Factory Planning

Factory planning

How do you find the optimal way to use your factory to increase your profits?

This sample shows how to find the optimal mix of products to manufacture, given production capacities and marketing limitations. You have a factory that makes seven different types of metal products. You use five different machines to process the products, and each product requires the use of certain machine processes for varying lengths of time. Some products are more profitable than others, but these often require greater utilization of the machinery. There are marketing limits to the products as well. You will not be able to sell more of certain products during certain months, even if you can manufacture more. Finally, the machines used in some processes will be down for maintenance during certain months.

This is an example of a multi-period production problem. A single-period production problem just evaluates the best manufacturing decisions for each month separately. However, in this model, it is possible to store certain products. So, you want to create a model that links all the months and takes into account the amounts of products held in storage.

This sample shows how to make a six-month plan that optimizes your factory's profits.

A multi-period production problem (`factoryPlanning.mod`)

In this problem, you have certain information that is known. Your factory can produce seven different products. Each product contributes to the profit, which is defined as a price per unit selling price minus the cost of material. You want to create a plan for six months. You know the number of hours the factory is operational each month.

There are five machines used to process the various products:
- grinder
- vertical drill
- horizontal drill
- borer
- planer

Each product requires the use of certain processing machines for specified amounts of time. For example, for Product 1, you need 0.5 hours of grinder use, 0.1 hours of vertical drill use, 0.2 hours of horizontal drill use, 0.05 hours of borer use, and 0 hours of planer use. You know this information for each product.

There are also marketing limitations on the products. For example, for the month of January, your company cannot sell more than 500 units of Product 1. You know these marketing limitations for all seven products.

You know how much it costs to hold products in your storage facility. You also know what is the maximum capacity at your storage facility.

All of this information is known and you model this known information as **data**.

What is not known? The items in the model that are not known are called **decision variables**. These decision variables are used to find the optimal solution to your problem. They can usually be thought of as questions to be answered. In this problem, there are three types of information that are not known:

- How much of each product to **make** in each of the six months?
- How much of each product to **sell** in each of the six months?
- How much of each product to **hold** in each of the six months?

How do you find the answers to the questions posed by the decision variables? How does the optimizer make the best decisions to find an optimal solution to your problem? You create an **objective**. In this problem, the objective is simple. You want to maximize the profit after subtracting the costs.

Finally, the data can be combined with the decision variables to form **constraints**. For example, there are limits on the storage capacity in your facility. You need to make sure that the solution to your problem does not violate any of the constraints. You have to make sure that the amount of each product that you decide to make, sell, and hold in each month does not require more storage space than is available in that month.

The model (`factoryPlanning.mod`)

In this problem, you model the data first. Then you model the decision variables. Then you model the objective. Finally, you model the constraints.

You want the optimizer to use certain **data** to find the answers to questions posed by the **decision variables** using the objective, which is subject to certain **constraints**.

Modeling the data

For DropSolve, all data must be defined in the form of tables (which in OPL implies using sets of tuples to define these tables).

Each product is defined by a tuple `TProduct` which contains the product name and its unit profit contribution. The table `Products` is then defined as a set of tuples, each tuple forming a row in the table.

```
{tuple TProduct {
  key string name;
  float profit;
}
{TProduct} Products = ...;
```

Another single row table `Calendar` formed from the tuple `TCalendar` contains the number of months for which you want to plan and the number of hours each month that the factory is operational.

```
tuple TCalendar {
  int nbMonths;
  float hoursPerMonth;
}
TCalendar Calendar = ...;
```

There is also an `Inventory` table which contains storage information. Each row (or tuple) defines how much is in storage at the beginning of the time period, `initialQuantity`; how much you want to have in storage at the end of the time period, finalQuantity; and also what is the maximum storage capacity, maxQuantity; and also the cost of storing a unit of a product costPerUnit.

```
tuple TInventoryData {
   float initialQuantity;
   float finalQuantity;
   float maxQuantity;
   float costPerUnit;
}
TInventoryData Inventory = ...;
```

The tuple Process defines the machines that are used in the production process. The table Processes thus contains a single row table with the different machine names as elements.

```
tuple TProcess {
  key string name;
}
{TProcess} Processes = ...;
```

In a similar way, you use tables to model other parts of the data. For example NumProcessPerMonth tells you how many machines are available each month for each process.

```
tuple TProcessNum {
   key string process;
   key int month;
   float num;}
{TProcessNum} NumProcessPerMonth = ...;
```

ProcessPerProduct is a table that represents the hours of each process required by each product.

```
tuple TProcessProduct {
   key string process;
   key string product;
   float required;
}
{TProcessProduct} ProcessPerProduct = ...;
```

MarketForecastPerMonth represents the information about marketing limitations on how many of each product you can sell each month.

```
tuple TProductForecastPerMonth {
  key string product;
  key int month;
  float forecast;
}
{TProductForecastPerMonth} MarketForecastPerMonth = ...;
```

Modeling the decision variables

There are three decision variables to model:

```
dvar float+ Make[Products][Months];
dvar float+ Hold[Products][0..NbMonths] in 0..Inventory.maxQuantity;
dvar float+ Sell[p in Products][m in Months] in 0..forecasts[p][m];
```

These decision variables will be used to answer the questions about how much of each product to make, sell or hold each month.

Modeling the objective

In this case, the objective is simple. You model the profit and the cost, and the objective is to maximize the profit minus the cost:

```
dexpr float totalProfit =
  sum (p in Products, m in Months) p.profit * Sell[p][m];
dexpr float totalInventoryCost =
  sum (p in Products, m in Months) Inventory.costPerUnit * Hold[p][m];

maximize totalProfit - totalInventoryCost;
```

Modeling the constraints

There are three constraints in this model. You want to find a solution that maximizes the objective: the profit minus the cost. However, you also want to make sure that the solution does not violate any of these constraints.

The first constraint reflects the fact that certain machines will be down for maintenance and unavailable for processing each month. For example, in the first month, there are only three machines available for the grinding process. You want to make sure that your solution does not require more than three grinding machines.

```
  forall(m in Months, r in Processes)
    ctCapacity:
      sum(p in Products) processUsages[r][p] * Make[p][m]
           <= processCapacities[r][m] * Calendar.hoursPerMonth;
```

The second constraint reflects the limited storage capacity at your facility. You want to make sure that the amount in storage never exceeds the limit.

```
  forall(p in Products, m in Months)
    ct_inventory_balance:
    Hold[p][m-1] + Make[p][m] == Sell[p][m] + Hold[p][m];
```

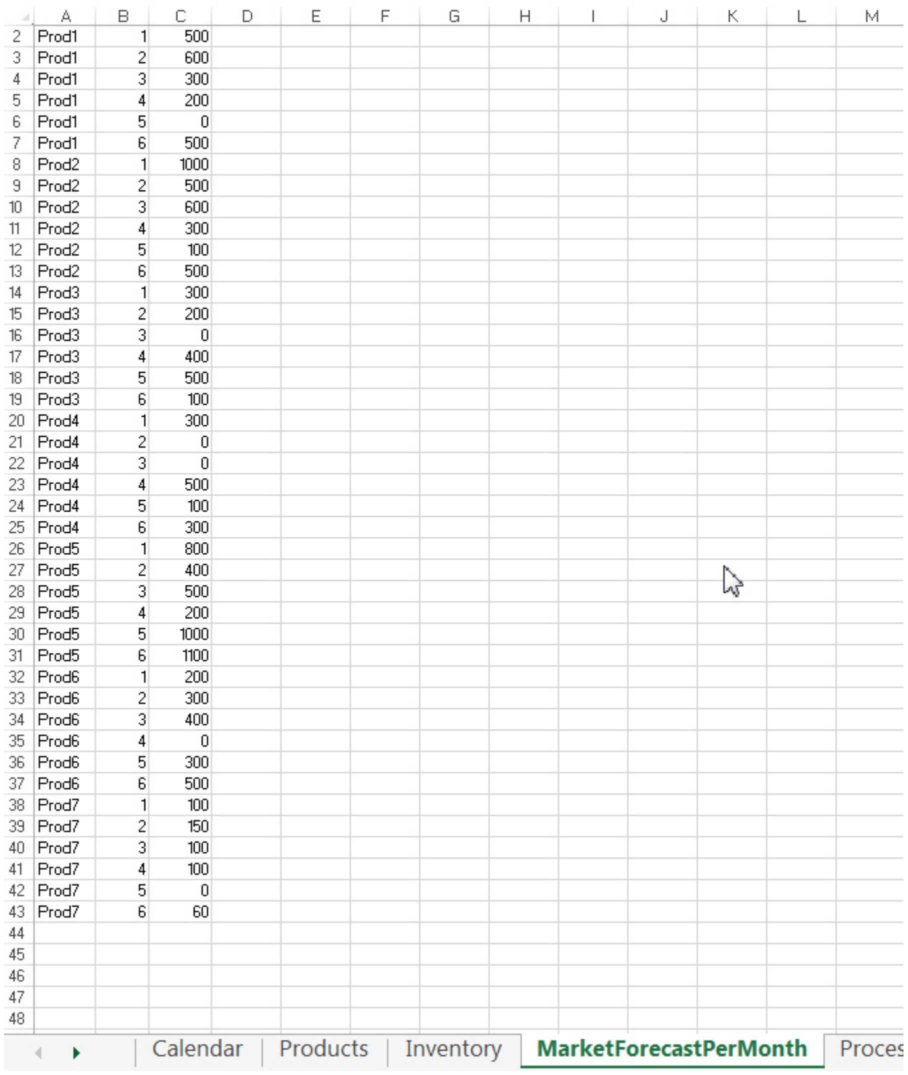
The third constraint requires that the amount of product in storage at the beginning of the period is equal to a defined starting amount `initialQuantity`. At the end of the six-month period, you state that the amount in storage is equal to a certain amount `finalQuantity`. You can vary these amounts by changing the data in the data input file.

```
  forall(p in Products) {
    ct_initial_inventory:
     Hold[p][0] == Inventory.initialQuantity;
    ct_final_inventory:
     Hold[p][NbMonths] == Inventory.finalQuantity;
  }
}
```

This model describes a problem from *Model Building in Mathematical Programming*, 3rd edition by H.P. Williams.

The data (`factoryPlanning.xlsx`)

All the data is contained in an Excel workbook. For example, the `MarketForecastPerMonth` worksheet shows that for Product 1 the forecasts are as follows; 500 units in month 1, 600 units in month 2, 300 units in month 3, 200 units in month 4, 0 in month 5 and 500 units in month 6.



	A	B	C
2	Prod1	1	500
3	Prod1	2	600
4	Prod1	3	300
5	Prod1	4	200
6	Prod1	5	0
7	Prod1	6	500
8	Prod2	1	1000
9	Prod2	2	500
10	Prod2	3	600
11	Prod2	4	300
12	Prod2	5	100
13	Prod2	6	500
14	Prod3	1	300
15	Prod3	2	200
16	Prod3	3	0
17	Prod3	4	400
18	Prod3	5	500
19	Prod3	6	100
20	Prod4	1	300
21	Prod4	2	0
22	Prod4	3	0
23	Prod4	4	500
24	Prod4	5	100
25	Prod4	6	300
26	Prod5	1	800
27	Prod5	2	400
28	Prod5	3	500
29	Prod5	4	200
30	Prod5	5	1000
31	Prod5	6	1100
32	Prod6	1	200
33	Prod6	2	300
34	Prod6	3	400
35	Prod6	4	0
36	Prod6	5	300
37	Prod6	6	500
38	Prod7	1	100
39	Prod7	2	150
40	Prod7	3	100
41	Prod7	4	100
42	Prod7	5	0
43	Prod7	6	60

The advantage of separating the model that uses the data from the actual data is that it is simple to make changes to the problem by changing the data file. For example, marketing initially forecasts that demand will be high for snow shovels in January (month 1). However, an unexpectedly warm winter leads marketing to change their forecast and lower the expected demand for snow shovels in January. To accommodate this change you would only have to change a value for `MarketProd` in the Excel file. You could then find a solution for the revised problem without having to make any changes to the model itself.

The solution

The solution appears in a simple table format. This table tells you how many units of each product to sell, hold, and make each month in order to make the most profitable use of your factory.

For example, the first line of the solution tells you how much of Product 1 to sell in each of the six months:

```
Sell = [[500 600 100 200 0 500]
```

For Product 1, you must sell 500 units in January, 600 units in February, 100 units in March, 200 units in April, 0 units in May, and 500 units in June. The next line tells you how much of Product 2 to sell in each month, and so on.

Similar tables exist to tell you how many units of each product to hold and to make in each month.

Customization

You can customize this model to help you make decisions about resource optimization problems that you may face at your factory. Here are some examples of ways to customize this model:

- Changing the number and type of machines or products. You can also change the number of months in the plan and the profit per product, as well as the marketing limitations of each product. All of this information is very easy to customize by simply changing the Excel file.
- Choosing the best maintenance schedule for the machinery. In this model, we have stated that maintenance for certain machines will be performed during certain months. If you have the freedom to choose when maintenance occurs, you can modify this model to choose the best months to perform maintenance on the various machines.
- This model could also be modified for a sawmill, a food production facility, or a chemical plant.