19

Design & Implementation of Optimization-Based Decision Support Systems

I don't want it perfect, I want it Thursday.
-John Pierpont Morgan

19.1 General Structure of the Modeling Process

The overall modeling process is one of:

1) Determining the need for a model.
2) Developing the model.
3) Implementing the model.

One should not skip over step (1) in one's enthusiasm to use a fancy model. Some questions to ask before deciding to use a model are:

a) Are the expected savings from using the model greater than the cost of developing and implementing the model?
b) Is there sufficient time to do (2) and (3) before the recommendation is needed?
c) Is it easier to do an experiment on the real system than to build a model? Sometime ago a question arose in a telephone company about the effect of serving certain telephone calls for information arising in city A from a central facility in city B, rather than from the existing facility in A. It was more simple to make a five minute wiring change to see what happened, than to construct an accurate statistical model. Similarly, Banks and Gibson (1997) describe a situation in which one fast food restaurant chain built a detailed model to evaluate the effect of a second drive-up window. In less time, a competitor tested the same idea by stationing a second person in the drive-up lane with a remote hand-held terminal and voice communication to the inside of the restaurant.
d) Do we have the input data needed to make plausible use of the model?

If the purpose of the model is to do a one-time analysis (e.g., to decide whether or not to make a certain investment), then step (3) will be relatively less laborious.

19.1.1 Developing the Model: Detail and Maintenance

Whether the model is intended for a one-time study or is to be used regularly has some influence on how you develop the model. If the model is to be used regularly, then you want to worry especially about the following:

Problem: The real world changes rapidly (e.g., prices, company structure, suppliers). We must be able to update the model just as fast.

Resolution: There are two relevant philosophies.

1) Keep worthless detail out of the model, follow the KISS (Keep It Simple, ...) admonition.
2) Put as much of the model into data tables rather than hard coded into the model structure.

19.2 Verification and Validation

The term *verification* is usually applied to the process of verifying that the model as implemented is actually doing what we think it should. Effectively, verification is checking the model has no unintentional “bugs”. *Validation* is the process of demonstrating that all the approximations to reality intentionally incorporated in the model are tolerable and do not sully the quality of the results from the model. Stated simply, verification is concerned with "solving the equations right", and validation is concerned with "solving the right equations", see Roache (1998). Several general approaches for verifying a model are to:

i) check the model results against solved special cases,
ii) check the model results against known extreme cases,
iii) check the model results on small examples that can be solved by hand,
iv) check that model results change in the proper direction as model inputs are changed (e.g., if the price of a raw material increases, we should not buy more of it), and
v) check that the model handles invalid cases robustly.
vi) if there are multiple dimensions, e.g. products, suppliers, customers, time periods, etc., the set of tests cases should give good coverage of all combinations.

Many of the methods used for verifying the quality of computer software apply equally well to verifying large models. For example, a useful concept in testing of computer software is that of "coverage" by the test. A good test for software should exercise or cover all sections of code in a software program. Similarly, good test data for a model should exercise all features of the model. If a model allows up to N different products, say, one should have test data for the cases of: just 1 product (e.g., see (i) above); exactly N products; an intermediate number of products, and more than N products. Elaborating on (vi) above, we might want to have tests of all pairs of combination of: 1 and N productes, 1 and M suppliers, 1 and P periods, etc.

19.2.1 Appropriate Level of Detail and Validation

Validation should begin with understanding the real world to be modeled. A common problem is the people who are willing to speak most authoritatively about the process to be modeled are not always the most informed. A good rule of thumb is always to check your “facts” with a second source. Rothstein (1985) mentions, in a conversation with a vice president of a major airline, the vice president assured him the airline did not engage in overbooking of any kind. A short time later, in a discussion with operating personnel, he learned in fact the airline had a sophisticated overbooking system used everywhere in that airline.

If unimportant details are kept out of the model, the model should be not only easier to modify, but also easier to use.

Example. In developing a long-range ship scheduling model, a question of appropriate unit of time arose. Tides follow a roughly 13-hour cycle and this is an important consideration in the scheduling of ships into shallow ports. Deep draft ships can enter a shallow port only at high tide. Thus, in developing a multiperiod model for ship scheduling, it appeared 13 hours should be the length of a period.

However, we found ship travel times were sufficiently random, so scheduling to the day was satisfactory. Thus, to model a month of activity, 30 time periods, rather than about 60, was satisfactory. Halving the number of periods greatly simplified the computations. The moral perhaps is that, when it comes to incorporating detail into the model, a little bit of selective laziness may be a good thing.

If there is an art to modeling, this is it: identifying the simplifications or approximations that can be made without sacrificing the useful accuracy of the model. Simplifying approximations can be categorized roughly as follows:

1) *Functional* - Use a linear function to approximate a slightly nonlinear one.
2) *Aggregation*
   (2.1) *Temporal* aggregation - All events occurring during a given day (or week, month, etc.) are treated as having occurred at the end of the day.
   (2.2) *Cross-sectional* aggregation - All customers in a given mail code region are lumped together to be treated as one large customer. In a consumer products firm, all detergents are treated as a single product.
3) *Statistical* - Replace a random variable by its expectation. For example, even though future sales are a random variable, most planners use a single number forecast in planning.
4) *Decomposition* - If a system is overwhelming in its complexity, then decomposition is an approach that may be useful for simplifying the structure. Under this approach, a sequence of models are solved, each nailing down more detail of the complete solution.

Rogers, Plante, Wong, and Evans (1991) give an extensive survey of techniques for aggregation in optimization problems. The steps in using an approximate model can be summarized as follows:

1) Obtain detailed input data.
2) Derive the approximate (hopefully small) model.
3) Solve the approximate model.
4) Convert the solution of the approximate model back to the real world.

The difficult step is (4). The worst thing that can happen is it is impossible to convert the approximate solution back to a feasible real world solution.

19.2.2 When Your Model & the RW Disagree, Bet on the RW

As part of the validation process, you compare the output of your model with what happened in the real world (RW). When there is a discrepancy, there are two possibilities: (a) People in the RW are not behaving optimally and you have an opportunity to make some money by using your model; or (b) your model still has some flaws.

Black (1989) described the situation quite well while he was trying to validate an option pricing model:

> "*We estimated the volatility of the stock for each of a group of companies... We noticed that several warrants looked like very good buys. The best buy of all seemed to be National General... I and others jumped right in and bought a bunch... Then a company called American Financial announced a tender offer for National General... the tender offer had the effect of sharply reducing the value of the warrants... In other words, the market knew something that our formula didn't know... and that's why the warrants seemed so low in price.*"

19.3 Separation of Data and System Structure

There are two reasons for separating data from model structure:

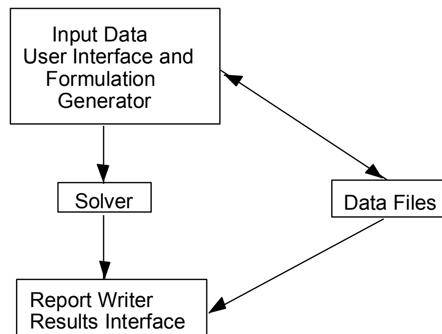
a) It allows us to adjust the model easily and quickly to changes in the real world,
b) The person responsible for making day-to-day changes in the data need not be familiar with the technical details of the model structure.

A flexible system is table driven. In powerful systems such as LINGO and What's*Best!*, factors such as interest rates can be input at a single place by a clerk, even though they appear numerous places in the model structure.

19.3.1 System Structure

In the typical case, a model will be used regularly (e.g., weekly in an operational environment). In this case, the model system can be thought of as having the structure shown in Figure 19.1:

Figure 19.1 System Structure



Notice there is a double-headed arrow between the data files and the formulation generator. This is because the generator may obtain parameters such as capacities from the data files. There is an arrow from the data files to the report writer because there are data, such as addresses of customers, that are needed for the output reports but are not needed in the formulation. The success of spreadsheet programs, such as Lotus 1-2-3, is due in part to the fact they incorporate all the above components in a relatively seamless fashion.

19.4 Marketing the Model

It is important to keep in mind: Who will be the users/clients? Frequently, there are two types of clients in a single organization:

1) The Model champion (e.g., a CEO),
2) Actual user (e.g., a foreman working 12 hours/day and whose major concern is getting the work out and meeting deadlines).

Client (1) will commit to model development based on expected profit improvement. Client (2) will actually use the model if it simplifies his/her life. He may get fired if he misses a production deadline. There is a modest probability of a raise if he improves profitability and takes the trouble to document it. Thus, for client (2), the input and output user interfaces are very important.

19.4.1 Reports

A model has an impact largely via the reports it produces. If a standard report already exists in the organization, try to use it. The model simply puts better numbers in it.

Example. An LP-based scheduling system was developed for shoe factories. It was a success in the first factory where it was tried. Production improved by about 15%. The system never "got off the ground" in a second factory. The reason for the difference in success was apparently as follows. The first factory had an existing scheduling report or work order. This report was retained. The results of the LP scheduling model simply put better numbers in it. The second factory had been using an informal scheduling system. The combination of installing both a new reporting system and a new scheduling system was too big a barrier to change.

19.4.1.1 Designing Reports

The proper attitude in designing reports is to ask: How will the results be used?

In operations settings, there frequently are three types of reports implied by the results of a model run:

a) *Raw material acquisition recommendations*. For example, in extreme cases, the model might generate purchase orders directly.
b) *Production orders*. For example, how are the raw materials to be processed into finished goods?
c) *Finished goods summaries*. If the production process is complicated, (e.g., several different alternative processes are used to achieve the total production of a specific product), then it may not be clear from (a) and (b) how much of a particular finished good was produced.

Example. *Reports in a Blending Facility*. In a facility that blends raw materials into finished goods, reports of the following type might be appropriate:

(a)

Raw Material Purchases			
Raw Material	Total Required	Beginning Inventory	Required Purchases

(b)

Production			
Batch 1 Product:		Batch 1 Product:	
Inputs: Raw Material	Amount	Inputs: Raw Material	Amount

(c)

Finished Goods Summary				
Product	Beginning Inventory	Goods Produced	Total Required	Surplus

19.4.1.2 Dimensional View of Reports

A more mechanical view of report generation is to take a dimensional view of a system (i.e., a problem and its solution have a number of dimensions). Each report is a sort and summary by dimensions.

Example: *Multiperiod Shipping*

Dimensions: Origins, destinations, time periods. The major decision variables might be of the form X_{ijt}, where X_{ijt} is the number of tons to be shipped from supplier *i*, to customer *j* in time period *t*. The types of reports might be:

Supplier's Report: Sorted by origin, time, destination (or summed over destination).

Shipping Manager's Report: Sorted by time, origin, destination.

Customer's Report: Sorted by destination, time, origin (or perhaps summed over origin).

Most spreadsheets and database systems have multi-level sorting capability.

19.4.1.3 Report Details/Passing the Snicker Test

Results should be phrased in terms the user finds easy to use.

For example, reporting a steel bar should be cut to a length of 58.36 inches may cause snickers in some places because "everybody knows" this commodity (like U.S. stock prices) is measured in multiples of 1/8 inches, or dollars, as the case may be. So, it would be better to round the result to 58.375 inches or, even better, report it as 58 and 3/8 inches.

Other examples: Dates should be reported not only in day of the month (taking into account leap years), but also day of the week. Different parts of the world use different formats for displaying dates (e.g., 14 March 1991 or 3/14/1991). Use a format appropriate for the location where used.

Example: *Vehicle Routing/Passing the Snicker Test*

Customers are grouped into trips, so the same vehicle serves customers on the same trip. The actual model decomposed the problem into two phases:

(1) Allocate to trips	←	Big savings here.
(2) Sequence each trip	←	Users notice this the most.

If your system does an excellent job of allocating customers to trips (where the big savings exist), but does not always get the optimal sequence of customers within a trip, users may notice the latter weakness. Even though there may be no big savings possible by improving the sequence, users may have less faith in the system because of this small weakness.

19.4.1.4 Models Should Always Have Feasible Solutions

In a large model where the input data are prepared by many people, there may be no assurance the data are perfectly consistent. For example, production capacity as estimated by the production department may be insufficient to satisfy sales forecasts as estimated by the marketing department. If the model has a constraint that requires production to equal forecasted sales, then there may be no feasible solution. The terse message "No feasible solution" is not very helpful.

A better approach is to have in the model a s*uperworker* or *superfacility* that can make any product at infinite speed, but at a rather high cost. There will always be a feasible solution although some parts of the solution may look somewhat funny.

Another device is to allow *demand to be backlogged* at a high cost.

In each case, the solution will give sensible indications of where one should install extra capacity or cut back on projected sales, etc.

A model may be fundamentally good, but incomplete in certain minor details. As a result, some of its recommendations may be slightly, but blatantly, incorrect.

For example, in reality almost every activity has a finite upper bound.

Similarly, there may be obvious *bounds on the dual* prices of certain resources. For example, if land is a scarce resource in the real world, then its dual price should never be zero. You should *include sellout or buy activities* corresponding to such things as renting out excess land to put lower and upper bounds on dual prices.

19.4.1.5 "Signing Off" on System Structure

If a prospective model (a) is likely to be complicated and (b) the group that will use the model is distinct from the group that will design the model, then it will be worthwhile to have beforehand a written document describing exactly what the model does. Effectively, the "User's Manual" is written before the system is implemented. The prospective users should "sign off" on the design (e.g., by providing a letter that says "Yes, this is what we want *and we will accept* if it provides this").

This document might include the following:

a) Form in which input will be provided.
b) Form in which output will be provided.
c) Test data sets that must be successfully processed. The model will be accepted if and only if these are satisfied.

19.4.2 Report Generation in LINGO

The default report format in LINGO is the three column: Variable, Value, Reduced Cost report format. You can generate somewhat arbitrary customized reports by using several functions available in a DATA section in LINGO. The functions are:

@TEXT() = output function. Allows you to specify a line to be output. If @TEXT( ) has no argument, then the line is output to the terminal display, else it is output to the filename listed in the argument as in @TEXT(myfile.txt) =.

@WRITEFOR output looping function. Analogous to the @FOR function in a model, it specifies looping over sets when generating output.

@WRITE(). Used for outputting a single line.

@NEWLINE(n) inserts n newlines or carriage returns in the output.

@FORMAT(field, value) specifies a field format, e.g., number of characters, and what value to insert in the field.

The following model based on the Sudoku puzzle illustrates how to use the above functions.

```
! The sudoku puzzle in LINGO.  Fill out a 9x9 grid with the digits
 1,2,...9, so that each digit appears once in
   a) each column,
   b) each row,
   c) each of the nine 3x3 subsquares,
   d) the main diagonal,
   e) in the reflected diagonal;
! Some versions of the puzzle do not require (d) and (e)
! Keywords: sudoku, Puzzles;
```

```
SETS:
 SIDE;
 SS( SIDE, SIDE): X;
 SSS( SIDE,SIDE,SIDE): Y;
ENDSETS
 DATA:
  SIDE = 1..9;
! Set diag = 1 if you want the diagonal constraints to be
  satisfied, else 0 if not required;
   diag = 0;
 ENDDATA
! Variables:
   X(i,j) = value in row i, col j of matrix,
   Y(i,j,k) = 1 if X(i,j) = k;

! Any pre-specified entries inserted here;
   X(1,1) = 5;
   X(2,6) = 8;
   X(3,4) = 5;
   X(3,9) = 1;
   X(4,2) = 1;
   X(4,7) = 6;
   X(4,8) = 3;
   X(9,1) = 9;
   X(9,2) = 8;
   X(9,5) = 6;

 ! Link X and Y;
   @FOR( SS(i,j):
     X(i,j) = @SUM(SIDE(k): k*y(i,j,k));
 ! Must choose something for cell i,j;
    @SUM(SIDE(k): y(i,j,k)) = 1;
 ! Make the Y's binary;
    @FOR(SIDE(k): @bin(y(i,j,k)));
       );

 ! Force each number k to appear once in each column j;
   @FOR( SIDE(j):
     @FOR( SIDE(k):
       @SUM( SIDE(i): Y(i,j,k)) = 1;
       );  );
 ! Force each number k to appear once in each row i;
   @FOR( SIDE(i):
     @FOR( SIDE(k):
       @SUM( SIDE(j): Y(i,j,k)) = 1;
        ); );
 ! Force each number k to appear once in each 3x3 subsquare;
   @FOR( SIDE(k):
     ! Upper left;
     @SUM( SS(i,j) | i #le#3 #and# j #le# 3: y(i,j,k)) = 1;
     ! Upper middle;
     @SUM( SS(i,j) | i #le#3 #and# j #gt# 3 #and# j#le# 6: y(i,j,k)) = 1;
     ! Upper right;
```

```
     @SUM( SS(i,j) | i #le#3 #and# j #gt# 6: y(i,j,k)) = 1;
     ! Middle left;
     @SUM( SS(i,j) | i #gt#3 #and# i #le#6 #and# j #le# 3: y(i,j,k)) = 1;
     ! Middle middle;
     @SUM( SS(i,j) | i #gt#3 #and# i #le#6 #and# j #gt# 3 #and# j #le# 6:
y(i,j,k)) = 1;
     ! Middle right;
     @SUM( SS(i,j) | i #gt#3 #and# i #le#6 #and# j #gt# 6 #and# j #le# 9:
y(i,j,k)) = 1;
     ! Lower left;
     @SUM( SS(i,j) | i #gt#6 #and# i #le#9 #and# j #gt# 0 #and# j #le# 3:
y(i,j,k)) = 1;
     ! Lower middle;
     @SUM( SS(i,j) | i #gt#6 #and# i #le#9 #and# j #gt# 3 #and# j #le# 6:
y(i,j,k)) = 1;
     ! Lower right;
     @SUM( SS(i,j) | i #gt#6 #and# i #le#9 #and# j #gt# 6 #and# j #le# 9:
y(i,j,k)) = 1;

   ! Force each number k to appear once in the main diagonal;
     @SUM( SS(i,j) | i #eq# j: diag*y(i,j,k)) = diag;
   ! Force each number k to appear once in the reflected diagonal;
     @SUM( SS(i,j) | i + j #eq# 10: diag*y(i,j,k)) = diag;
       );

  DATA:
 ! Write the solution in matrix form;
   @TEXT() =
    @WRITE( @NEWLINE( 1), 25*' ', 'Sudoku Puzzle Solution',
       @NEWLINE( 1));
   @TEXT() =
      @WRITEFOR( SIDE( i):
         @WRITEFOR( SIDE( j):
            @FORMAT( '#8.0g', x( i, j) )
                   ) , @NEWLINE( 1)
                );
   @TEXT() = ' ';
  ENDDATA
```

19.5 Reducing Model Size

Practical LP models tend to be large. Thus, it makes sense to talk about the management of these models. Some of the important issues are:

1. Choosing an appropriate formulation. Frequently, there are two conflicting considerations: (a) the model should be large enough to capture all important details of reality, and (b) the model should be solvable in reasonable time.
2. What input data are needed? How is it collected?
3. How do we create an explicit model from the current data? This process has traditionally been called matrix generation.
4. How is the model solved? Is it solvable in reasonable time? In reality, some optimization program must be selected.

In this section, we discuss issues (1) and (3). The selection of an appropriate formulation also has implications for how easily a model is solved (issue 4).

We begin our discussion with how to choose a formulation that is small and thus more easily solved (usually).

The computational difficulty of an LP is closely related to three features of the LP: the number of rows, the number of columns, and the number of nonzeroes in the constraint matrix. For linear programs, the computation time tends to increase with the square of the number of nonzeroes. Thus, there is some motivation to (re)formulate LP models, so they are small in the above-mentioned three dimensions.

A number of commercial LP solvers have built-in commands, with names like REDUCE, that will mechanically do simple kinds of algebraic substitutions and eliminations necessary for reduction. Brearley, Mitra, and Williams (1975) give a thorough description of these reductions.

19.5.1 Reduction by Aggregation

We say we aggregate a set of variables if we replace a set of variables by a single variable. We aggregate a set of constraints if we replace a set of constraints by a single constraint. If we do aggregation, we must resolve several issues:

1. After solving the LP, there must be a postprocessing/disaggregation phase to deduce the disaggregate values from the aggregate values.
2. If row aggregation was performed, the solution to the aggregate problem may not be feasible to the true disaggregate problem.
3. If variable aggregation was performed, the solution to the aggregate problem may not be optimal to the true disaggregate problem.

To illustrate (2), consider the LP:

$$\begin{array}{ll} \text{Maximize} & 2x + y \\ \text{subject to} & x \leq 1 \\ & y \leq 1 \\ & x, y \geq 0 \end{array}$$

The optimal solution is $x = y = 1$; with objective value equal to 3. We could aggregate the rows to get:

$$\begin{array}{ll} \text{Maximize} & 2x + y \\ \text{subject to} & x + y \leq 2 \\ & x, y \geq 0 \end{array}$$

The optimal solution to this aggregate problem is $x = 2$, $y = 0$, with objective value equal to 4. However, this solution is not feasible to the original problem.

To illustrate (3), consider the LP:

$$\begin{array}{ll} \text{Minimize} & x_1 + x_2 \\ \text{subject to} & x_1 \geq 2 \\ & x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{array}$$

The optimal solution is $x_1 = 2$, $x_2 = 1$, with objective value equal to 3. We could aggregate variables to get the LP:

$$\begin{array}{ll} \text{Minimize} & 2x \\ \text{subject to} & x \geq 2 \\ & x \geq 1 \\ & x \geq 0 \end{array}$$

The optimal solution to the aggregate problem is $x = 2$, with objective value equal to 4. This solution is, however, not optimal for the original, disaggregate LP.

19.5.1.1 Example: The Room Scheduling Problem

We will illustrate both variable and constraint aggregation with a problem that confronts any large hotel that has extensive conference facilities for business meetings. The hotel has r conference rooms available of various sizes. Over the next t time periods (e.g., days), the hotel must schedule g groups of people into these rooms. Each group has a hard requirement for a room of at least a certain size. Each group may also have a preference of certain time periods over others. Each group requires a room for exactly one time period. The obvious formulation is:

V_{gtr} = value of assigning group g to time period t in room r. This value is provided by group g, perhaps as a ranking. The decision variables are:

X_{gtr} = 1 if group g is assigned to room r in time period t. This variable is defined for each group g, each time period t, and each room r that is big enough to accommodate group g.
= 0 otherwise.

The constraints are:

$$\sum_t \sum_r x_{gtr} = 1 \quad \text{for each group } g$$

$$\sum_g x_{gtr} \leq 1 \quad \text{for each room } r\text{, time period } t$$

$$x_{gtr} = 0 \text{ or } 1 \quad \text{for all } g\text{, } t\text{, and } r$$

The objective is:

$$\text{Maximize} \quad \sum_g \sum_t \sum_r V_{gtr}\, x_{gtr}$$

The number of constraints in this problem is $g + r \times t$. The number of variables is approximately $g \times t \times r/2$. The 1/2 is based on the assumption that, for a typical group, about half of the rooms will be big enough.

A typical problem instance might have $g = 250$, $t = 10$, and $r = 30$. Such a problem would have 550 constraints and about 37,500 variables. A problem of that size is nontrivial to solve, so we might wish to work with a smaller formulation.

Aggregation of variables can be used validly if a group is indifferent between rooms b and c, as long as both rooms b and c are large enough to accommodate the group. In terms of our notation, $V_{gtb} = V_{gtc}$ for every g and t if both rooms b and c are large enough for g. More generally, two variables can be aggregated if, in each row of the LP, they have the same coefficients. Two constraints in an LP can be validly aggregated if, in each variable, they have the same coefficients. We will do constraint aggregation by aggregating together all rooms of the same size. This aggregation process is

representative of a fundamental modeling principle: when it comes to solving the model, *do not distinguish things that do not need distinguishing*.

The aggregate formulation can now be defined:

K	=	number of distinct room sizes
N_k	=	number of rooms of size k or larger
S_k	=	the set of groups, which require a room of size k or larger
V_{gt}	=	value of assigning group g to time period t
x_{gt}	=	1 if group g is assigned to a room in time period t
	=	0 otherwise

The constraints are:

$$\sum_{i} x_{gt} = 1 \qquad \text{for each group } g$$

$$\sum_{g \in S_k} x_{gt} \leq N_k \qquad \text{for each room size } k.$$

The objective is:

$$\text{Maximize } \sum_{g} \sum_{t} V_{gt} X_{gt}$$

This formulation will have $g + k \times t$ constraints and $g \times t$ decision variables. For the case $g = 250$, $t = 10$, and $r = 30$, we might have $k = 4$. Thus, the aggregate formulation would have 290 constraints and 2500 variables, compared with 550 constraints and 37,500 variables for the disaggregate formulation.

The post processing required to extract a disaggregate solution from an aggregate solution to our room scheduling problem is straightforward. For each time period, the groups assigned to that time period are ranked from largest to smallest. The largest group is assigned to the largest room, the second largest group to the second largest room, etc. Such an assignment will always be feasible as well as optimal to the original problem.

19.5.1.2 Example 2: Reducing Rows by Adding Additional Variables

If two parties, A and B, to a financial agreement, want the agreement to be treated as a lease for tax purposes, the payment schedule typically must satisfy certain conditions specified by the taxing agency. Suppose P_i is the payment A is scheduled to make to B in month i of a seven-year agreement. Parties A and B want to choose at the outset a set of P_j's that satisfy a tax regulation that no payment in any given month can be less than 2/3 of the payment in any earlier month. If there are T periods, the most obvious way of writing these constraints is:

For $i = 2, T$:
For $j = 1, i - 1$:
$\quad P_i \geq 0.66666\, P_j$

This would require $T(T-1)/2$ constraints. A less obvious approach would be to define PM_i as the largest payment occurring any period before i. The requirement could be enforced with:

$PM_1 = 0$
For $i = 2$ to T:
$P_i \geq 0.66666\ PM_i$
$PM_i \geq PM_{i-1}$
$PM_i \geq P_{i-1}$

This would require $3T$ constraints rather than $T(T-1)/2$. For $T = 84$, the difference is between 3486 constraints and 252.

19.5.2 Reducing the Number of Nonzeroes

If a certain linear expression is used more than once in a model, you may be able to reduce the number of nonzeroes by substituting it out. For example, consider the two-sided constraints frequently encountered in metal blending models:

$$L_i \leq \frac{\sum_j q_{ij} X_j}{\sum_j X_j} \leq U_i \qquad \text{(for each quality characteristic } i\text{).}$$

In these situations, L_k and U_k are lower and upper limits on the ith quality requirement, and q_{ij} is the quality of ingredient j with respect to the ith quality. The "obvious" way of writing this constraint in linear form is:

$$\sum_j (q_{ij} - L_i)\ X_j \geq 0,$$
$$\sum_j (q_{ij} - U_k)\ X_j \leq 0.$$

By introducing a batch size variable B and a slack variable s_i, this can be rewritten:

$$B - \sum_j X_j = 0$$
$$\sum_j q_{ij} X_j + s_i = U_i \times B$$
$$s_i \leq (U_k - L_i) \times B$$

If there are m qualities and n ingredients, the original formulation had $2 \times m \times n$ nonzeroes. The modified formulation has $n + 1 + m \times (n + 2) + m \times 2 = n + 1 + m \times (n + 4)$ nonzeroes. For large n, the modified formulation has approximately 50% fewer nonzeroes.

19.5.3 Reducing the Number of Nonzeroes in Covering Problems

A common feature in some covering and multiperiod financial planning models is each column will have the same coefficient (e.g., + 1) in a large number of rows. A simple transformation may substantially reduce the number of nonzeroes in the model. Suppose row i is written:

$$\sum_{j=1}^{n} a_{ij} X_j = a_{i0}$$

Now, suppose we subtract row $i - 1$ from row i, so row i becomes:

$$\sum_{j=1}^{n} (a_{ij} - a_{i-1,j}) X_j = a_{i0} - a_{i-1,0}$$

If $a_{ij} = a_{i-1,j} \neq 0$ for most j, then the number of nonzeroes in row i is substantially reduced.

Example

Suppose we must staff a facility around the clock with people who work eight-hour shifts. A shift can start at the beginning of any hour of the day. If r_i is the number of people required to be on duty from hour i to hour $i + 1$, X_i is the number of people starting a shift at the beginning of hour i, and s_i is the surplus variable for hour i, then the constraints are:

$$\begin{aligned}
&X_1 + X_{18} + X_{19} + X_{20} + X_{21} + X_{22} + X_{23} + X_{24} - s_1 = r_1 \\
&X_1 + X_2 + X_{19} + X_{20} + X_{21} + X_{22} + X_{23} + X_{24} - s_2 = r_2 \\
&X_1 + X_2 + X_3 + X_{20} + X_{21} + X_{22} + X_{23} + X_{24} - s_3 = r_3 \\
&\quad\vdots
\end{aligned}$$

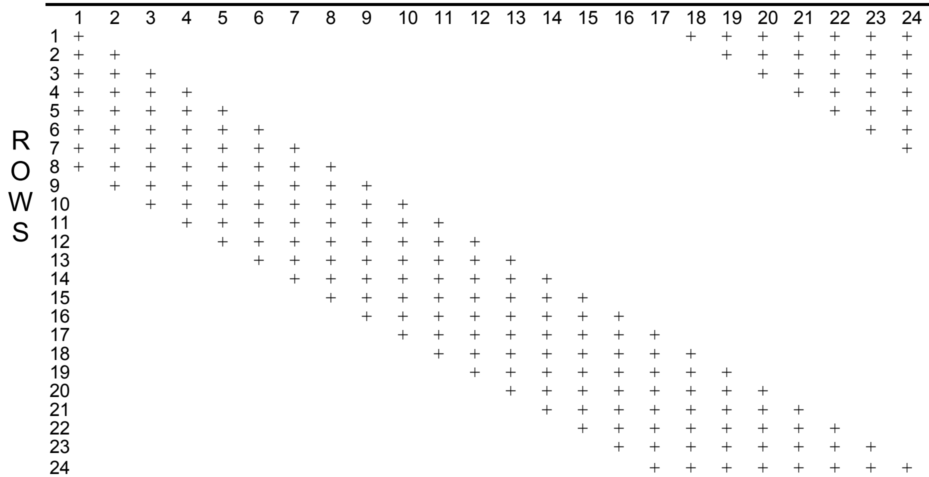
Suppose we subtract row 23 from row 24, row 22 from row 23, etc. The above constraints will be transformed to:

$$\begin{aligned}
&X_1 + X_{18} + X_{19} + X_{20} + X_{21} + X_{22} + X_{23} + X_{24} - s_1 = r_1 \\
&X_2 - X_{18} + s_1 - s_2 = r_2 - r_1 \\
&X_1 - X_{19} + s_2 - s_3 = r_3 - r_2 \\
&\quad\vdots
\end{aligned}$$

Thus, a typical constraint will have four nonzeroes rather than nine.

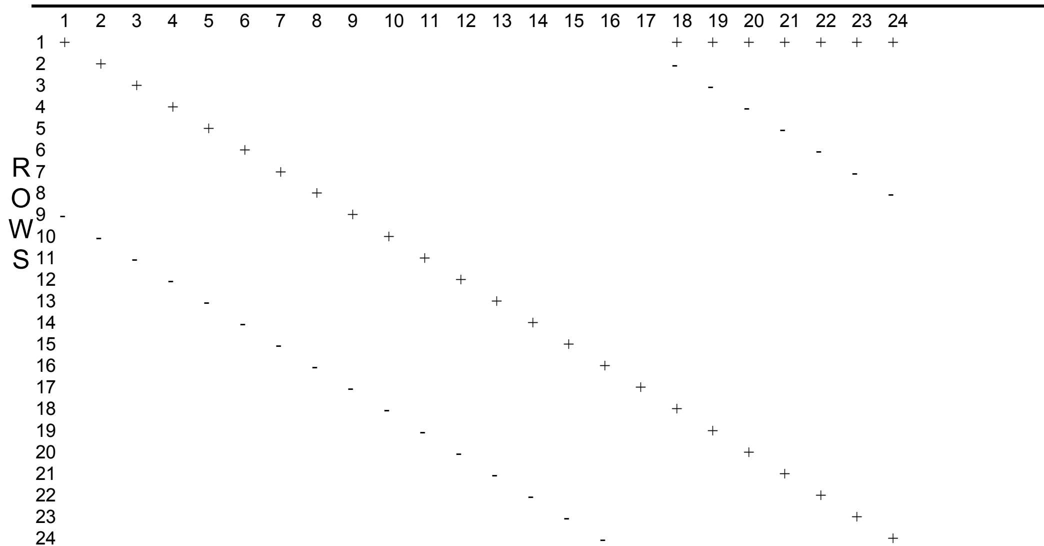
The pattern of nonzeroes for the X variables in the original formulation is shown in Figure 19.1. The pattern of the nonzeroes for the X variables in the transformed formulation is shown in Figure 19.2. The total constraint nonzeroes for X and s variables in the original formulation is 216. The analogous count for the transformed formulation is 101, a very attractive reduction.

Figure 19.2
Nonzero Pattern for X Variables in Original Formulation.
X Variables



ROWS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	+																	+	+	+	+	+	+	+
2	+	+																	+	+	+	+	+	+
3	+	+	+																	+	+	+	+	+
4	+	+	+	+																	+	+	+	+
5	+	+	+	+	+																	+	+	+
6	+	+	+	+	+	+																	+	+
7	+	+	+	+	+	+	+																	+
8	+	+	+	+	+	+	+	+																
9		+	+	+	+	+	+	+	+															
10			+	+	+	+	+	+	+	+														
11				+	+	+	+	+	+	+	+													
12					+	+	+	+	+	+	+	+												
13						+	+	+	+	+	+	+	+											
14							+	+	+	+	+	+	+	+										
15								+	+	+	+	+	+	+	+									
16									+	+	+	+	+	+	+	+								
17										+	+	+	+	+	+	+	+							
18											+	+	+	+	+	+	+	+						
19												+	+	+	+	+	+	+	+					
20													+	+	+	+	+	+	+	+				
21														+	+	+	+	+	+	+	+			
22															+	+	+	+	+	+	+	+		
23																+	+	+	+	+	+	+	+	
24																	+	+	+	+	+	+	+	+

Figure 19.3
Nonzero Pattern for X Variables in Transformed Formulation
X Variables



ROWS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	+																	+	+	+	+	+	+	+
2		+																-						
3			+																-					
4				+																-				
5					+																-			
6						+																-		
7							+																-	
8								+																-
9	-								+															
10		-								+														
11			-								+													
12				-								+												
13					-								+											
14						-								+										
15							-								+									
16								-								+								
17									-								+							
18										-								+						
19											-								+					
20												-								+				
21													-								+			
22														-								+		
23															-								+	
24																-								+

19.6 On-the-Fly Column Generation

There are a number of generic LP models that have a modest number of rows (e.g., a hundred or so), but a large number of columns (e.g., a million or so). This is frequently the case in cutting stock

problems. This could also be the case in staffing problems, where there might be many thousands of different work patterns people could work. Explicitly generating all these columns is not a task taken lightly. An alternative approach is motivated by the observation that, at an optimum, there will be no more positive columns than there are rows.

The following iterative process describes the basic idea:

1. Generate and solve an initial LP that has all the rows of the full model defined, but only a small number (perhaps even zero) of the columns explicitly specified.
2. Given the dual prices of the current solution, generate one or more columns that price out attractively. That is, if a_{0j} is the cost of column j, a_{ij} is its usage of resource i (i.e., its coefficient in rows i for i = 1, ..., m), and p_i is the dual price of row i, then generate or find a new column a such that:
   $$a_{0j} + p_1 a_{1j} + p_2 a_{2j} + \ldots + p_m a_{mj} < 0.$$
   If no such column exists, then stop. The solution is optimal.
3. Solve the LP with the new column(s) from (2) added.
4. Go to (2).

The crucial step is (2). To use column generation for a specific problem, you must be able to solve the column generation subproblem in (2). In mathematical programming form, the subproblem in (2) is:

Given $\{p_j\}$, solve

Min $a_{0j} + p_1 a_{1j} + p_2 a_{2j} + \ldots + p_m a_{mj}$

subject to:

The a_{ij} satisfy the conditions defining a valid column.

19.6.1 Example of Column Generation Applied to a Cutting Stock Problem

A common problem encountered in flat goods industries, such as paper, textiles, and steel, is the cutting of large pieces of raw material into smaller pieces needed for producing a finished product. Suppose raw material comes in 72" widths and it must be cut up into eight different finished good widths described by the following table:

Product	Width in Inches	Linear feet Required
1	60	500
2	56	400
3	42	300
4	38	450
5	34	350
6	24	100
7	15	800
8	10	1000

We start the process somewhat arbitrarily by defining the eight pure cutting patterns. A pure pattern produces only one type of finished good width. Let P_i = number of feet of raw material to cut according to the pattern i. We want to minimize the total number of feet cut. The LP with these patterns is:

```
MIN =P001 + P002 + P003 + P004 + P005 + P006 + P007 + P008;
 [W60]      P001 >=  500;  !       (60 inch width);
 [W56]      P002 >=  400;  !       (56 inch width);
 [W42]      P003 >=  300;  !       (42 inch width);
 [W38]      P004 >=  450;  !       (38 inch width);
 [W34] 2 * P005 >=  350;  !       (34 inch width);
 [W24] 3 * P006 >=  100;  !       (24 inch width);
 [W15] 4 * P007 >=  800;  !       (15 inch width);
 [W10] 7 * P008 >=  1000; !       (10 inch width);
END
```

The solution is:

```
Optimal solution found at step:         0
Objective value:                 2201.190
Variable            Value        Reduced Cost
    P001         500.0000           0.0000000
    P002         400.0000           0.0000000
    P003         300.0000           0.0000000
    P004         450.0000           0.0000000
    P005         175.0000           0.0000000
    P006         33.33333           0.0000000
    P007         200.0000           0.0000000
    P008         142.8571           0.0000000
     Row    Slack or Surplus      Dual Price
       1         2201.190            1.000000
     W60        0.0000000           -1.000000
     W56        0.0000000           -1.000000
     W42        0.0000000           -1.000000
     W38        0.0000000           -1.000000
     W34        0.0000000          -0.5000000
     W24        0.0000000          -0.3333333
     W15        0.0000000          -0.2500000
     W10        0.0000000          -0.1428571
```

The dual prices provide information about which finished goods are currently expensive to produce. A new pattern to add to the problem can be found by solving the problem:

Minimize $1 - y_1 - y_2 - y_3 - y_4 - .5y_5 - 0.333333y_6 - 0.25y_7 - 0.142857y_8$

subject to

$$60y_1 + 56y_2 + 42y_3 + 38y_4 + 34y_5 + 24y_6 + 15y_7 + 10y_8 \le 72$$

$$y_1 = 0, 1, 2, \ldots \quad \text{for} \quad i = 1, \ldots 8.$$

Note the objective can be rewritten as:

Maximize $y_1 + y_2 + y_3 + y_4 + .5y_5 + 0.333333y_6 + 0.25y_7 + 0.142857y_8$

This is a knapsack problem. Although knapsack problems are theoretically difficult to solve, there are algorithms that are quite efficient on typical practical knapsack problems. An optimal solution to this knapsack problem is y_4 = 1, y_7 = 2 (i.e., a pattern that cuts one 38” width and two 15” widths). When this column, P009, is added to the LP, we get the formulation (in Picture form):

```
   P P P P P P P P P
   0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0
   1 2 3 4 5 6 7 8 9
1: 1 1 1 1 1 1 1 1 1 MIN
2: 1     '     '     > C
3: ' 1'  '  '  '  '  > C
4:     1 '     '     > C
5:       1     '   1 > C
6: '  '  ' 2'  '  '  > C
7:       '   3 '     > B
8:       '     4   2 > C
9: '  '  '  '  ' 7'  > C
```

The solution is:

```
Optimal solution found at step:          3
Objective value:                  2001.190
Variable           Value        Reduced Cost
    P001        500.0000           0.0000000
    P002        400.0000           0.0000000
    P003        300.0000           0.0000000
    P004        50.00000           0.0000000
    P005        175.0000           0.0000000
    P006        33.33333           0.0000000
    P007       0.0000000            1.000000
    P008        142.8571           0.0000000
    P009        400.0000           0.0000000
     Row    Slack or Surplus      Dual Price
       1        2001.190            1.000000
     W60       0.0000000           -1.000000
     W56       0.0000000           -1.000000
     W42       0.0000000           -1.000000
     W38       0.0000000           -1.000000
     W34       0.0000000          -0.5000000
     W24       0.0000000          -0.3333333
     W15       0.0000000           0.0000000
     W10       0.0000000          -0.1428571
```

The column generation subproblem is:

Minimize $y_1 + y_2 + y_3 + y_4 + .5 y_5 + 0.333333 y_6 + 0.142857 y_8$

subject to

$$60 y_1 + 56 y_2 + 42 y_3 + 38 y_4 + 34 y_5 + 24 y_6 + 15 y_7 + 10 y_8 \leq 72$$

$y_1 = 0, 1, 2,...$ for $i = 1,... 8$.

An optimal solution to this knapsack problem is y_4 = 1, y_5 = 1 (i.e., a pattern that cuts one 38” width and one 34” width).

We continue generating and adding patterns for a total of eight iterations. At this point, the LP formulation is:

```
   P P P P P P P P P P P P P P P
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
   1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 MIN
2: 1     '     '     '     '   1 > C
3: ' 1'  '  '  '  '  '  '1 '  '  > C
4:     1 '     '     ' 1   ' 1   > C
5:       1     '   1 1     1     > C
6: '  '  ' 2'  '  '  1  '  '  '  > C
7:       '   3 '     '     1     > B
8:       '     4   2 ' 2 1 '     > C
9: '  '  '  '  ' 7'  '  '  1 3'1 > C
```

The solution is:

```
Optimal solution found at step:        10
Objective value:                 1664.286

Variable           Value        Reduced Cost
    P001       0.0000000           0.1428571
    P002       0.0000000           0.2142857
    P003       0.0000000           0.4285714
    P004       0.0000000           0.4285714
    P005       0.0000000           0.1428571
    P006       0.0000000           0.1428571
    P007       0.0000000           0.1428571
    P008        14.28571           0.0000000
    P009       0.0000000           0.0000000
    P010        350.0000           0.0000000
    P011        200.0000           0.0000000
    P012        400.0000           0.0000000
    P013        100.0000           0.0000000
    P014        100.0000           0.0000000
    P015        500.0000           0.0000000

     Row    Slack or Surplus      Dual Price
       1        1664.286            1.000000
     W60       0.0000000          -0.8571429
     W56       0.0000000          -0.7857143
     W42       0.0000000          -0.5714286
     W38       0.0000000          -0.5714286
     W34       0.0000000          -0.4285714
     W24       0.0000000          -0.2857143
     W15       0.0000000          -0.2142857
     W10       0.0000000          -0.1428571
```

The relevant knapsack problem is:

Maximize
$0.857143y_1 + 0.785714y_2 + 0.571429y_3 + 0.523809y_4$
$+ 0.476191y_5 + 0.333333y_6 + 0.214285y_7 + 0.142857y_8$
subject to
$60y_1 + 56y_2 + 42y_3 + 38y_4 + 34y_5 + 24y_6 + 15y_7 + 10y_8 \le 72$
$y_1 = 0, 1, 2,...$ for $i = 1,... 8.$

The optimal solution to the knapsack problem has an objective function value less-than-or-equal-to one. Because each column, when added to the LP, has a "cost" of one in the LP objective, when the proposed column is priced out with the current dual prices, it is unattractive to enter. Thus, the previous LP solution specifies the optimal amount to run of all possible patterns. There are in fact 29 different efficient patterns possible, where efficient means the edge waste is less than 10". Thus, the column generation approach allowed us to avoid generating the majority of the patterns.

If an integer solution is required, then a simple rounding up heuristic tends to work moderately well. In our example, we know the optimal integer solution costs at least 1665. By rounding P008 up to 15, we obtain a solution with cost 1665.

The following is a LINGO program that automates this column generation process.

```
MODEL:          ! Loopcut72.lng;
! Uses Lingo's programming capability to do
  on-the-fly column generation for a
  cutting-stock problem;
! Keywords: Column Generation, Knapsack Model, Cutting Stock;
SETS:
   PATTERN: COST, X;
  FG: WIDTH, DEM, PRICE, Y, YIELD;
  FXP( FG, PATTERN): NBR;
ENDSETS

DATA:
  PATTERN = 1..20; ! Allow up to 20 patterns;
  RMWIDTH = 72;    ! Raw material width;
  FG =  F60  F56  F42  F38  F34  F24  F15  F10;!Finished goods...;
  WIDTH= 60   56   42   38   34   24   15   10;!their widths...;
  DEM = 500  400  300  450  350  100  800 1000;!and demands;
  BIGM = 999;
ENDDATA

SUBMODEL MASTER_PROB:
  [MSTROBJ] MIN= @SUM( PATTERN( J)| J #LE# NPATS: COST( J)*X( J));
  @FOR( FG( I):
   [R_DEM]  @SUM( PATTERN( J)| J #LE# NPATS:
                        NBR( I, J) * X( J)) >= DEM( I);
       );
ENDSUBMODEL

SUBMODEL INTEGER_REQ:
  @FOR( PATTERN: @GIN( X));
ENDSUBMODEL
```

```
SUBMODEL PATTERN_GEN:
  [SUBOBJ] MAX = @SUM( FG( I): PRICE( I)* Y( I));
   @SUM( FG( I): WIDTH( I)*Y( I)) <= RMWIDTH;
  @FOR( FG( I): @GIN(Y( I)));
ENDSUBMODEL

CALC:
  ! Set some parameters;
  @SET( 'DEFAULT'); ! Set all parameters to defaults;
  @SET( 'TERSEO', 2); ! Turn off default output;

 ! Max number of patterns we'll allow;
  MXPATS = @SIZE( PATTERN);
 ! Make first pattern an expensive super pattern;
  COST( 1) = BIGM;
  @FOR( FG( I): NBR( I, 1) = 1);

 ! Loop as long as the reduced cost is
   attractive and there is space;
  NPATS = 1;
  RC = -1;   ! Clearly attractive initially;
  @WHILE( RC #LT# 0 #AND# NPATS #LT# MXPATS:
   ! Solve for best patterns to run among ones
     generated so far;
  @SOLVE( MASTER_PROB);
  ! Copy dual prices to PATTERN_GEN submodel;
  @FOR( FG( I): PRICE( I) = -@DUAL( R_DEM( I)));
  ! Generate the current most attractive pattern;
  @SOLVE( PATTERN_GEN);
  ! Marginal value of current best pattern;
  RC = 1 - SUBOBJ;
  ! Add the pattern to the Master if it is attractive;
  @IFC( RC #LT# 0:
     NPATS = NPATS + 1;
     @FOR( FG( I): NBR( I, NPATS) = Y( I));
     COST( NPATS) = 1;
      );
  );

 ! Finally solve Master as an IP;
  @SOLVE( MASTER_PROB, INTEGER_REQ);

! This following calc section displays the
  solution in a tabular format;
! Compute yield of each FG;
  @FOR( FG( F): YIELD( F) =
   @SUM( PATTERN( J)| J #LE# NPATS:
      NBR( F, J) * X(J))
      );

! Compute some stats;
  TOTAL_FT_USED = @SUM( PATTERN(j)| j #LE# NPATS: X(j)) * RMWIDTH;
  TOTAL_FT_YIELD = @SUM( FG: YIELD * WIDTH);
```

```
   PERC_WASTE =  100 * ( 1 - ( TOTAL_FT_YIELD / TOTAL_FT_USED)) ;
  ! Display the table of patterns and their usage;
   FW = 6;
   @WRITE( @NEWLINE( 1));
   @WRITE( ' Total raws used:    ', @SUM(PATTERN(j) | j #LE# NPATS:
                                   X(j)), @NEWLINE( 2),
         ' Total feet yield:   ', TOTAL_FT_YIELD , @NEWLINE( 1),
         ' Total feet used:    ', TOTAL_FT_USED , @NEWLINE( 2),
         ' Percent waste:    ', @FORMAT( PERC_WASTE, '#5.2G'),
         '%',   @NEWLINE( 1));
   @WRITE( @NEWLINE( 1), 24*' ', 'Pattern:', @NEWLINE( 1));
   @WRITE( '   FG  Demand Yield');
   @FOR( PATTERN( I) | I #LE# NPATS: @WRITE( @FORMAT( I, '6.6G')));
   @WRITE( @NEWLINE( 1));
   @WRITE( ' ',FW*( NPATS+3)*'=', @NEWLINE( 1));
   @FOR( FG( F):
     @WRITE((FW - @STRLEN( FG( F)))*' ', FG( F), ' ',
       @FORMAT( DEM( F), '6.6G'), @FORMAT( YIELD( F), '6.6G'));
     @FOR( FXP( F, P) | P #LE# NPATS:
       @WRITE( @IF( NBR( F, P) #GT# 0,
       @FORMAT( NBR( F, P), "6.6G"), '     .')));
     @WRITE( @NEWLINE( 1))
   );
   @WRITE( ' ',FW*( NPATS+3)*'=', @NEWLINE( 1));
   @WRITE( 2*FW*' ', ' Usage:');
   @WRITEFOR( PATTERN( P) | P#LE# NPATS: @FORMAT( X( P), '6.6G'));
   @WRITE( @NEWLINE( 1));
  ENDCALC
  END
```

19.6.2 Column Generation and Integer Programming

Column generation can be used to easily find an optimum solution to an LP. This is not quite true with IP's. The problem is, with an IP, there is no simple equivalent to dual price. Dual prices may be printed in an IP solution, but they have an extremely limited interpretation. For example, it may be that all dual prices are 0 in an IP solution.

Thus, the usual approach, when column generation is used to attack IP's, is to use column generation only to solve the LP relaxation. A standard IP algorithm is then applied to the problem composed of only the columns generated during the LP. However, it may be that a true IP optimum includes one or more columns that were not generated during the LP phase. The LP solution, nevertheless, provides a bound on the optimum LP solution. In our previous cutting stock example, this bound was tight.

There is a fine point to be made with regard to the stopping rule. We stop, *not* when the previous column added leads to no improvement in the LP solution, but when the latest column generated prices out unattractively.

19.6.3 Row Generation

An analogous approach can be used if the problem intrinsically has many thousands of constraints even though only a few of them will be binding. The basic approach is:

1. Generate some of the constraints.
2. Solve the problem with the existing constraints.
3. Find a constraint that has not been generated, but is violated. If none, we are done.
4. Add the violated constraint and go to (2)

19.7 Problems

1. A rope, 30 meters long is suspended between two vertical poles, each 20 meters high. The lowest point of the hanging rope is 5 meters above the (level) ground. How far apart are the two poles? Use the model verification technique of checking extreme cases, to answer this question.

2. Consider the LP:

```
MAX = W + 20 * X;
               X               + Z <= 70;
                       Y       + Z <= 55;
      W         - 30 * Y   - 46 * Z = 0;
               X + 2 * Y + 3.1 * Z <= 130;
END
```

   All variables are nonnegative. Show how this model can be reduced to a smaller, but equivalent LP.

3. One of the most important and creative steps in analyzing or modeling a problem is identifying the decision variables or, more generally, identifying the options available in solving the problem. Show how paying close attention to the options available allows you to solve the following two problems:

   a) On a sheet of paper, arrange nine dots in a 3 by 3 square grid. Without removing your pen from the paper, draw four straight connected line segments that pass through all nine points.

   b) Rearrange the letters in "new door" to create one word.

4. Important skills in decision making and model formulation are identifying decision options and constraints. Consider the figure below. Draw the shortest path from *A* to *B*.

A

B

5. A common footnote seen in financial reports is a phrase like: “numbers may not sum to 100 because of rounding”. This is an example the rounding problem frequently encountered when preparing reports. In its simplest form, one is given a column of numbers, some of which have fractional parts. One wants to round the numbers to integers, or to numbers with fewer fractional digits, so that the sum of the rounded numbers equals the sum of the original numbers. Some variations of this problem are: a) in regulated utilities where a firm is allowed to round certain charges to a multiple of a nickel, subject to having the total equal a certain quantity; b) in some parliaments, e.g., the U.S. House of Representatives, each state is supposed to have an integer number of representatives(out of a fixed total of 435) proportional to the state’s population. To illustrate, consider the following two sets of numbers and their sums:

Set 1	Set 2
23.3	3.7
15.4	11.6
61.3	9.8
100.0	47.7
	9.6
	11.5
	6.1
	100.0

Notice that for both examples, if you round each component to the nearer integer, the results will not sum to 100.

a) Specify a method for rounding the components of an arbitrary set of numbers, extolling the virtues of your method.

b) Illustrate your method on the two examples.

6. You have created a supply chain modeling system that can accommodate from 1 to 10 products, from 1 to 6 suppliers, from 1 to 25 customers, and from 1 to 12 time periods. In order to test your modeling system you want run at least one case for each extreme of each dimension. In case there is a bug in your modeling system that depends upon interactions between two dimension you also want to run at least one case for each pair of extremes. For example, considering products and suppliers, you want to run at least four cases covering: 1) 1 product and 1 supplier, 2) 1 product and 6 suppliers, 3) 10 products and 1 supplier, 4) 10 products and 6 suppliers. Note that a given test case in fact covers one product setting, one supplier setting, one customer setting, and one number-of-periods setting. What is the minimum number of test cases you need to run, so that if you consider any pair of dimensions, each of the four combinations of extreme cases have been run?