Formulation IV: Recourse Models

- Have stuck with basic LP assumptions so far
  - Proportionality, additivity, divisibility, certainty
  - Have vaguely discussed relaxing divisibility with integer variables
- Can we relax the certainty assumption?
  - Examples so far contain many things that could (or are) random
  - Seems particularly bad to take randomness out of things like customer demand
  - Are there any ways to represent randomness in an LP?
- Falls into a general area called *stochastic programming*
  - Models range from straightforward to **very, very tough**
  - Math can be very difficult
  - I will present one *simple* type of SP
  - Good references: Schrage (LINDO), Kall and Wallace

General Motivation: Recourse Models

- Model type considered here is as follows:
  - We make some decision
  - Nature chooses an outcome *scenario* (we know the distribution of the scenarios, though)
  - We take some action (called *recourse*) based on the natural outcome
- Corrects a very basic flaw in a great deal of OR work
  - **Analyst to decision maker**: "You tell me the future, and I'll tell you what you should do. If you predict the future wrong, it's your problem."
  - **Decision maker to analyst**: "If I knew what the future was, I wouldn't need you. Get the %$#@^&!! out of my office."

Formulation Rules

- Certainty Equivalence Theorem:
  - If the randomness or unpredictability in problem data exists solely in the objective function coefficients, it is correct to solve the LP in regular form after simply using expected values for the random coefficients in the objective (Schrage, p. 212)
- Random elements elsewhere? Not covered in this course
- So, we set the problem up to optimize

$$\min \quad cx + EXP_s\left[h(x,s)\right]$$

cost of initial decisions → cx

expected recourse cost over all scenarios ***s*** → $EXP_s\left[h(x,s)\right]$

Example: Winston, p. 76, #6, Modified

- The issue with a lot of these problems is that the demand is random (cops per shift)
- Consider a modified problem:
  - Normal shifts are 12 hours (no 18-hour shifts)
  - We can, however, hire adjunct cops at the following rates:
    - 12am - 6am, 6pm - 12am: double time ($8 an hour)
    - 6am-12pm, 12pm-6pm: time-and-a-half ($6 an hour)
  - **Don't know if we need adjuncts in advance**
  - However, we have 3 typical scenarios with probabilities:

Shift	**Demand**		
12am -6 am	12	10	6
6am -12 pm	8	4	10
12pm -6pm	6	18	11
6pm -12am	15	12	16
scenario prob	0.5	0.25	0.25

Now, What Do We Do?

- First attempt: solve normal LP for different scenarios
- Indices
  - **t** = shifts {a12,a6,p12,p6}
- Data
  - $\mathbf{REQ_t}$ = cops/shift required
  - **NCOST** = cost/hr for scheduled shift cops ($4)
  - **NSHIFT** = length of a normal shift in hours (12)
- Variables
  - $\mathbf{cop_t}$ = number of cops starting on shift t
  - **totcost** = total cost of cops
- Objective

$$\min_{cop} \quad totcost = \sum_t NSHIFT * NCOST * cop_t$$

Example, cont'd

- Constraints

note

$$cop_{t-1} + cop_t \geq REQ_t \quad \text{for all } t \text{ (meet demands)}$$

note

$$0 \leq cop_t \leq \max_t REQ_t \quad \text{for all } t \text{ (nonnegativity + upper bounds)}$$

- Solutions for the 3 scenarios:

	Scenario		
Shift	**1**	**2**	**3**
12am -6 am	8	4	10
6am -12 pm	0	0	0
12pm -6pm	11	18	11
6pm -12am	4	6	5
Total cost	**1104**	**1344**	**1248**

- What if we choose the scenario 1 answer?

Performance of Scenario 1 Answer

- We can compute the overtime we'd see for the other scenarios, using the Scenario 1 solution:

	Scenario		
Shift	**1**	**2**	**3**
12am -6 am	0	0	0
6am -12 pm	0	0	2
12pm -6pm	0	7	0
6pm -12am	0	0	1
total overtime cost	0	252	120

- So, the total *expected* cost is $1104 + expected overtime
- This is $1104 + (0.5*0 +0.25*252 + 0.25*120) = $1197
- *An 8% increase*

Other Schemes; Introducing the Recourse LP

- Some alternative approaches
  - Optimize for max demand: expected total cost = $1440
  - Optimize for average demand: expected total cost = $1239
  - Average the optimal answers for each scenario: expected total cost = $1311
- Clearly, we're thrashing - try a new formulation
- Added indicies:
  - **s** = scenarios {s1,s2,s3}
- Added data:
  - $\mathbf{REQ_{st}}$ = cops/shift required for scenario **s**
  - $\mathbf{OTCOST_{t}}$ = overtime cost/hr for adjunct cops
  - **OSHIFT** = length of an overtime shift in hours (4)
  - $\mathbf{PROB_{s}}$ = probability of scenario **s**

Remainder of New Model

- Added variables
  - **ovr_{st}** = number adjunct cops for shift **t**, scenario **s**
- New objective function

cost of initial decisions

$$\min_{cop,ovr} \quad totcost = \sum_t NSHIFT * NCOST * cop_t +$$

$$\sum_s PROB_s * \left[\sum_t OSHIFT * OTCOST_t * ovr_{st} \right]$$

expected recourse cost
over all scenarios ***s***

Remainder of New Model (cont'd)

- Constraints

$$cop_{t-1} + cop_t + ovr_{st} \geq REQ_{st} \quad \text{for all } s, t \text{ (meet demands)}$$

$$0 \leq cop_t \leq \max_{st} REQ_{st} \quad \text{for all } s, t \text{ (nonnegativity + upper bounds)}$$

$$0 \leq ovr_{st} \leq \max_{st} REQ_{st} \quad \text{for all } s, t \text{ (nonnegativity + upper bounds)}$$

- Now, how hard was that?

And, What Answer Do We Get?

- A totally unexpected one:

		ovr(s,t)		
Shift	**cop(t)**	**1**	**2**	**3**
12am -6 am	4	0	0	0
6am -12 pm	0	4	0	6
12pm -6pm	7	0	11	4
6pm -12am	8	0	0	1
Costs	912	144	396	408

- Total expected cost: $1185
- Would have been difficult (or impossible) to come up with this using some "external" method

Commercial Modeling Languages & Solvers

- These days are over:
  - The days of writing your own LP code
  - The days of never solving *anything* as a student
  - The days of writing FORTRAN for model development (NOTE: you still may write code for model *implementation)*
  - The days of renting time on a Cray to solve a big problem
- What is the common architecture?
  - Commercial solver (e.g., CPLEX, XPRESS, OSL, MINOS, CONOPT)
    - Require a model be fed to them in a particular format
    - Allow considerable access to solver options
  - Algebraic modeling language (e.g. MPL, GAMS, AMPL)
    - Express the optimization in *algebraic* form
    - Translate input data into model coefficients
    - Generate model in solver's native form
    - Retrieve solver output and allow manipulation

What We Will Use in this Course

- Modeling language: MPL
  - Available for download from Maximal Software (www.maximalsoftware.com)
  - PDF's for user's manual, tutorial on course home page
- Commercial Solver: CPLEX
  - Extremely powerful LP and MIP solver
  - Tremendous solution time improvements over last 10 years
  - Available via Maximal Software with a 6-month student license

An Example of Commercial Solver Progress

- USAF Patient Distribution System (PDS) problems
  - From Bixby (Operations Research, vol. 50, No. 1)
  - Modeled patient evacuation from a major war
  - PDS 90: 507,771 variables, 142,823 constraints, 1.2M nonzeros
  - Unsolvable when first proposed (1990) by *any* code
- CPLEX progress on PDS 90 (run on 300MHZ SPARC)
  - CPLEX 1.0 (1988): could not solve problem
  - CPLEX 5.0 (1994): 16.67 hours
  - Special Code (Castro 2000) 6 hours
  - CPLEX 7.1 primal (2000): 41 minutes
  - CPLEX 7.1 dual (2000): 320 *seconds*
- *99.5% reduction in solve time!*

When to Use an Algebraic Language

- Development
  - For almost any straightforward application
  - Makes debugging much easier
  - Leaves behind a tool for the inevitable future changes
  - Much faster to use than writing C or VB code
- Implementation
  - Algebraic modeling languages very slow (order of magnitude worse than writing your own generator)
  - Don't use them if you require real-time response
  - Can't be used for "indirect" methods (like decomposition)
  - Generally do not allow access to all solver options
  - Are niche products, with uneven levels of support
- NOTE: these languages are improving, though

An MPL Example - the Cop Scheduling SP

- MPL file structure
  - TITLE (optional)
  - INDEX
  - DATA
  - DECISION VARIABLES
  - MODEL (this is the objective function)
  - SUBJECT TO (these are the constraints)
  - BOUNDS (as needed)
  - END
- Has a close correspondence to NPS standard format

MPL Code for SP Problem; Indices and Data

```
TITLE
    RecourseScheduling;

INDEX
    s := (s1,s2,s3);                { scenarios }
    t := (a12,a6,p12,p6) CIRCULAR;  { shifts }

DATA
    PROB[s]  := (0.5,0.25,0.25); { probability of scenario }

    REQ[s,t] := (12,8,6,15,  { cops/shift req'd, by scenario }
                 10,4,18,12,
                 6,10,11,16);

    OTCOST[t] := (8,6,6,8);  { overtime cost/hr in $ }

    NCOST := 4; {cost/hr of straight (scheduled) time in $ }

    NSHIFT := 12; { length of a normal shift in hours }

    OSHIFT := 6; {length of an overtime shift in hours }

    MAXREQ := 18; { max requirement for any shift }
```

Emphasize:
- comments
- order of indicies
- CIRCULAR definition
- data input order
- declaration of scalars

MPL Variable and Model Definitions

```
DECISION VARIABLES

    cop[t];    { number of scheduled cops starting on shift t }
    ovr[s,t];  { number of overtime cops working shift t under scenario s }

MODEL

    MIN  totexpcost = SUM(t: NSHIFT*NCOST*cop[t]) +
                      SUM(s,t: PROB[s]*OSHIFT*OTCOST[t]*ovr[s,t]);
```

Emphasize:

- use of lower case for variables, upper case for data (style, not req'd by MPL)
- MPL is CASE SENSITIVE (by default – there is a switch to turn this off)
- no explicit variable declaration for objective function value
- explicit use of indicies in sums (again, style; MPL doesn't seem to care)
- no explicit constants in equations (style)

MPL Constraints and Bounds

```
SUBJECT TO

    demand[s,t]:  {shift demand constraints}

      cop[t]+cop[t-1]+ovr[s,t]  <  REQ[s,t] ;

BOUNDS

    cop[t]   <= MAXREQ;
    ovr[s,t] <= REQ[s,t];

END
```

Some notes:

- use of circular set index in constraint
- implicit assumption that variables are nonnegative
- mistake in inequality constraint (save for debugging)

So We Run It, and It's Wrong

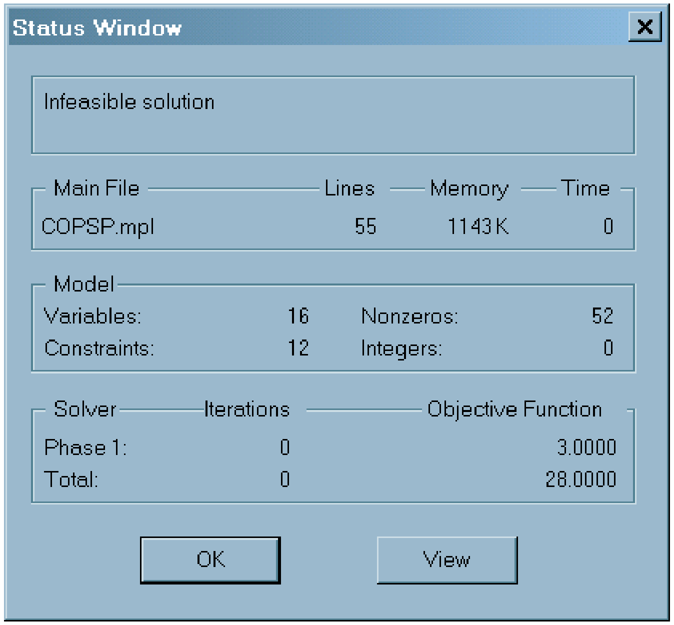
- Initial answer: 0 (look at View/Files/COPSP.sol)
  - No money spent, but no cops scheduled
  - No overtime either
  - What the hell is going on?
- Common trick: fix a variable and see what blows up
- Change one bound:

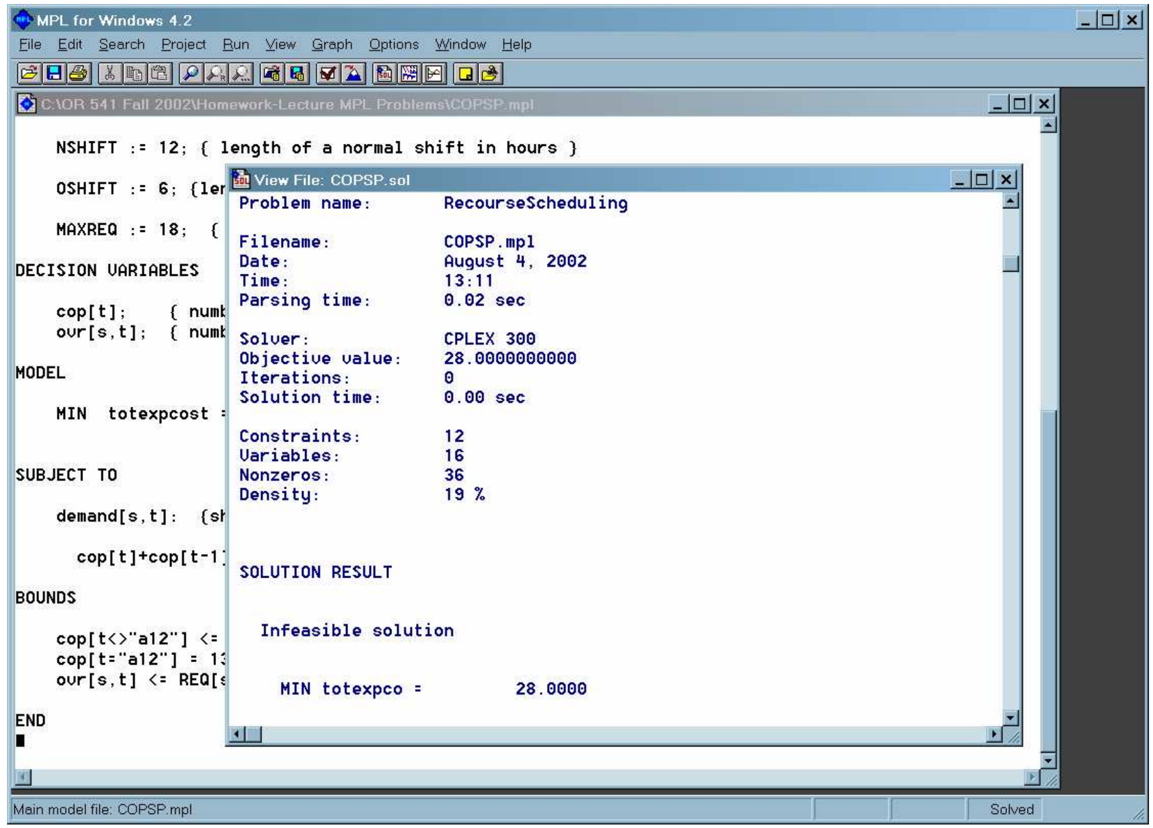
```
BOUNDS

    cop[t<>"a12"]   <= MAXREQ;
    cop[t="a12"] = 13;
    ovr[s,t] <= REQ[s,t];

END
```

Note:

- should be legit; can always hire more than required
- MPL mechanisms to restrict certain indicies

Now It's Infeasible





Now, How Do We Find What Blew Up?

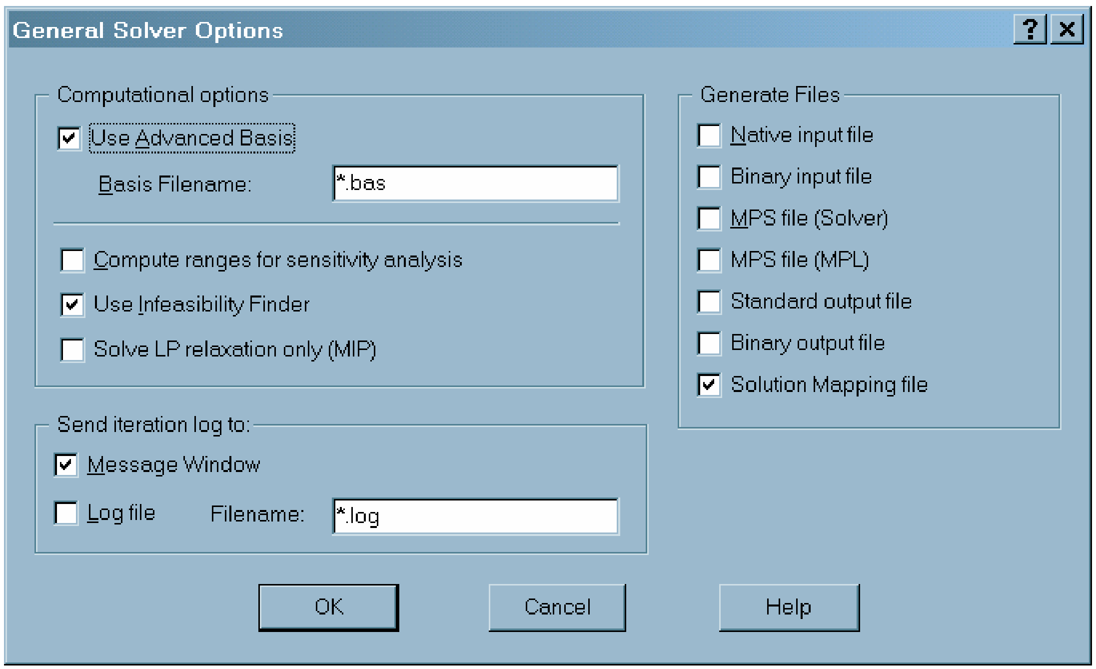
- Go to View/Files/COPSP.iis
  - “IIS” is a CPLEX option that generates a the set of inconsistent constraints
  - Here’s what’s in it:

```
\Problem name: RecourseScheduling

Minimize
subject to
\Rows in the iis:
 R10: C1 + C2 + C14 <= 10
\Columns in the iis:
Bounds
 C1 = 13
 C2 >= 0
 C14 >= 0
End
```

- What the #$%^&@!! does this mean?
  - MPL does not pass actual row and variable names to CPLEX; it maps names to generic row and column indices
  - We need to look at the mapping to see what’s what

MPL Mapping File

- You have to turn on the option to generate a map file
  - Go to Options/General Solver and check "Solution Mapping File" box under "Generate Files"



Looking at the Map File

```
MAPPING RecourseScheduling

VARIABLE DEFINITIONS
    cop[t]  (4)
    ovr[s,t]  (12)

CONSTRAINT DEFINITIONS
    demand[s,t]  (12)

VARIABLE MAPPINGS
       1)    C1,           cop,          a12, (we fixed at 13)
       2)    C2,           cop,          a6,

      14)    C14,          ovr,          s3, a6,

CONSTRAINT MAPPINGS

      10)    R10,          demand,       s3, a6,

END
```

Now We See It

- Look at the IIS again:

```
\Problem name: RecourseScheduling

Minimize
subject to
\Rows in the iis:
 R10: C1 + C2 + C14 <= 10
\Columns in the iis:
Bounds
 C1 = 13
 C2 >= 0
 C14 >= 0
End
```

- We've reversed the inequality! We should make sure we have *more* cops than the requirement, not less!

With the Fix, It Runs Fine

- Repair constraints and restore bounds:

```
SUBJECT TO

    demand[s,t]:  {shift demand constraints}

      cop[t]+cop[t-1]+ovr[s,t]  > REQ[s,t] ;

BOUNDS

    cop[t]   <= MAXREQ;
    ovr[s,t] <= REQ[s,t];

END
```

- And you get the optimal solution of $1185

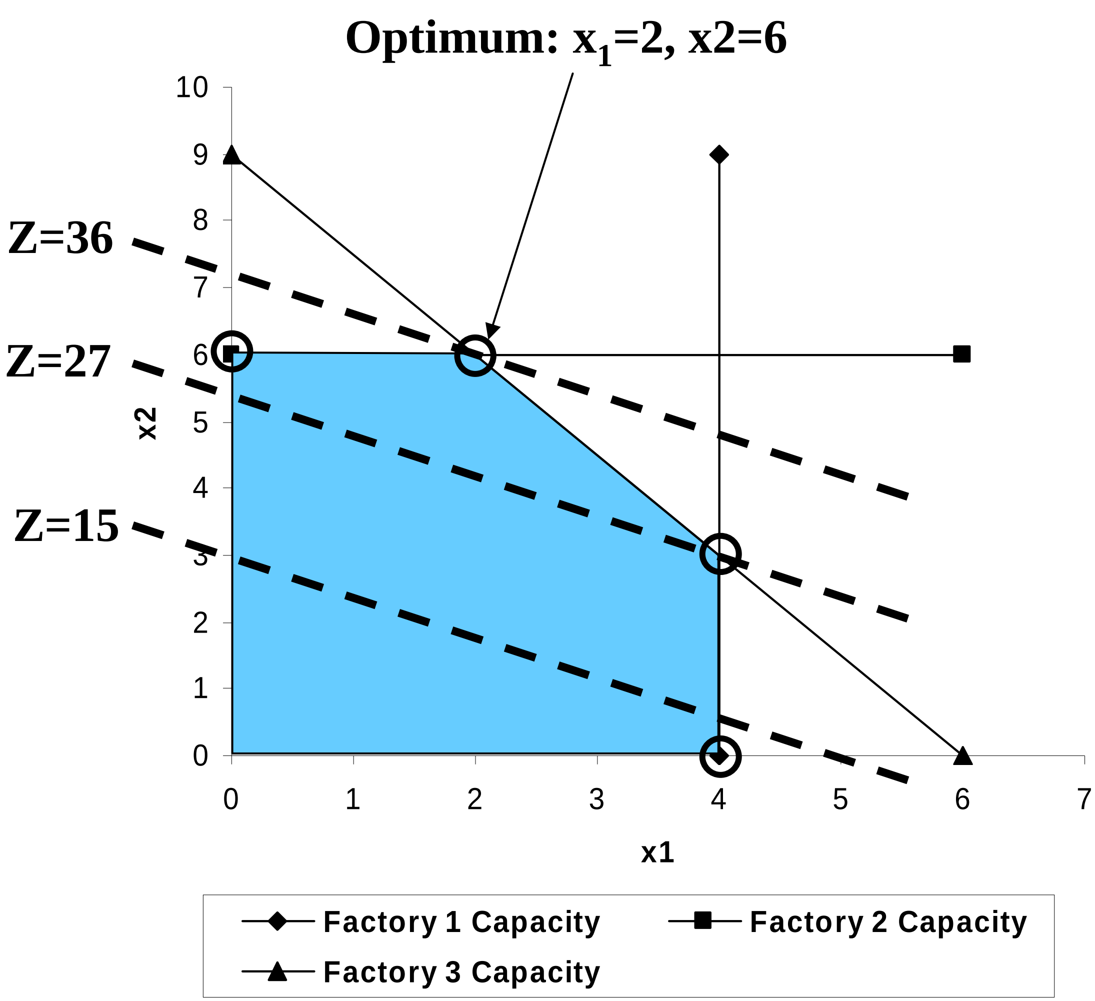
Other MPL Gotchas

- Pay attention to the unusual rules for formulas with parentheses (user manual, sec. 9.8)
- Be careful with set manipulations; lots of power available, lots of unintended effects possible
- Don't start index names with a number and then append letters (e.g, "12a"); MPL doesn't like this
- I'm learning MPL as well; I'll transmit more as the course goes along

Introduction to the Simplex Method

- By now, you suspect there is some algorithm for solving LPs
  - Can't use graphical methods in n-space
  - CPLEX shows things like "Phase I" and "iterations"
- Dantzig (1947) developed original simplex method
  - Implementation not really useable until mid-50's
  - Improvement has come from intense development of numerical linear algebra (plus many other tricks)
- Despite competition from interior-point schemes, the simplex method hangs on, because it's:
  - Fast
  - Well-understood
  - Has good restart properties (essential for IP)

Recall the Graphical Scheme



- We drew the feasible region, and then moved to objective function contour to the “best” extreme point

Recall Also the Linear Algebra Ideas

- A typical LP is in this form:

$$\max z = cx$$
$$\text{subject to } Ax \le b$$
$$x \ge 0$$

- To get it into something that looks like a linear algebra problems, we add slack variables make the constraints equalities:

$$\max z = cx$$
$$\text{subject to } Ax + s = b$$
$$x \ge 0$$
$$s \ge 0$$

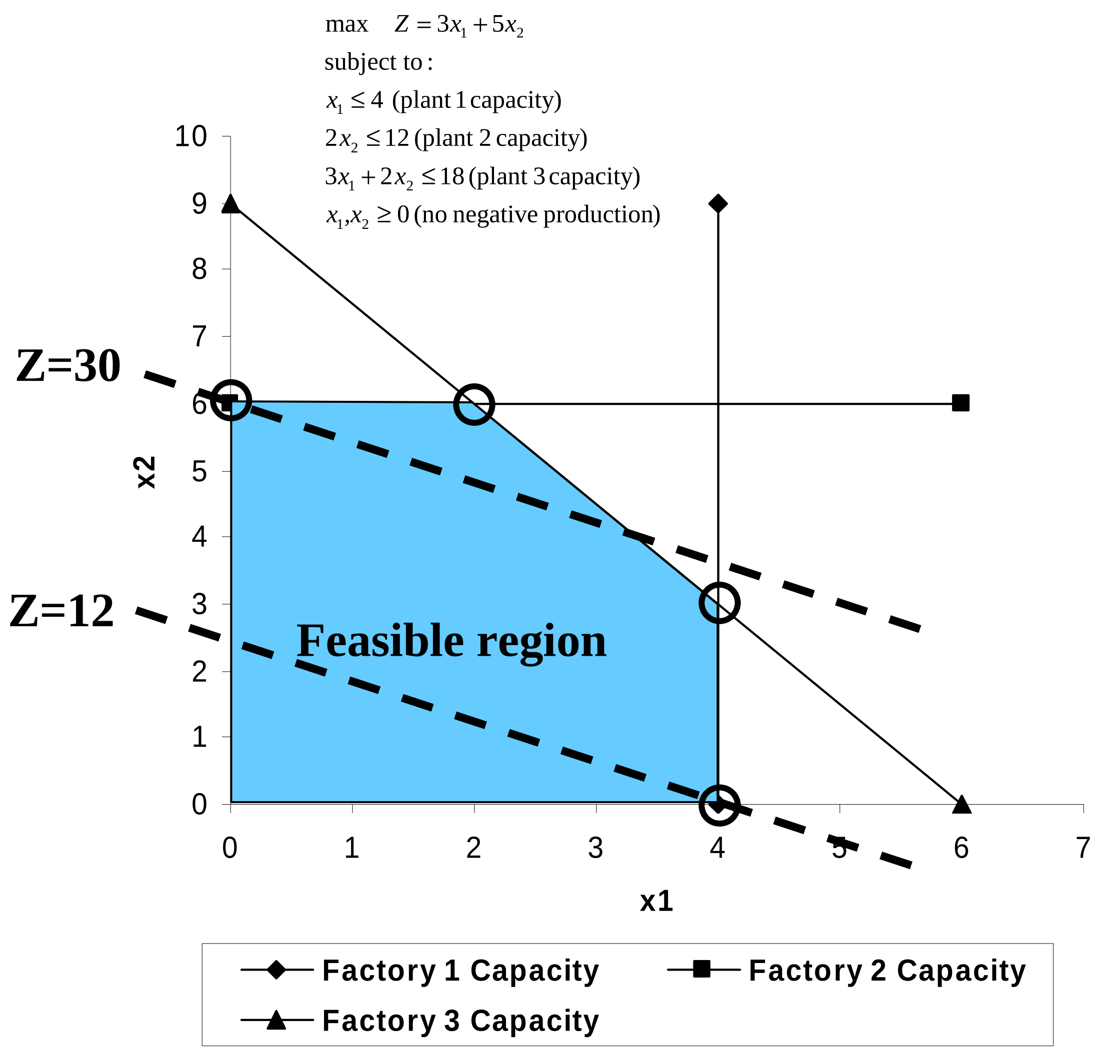
Basic Solutions

- We know how to score any proposed solution
- The question is, how do we find solutions that obey the constraints?
- Terminology for the system **Ax + s = b**:
  - **n** variables, **m** equations (constraints), assume n > m
  - **Basic solution:** a set of m variables that satisfies **Ax+s=b**; the others are set to 0 (note that a basic variable may be 0 also)
  - **Basic (nonbasic) variable**: variable (not) in the basis
  - **Basic feasible solution:** a basic solution that also satisfies the nonnegativity conditions **x >0, s>0**
  - **Adjacent basic feasible solutions:** solutions with **m-1** basic variables in common

Bases and Extreme Points: Where to Look

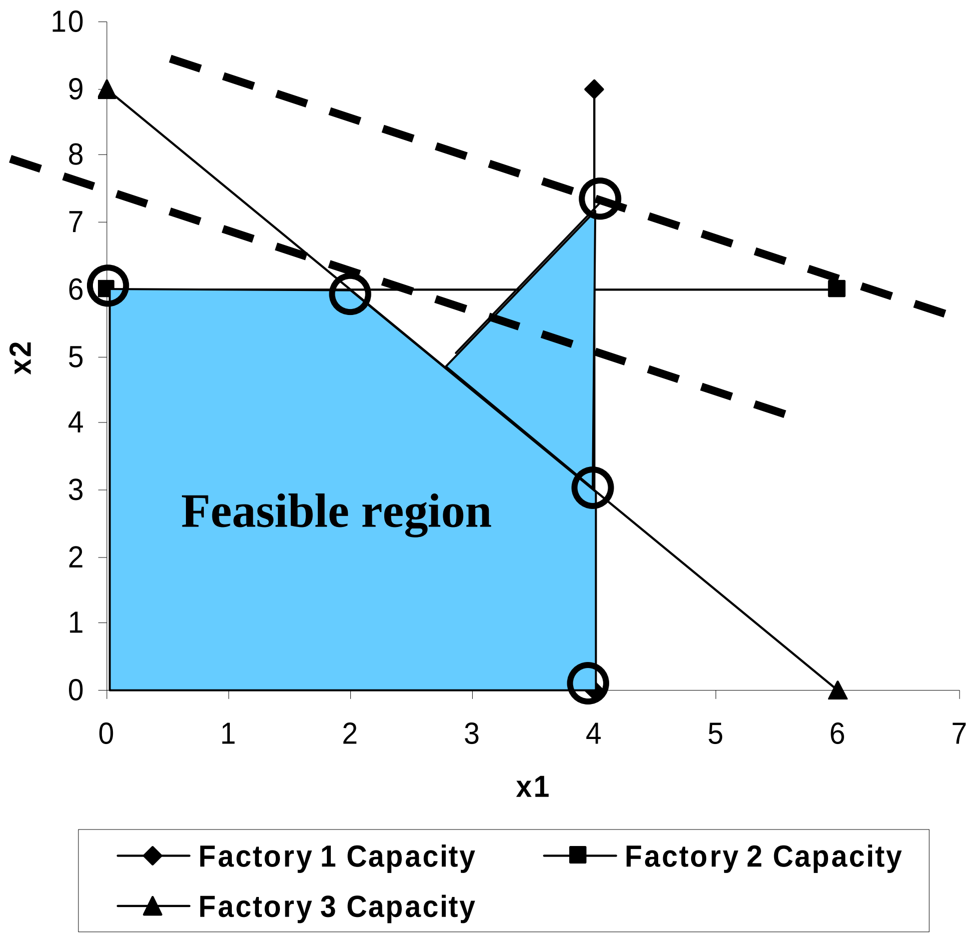
- We've shown before that the optimal solution (if there is one) will occur at an extreme point
- It turns out that BFSs are, in fact, extreme points!
- So, all we have to do is search $\binom{n}{m}$ possible bases!
- But doing it this way would be bad …
  - n = 1000, m = 200 means 6.62 x 10^{215} possible bases
  - Clearly, simplex is more efficient than that
- To build an algorithm, we need:
  - A way to start
  - A way to take improving steps
  - A way to terminate with a guaranteed optimum (if problem is feasible)

Look Again at the Graphical LP



- Suppose we start at (0,0)
  - What's the BFS, by the way?
  - It's s1 and s2
- We have two adjacent extreme points
  - Which one would be the better one to move to?
  - (0,6); why?
- Once we get to the best extreme point, how do we know we're there?
  - Convexity

How Do We Know That We're Optimal?



- Here, we thought we were done, but there was a better point!
- What's the problem with this feasible region?
  - Not convex
- What LP assumption is violated?
  - Linearity (in new constraint)
- So, we can guarantee that if all adjacent points are worse (or equal), we're optimal!