

---

## Building Larger Models

The linear programs that we have presented so far have been quite small, so their data and solutions could fit onto a page. Most of the LPs found in practical applications, however, have hundreds or thousands of variables and constraints, and some are even larger.

How do linear programs get to be so large? They might be like the ones we have shown, but with larger indexing sets and more data. A steel mill could be considered to make hundreds of different products, for example, if every variation of width, thickness, and finish is treated separately. Or a large organization could have thousands of people involved in one assignment problem. Nevertheless, these kinds of applications are not as common as one might expect. As a model is refined to greater levels of detail, its data values become harder to maintain and its solutions harder to understand; past a certain point, extra detail offers no benefit. Thus to plan production for a few lines, considerable detail may be justifiable; but to plan for an entire company, it may be better to have a small aggregated, plant-level model that be run many times with different scenarios.

A more common source of large linear programs is the linking together of smaller ones. It is not unusual for an application to give rise to many simple LPs of the kinds we have discussed before; here are three possibilities:

Many products are to be shipped, and there is a transportation problem (as in Chapter 3) for each product.

Manufacturing is to be planned over many weeks, and there is a production problem (as in Chapter 1) for each week.

Several products are made at several mills, and shipped to several factories; there is a production problem for each mill, and a transportation problem for each product.

When variables or constraints are added to tie these LPs together, the result can be one very large LP. No individual part need be particularly detailed; the size is more due to the large number of combinations of origins, destinations, products and weeks.

This chapter shows how AMPL models might be formulated for the three situations outlined above. The resulting models are necessarily more complicated than our previous ones, and require the use of a few more features from the AMPL language. Since they build on the terminology and logic of smaller models that have already been introduced, however, these larger models are still manageable.

## 4.1 A multicommodity transportation model

The transportation model of the previous chapter was concerned with shipping a single commodity from origins to destinations. Suppose now that we are shipping several different products. We can define a new set, `PROD`, whose members represent the different products, and we can add `PROD` to the indexing of every component in the model; the result can be seen in Figure 4-1. Because `supply`, `demand`, `cost`, and `Trans` are indexed over one more set in this version, they take one more subscript: `supply[i,p]` for the amount of product `p` shipped from origin `i`, `Trans[i,j,p]` for the amount of `p` shipped from `i` to `j`, and so forth. Even the `check` statement is now indexed over `PROD`, so that it verifies that supply equals demand for each separate product.

If we look at `Supply`, `Demand` and `Trans`, there are (origins + destinations) (products) constraints in (origins) (destinations) (products) variables. The result could be quite a large linear program, even if the individual sets do not have many members. For example, 5 origins, 20 destinations and 10 products give 250 constraints in 1000 variables. The size of this LP is misleading, however, because the shipments of the products are independent. That is, the amounts we ship of one product do not affect the amounts we can ship of any other product, or the costs of shipping any other product. We would do better in this case to solve a smaller transportation problem for each individual product. In AMPL terms, we would use the simple transportation model from the previous chapter, together with a different data file for each product.

The situation would be different if some additional circumstances had the effect of tying together the different products. As an example, imagine that there are restrictions on the total shipments of products from an origin to a destination, perhaps because of limited shipping capacity. To accommodate such restrictions in our model, we declare a new parameter `limit` indexed over the combinations of origins and destinations:

```
param limit {ORIG,DEST} >= 0;
```

Then we have a new collection of (origins) (destinations) constraints, one for each origin `i` and destination `j`, which say that the sum of shipments from `i` to `j` of all products `p` may not exceed `limit[i,j]`:

```
subject to Multi {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Subject to these constraints (also shown in Figure 4-1), we can no longer set the amount of one product shipped from `i` to `j` without considering the amounts of other products also shipped from `i` to `j`, since it is the sum of all products that is limited. Thus we have no choice but to solve the one large linear program.

For the steel mill in Chapter 1, the products were bands, coils, and plate. Thus the data for the multicommodity model could look like Figure 4-2. We invoke AMPL in the usual way to get the following solution:

```
ampl: model multi.mod; data multi.dat; solve;
CPLEX 8.0.0: optimal solution; objective 199500
41 dual simplex iterations (0 in phase I)
```

---

```

set ORIG;    # origins
set DEST;    # destinations
set PROD;    # products

param supply {ORIG,PROD} >= 0; # amounts available at origins
param demand {DEST,PROD} >= 0; # amounts required at destinations

    check {p in PROD}:
        sum {i in ORIG} supply[i,p] = sum {j in DEST} demand[j,p];

param limit {ORIG,DEST} >= 0;

param cost {ORIG,DEST,PROD} >= 0; # shipment costs per unit
var Trans {ORIG,DEST,PROD} >= 0;  # units to be shipped

minimize Total_Cost:
    sum {i in ORIG, j in DEST, p in PROD}
        cost[i,j,p] * Trans[i,j,p];

subject to Supply {i in ORIG, p in PROD}:
    sum {j in DEST} Trans[i,j,p] = supply[i,p];

subject to Demand {j in DEST, p in PROD}:
    sum {i in ORIG} Trans[i,j,p] = demand[j,p];

subject to Multi {i in ORIG, j in DEST}:
    sum {p in PROD} Trans[i,j,p] <= limit[i,j];

```

**Figure 4-1:** Multicommodity transportation model (`multi.mod`).

---

```

ampl: display {p in PROD}: {i in ORIG, j in DEST} Trans[i,j,p];

Trans[i,j,Wands] [*,*] (tr)
:      CLEV  GARY  PITT    :=
DET      0      0    300
FRA    225      0     75
FRE      0      0    225
LAF    225      0     25
LAN      0      0    100
STL    250    400     0
WIN      0      0     75
;

Trans[i,j,Boils] [*,*] (tr)
:      CLEV  GARY  PITT    :=
DET    525      0    225
FRA      0      0    500
FRE    225    625     0
LAF      0    150    350
LAN    400      0     0
STL    300     25    625
WIN    150      0    100
;

```

---

```

set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;

param supply (tr):  GARY    CLEV    PITT :=
                    bands    400     700     800
                    coils    800    1600    1800
                    plate    200     300     300 ;

param demand (tr):
                    FRA    DET    LAN    WIN    STL    FRE    LAF :=
    bands    300    300    100    75    650    225    250
    coils    500    750    400    250    950    850    500
    plate    100    100     0    50    200    100    250 ;

param limit default 625 ;

param cost :=

    [*,*,bands]:  FRA    DET    LAN    WIN    STL    FRE    LAF :=
        GARY     30     10     8     10     11     71     6
        CLEV     22     7     10     7     21     82     13
        PITT     19     11     12     10     25     83     15

    [*,*,coils]:  FRA    DET    LAN    WIN    STL    FRE    LAF :=
        GARY     39     14     11     14     16     82     8
        CLEV     27     9     12     9     26     95     17
        PITT     24     14     17     13     28     99     20

    [*,*,plate]:  FRA    DET    LAN    WIN    STL    FRE    LAF :=
        GARY     41     15     12     16     17     86     8
        CLEV     29     9     13     9     28     99     18
        PITT     26     14     17     13     31    104     20 ;

```

---

**Figure 4-2:** Multicommodity transportation problem data (multi.dat).

---

```

Trans[i,j,plate%] [*,*] (tr)
:      CLEV  GARY  PITT      :=
DET    100     0     0
FRA     50     0    50
FRE    100     0     0
LAF     0     0   250
LAN     0     0     0
STL     0    200     0
WIN     50     0     0
;

```

In both our specification of the shipping costs and AMPL's display of the solution, a three-dimensional collection of data (that is, indexed over three sets) must be represented on a two-dimensional screen or page. We accomplish this by slicing the data along one index, so that it appears as a collection of two-dimensional tables. The display command will make a guess as to the best index on which to slice, but by use of an

---

```

set PROD;      # products
param T > 0;    # number of weeks

param rate {PROD} > 0;      # tons per hour produced
param avail {1..T} >= 0;    # hours available in week
param profit {PROD,1..T};   # profit per ton
param market {PROD,1..T} >= 0; # limit on tons sold in week

var Make {p in PROD, t in 1..T} >= 0, <= market[p,t];
                                # tons produced

maximize Total_Profit:
    sum {p in PROD, t in 1..T} profit[p,t] * Make[p,t];
    # total profits from all products in all weeks

subject to Time {t in 1..T}:
    sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];
    # total of hours used by all products
    # may not exceed hours available, in each week

```

---

**Figure 4-3:** Production model replicated over periods (steelT0.mod).

---

explicit indexing expression as shown above, we can tell it to display a table for each product.

The optimal solution above ships only 25 tons of coils from GARY to STL and 25 tons of bands from PITT to LAF. It might be reasonable to require that, if any amount at all is shipped, it must be at least, say, 50 tons. In terms of our model, either  $\text{Trans}[i, j, p] = 0$  or  $\text{Trans}[i, j, p] \geq 50$ . Unfortunately, although it is possible to write such an ~~either/or~~ constraint in AMPL, it is not a linear constraint, and so there is no way that an LP solver can handle it. Chapter 20 explains how more powerful (but costlier) integer programming techniques can deal with this and related kinds of discrete restrictions.

## 4.2 A multiperiod production model

Another common way in which models are expanded is by replicating them over time. To illustrate, we consider how the model of Figure 1-4a might be used to plan production for the next  $T$  weeks, rather than for a single week.

We begin by adding another index set to most of the quantities of interest. The added set represents weeks numbered 1 through  $T$ , as shown in Figure 4-3. The expression  $1..T$  is AMPL's shorthand for the set of integers from 1 through  $T$ . We have replicated all the parameters and variables over this set, except for `rate`, which is regarded as fixed over time. As a result there is a constraint for each week, and the `profit` terms are summed over weeks as well as products.

So far this is merely a separate LP for each week, unless something is added to tie the weeks together. Just as we were able to find constraints that involved all the products, we

could look for constraints that involve production in all of the weeks. Most multiperiod models take a different approach, however, in which constraints relate each week's production to that of the following week only.

Suppose that we allow some of a week's production to be placed in inventory, for sale in any later week. We thus add new decision variables to represent the amounts inventoried and sold in each week. The variables `Make[j,t]` are retained, but they represent only the amounts produced, which are now not necessarily the same as the amounts sold. Our new variable declarations look like this:

```
var Make {PROD,1..T} >= 0;
var Inv {PROD,0..T} >= 0;

var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t];
```

The bounds `market[p,t]`, which represent the maximum amounts that can be sold in a week, are naturally transferred to `Sell[p,t]`.

The variable `Inv[p,t]` will represent the inventory of product `p` at the end of period `t`. Thus the quantities `Inv[p,0]` will be the inventories at the end of week zero, or equivalently at the beginning of the first week—in other words, now. Our model assumes that these initial inventories are provided as part of the data:

```
param inv0 {PROD} >= 0;
```

A simple constraint guarantees that the variables `Inv[p,0]` take these values:

```
subject to Init_Inv {p in PROD}: Inv[p,0] = inv0[p];
```

It may seem inefficient to devote a constraint like this to saying that a variable equals a constant, but when it comes time to send the linear program to a solver, AMPL will automatically substitute the value of `inv0[p]` for any occurrence of `Inv[p,0]`. In most cases, we can concentrate on writing the model in the clearest or easiest way, and leave matters of efficiency to the computer.

Now that we are distinguishing sales, production, and inventory, we can explicitly model the contribution of each to the profit, by defining three parameters:

```
param revenue {PROD,1..T} >= 0;
param prodcost {PROD} >= 0;
param invcost {PROD} >= 0;
```

These are incorporated into the objective as follows:

```
maximize Total_Profit:
    sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
                                prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);
```

As you can see, `revenue[p,t]` is the amount received per ton of product `p` sold in week `t`; `prodcost[p]` and `invcost[p]` are the production and inventory carrying cost per ton of product `p` in any week.

Finally, with the sales and inventories fully incorporated into our model, we can add the key constraints that tie the weeks together: the amount of a product made available in

a week, through production or from inventory, must equal the amount disposed of in that week, through sale or to inventory:

```
subject to Balance {p in PROD, t in 1..T}:
    Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
```

Because the index  $t$  is from a set of numbers, the period previous to  $t$  can be written as  $t-1$ . In fact,  $t$  can be used in any arithmetic expression; conversely, an AMPL expression such as  $t-1$  may be used in any context where it makes sense. Notice also that for a first-period constraint ( $t$  equal to 1), the inventory term on the left is  $\text{Inv}[p, 0]$ , the initial inventory.

We now have a complete model, as shown in Figure 4-4. To illustrate a solution, we use the small sample data file shown in Figure 4-5; it represents a four-week expansion of the data from Figure 1-4b.

If we put the model and data into files `steelT.mod` and `steelT.dat`, then AMPL can be invoked to find a solution:

```
ampl: model steelT.mod;
ampl: data steelT.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
20 iterations, objective 515033

ampl: option display_lcol 0;

ampl: display Make;
Make [*,*] (tr)
:  bands  coils      :=
1   5990   1407
2   6000   1400
3   1400   3500
4   2000   4200
;

ampl: display Inv;
Inv [*,*] (tr)
:  bands  coils      :=
0    10      0
1     0    1100
2     0       0
3     0       0
4     0       0
;

ampl: display Sell;
Sell [*,*] (tr)
:  bands  coils      :=
1   6000    307
2   6000   2500
3   1400   3500
4   2000   4200
;
```

---

```

set PROD;      # products
param T > 0;    # number of weeks

param rate {PROD} > 0;      # tons per hour produced
param inv0 {PROD} >= 0;     # initial inventory
param avail {1..T} >= 0;    # hours available in week
param market {PROD,1..T} >= 0; # limit on tons sold in week

param prodcost {PROD} >= 0; # cost per ton produced
param invcost {PROD} >= 0;  # carrying cost/ton of inventory
param revenue {PROD,1..T} >= 0; # revenue per ton sold

var Make {PROD,1..T} >= 0;   # tons produced
var Inv {PROD,0..T} >= 0;   # tons inventoried
var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t]; # tons sold

maximize Total_Profit:
    sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
        prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);
        # Total revenue less costs in all weeks

subject to Time {t in 1..T}:
    sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];
        # Total of hours used by all products
        # may not exceed hours available, in each week

subject to Init_Inv {p in PROD}: Inv[p,0] = inv0[p];
        # Initial inventory must equal given value

subject to Balance {p in PROD, t in 1..T}:
    Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
        # Tons produced and taken from inventory
        # must equal tons sold and put into inventory

```

**Figure 4-4:** Multiperiod production model (steelT.mod).

```

param T := 4;
set PROD := bands coils;

param avail := 1 40 2 40 3 32 4 40 ;

param rate := bands 200 coils 140 ;
param inv0 := bands 10 coils 0 ;

param prodcost := bands 10 coils 11 ;
param invcost := bands 2.5 coils 3 ;

param revenue: 1 2 3 4 :=
    bands 25 26 27 27
    coils 30 35 37 39 ;

param market: 1 2 3 4 :=
    bands 6000 6000 4000 6500
    coils 4000 2500 3500 4200 ;

```

**Figure 4-5:** Data for multiperiod production model (steelT.dat).

---



Production of coils in the first week is held over to be sold at a higher price in the second week. In the second through fourth weeks, coils are more profitable than bands, and so coils are sold up to the limit, with bands filling out the capacity. (Setting option `display_1col` to zero permits this output to appear in a nicer format, as explained in Section 12.2.)

### 4.3 A model of production and transportation

Large linear programs can be created not only by tying together small models of one kind, as in the two examples above, but by linking different kinds of models. We conclude this chapter with an example that combines features of both production and transportation models.

Suppose that the steel products are made at several mills, from which they are shipped to customers at the various factories. For each mill we can define a separate production model to optimize the amounts of each product to make. For each product we can define a separate transportation model, with mills as origins and factories as destinations, to optimize the amounts of the product to be shipped. We would like to link all these separate models into a single integrated model of production and transportation.

To begin, we replicate the production model of Figure 1-4a over mills ~~that is~~, origins ~~rather than over weeks as in the previous example~~:

```

set PROD;      # products
set ORIG;      # origins (steel mills)

param rate {ORIG,PROD} > 0; # tons per hour at origins
param avail {ORIG} >= 0;    # hours available at origins

var Make {ORIG,PROD} >= 0;  # tons produced at origins

subject to Time {i in ORIG}:
    sum {p in PROD} (1/rate[i,p]) * Make[i,p] <= avail[i];

```

We have temporarily dropped the components pertaining to the objective, to which we will return later. We have also dropped the market demand parameters, since the demands are now properly associated with the destinations in the transportation models.

The next step is to replicate the transportation model, Figure 3-1a, over products, as we did in the multicommodity example at the beginning of this chapter:

```

set ORIG;      # origins (steel mills)
set DEST;      # destinations (factories)
set PROD;      # products

param supply {ORIG,PROD} >= 0; # tons available at origins
param demand {DEST,PROD} >= 0; # tons required at destinations

var Trans {ORIG,DEST,PROD} >= 0; # tons shipped

```

---

```

set ORIG;    # origins (steel mills)
set DEST;    # destinations (factories)
set PROD;    # products

param rate {ORIG,PROD} > 0;      # tons per hour at origins
param avail {ORIG} >= 0;         # hours available at origins
param demand {DEST,PROD} >= 0;  # tons required at destinations

param make_cost {ORIG,PROD} >= 0;      # manufacturing cost/ton
param trans_cost {ORIG,DEST,PROD} >= 0; # shipping cost/ton

var Make {ORIG,PROD} >= 0;      # tons produced at origins
var Trans {ORIG,DEST,PROD} >= 0; # tons shipped

minimize Total_Cost:
    sum {i in ORIG, p in PROD} make_cost[i,p] * Make[i,p] +
    sum {i in ORIG, j in DEST, p in PROD}
        trans_cost[i,j,p] * Trans[i,j,p];

subject to Time {i in ORIG}:
    sum {p in PROD} (1/rate[i,p]) * Make[i,p] <= avail[i];

subject to Supply {i in ORIG, p in PROD}:
    sum {j in DEST} Trans[i,j,p] = Make[i,p];

subject to Demand {j in DEST, p in PROD}:
    sum {i in ORIG} Trans[i,j,p] = demand[j,p];

```

---

**Figure 4-6:** Production/transportation model, 3rd version (steelP.mod).

---

```

subject to Supply {i in ORIG, p in PROD}:
    sum {j in DEST} Trans[i,j,p] = supply[i,p];

subject to Demand {j in DEST, p in PROD}:
    sum {i in ORIG} Trans[i,j,p] = demand[j,p];

```

Comparing the resulting production and transportation models, we see that the sets of origins (ORIG) and products (PROD) are the same in both models. Moreover, the ~~tons~~ available at origins~~to supply~~ in the transportation model are really the same thing as the ~~tons~~ produced at origins~~to Make~~ in the production model, since the steel available for shipping will be whatever is made at the mill.

We can thus merge the two models, dropping the definition of supply and substituting `Make[i,p]` for the occurrence of `supply[i,p]`:

```

subject to Supply {i in ORIG, p in PROD}:
    sum {j in DEST} Trans[i,j,p] = Make[i,p];

```

There are several ways in which we might add an objective to complete the model. Perhaps the simplest is to define a cost per ton corresponding to each variable. We define a parameter `make_cost` so that there is a term `make_cost[i,p] * Make[i,p]` in the objective for each origin `i` and product `p`; and we define `trans_cost` so that there is a term `trans_cost[i,j,p] * Trans[i,j,p]` in the objective for each origin `i`, destination `j` and product `p`. The full model is shown in Figure 4-6.

---

```

set ORIG := GARY CLEV PITT ;
set DEST := FRA DET LAN WIN STL FRE LAF ;
set PROD := bands coils plate ;

param avail :=  GARY 20  CLEV 15  PITT 20 ;

param demand (tr):
      FRA  DET  LAN  WIN  STL  FRE  LAF :=
bands  300  300  100   75  650  225  250
coils   500  750  400  250  950  850  500
plate   100  100   0   50  200  100  250 ;

param rate (tr):  GARY  CLEV  PITT :=
      bands    200    190    230
      coils    140    130    160
      plate    160    160    170 ;

param make_cost (tr):
      GARY  CLEV  PITT :=
      bands  180   190   190
      coils  170   170   180
      plate  180   185   185 ;

param trans_cost :=
[*,*,bands]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
      GARY    30   10   8   10   11   71   6
      CLEV    22    7   10   7   21   82  13
      PITT    19   11   12   10  25   83  15

[*,*,coils]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
      GARY    39   14   11   14   16   82   8
      CLEV    27    9   12    9   26   95  17
      PITT    24   14   17   13   28   99  20

[*,*,plate]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
      GARY    41   15   12   16   17   86   8
      CLEV    29    9   13    9   28   99  18
      PITT    26   14   17   13   31  104  20 ;

```

---

**Figure 4-7:** Data for production/transportation model (steelP.dat).

---

Reviewing this formulation, we might observe that, according to the Supply declaration, the nonnegative expression

$$\sum \{j \text{ in DEST}\} \text{Trans}[i,j,p]$$

can be substituted for  $\text{Make}[i,p]$ . If we make this substitution for all occurrences of  $\text{Make}[i,p]$  in the objective and in the Time constraints, we no longer need to include the Make variables or the Supply constraints in our model, and our linear programs will be smaller as a result. Nevertheless, in most cases we will be better off leaving the model as it is shown above. By substituting out the Make variables we render the model harder to read, and not a great deal easier to solve.

As an instance of solving a linear program based on this model, we can adapt the data from Figure 4-2, as shown in Figure 4-7. Here are some representative result values:

```

ampl: model steelP.mod; data steelP.dat; solve;
CPLEX 8.0.0: optimal solution; objective 1392175
27 dual simplex iterations (0 in phase I)

ampl: option display_1col 5;
ampl: option omit_zero_rows 1, omit_zero_cols 1;

ampl: display Make;
Make [*,*]
:      bands   coils plate      :=
CLEV      0    1950      0
GARY    1125    1750    300
PITT      775     500    500
;

ampl: display Trans;
Trans [CLEV,*,*]
:      coils      :=
DET      750
LAF      500
LAN      400
STL      50
WIN      250

[GARY,*,*]
:      bands coils plate      :=
FRE    225    850    100
LAF    250      0      0
STL    650    900    200

[PITT,*,*]
:      bands coils plate      :=
DET    300      0    100
FRA    300    500    100
LAF      0      0    250
LAN    100      0      0
WIN     75      0    50
;

ampl: display Time;
Time [*] :=
CLEV  -1300
GARY  -2800
;

```

As one might expect, the optimal solution does not ship all products from all mills to all factories. We have used the options `omit_zero_rows` and `omit_zero_cols` to suppress the printing of table rows and columns that are all zeros. The dual values for `Time` show that additional capacity is likely to have the greatest impact on total cost if it is placed at GARY, and no impact if it is placed at PITT.

We can also investigate the relative costs of production and shipping, which are the two components of the objective:

```

ampl: display sum {i in ORIG, p in PROD}
      make_cost[i,p] * Make[i,p];
sum{i in ORIG, p in PROD} make_cost[i,p]*Make[i,p] = 1215250
ampl: display sum {i in ORIG, j in DEST, p in PROD}
      trans_cost[i,j,p] * Trans[i,j,p];
sum{i in ORIG, j in DEST, p in PROD}
  trans_cost[i,j,p]*Trans[i,j,p] = 176925

```

Clearly the production costs dominate in this case. These examples point up the ability of AMPL to evaluate and display any valid expression.

## Bibliography

H. P. Williams, *Model Building in Mathematical Programming* (4th edition). John Wiley & Sons (New York, 1999). An extended compilation of many kinds of models and combinations of them.

## Exercises

**4-1.** Formulate a multi-period version of the transportation model, in which inventories are kept at the origins.

**4-2.** Formulate a combination of a transportation model for each of several foods, and a diet model at each destination.

**4-3.** The following questions pertain to the multiperiod production model and data of Section 4.2.

(a) Display the marginal values associated with the constraints `Time[t]`. In which periods does it appear that additional production capacity would be most valuable?

(b) By soliciting additional sales, you might be able to raise the upper bounds `market[p,t]`. Display the reduced costs `Sell[p,t].rc`, and use them to suggest whether you would prefer to go after more orders of bands or of coils in each week.

(c) If the inventory costs are all positive, any optimal solution will have zero inventories after the last week. Why is this so?

This phenomenon is an example of an ~~end effect~~ <sup>end effect</sup>. Because the model comes to an end after period  $T$ , the solution tends to behave as if production is to be shut down after that point. One way of dealing with end effects is to increase the number of weeks modeled; then the end effects should have little influence on the solution for the earlier weeks. Another approach is to modify the model to better reflect the realities of inventories. Describe some modifications you might make to the constraints, and to the objective.

**4-4.** A producer of packaged cookies and crackers runs several shifts each month at its large bakery. This exercise is concerned with a multiperiod planning model for deciding how many crews to employ each month. In the algebraic description of the model, there are sets  $S$  of shifts and  $P$  of

products, and the planning horizon is  $T$  four-week periods. The relevant operational data are as follows:

- $l$  number of production lines: maximum number of crews that can work in any shift
- $r_p$  production rate for product  $p$ , in crew-hours per 1000 boxes
- $h_t$  number of hours that a crew works in planning period  $t$

The following data are determined by market or managerial considerations:

- $w_s$  total wages for a crew on shift  $s$  in one period
- $d_{pt}$  demand for product  $p$  that must be met in period  $t$
- $M$  maximum change in number of crews employed from one period to the next

The decision variables of the model are:

- $X_{pt} \geq d_{pt}$  total boxes (in 1000s) of product  $p$  baked in period  $t$
- $0 \leq Y_{st} \leq l$  number of crews employed on shift  $s$  in period  $t$

The objective is to minimize the total cost of all crews employed,

$$\sum_{s \in S} \sum_{t=1}^T w_s Y_{st}.$$

Total hours required for production in each period may not exceed total hours available from all shifts,

$$\sum_{p \in P} r_p X_{pt} \leq h_t \sum_{s \in S} Y_{st}, \text{ for each } t = 1, \dots, T.$$

The change in number of crews is restricted by

$$M \leq \sum_{s \in S} (Y_{s,t+1} - Y_{st}) \leq M, \text{ for each } t = 1, \dots, T-1.$$

As required by the definition of  $M$ , this constraint restricts any change to lie between a reduction of  $M$  crews and an increase of  $M$  crews.

(a) Formulate this model in AMPL, and solve the following instance. There are  $T=13$  periods,  $l=8$  production lines, and a maximum change of  $M=3$  crews per period. The products are 18REG, 24REG, and 24PRO, with production rates  $r_p$  of 1.194, 1.509 and 1.509 respectively. Crews work either a day shift with wages  $w_s$  of \$44,900, or a night shift with wages \$123,100. The demands and working hours are given as follows by period:

Period $t$	$d_{18REG,t}$	$d_{24REG,t}$	$d_{24PRO,t}$	$h_t$
1	63.8	1212.0	0.0	156
2	76.0	306.2	0.0	152
3	88.4	319.0	0.0	160
4	913.8	208.4	0.0	152
5	115.0	298.0	0.0	156
6	133.8	328.2	0.0	152
7	79.6	959.6	0.0	152
8	111.0	257.6	0.0	160
9	121.6	335.6	0.0	152
10	470.0	118.0	1102.0	160
11	78.4	284.8	0.0	160
12	99.4	970.0	0.0	144
13	140.4	343.8	0.0	144

Display the numbers of crews required on each shift. You will find many fractional numbers of crews; how would you convert this solution to an optimal one in whole numbers?

(b) To be consistent, you should also require at most a change of  $M$  between the known initial number of crews (already employed in the period before the first) and the number of crews to be employed in the first planning period. Add a provision for this restriction to the model.

Re-solve with 11 initial crews. You should get the same solution.

(c) Because of the limit on the change in crews from period to period, more crews than necessary are employed in some periods. One way to deal with this is to carry inventories from one period to the next, so as to smooth out the amount of production required in each period. Add a variable for the amount of inventory of each product after each period, as in the model of Figure 4-4, and add constraints that relate inventory to production and demand. (Because inventories can be carried forward, production  $X_{pt}$  need not be  $\geq$  demand  $d_{pt}$  in every period as required by the previous versions.) Also make a provision for setting initial inventories to zero. Finally, add an inventory cost per period per 1000 boxes to the objective.

Let the inventory costs be \$34.56 for product 18REG, and \$43.80 for 24REG and 24PRO. Solve the resulting linear program; display the crew sizes and inventory levels. How different is this solution? How much of a saving is achieved in labor cost, at how much expense in inventory cost?

(d) The demands in the given data peak at certain periods, when special discount promotions are in effect. Big inventories are built up in advance of these peaks, particularly before period 4. Baked goods are perishable, however, so that building up inventories past a certain number of periods is unrealistic.

Modify the model so that the inventory variables are indexed by product, period and age, where age runs from 1 to a specified limit  $A$ . Add constraints that the inventories of age 1 after any period cannot exceed the amounts just produced, and that inventories of age  $a > 1$  after period  $t$  cannot exceed the inventories of age  $a - 1$  after period  $t - 1$ .

Verify that, with a maximum inventory age of 2 periods, you can use essentially the same solution as in (c), but that with a maximum inventory age of 1 there are some periods that require more crews.

(e) Suppose now that instead of adding a third index on inventory variables as in (d), you impose the following inventory constraint: The amount of product  $p$  in inventory after period  $t$  may not exceed the total production of product  $p$  in periods  $t - A + 1$  through  $t$ .

Explain why this constraint is sufficient to prevent any inventory from being more than  $A$  periods old, provided that inventories are managed on a *first-in, first-out* basis. Support your conclusion by showing that you get the same results as in (d) when solving with a maximum inventory age of 2 or of 1.

(f) Explain how you would modify the models in (c), (d), and (e) to account for initial inventories that are *not* zero.

**4-5.** Multiperiod linear programs can be especially difficult to develop, because they require data pertaining to the future. To hedge against the uncertainty of the future, a user of these LPs typically develops various scenarios, containing different forecasts of certain key parameters. This exercise asks you to develop what is known as a stochastic program, which finds a solution that can be considered robust over all scenarios.

(a) The revenues per ton might be particularly hard to predict, because they depend on fluctuating market conditions. Let the revenue data in Figure 4-5 be scenario 1, and also consider scenario 2:

```

param revenue:  1      2      3      4 :=
    bands      23      24      25      25
    coils      30      33      35      36 ;

```

and scenario 3:

```

param revenue:  1      2      3      4 :=
    bands      21      27      33      35
    coils      30      32      33      33 ;

```

By solving the three associated linear programs, verify that each of these scenarios leads to a different optimal production and sales strategy, even for the first week. You need one strategy, however, not three. The purpose of the stochastic programming approach is to determine a single solution that produces a good profit ~~in~~ average ~~in~~ a certain sense.

(b) As a first step toward formulating a stochastic program, consider how the three scenarios could be brought together into one linear program. Define a parameter  $S$  as the number of scenarios, and replicate the revenue data over the set  $1..S$ :

```

param S > 0;
param revenue {PROD,1..T,1..S} >= 0;

```

Replicate all the variables and constraints in a similar way. (The idea is the same as earlier in this chapter, where we replicated model components over products or weeks.)

Define a new collection of parameters  $\text{prob}[s]$ , to represent your estimate of the probability that a scenario  $s$  takes place:

```

param prob {1..S} >= 0, <= 1;
check: 0.99999 < sum {s in 1..S} prob[s] < 1.00001;

```

The objective function is the expected profit, which equals the sum over all scenarios of the probability of each scenario times the optimum profit under that scenario:

```

maximize Expected_Profit:
    sum {s in 1..S} prob[s] *
        sum {p in PROD, t in 1..T} (revenue[p,t,s]*Sell[p,t,s] -
            prodcost[p]*Make[p,t,s] - invcost[p]*Inv[p,t,s]);

```

Complete the formulation of this multiscenario linear program, and put together the data for it. Let the probabilities of scenarios 1, 2 and 3 be 0.45, 0.35 and 0.20, respectively. Show that the solution consists of a production strategy for each scenario that is the same as the strategy in (a).

(c) The formulation in (b) is no improvement because it makes no connection between the scenarios. One way to make the model usable is to add ~~nonanticipativity~~ constraints that require each week-1 variable to be given the same value across all scenarios. Then the result will give you the best single strategy for the first week, in the sense of maximizing expected profit for all weeks. The strategies will still diverge after the first week ~~but~~ a week from now you can update your data and run the stochastic program again to generate a second week ~~strategy~~.

A nonanticipativity constraint for the Make variables can be written

```

subject to Make_na {p in PROD, s in 1..S-1}:
    Make[p,1,s] = Make[p,1,s+1];

```

Add the analogous constraints for the Inv and Sell variables. Solve the stochastic program, and verify that the solution consists of a single period-1 strategy for all three scenarios.

(d) After getting your solution in (c), use the following command to look at the profits that the recommended strategy will achieve under the three scenarios:



```
display {s in 1..S}
  sum {p in PROD, t in 1..T} (revenue[p,t,s]*Sell[p,t,s] -
    prodcost[p]*Make[p,t,s] - invcost[p]*Inv[p,t,s]);
```

Which scenario will be most profitable, and which will be least profitable?

Repeat the analysis with probabilities of 0.0001, 0.0001 and 0.9998 for scenarios 1, 2 and 3. You should find that profit from strategy 3 goes up, but profits from the other two go down. Explain what these profits represent, and why the results are what you would expect.