ORIE 4820: Spreadsheet-Based Modeling and Data Analysis
Box Packing Lab Exercise
Spring 2013

During this lab exercise, we will build a decision tool to solve a simple box packing problem. The primary purpose is to give you practice ***using VBA to implement algorithms within the Excel environment***.

The template file for this exercise is ***Box-Packing.xlsm***. Download a copy of the file from the course Blackboard site and *save it on your computer*. There is one worksheet in the workbook and two Sections to this exercise:

- **Section 1** focuses on developing a model on the *Box Packing* worksheet to facilitate the automation of an algorithm that packs items into boxes
- **Section 2** focuses on implementing the algorithm in a modular fashion using VBA

Topics and tools we will cover:

- ***Building structured decision models*** in Excel (includes review of numerous ***lookup and reference*** functions, ***conditional logic*** functions, ***array functions***, ***conditional formatting***, and ***error checking*** protocols)
- ***Writing, modifying, and running code to automate an iterative algorithm*** using the Visual Basic Editor (VBE)
  - ***Defining, initializing, and updating variables*** within VBA subroutines
  - ***Using logical, case, and looping constructs*** within VBA subroutines

If you have problems or questions at any point during the exercise, please raise your hand.

Background:

The ***Box Packing Problem*** we consider in this exercise is one that surfaces in numerous business settings, with applications in manufacturing, transportation and logistics, staff scheduling, and many other areas. The statement of the problem is straightforward:

Given a set of items having different weights and a set of boxes with weight capacities, pack all items into boxes using as few boxes as possible.

We will build a decision model that can handle a problem with 20 items and 10 boxes, but the model can easily be extended to handle larger problems. Our model also allows the weight capacities of the boxes to differ.

The picture below shows the elements of the problem, as well as the packing matrix that will be used to capture the assignment of items to boxes:

Packing Assignment Matrix			Boxes										
	Weight (lbs)	Item	A	B	C	D	E	F	G	H	I	J	Item Packed
Items to be Packed	1.2	1											0
	2.3	2											0
	3.8	3											0
	3.3	4											0
	1.1	5											0
	3.8	6											0
	2.3	7											0
	0.2	8											0
	0.8	9											0
	3.2	10											0
	3.1	11											0
	3.5	12											0
	0.6	13											0
	2.3	14											0
	0.9	15											0
	2.7	16											0
	3.6	17											0
	3.2	18											0
	4.3	19											0
	3.2	20											0
	Box Used												
	Current Weight of Box												
	Maximum Weight for Box		10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	

Before we begin, there are several things to note about the *Box Packing* worksheet:

- This worksheet contains several ***named cell ranges*** that will facilitate modeling and implementation. Take a few moments to review all of the defined names in the workbook (Formulas->Defined Names->Name Manager).
- ***Conditional formatting*** has been used in several places to indicate when exceptions exist (such as when box weight constraints are violated). Take a few moments to review all of the conditional formatting on the worksheet.
- ***Text concatenation*** has been used in a few places to create dynamic labels. To see this, press Ctrl-` to toggle between regular and formula views of the worksheet.

Section 1: Completing the Box Packing Model

The model is designed with the expectation that ***every row of the packing matrix will be populated with exactly one "1" entry***. The column of the "1" entry indicates which box the item is being packed into. There can be multiple "1"s in a column, since each box can hold up to its capacity limit in weight.

The goal of the exercise is to pack all items using as few boxes as possible, without exceeding any of the box weight capacities. Therefore, as we start to assign items to boxes, our model should explicitly indicate ***which boxes are currently being used***, and ***how much weight each box is currently carrying***:

(1) ***Populate cell E31*** with a "1" if there are any items in the box A, and a "0" otherwise. Copy the formula across to cells F31:N31.

(2) ***Populate cell E32*** with the current total weight of items in box A. Copy the formula across to cells F32:N32.

(3) ***Find the best solution that you can manually*** using the tool as it currently is. As you go through the assignment process, make a mental note of the process you are using to determine whether or not to assign an item to a box. Note that you can use the "Reset Matrix" button to clear all matrix entries.

Now that you have practiced making manual assignments on a small problem, it is easy to see how complicated problems of this type can become. In most cases, it can be helpful (and far more efficient) to design an algorithm that uses simple rules to make assignments. The user then has the option of using this solution as it is, or using it as a starting point for finding a better one.

The box packing algorithm that we will implement is straightforward. Boxes will be packed one-by-one, in the order listed in the packing matrix, until all items are packed (or until we run out of boxes). For each box, we will do the following:

- *Find the heaviest remaining item that will fit into the current box, and assign it to the box.*
- *If no remaining item fits, then the box is "Full" – move on to the next box.*

Although we will use VBA code to automate the actual *assignment* of the item (i.e., we will automate placing a "1" in the appropriate packing matrix cell), the *identification* of the next item to be placed in a particular box is much more easily accomplished directly on our worksheet. Hence, we will use VBA ***in conjunction with*** resident spreadsheet logic to implement the box packing algorithm. Here is what we need to do to identify the next item to assign:

(4) ***Populate cell K4 with the current weight of the Selected_Box*** (K2).

"=IFERROR(INDEX($E$32:$N$32,MATCH(Selected_Box,Boxes,0)),"")"

Note that **index** and **match** are used to look up the appropriate entry from row 32. The **iferror** ensures that if no box is selected in cell K2, the user does not see an error displayed.

(5) ***Populate cell K5 with the remaining weight that can be assigned to the Selected_Box*** (K2). This is simply the difference between the box's weight capacity (from row 33) and the current weight assigned.

(6) ***Populate cell K6 with the next item to add to the Selected_Box*** (K2):

a. It is best to do this in several steps. First, using embedded ***array functions***, determine the weight of the heaviest remaining item that will fit in the box:

"=MAX(IF((Items_Packed=0)*(Weights<=$K$5)=1,Weights,""))"

b. Next, determine the item that corresponds to this weight:

"=INDEX(Items,MATCH([*from part a*],IF((Items_Packed=0)*(Weights<=$K$5)=1, Weights,""),0))"

c. Wrap around the current formula an **iferror** statement that will mark the cell as "Full" if the **index** function is unsuccessful:

"=IFERROR([*from part b*],"Full")"

d. Finally, wrap around the current formula an **if** statement that will blank out the cell if Selected_Box is blank or if all items have been assigned:

"=IF(OR(Selected_Box="",Num_Remaining_Items<=0),"",[*from part c*])"

Note that conditional formatting has been used on the packing matrix cells to highlight the indicated box/item combination in cells K2 and K6.

Section 2: Automating the Box Packing Algorithm

In this part of the exercise, ***you will complete three VBA subroutines*** that collectively automate the box packing algorithm described in the previous section.

First, press Alt-F11 to open the Visual Basic Editor (VBE). In the left-hand side project window, select Modules within the project hierarchy and double-click on ***PackingModule*** to open it. This module contains code for the four VBA subroutines that you will be working on. The first one, `Reset_Matrix`, has already been completed. You can begin with the `Add_Item_To_Box` subroutine.

(1) `Reset_Matrix`: This subroutine consists of a single line of code that simply clears all entries of the "Packing_Matrix" range (as if the range had been highlighted and Delete key had been pressed).

(2) `Add_Item_To_Box`: This subroutine's purpose is to *add the item* specified by the "Next_Item" cell on the worksheet to the box specified by the "Selected_Box" cell. Therefore, the code needs to determine the row and column indices of the "Packing_Matrix" that correspond to the item and box in question, and it needs to update the entry. (This subroutine will be called iteratively by `Pack_Box`.) Using the skeleton code in `Add_Item_To_Box` as a starting point, it remains for you to:

- Determine the <u>indices</u> of the "Packing_Matrix" cell range that correspond to the item to be packed. Note that you should not simply use the item number as the row index because the items might not be listed in the table in sequential order (e.g., it is possible that item 4 might be in position 10 of the table). <u>Hint</u>: Use the `Application.WorksheetFunction` prefix to make use of Excel's **match** function.
- Set the corresponding "Packing_Matrix" cell value to 1.

(3) `Pack_Box`: This subroutine's purpose is to pack a *single box* with items. It will do so by iteratively calling `Add_Item_To_Box` as long as none of the termination conditions have been met. Using the skeleton code in `Pack_Box` as a starting point, it remains for you to:

- Implement a **Do While** loop that calls the `Add_Item_To_Box` subroutine until we reach a point where either the box is full (i.e., no remaining items will fit) or there are no more items left to pack.

(4) `Run_Packing_Algorithm`: This subroutine's purpose is to pack boxes until all items are packed (or we run out of boxes). It will do so by iteratively calling `Pack_Box`, in the order specified by the "Boxes" named range. It continues as long as none of the termination conditions have been met. Using the skeleton code in `Run_Packing_Algorithm` as a starting point, it remains for you to:

- Implement an **If Then** statement that calls the `Pack_Box` subroutine provided that there is an item that can be packed into the box in question.

Exercise:

Write down a formulation of the Box Packing Problem as an integer linear program. Be sure to ***fully define*** all decision variables and problem parameters.

```
Sub Reset_Matrix()
'Deletes all entries in the packing matrix

    Range("Packing_Matrix").ClearContents

End Sub
```

```
Sub Add_Item_To_Box()
'Assigns "Next_Item" to the currently "Selected_Box" by setting the corresponding Packing_Matrix entry to 1.
'The field "Next_Item" holds the heaviest remaining item that will fit into the current box.
'

Dim next_item_index As Integer
Dim box_index As Integer

'Set the appropriate matrix entry to 1

    If Range("Next_Item").Value = "" Then
        MsgBox ("No item")
    Else
        If Range("Next_Item").Value = "Full" Then
           MsgBox ("Box is Full")
        Else
           next_item_index = Application.WorksheetFunction.Match(Range("Next_Item").Value, Range("Items"), 0)
           box_index = Application.WorksheetFunction.Match(Range("Selected_Box").Value, Range("Boxes"), 0)
           Range("Packing_Matrix").Cells(next_item_index, box_index) = 1
        End If
    End If

End Sub
```

```
Sub Pack_Box()

'Adds items to "Selected_Box" until the box is full or no items remain to pack

    If Range("Next_Item").Value = "" Then
        MsgBox ("No item")
    Else
        If Range("Next_Item").Value = "Full" Then
            MsgBox ("Box is Full")
        Else
            Do While (Range("Next_Item").Value <> "" And Range("Next_Item").Value <> "Full")
                Call Add_Item_To_Box
            Loop
        End If
    End If

End Sub
```

```
Sub Run_Packing_Algorithm()
'Runs the packing algorithm in "Boxes" order.
'The field "Next_Item" holds the heaviest remaining item that will fit into the current box.
'
Dim hold_box As String

    'If all items are packed, then exit the subroutine

    If Range("Num_Remaining_Items").Value <= 0 Then
        MsgBox ("No remaining items to pack")
        Exit Sub
    End If

    Application.ScreenUpdating = False

    hold_box = Range("Selected_Box").Value

    'Pack each box in "Boxes" order

    For Each cell In Range("Boxes")
       Range("Selected_Box").Value = cell.Value

        'Must have an item to put in the box, else skip the box
        If Range("Next_Item").Value <> "" And Range("Next_Item").Value <> "Full" Then
            Call Pack_Box
        End If

        If Range("Num_Remaining_Items").Value <= 0 Then
            Exit For
        End If
    Next

    Range("Selected_Box").Value = hold_box

End Sub
```