Warehouse location

Warehouse location

Where is the best location to build a warehouse so that it can supply its existing stores at a minimal cost?

A retail company is considering a number of locations for building warehouses to supply existing stores. Each possible warehouse has a fixed maintenance cost and a maximum capacity specifying how many stores it can support. In addition, each store can be supplied by only one warehouse and the supply cost to the store differs according to the warehouse selected.

The problem consists of choosing which warehouses to build and which of them should supply the various stores while minimizing the total cost, that is, the sum of the fixed and supply costs.

Warehouse location is a typical discrete optimization problem that uses Integer Programming (IP). Integer Programming is the class of problems defined as the optimization of a linear function, subject to linear constraints over integer variables. IP programs are generally more difficult to solve than linear programs.

The data

Consider an example with five warehouses and ten stores. The fixed costs for the warehouses are all identical and equal to 30. The instance data for the problem, shown in the table below, reflects the transportation costs (defined by the tuple `supplyCosts`) and the capacity constraints (defined by the tuple `warehouses`) in the data files. For DropSolve, all data must be defined in the form of tables (which in OPL implies using sets of tuples to define these tables). The `warehouse_data.mod` file declares the tuples `TWarehouse`, `TSupplyCost` and `TPlan`. These tuples form the rows of the tables `plan`, `warehouses`, and `supplyCosts`.

The tuple `TPlan` only contains a single element (the number of stores) and the table `plan` therefore consists of a single cell. The `warehouse_cloud.dat` file provides the values for the costs and capacities in the form of tables based on these declared tuples (rows).

Warehouses	**Bonn**	**Bordeaux**	**London**	**Paris**	**Rome**
Capacity	1	4	2	1	3
Store 1	20	24	11	25	30
Store 2	28	27	82	83	74
Store 3	74	97	71	96	70
Store 4	2	55	73	69	61
Store 5	46	96	59	83	4
Store 6	42	22	29	67	59
Store 7	1	5	73	59	56
Store 8	10	73	13	43	96
Store 9	93	35	63	85	46
Store 10	47	65	55	71	95

The model (`warehouse_cloud.mod`)

The variables

To represent the warehouse location problem as an integer program, the model, `warehouse_cloud.mod`, uses a 0-1 Boolean variable for each combination of warehouse and store to represent whether or not a warehouse supplies a store. In addition, the model associates a variable with each warehouse to indicate whether or not the warehouse is open.

```
dvar boolean Open[warehouses];
dvar boolean Supply[stores][warehouses];
```

represent which warehouses supply the stores, that is, `Supply[s][w]` is 1 if warehouse `w` supplies store `s`, and zero otherwise.

The constraints

- each store must be supplied by a warehouse
- each store can be supplied by only an open warehouse
- each warehouse cannot deliver more stores than its allowed capacity

The constraint:

```
  forall( s in Stores )
    ctEachStoreHasOneWarehouse:
      sum( w in  Warehouses )
        Supply[s][w] == 1;
```

states that a store must be supplied by exactly one warehouse.

To express that a warehouse can supply a store only when the warehouse is open an inequality is used as follows:

```
  forall( w in warehouses, s in stores )
    ctUseOpenWarehouses:
      Supply[s][w] <= Open[w];
```

which ensures that when warehouse `w` is not open, it cannot supply store `s`. This follows from the fact that `Open[w] == 0` implies `Supply[w][s] == 0`.

This constraint is combined with the capacity constraint for the warehouses:

```
  forall( w in warehouses )
    ctMaxUseOfWarehouse:
      sum( s in stores )Supply[s][w] <= w.capacity;
```

The objective function

After the variables are declared, the objective function:

```
minimize
  totalOpeningCost + totalSupplyCost;
```

expresses the goal that the model minimizes the fixed cost of the selected warehouse and the variable costs of supplying stores. These costs are defined as follows:

```
// expression
dexpr float totalOpeningCost = sum( w in warehouses ) w.fixedCost * Open[w];
dexpr float totalSupplyCost  = sum( w in warehouses , s in stores, k in supplyCosts :
k.storeId == s && k.warehouseName == w.name ) Supply[s][w] * k.cost;
```

The result

For the instance data depicted in the table above, the model returns the following optimal solution with a total cost of 383:

```
Open = [1 1 1 0 1];
Supply =    [[0 0 0 0 1]
             [0 1 0 0 0]
             [0 0 0 0 1]
             [1 0 0 0 0]
             [0 0 0 0 1]
             [0 1 0 0 0]
             [0 1 0 0 0]
             [0 0 1 0 0]
             [0 1 0 0 0]
             [0 0 1 0 0]];
```

This shows that warehouse number 4, in Paris, is not open and therefore it cannot supply any of the 10 stores. The proposed solution for the open warehouses is:

- The Bonn warehouse can supply store number 4
- The Bordeaux warehouse can supply stores 2, 6, 7, 9
- The London warehouse can supply stores 8, 10
- The Rome warehouse can supply stores 1, 3, 5