Decision Support systems

**PART A: Database and Linear Programming**

**Objectives**:

* How to use database to supply required inputs to be used in an LP Problem.
* How to use a form to supply inputs interactively to an LP problem.
* How to optimize a given LP problem by the click of a button.

**0. Preparation:**

Read and carry out the following practice problem first.

GrillRite Product Mix Problem

**Setting:** GrillRite of Tucson, Arizona, is one of the biggest manufacturers of grills in the country. Dawn Li is a new product manager for GrillRite’s southwest sales region. Her job is to manage the orders for stores in Arizona, New Mexico, and Nevada. It is spring and Dawn has to prepare an order to be ready for the heavy early summer sales.

GrillRite makes four models of grills: *Standard*, *Deluxe*, *Dual*, and *Extended*. Dawn has to decide how many of each model to order for her stores. There are many factors involved in making up her order. She needs to order enough of each model to satisfy store demand. Each grill has to be assembled at a regional plant, so Dawn can only order models that require parts that are currently in stock. She can only order what the plant can produce. Finally, she has to order a mix of models that will *maximize* GrillRite’s *net income* for the coming months. Some of the models have a higher profit margin than others.

Trying to balance all of these factors is a great challenge. Dawn has asked your help in putting together an order that will satisfy her customers, the plant managers, and her bosses. You meet with Dawn to start work on her grill order. Your job will be to find the purchase order that maximizes GrillRite’s profits while meeting consumer demand and satisfying the limitations of the production process. Dawn has put together a workbook containing all of the factors you need for your work. You decide to examine this work book to get a better grasp on the issues confronting her.

Grill.xls

Open the **Grill.xls** file and study it carefully. There are two sections in the **Order** worksheet. The left section, shown below, contains an income analysis of the four grill models: Standard, Deluxe, Dual, and Extended. GrillRite has fixed expenses of $30,000 that Dawn will have to account for when she determines the profitability of her order. Dawn has already received requests for grills from various stores in her region as follows:

* Standard >= 125 units
* Deluxe >= 100 units
* Dual >= 100 units
* Extended >= 75 units

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decision Variables (Xi); note all values are zeros (or Null) before **optimization**.  Variable Expenses for the Grill Order  The Region’s Profit on the Order  Revenue generated by an order of Grills | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **GrillRite** |  |  |  |  | | ***Product Mix Analysis*** |  |  |  |  | |  |  |  |  |  | | **Income Analysis** |  |  |  |  | | Variable Expenses | Standard | Deluxe | Dual | Extended | | Models Ordered | 0 | 0 | 0 | 0 | | Cost of Parts | $96.50 | $105.15 | $155.20 | $178.00 | | Cost of Assembly | $10.00 | $10.00 | $20.00 | $25.00 | | Total Model Cost | $106.50 | $115.15 | $175.20 | $203.00 | | **Total Variable Expenses** | **$0.00** | |  |  | |  |  |  |  |  | | Revenue | Standard | Deluxe | Dual | Extended | | Models Sold | 0 | 0 | 0 | 0 | | Price per Model | $155.00 | $175.00 | $245.00 | $315.00 | | **Total Revenue** | **$0.00** | |  |  | |  |  |  |  |  | | **Summary** |  |  |  |  | | Total Revenue | $0.00 | |  |  | | Total Variable Expenses | $0.00 | |  |  | | Total Fixed Expenses | $30,000.00  Optimized max Net Income will display here | |  |  | | **Net Income** | **-$30,000.00** | |  |  | |

When Dawn orders a grill, it has to be assembled at a regional plant. On the right side of the **Order** sheet, Dawn has inserted the parts inventory data from the plant (see image below).

Number of units **used** up in production by orders during the optimization process will display here. Note that “**used**” must be <= “**available**”.

|  |
| --- |
| Cost per unit  Number of parts required to satisfy Dawn’s order and the number of parts remaining. Note all orders 0 now before optimization.  Parts used to assemble grills |

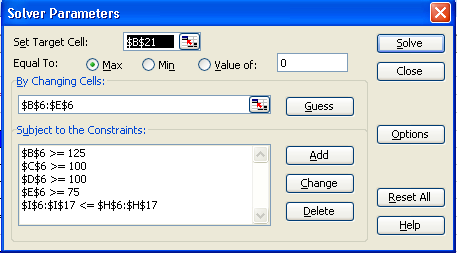
|  |
| --- |
| Number of parts needed in assembling each grill order. |

**Check Formulas in Each Relevant Cell**

Check all the requisite formulas in appropriate cells. *Please check before you continue*. If the formulas are missing or incorrect, please correct them or your **Solver** will not work properly.

**LP Formulation**

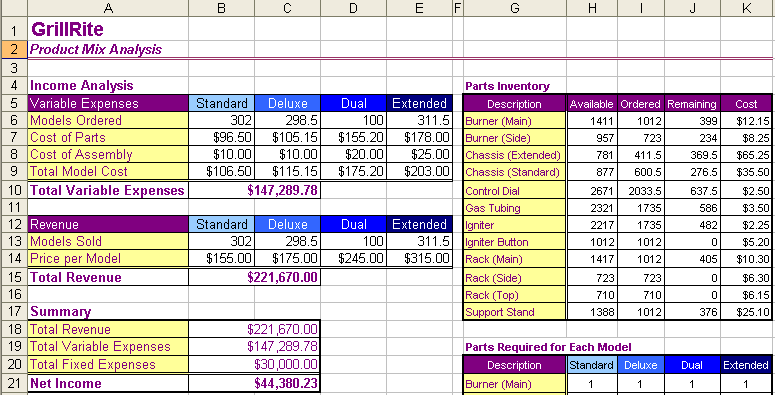
For correct formulation, see the Solver image below.



Note that you *do not need to specify the non-negativity constraint*. Do you know why?

**Run Solver**

Run the Solver with the above formulation and verify that you get the following results shown in the image below. If you do not get these results, please do not proceed further. Check all the appropriate formulas and the Solver formulation one more time.



Values after optimization by Solver

**1. Hands-On Problem:**

Read and carry out the following problem following the given instructions..

LP Problem Interfaces with Database

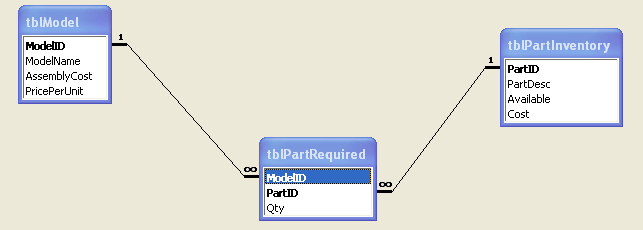
The LP problem above stores all the needed data in the spreadsheet itself. *But this is not realistic*. Data is best stored and managed in a database. In this section, we will enhance the above problem with **Access** database access. All the data for the LP Problem will come from an **Access** database (except for some data that will be input via a form as the program runs; for example, the number of grills ordered for each model).

**1. Check the Excel File**

Open the **SolverGrill.xls** (note the name change) file and study it carefully. It has three worksheets: **Documentation**, **GUI**, and **Order**. Take a look at the **Order** worksheet. You will see just a “shell” of the Solver set up and all the relevant data is missing. To be more realistic, these data have been transferred to an **Access** database. Even though the worksheet **Order** is empty, it still has all the requisite *formulas* in appropriate cells. *Please check before you continue*. If the formulas are missing or incorrect, please correct them or your Solver will not work properly. NOTE: *I am using range* ***B30:E30*** *to store the number of grill orders for each grill model.*

**2. Create an Access Database**

* For your convenience, an **Access** database for all the needed data for the Solver scenario has been created and saved under the file name of **Solver.mdb**. Open this file and try to familiarize yourself with the tables, their structure and the data stored in them. Pay careful attention to field names, primary keys, foreign keys, and how tables are related to one another through a common field. Also, check the relationship diagram.



* When we want to access particular data from the database, we use a *query*. Queries can be designed and run *graphically* as well as via a programming language called *SQL* (Structured Query Language). Writing a query with SQL is the preferred choice to access required data, when you want this data retrieval to happen *automatically* as your program is running.
* To *practice* the following 2 *examples,* you must be in SQL window. To get there:

**ACCESS 2003**

* Queries 🡪 Design 🡪 Close 🡪 View 🡪 SQL View.

**ACCESS 2007**

* Click **Create** tab 🡪 **Query Design** option in **Other** group 🡪 **Close** in the **Show Table** dialog box 🡪 and you will see a blank query screen. Click **SQL View** in **Results** group 🡪 you will see the section in which you can type in and run your SQL codes.

**SQL Codes**

* Type the SQL statements in this window and press the run button that looks like a red “exclamation mark!” Type in and run the following two examples. Check the outcomes. The outcome that you get when you run a SQL program is called a “**Record Set**”. The nature of this record set could be a single value ( 1 x 1 table), a table with many rows and one column only (n x 1 table), or a table with many rows and many columns (m x n table).

**a). Single Table Query Example**:

**SELECT AssemblyCost**

**FROM tblModel**

**WHERE ModelID = “ST”;**

**b).** **Two Table Query Example**:

When you want to extract data that exist separately in *two* directly “related” tables, you must *join* the table first before you can do it.

**SELECT Qty**

**FROM tblModel *INNER JOIN* tblPartRequired**

***ON* (tblModel.ModelID = tblPartRequired.ModelID);**

**3. The GUI Worksheet**

This worksheet is already designed for you. Read the instructions and do the needful.

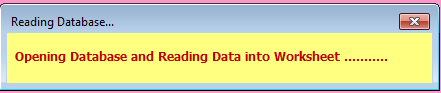
**a). Buttons:** The ***GUI*** Worksheet contains a couple of **buttons** that require some coding. Write the following codes in appropriate places.

* + When the **START** button is clicked, it will display the user form named frmGrill. This form is already designed for you. The purpose of this form is to enter the data required by the application “on the fly” (i.e., data that is not practical to store in the database in advance).
  + When the **QUIT** button is clicked, it will reinitialize some of the ranges on the Order sheet, thus deleting the values from the previous optimization run. Also, it will use the VBA code “Application.Quit” to quit Excel

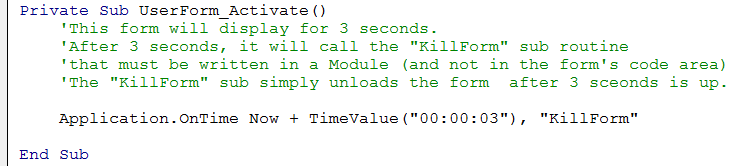
|  |
| --- |
| **Private Sub cmdQuit\_Click()**  ***'Re-initialize relevant values to zeroes on Order worksheet***  Sheets("Order").Select    Sheets("Order").Range("B6:E6") = 0  Consider writing these lines in a library subroutine so that you can call and use this library subroutine not only in this **cmdQuit\_Click( )** routine, but also from other places if required.  Sheets("Order").Range("B8:E8") = 0  Sheets("Order").Range("B6:E6") = 0  Sheets("Order").Range("B14:E14") = 0  Sheets("Order").Range("H6:H17") = 0    Sheets("Order").Range("K6:K17") = 0  Sheets("Order").Range("H21:K32") = 0  Sheets("Order").Range("B20") = 0    Sheets("GUI").Select  ActiveWorkbook.Save    *'Quit Excel application*  Application.Quit    **End Sub** |

**b). The Form *frmMsg***: This is a second form (see image below) that will display on the **GUI** worksheet when the CONTINUE button on the user form **frmGrill** has been clicked. This form is already designed for you.

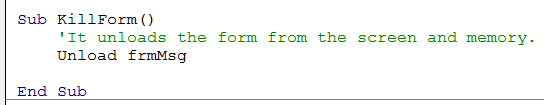
The purpose of this small form is to act as a **splash screen** that informs the user the program is opening the database and transferring the data from the database into the **Order** worksheet. Actions like transferring data from database to Excel can take some time, and it is a good etiquette to inform the user what is going on behind the scenes. This form will display for about 3 seconds only.



Write the following code in the **frmMsg** code area for the “**Activate**” event.



Write the following code in a module:



**c). The Form *frmGrill*:** The form will display on the **GUI** worksheet. The purpose of the form is to capture required *interactive* data from the user as the program is running. This type of data *cannot* be conveniently stored in the database such as: *Order quantities* currently received for each model and the current *total fixed cost*.



The form should validate the data (not Null, not negative) before it ships it to the **Order** Worksheet. For example, quantities cannot be Null or negative. This validation activity happens when the **Continue** button is clicked by the user.

After the validation is successful,

* it will show the splash screen **frmMsg**
* it will write each model order quantity and total fixed cost captured from the form into ORDER worksheet in the correct cells.
* It will connect and read data from the database and fill the ORDER worksheet in the correct cells. In particular, you will be writing the following separate SQL statements to retrieve required data from the database and populate the **Order** worksheet cells:
  + set SQL statement to retrieve *Cost of Assembly* in *tblModel*
  + set SQL statement to retrieve *Model Price per unit* in *tblModel*
  + set SQL statement to retrieve *Available parts* in *tblPartInventory*
  + set SQL statement to retrieve *Cost of parts* in *tblPartInventory*
  + set SQL statement to retrieve *Parts Required* for "ST" model
  + set SQL statement to retrieve *Parts Required* for "DX" model
  + set SQL statement to retrieve *Parts Required* for "DL" model
  + set SQL statement to retrieve *Parts Required* for "EX" model

Do your own coding for the CONTINUE and the CANCEL buttons’ click events.

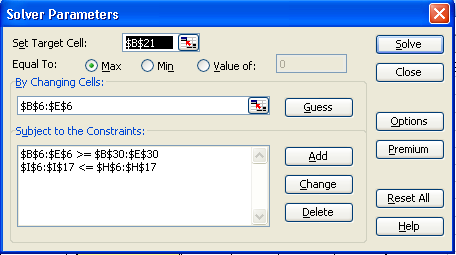
**4. The ORDER Worksheet**

This worksheet has two buttons and BOTH them are **ActiveX** controls (not the Form Controls buttons that we have been using. Google and read the benefits of ActiveX Controls).

If all the components on the **GUI** sheet worked properly, you should see all the required data being pulled correctly into the **ORDER** sheet from the Access database. Please check the data carefully. You might want to load **Grill.xls** that you did at the beginning and counter check the cell values against the values in your current ORDER sheet (adjust for the order quantities for each model). *Please do not continue any further if the values are not matching*. Do the needful to correct the discrepancies.

This worksheet has a *button* labeled **Call Solver** that calls and runs the **Solver** model. The Solver model should automatically pull in required data from the appropriate cells, write needed formulation and constraints, run and give the final “optimal” answer. To write code to do all this is pretty rough. So, we will use a trick. We will *record a macro* and let the Macro do the code-writing for us. We will then copy and paste the code in the Click event of the button procedure.

**a). Record the Macro:** Go through the process of running Solver and record the corresponding Macro under the name of **RunSolver**. Use the following Solver parameters:



**a). Enhance the Recorded Macro:** The recorded Macro can be further improved to give it a user-friendly feel. Locate the **SolverSolve** line in the recorded Macro (it should be near the bottom) and modify it as follows. Note that I added the “True” argument so that it will suppress the display of **Solver Results** window that always pops up and it will confuse the user.

|  |
| --- |
| **SolverSolve UserFinish:=True** *'this will suppress the Solver Results dialog box being displayed* |

Next, add the codes shown between the two lines of asterisks under the **SolverSolve** line.

|  |
| --- |
| '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SolverFinish KeepFinal:=1 *‘keep the solver’s final results in the cells*    *‘Check if Solver successfully completed; returned codes (0 or 1 or 2) mean a successful optimization*  If SolverSolve(UserFinish:=True) <= 2 Then    MsgBox "Solver has found an optimum product mix solution." & Chr(10) & Chr(10) & "\* Optimum Quantities are in Cells ” & \_  “B6 ­through E6" & Chr(10) & Chr(10) & "\* Max Net Income in “ & \_ “Cell B21" & Chr(10) & Chr(10) & "\* All Constraints are Satisfied."  Else  MsgBox "The Solver couldn't find a solution. Try again with a different set of inputs! "    End If  '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

Next, add the following 2 lines at the top of the macro body.

The first function **SolverReset** is very important because it re-sets the target cells, changing cells and constraints before a new run so that the old parameters do not interfere with the new ones.

|  |
| --- |
| SolverReset *'resets the solver and deletes any leftover constraints from previous runs*  Application.ScreenUpdating = False |

Attach the following Macro to the **Call Solver** button’s click sub procedure on the ORDER Worksheet. Test the **Call Solver** button.

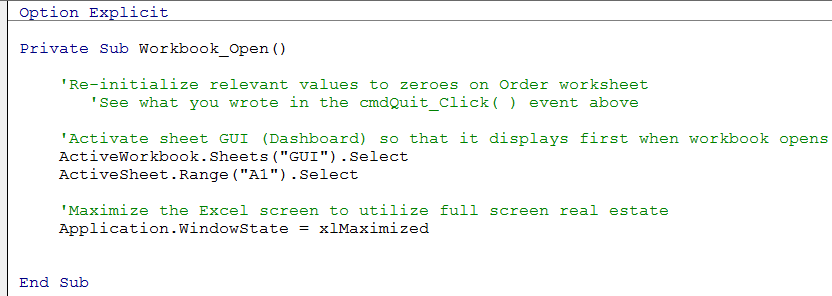
|  |
| --- |
| Private Sub cmdSolver\_Click()  Call RunSolver    End Sub |

Ideally, in an actual DSS, the **Order** worksheet should be hidden from the end-user, and only the final optimal results should be displayed to him/her on a separate user form. But in this assignment, we will display the **Order** worksheet for the developer’s verification purposes.

The **DASHBOARD** button will send the user back to the **GUI** worksheet.

**4. Workbook.Open Event**

There are certain actions we may need to undertake when an Excel application workbook is opened for the first time. Therefore, do not forget to write codes for this event. In the VBE, in the left Project Explorer pane, look for “ThisWorkbook”. Right-click on it and select **View Code**. This is the place where you write codes for the Workbook.Open event.



.

**PART B: Pivot Table, Slicers, and Dashboard**

Before you start, please watch a short but very helpful video at the following link:

***How to make an Excel Pivot Table SALES DASHBOARD***

<http://www.youtube.com/watch?v=v-0Z4qGS0xk>

After watching the video, you are now ready for an adventure.

Open the **SalesData** worksheet in the **SolverGrill** workbook that includes sales data for an imaginary stationery company, and each row represents an order. It shows:

-- when the order was placed,

-- which region in which the sale was made

-- the sales representative's name

-- the item sold

-- the number of units sold

-- the cost of a single unit

-- the total cost of the order

Using your own imagination and creativity, create a nice and helpful DASBOARD for the decision maker for the above business. This is an open-ended question since the requirements and options are not precisely listed. Please try your best since this experience will be very useful in your own project. All I want is a good Dashboard that can answer relevant and helpful questions about the business for the decision maker. Just follow the ideas as much as you an in the Video that you watched.