): If all $\boldsymbol{A}_{ij}$'s are 0 in row i, the problem's infeasible

$$
\begin{array}{lccccccl}
\text{cover} & 0 & 0 & 0 & 0 & 0 & 0 & \geq 1 \\
\text{partition} & 0 & 0 & 0 & 0 & 0 & 0 & = 1
\end{array}
$$

- (2)(cover, partition) If row i has 1 nonzero $\boldsymbol{A}_{ij}$ (say, $\boldsymbol{A}_{ik}$), then set $\boldsymbol{x}_{ik}$ = 1, delete column k, and delete all rows r with $\boldsymbol{A}_{rk}$ = 1

partition

$$
\begin{array}{ccccccccl}
 & & & \vdots & & & & & \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & \geq 1, =1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & \geq 1, =1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & \geq 1, =1 \\
 & & & \vdots & & & & & \\
 & & & k & & & & &
\end{array}
$$

cover

More Reduction Rules

- (2a) (partition) In addition to the row deletions in (2), delete every column where $\boldsymbol{A_{tj}} = \boldsymbol{A_{tk}} = 1$, $\boldsymbol{j}$ <> $\boldsymbol{k}$, for every row $\boldsymbol{r}$ deleted

$$
\begin{array}{ccccccccc}
 & & & \vdots & & & & & \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & = & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & = & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & = & 1 \\
 & & & \vdots & & & & &
\end{array}
$$

Variable for this column = 1, forces all others to 0

- (3) (cover, partition) If $\boldsymbol{A_{rj}} >= \boldsymbol{A_{ij}}$ for all $\boldsymbol{j}$ for rows $\boldsymbol{r}$ and $\boldsymbol{i}$, delete row $\boldsymbol{r}$

$$
\begin{array}{cccccccc}
(r) & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
(i) & 0 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

Yet More Reduction Rules

- (3a) (partition) As in (3), but also delete all columns with $\boldsymbol{A_{rk}} = 1$ and $\boldsymbol{A_{ik}} = 0$

$$\begin{array}{c|c|cc|ccc|c|}
 & & & & & \vdots & & \\
(r) & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
(i) & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
 & & & & & \vdots & &
\end{array}$$

One of these variables will be = 1, forces all others to 0

- (4) (cover,partition) If ***S*** is a set of columns, and

$$\sum_{j \in S} A_{ij} = A_{ik} \text{ for all } i,$$

$$k \notin S, \text{ and } \sum_{j \in S} C_j \leq C_k$$

then, delete column ***k***

Last of the Reductions

- **Reduction (4)**:

obj	2	4	2	1	1	10
row 1	1	0	0	1	0	1
row 2	0	1	0	1	1	1
row 3	0	0	0	0	1	0

- **Reduction (4a)** (cover) as in (4), but with condition

$$\sum_{j \in S} A_{ij} > A_{ik} \text{ for all } i$$

Example: Winston p. 477, ex. 5 (cover)

row 1	1	1	0	0	0	0
row 2	1	1	0	0	0	1
row 3	0	0	1	1	0	0
row 4	0	0	1	1	1	0
row 5	0	0	0	1	1	1
row 6	0	1	0	0	1	1

Rule 3: Delete row 2 (covered by row 1)

Rule 3: Delete row 4 (covered by row 3)

Example (cont'd)

- No more reductions, but can you solve the problem?

$$\begin{array}{lcccccc}
\text{row } 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\text{row } 3 & 0 & 0 & 1 & 1 & 0 & 0 \\
\text{row } 5 & 0 & 0 & 0 & 1 & 1 & 1 \\
\text{row } 6 & 0 & 1 & 0 & 0 & 1 & 1
\end{array}$$

- $\boldsymbol{x_2} = 1$, $\boldsymbol{x_4} = 1$

Strong Versus Weak Formulations

- An example from my past:
  - Job was associated with an airlift analysis
  - Had 100 possible onload locations in the U.S.
  - Needed to reduce locations to 10-20; all cargo from other locations would go to one of the chosen “hubs”
  - Wanted to minimize total tonnage*distance to move cargo to hubs
  - Known as a “k-median” problem
- First used a heuristic on the problem
- Was learning GAMS at the time, so I set it up as an integer program

The First K-Median Formulation

- Indicies
  - ***i,j*** = locations
- Data
  - $\boldsymbol{STONS_i}$ = short tons to be moved from location ***i***
  - $\boldsymbol{DIST_{ij}}$ = distance between ***i*** and ***j***
  - ***MAXHUBS*** = maximum number of hubs
  - ***NUM*** = total number of locations
- Variables
  - $\boldsymbol{assign_{ij}}$ = 1 if location ***i*** assigned to hub ***j***, 0 otherwise
  - $\boldsymbol{choose_j}$ = 1 if location ***j*** chosen as a hub, 0 otherwise

The First Model

- Objective and constraints:

$$\min z = \sum_{ij} DIST_{ij} * STONS_i * assign_{ij}$$

subject to

$$\sum_j assign_{ij} = 1 \text{ for all } i$$

$$\sum_j choose_j \leq MAXHUBS$$

$$\sum_i assign_{ij} \leq NUM * choose_j \text{ for all } j$$

$$assign_{ij} \in \{0,1\} \text{ for all } i,j$$

$$choose_j \in \{0,1\} \text{ for all } j$$

What do these constraints do?

What does this constraint do?

What do these constraints do?

No Luck

- Tried to solve this in OSL
  - Still didn't meet integrality gap requirements after 100,000 iterations
  - Ran for several hours
  - No progress
  - Went back to heuristic, wondered what I did wrong
- Asked an optimization professor a year later at a meeting
  - He sent back an answer the next day
  - His change allowed OSL to solve the problem in about 10 seconds
  - What was it?

A Stronger Formulation

- All he suggested was the following:

$$\min z = \sum_{ij} DIST_{ij} * STONS_{ij} * assign_{ij}$$

subject to

$$\sum_{j} assign_{ij} = 1 \text{ for all } i$$

$$\sum_{j} choose_{j} \leq MAXHUBS$$

$$\boxed{assign_{ij} \leq choose_{j} \text{ for all } i,\ j}$$

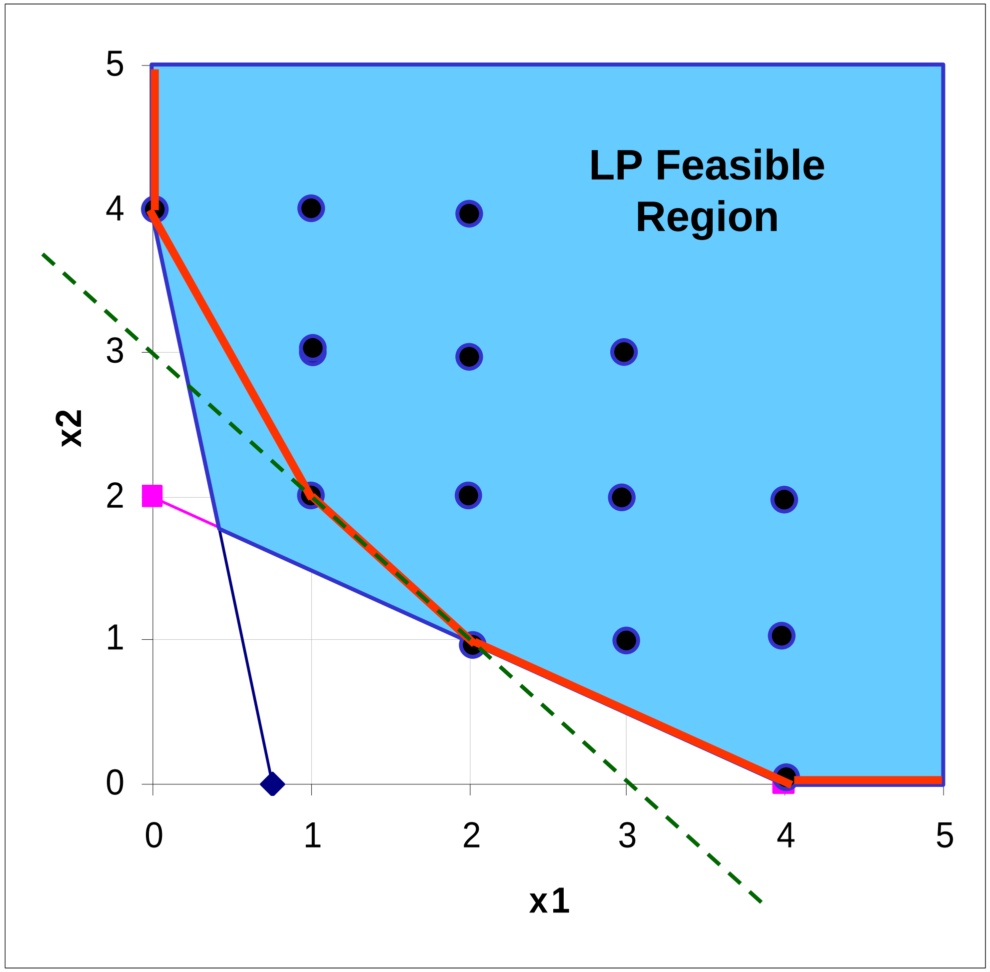
$$assign_{ij} \in \{0,1\} \text{ for all } i,j$$

$$choose_{j} \in \{0,1\} \text{ for all } j$$

- Note that this increased the number of constraints by 100 x 100 - 100 = 9900
- How could it be so much faster?

With MIPs, More Constraints Are Better

- The first formulation encouraged “fractionation” of the binary variables
- The second cuts off many possible fractional solutions
- Want to get as close to the “integer hull” as possible



Another Strengthening Example

- From the mining example:

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

$$e.g.,$$

$$o_{i1} \geq o_{i2}$$

$$o_{i2} \geq o_{i3}$$

$$\vdots$$

- A stronger set of constraints:

$$o_{it} \geq o_{i,t'} \text{ for all } i, t < 5, t' > t$$

$$e.g.,$$

$$o_{i1} \geq o_{i2}$$

$$o_{i1} \geq o_{i3}$$

$$o_{i1} \geq o_{i4}$$

$$o_{i1} \geq o_{i5}$$

$$o_{i2} \geq o_{i3}$$

$$\vdots$$

Cuts

- See Winston, Sec. 9-8
- Note that branching requires solving two LPs
  - One for the integer floor of the branching variable
  - One for the integer ceiling of the branching variable
- An alternative approach is called a *cut*
  - The idea here is to “cut off” the fractional solution, but don’t cut off any feasible integer solutions
  - The aim is to generate constraints that form the integer hull of the feasible region
  - Such constraints are called *facets*

From the Dual Simplex Lesson (6-1)

- Recall this was the optimal (fractionated) tableau:

Row	z	x1	x2	s1	s2	RHS	BV
0	1	0	0	2/5	9/5	44/5	z
1		1	0	-2/5	1/5	4/5	x1
2		0	1	1/5	-3/5	8/5	x2

- Row 2 can be written as:

$$x_2 + \frac{1}{5}s_1 - \frac{3}{5}s_2 = \frac{8}{5}$$

- In Lesson 6-1, I used this row (called a *source row*) to generate a mysterious constraint; how did I do that?

Generating a Gomory Cut

- We rewrite this constraint by recognizing that any fraction can be written as

$$x = \lfloor x \rfloor + f, 0 < f < 1$$

- So, applying this to Row 2, we get:

$$x_2 + \left(0s_1 + \frac{1}{5}s_1\right) + \left(-s_2 + \frac{2}{5}s_2\right) = \left(1 + \frac{3}{5}\right)$$

- Now, group the integral terms on the left and the fractional terms on the right:

$$x_2 + 0s_1 - s_2 - 1 = \underbrace{-\frac{1}{5}s_1 - \frac{2}{5}s_2 + \frac{3}{5}}_{\textbf{Part we would like to get rid of}}$$

Some Arguments

- For integer feasibility:
  - The left-hand side must be integer
  - Therefore, the right-hand side must be integer
  - s_1 and s_2 must be >= 0
- So, what's the biggest the right-hand side can be and still be feasible?
- Result: we add the cut:

$$-\frac{1}{5}s_1 - \frac{2}{5}s_2 + \frac{3}{5} \le 0, \text{ or}$$
$$-\frac{1}{5}s_1 - \frac{2}{5}s_2 + s_3 = -\frac{3}{5}$$

- Is this cool, or what?

More Info on Cuts

- Cutting plane algorithms had a bad reputation early
  - Algorithms only added one cut at a time
  - Had very slow convergence
- Have recently become very popular
  - No reason to add cuts one at a time
  - Can add a cut for virtually any fractional row
  - Can combine with branch-and-bound (branch on one variable, generate cuts for others)
  - Easy to implement, run very quickly
- Bixby article shows that installing these cuts in CPLEX gives tremendous improvements

A (Very) Quick Tour of CPLEX MIP Switches

- For a small MIP or one known to be easy, you can stick with the defaults
- For anything else, you should ***always*** set the following:
  - **Time limit** (p. 95): CPLEX has a huge default (100,000,000 hours, a bit longer than I'd wait)
  - MIP strategy (p. 98): choose depth-first to emphasize feasibility, others to search for better solutions
  - Upper cutoff/lower cutoff (p. 106): *if you have a solution*, set these to avoid unproductive parts of the b-b tree
  - **Relative/absolute gap** (p. 106): a good starting relative gap is 0.10; absolute gap depends on the problem

CPLEX Switches You Can Play With

- Bound strengthening, coefficient reduction (p. 90)
  - These are more aggressive prereduce options
  - You should consider them if you have lots of binary variables and “chains” of relationships
- MIP probing (p. 99)
  - Explores implications of binary settings at every node
  - Time consuming, but may crack the problem early
- Variable selection (p. 99)
  - Strong branching is “probing lite” - can be very helpful
  - Maximum infeasibility branching is useful if you have feasible solutions and want to get faster improvement

CPLEX Cuts

- CPLEX can employ 9 different types of cuts
  - Some are easy (like Gomory fractional cuts)
  - Some involve substantial math (disjunctive cuts)
  - Not easy to figure out *a priori* which will work
- Some general advice
  - CPLEX is fairly intelligent on when to apply cuts
  - If you're really having trouble, go aggressive on everything (kitchen sink approach)
  - Bixby's article gives good statistics on general performance of cuts on a large suite of MIPs
  - Clique cuts good for partition problems; cover cuts good for covers
  - Implied bound cuts good for problems with lots of general integer variables

Conclusion

- Be prepared for a lot of work with a big MIP
  - Exploit as much problem structure as you can
  - Use strong formulations; when in doubt, add more constraints
  - Help the solver with cutoff values and branch priorities
  - **First get a feasible answer**, then work from there
- Once you're feasible, work on improvement
  - Throw more switches to drive down the integrality gap
  - Recognize that some problems have "loose" LP formulations and require *very* long b-b solves to tighten the gap
  - Pay close attention to the structure of the interim feasible solutions
  - Add more constraints if you see opportunities (like the *NOSWOT* problem)

Constraint-Satisfaction Problems (CSPs)

- Sometimes we just want to find a *feasible* solution
- Map-coloring problem:
  - assign colors to maps so no adjacent countries have the same color
- Stable marriage problem
  - Have a group of N men, and a group of N women
  - Each woman has rated the men 1-N, as have each of the men
  - Assign men to the women so that if Man A prefers Man B's wife, Man B's wife prefers her husband to Man A
- Scene labeling
  - Recognize 3-D objects by assigning lines in 2-D drawings

The Idea of Constraint Programming

- Basic algorithm
  - You have a set of variables, each with a finite domain
  - You have a set of constraints that determine allowable settings on combinations of variables
  - Successive applications of those constraints reduce the domains of the variables
  - Stop when you come up with variable settings that satisfy all constraints
- Several commercial products, such as ILOG's OPL, provide a language for constraint programming

Integer Programming for CSPs

- In some cases, we can write integer programs to solve CSPs
- Consider SuDoKu
  - Problems consist of a 9 x 9 grid
  - Have to assign numbers 1-9 so that each row, column, and the 9 3 x 3 subgrids contains each number exactly once
- How do you solve these manually?
- Chances are, you use your own version of constraint programming

The Challenge

- Formulate an integer program in MPL to solve the SuDoKu problem shown to the right
- Furthermore, SuDoKu puzzles are advertised to have a *single* solution
- Does this one have a single solution? Modify your formulation to find out

7		3		1				
	6		8	4		3		
		5					8	
						2		8
	2		1				6	
6		9						
	5					1		
		6		3	5		4	
				2		7		9