Gurobi Optimization in Excel

Lukas Sykora[1*]

[1*]Department of Information and Knowledge Engineering, Prague University of Economics and Business.

Corresponding author(s). E-mail(s): lukas.sykora@vse.cz;

Abstract

This paper aims to verify the possibility of using Excel with Gurobi Optimizer. This use is possible thanks to the third-party application called SolverStudio. SolverStudio is an Excel add-in that allows the use of Gurobi Python API. Using this tool it is possible to solve various optimization problems. The paper covers the installation of the necessary software, using SolverStudio, using Guropi Python API and optimization problem examples.

Keywords: Linear Programming, Gurobi, Excel, Mathematical Optimization, Operations Research

1 Introduction

Operations Research can be used wherever operations analysis and coordination are involved. Operations research is a tool to find the optimal solution to a problem while respecting various constraints. The basic tool of operations research is mathematical modeling. In applying operations research, the following steps can be followed [1]:

1. Definition of the problem in a real system.
2. Create a simplified literal description of the real system (essential elements and links between them and the goal of the analysis). In operations research, this simplified description of reality is called an economic model.
3. From the economic model, a mathematical model may emerge that contains everything that the economic model does.
4. Solving a mathematical model is a technical matter. In this paper, the Gurobi is used for it.

5. The last stage is the interpretation and verification of the results.

The optimization problems are solved using the mathematical optimization methodology. The methodology entails three steps [2]:

1. The formulation of a real problem as a mathematical model.
2. The development of algorithms to solve those mathematical models.
3. The use of software and hardware to run these algorithms and develop mathematical programming applications.

This paper deals only with the first and third steps. There is no ambition to develop new algorithms. Instead, it uses the practice-tested Gurobi Optimizer.

The second chapter covers the necessary tools and their installation. The third chapter describes the SolverStudio interface. The fourth chapter describes the Gurobi python API. Chapters five and six describe solutions to specific optimization problems. The last section is the conclusion.

2 Tools and Installation

The mathematical model solving tool used in this text is Gurobi Solver Studio. Gurobi Solver Studio is a commercial solver (provides an academic license for students).

Gurobi provides a wide range of functionalities [3]:

- Linear programming (LP)
- Mixed-integer linear programming (MILP)
- Quadratic programming (QP) Non-Convex
- Mixed-integer quadratic programming (MIQP) Convex and Non-Convex
- Quadratically-constrained programming (QCP) Convex and Non-Convex
- Mixed-integer quadratically-constrained programming (MIQCP) Convex and Non-Convex

This text covers linear programming only.

Using the Gurobi Optimizer in Excel is made possible by the third-party tool SolverStudio. While Gurobi Optimizer can be installed on various platforms [1], SolverStudio requires a Windows operating system [2].

First, you need to install Microsoft Excel [3] and Gurobi Optimizer. Subsequently, it is possible to install SolverStudio. All three programs have an installation wizard. Gurobi Optimizer can be accessed from SolverStudio using the Gurobi Python modelling interface. To use the Gurobi Python modelling interface you need Python [4] and the gurobipy [5] package, they both are installed together with Gurobi Optimizer.

This paper describes the functionality of the following software versions:

[1] https://www.gurobi.com/downloads/gurobi-software/
[2] https://solverstudio.org/download-install/
[3] https://www.microsoft.com/en-us/microsoft-365/excel
[4] https://www.python.org/downloads/
[5] https://pypi.org/project/gurobipy/

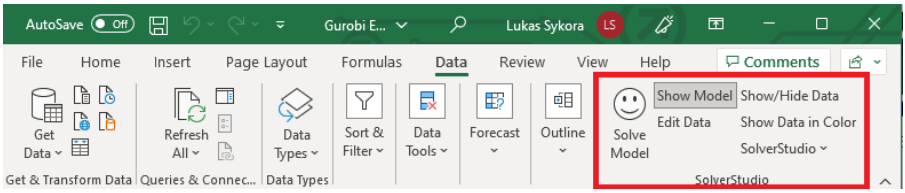

Fig. 1 SolverStudio section in Excel.

- Gurobi Optimizer 9.5.2
- SolverStudio 0.09.03
- Microsoft Excel 2207
- Python 3.7.11
- gurobipy 9.5.2

The author of this article uses the operating system Windows 10 Enterprise. If the installation is successful, Excel should display the SolverStudio section on the Data tab, see Figure 1.

3 SolverStudio Interface

The following buttons are in the SolverStudio section (see Figure 1) of Excel (in the order in which the user should use them):

1. SolverStudio - Press this button to bring up a submenu with 3 tabs:
   (a) SolverStudio Examples - Examples for Gurobi use the syntax for Python 2. The same examples modified for Python 3 can be found at https://github.com/lukassykora/gurobi
   (b) SolverStudioSetting - If the automatic Python or Gurobi searches don't work well, you can specify file paths.
   (c) About SolverStudio
2. Edit Data - Shows/hides the Data Items Editor. The Data Items Editor enables the user to map Excel cells to Python variables that can be used in models.
3. Show/Hide Data - Turns on and off the highlighting of data items, that has been selected in the Data Items Editor.
4. Show Data in Color - It is used in conjunction with Show/Hide data. If Show Data in Color, the names of the data items are colored.
5. Show Model - Shows/hides the Model Editor. Python code can be written in the Model Editor using the gurobipy package. In the Language menu it is important to select the programming language in which the model is written, that is Gurobi (Python), see Figure 2.
6. Solve Model - Solves the model in current Excel sheet.

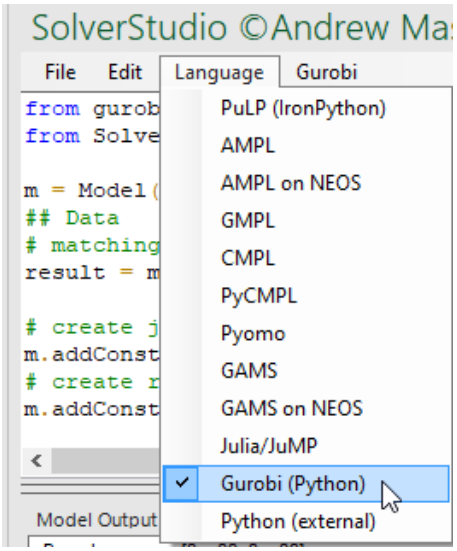

Fig. 2 Select Language.

4 Model in Gurobi

If a problem has been defined and a simplified description of it exists, it is possible to create a mathematical description of it. In this case, the Gurobi Python API is used to create this model. The examples in this paper lead to problems solvable by linear programming, meaning that all equations and inequalities used in the models are linear [4].

The following steps can be used to create a model in the Gurobi Python API:

1. `Model()` - The Gurobi Model object is initialised.
2. `Model.addVar()` or `Model.addVars()` - Add a decision variable or decision variables to the model. Decision variables represent the unknown that will be revealed by the optimization. In this method, the data type of the solution can be constrained by `vtype` parameter to `GRB.INTEGER`, `GRB.BINARY` or `GRB.CONTINUOUS`.
3. `Model.addConstr()` or `Model.addConstrs()` - Add a constraint or constraints to the model. The constraints can be equations and inequalities. The solution obtained by optimization must stay within the bounds of these constraints.
4. `Model.setObjective` - Set the model objective function equal to a linear expression. This method takes as its first parameter the objective function and, as a second parameter, information about whether the objective function should be minimized or maximized.
5. `Model.optimize()` - Optimize the model by Gurobi.

5 Transport problem

This is a very popular problem in operations research books [1, 4, 5]. The production capacity for the Plzen, Pardubice and Olomouc branches is 330, 180 and 220 products per month, respectively. These products are distributed to 4 customers. According to the contracts, the customers in Brno, Prague, Ostrava and Liberec receive 180, 250, 160 and 110 products, respectively. The transport costs are in the table for each combination of supplier and customer, see Figure 3. For example, transport from Pardubice to Prague costs 3 units per product. The goal is to minimize transportation costs.

	A	B	C	D	E	F	G
1		Transport Problem					
2							
3		Assignment					
4			Brno	Praha	Ostrava	Liberec	Capacity
5		Plzen	10	3	14	6	330
6		Pardubice	5	3	7	4	180
7		Olomouc	2	8	5	11	220
8		Request	180	250	160	110	
9							
10							
11		Result:					
12			Brno	Praha	Ostrava	Liberec	Capacity
13		Plzen					0
14		Pardubice					0
15		Olomouc					0
16		Request	0	0	0	0	
17							

Fig. 3 Transport costs.

If the input data and the output table are prepared in Excel, we can use the Data Items Editor to map them to variables in Python, see Figure 4.

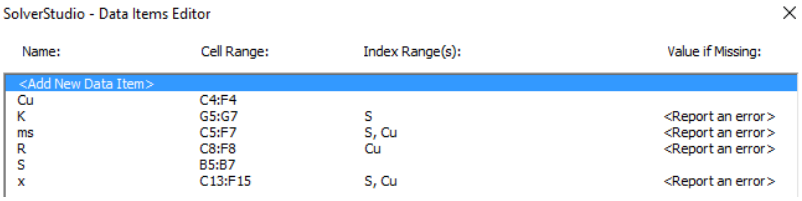

Fig. 4 Data Items Editor.

The final mapping visualisation for this example is in Figure 5.

Once the mapping is complete, Python can be used. First we need to load the necessary gurobipy and SolverStudio packages. Next, initialize `Model()`. The decision variables that should be optimized can be assigned to the model.

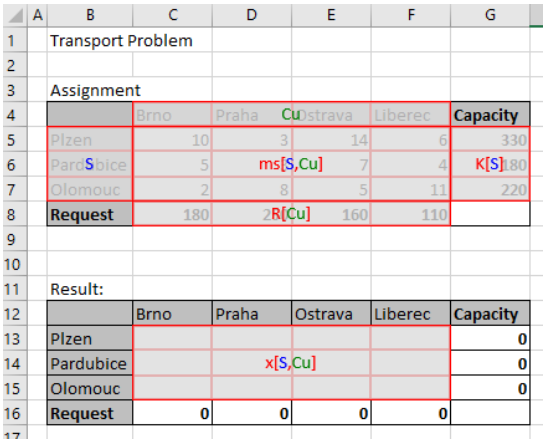

Fig. 5 Mapping.

```
from gurobipy import *
from SolverStudio import *

m = Model("transport")

# decision variables
x = m.addVars(x.keys(), vtype=GRB.INTEGER, name="assign")
```

Let S is the list of suppliers and C is the list of customers. A supplier $s \in S$ and a customer $c \in C$. K_s is a capacity for a supplier and R_c is a requested amount from a customer. Let $x_{s,c}$ is a decision variable intended for optimization, it expresses how many products will be transported from a supplier to a cistomer.

The constraint for suppliers is:

$$\sum_{c \in C} x_{s,c} \leq K_s, \qquad s \in S \tag{1}$$

The constraint for capacity is:

$$\sum_{s \in S} x_{s,c} = R_c, \qquad c \in C \tag{2}$$

These constraints can be written in Python as follows.

```
# create suppliers constraints
m.addConstrs((x.sum(s,'*') <= K[s] for s in S), 'suppliers')
# create customers constraints
m.addConstrs((x.sum('*',c) == R[c] for c in Cu), 'customers')
```

Let matching score ms is the cost for transportation of one product. The goal is to minimize the value per transport.

The objective function is:

$$Min \sum_{s \in S} \sum_{c \in C} ms_{s,c} x_{s,c}, \tag{3}$$

The objective function in Python:

```
# The objective is to minimize total matching score of the assignments
m.setObjective(x.prod(ms), GRB.MINIMIZE)
```

The mathematical formulation of the problem is completed so the optimization can be run.

```
# Find the optimal solution
m.optimize()
```

Finally, we can output the optimal results and the optimal value of the objective function to the console. At the same time, the numerical value must be assigned to the excel table for the resulting decisions.

```
def print_solution(variables1, variables2, result, m):
  if m.status == GRB.status.OPTIMAL:
    print('\nSolverResult:', m.objVal)
    print('\nflow:')
    for var1 in variables1:
      for var2 in variables2:
        if result[var1,var2].x > 1e-6:
          print('result(%s.%s)' % (var1, var2), result[var1,var2].x)
        result[var1,var2]=result[var1,var2].x
  else:
    print('No solution')

print_solution(S, Cu, x, m)
```

If everything went well, the result should be displayed in Excel, see Figure 6.

11	Result:					
12		Brno	Praha	Ostrava	Liberec	Capacity
13	Plzen	0	250	0	50	300
14	Pardubice	0	0	120	60	180
15	Olomouc	180	0	40	0	220
16	Request	180	250	160	110	

Fig. 6 Result.

6 Resource Assignment Problem

This example comes from Gurobi's materials [2]. The company needs to place its employees in optimal jobs. The employees are Carlos, Joe and Monika. The positions they need to place them in are Tester, Java-Developer and Architect. Based on historical data, the company is able to determine the performance of the employees in these positions, see Figure 7. Each employee can only have one job. Each job position must be filled.

	A	B	C	D	E
1		Resource Assignment Problem			
2					
3		Assignment			
4			Tester	Java Deve	Architect
5		Carlos	53	27	13
6		Joe	80	47	67
7		Monika	53	73	47
8					
9					
10		Result			
11			Tester	Java Deve	Architect
12		Carlos			
13		Joe			
14		Monika			
15					

Fig. 7 Performance of employees.

The Data Items Editor is used to map the data from Excel to Python variables, see Figure 8.

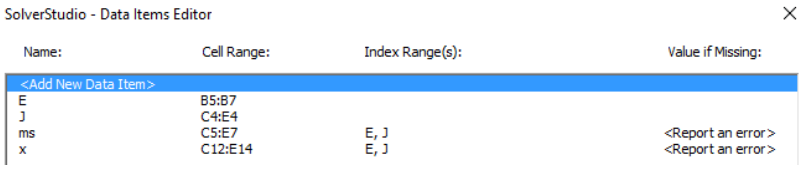

Fig. 8 Data Items Editor.

The final mapping visualisation for this example is in Figure 9.

Next, the SolverStudio Editor can be used and the following Python code can be used to initialize the `Model()`. It is also defined that the decision variables can only take binary values.

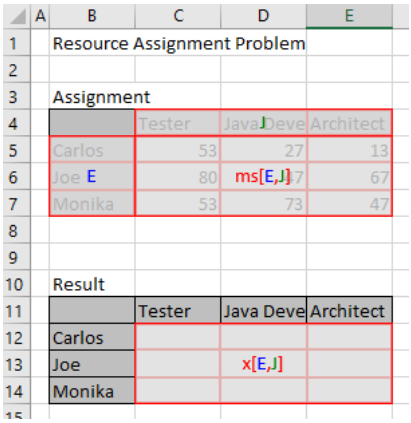

Fig. 9 Performance of employees.

```
from gurobipy import *
from SolverStudio import *

m = Model("resources")

# decision variables
x = m.addVars(x.keys(), vtype=GRB.BINARY, name="assign")
```

The list E contains the names of the three resources: Carlos, Joe, and Monika. The list J contains the names of the job positions: tester, java-developer, and architect. An employee resource $e \in E$ and a job $j \in J$. Let $x_{e,j}$ is a decision variable intended for optimization, When it is equal to 1, the employee is assigned to the job..

The job constraint defines that each position must be filled.

$$\sum_{e \in E} x_{e,j} = 1, \qquad j \in J \tag{4}$$

The resource constraint defines that resources can be allocated at most once. If we had more employees than jobs, some employees would be without jobs.

$$\sum_{j \in J} x_{e,j} \leq 1, \qquad e \in E \tag{5}$$

These constraints can be written in Python as follows.

```
# create jobs  constraints
m.addConstrs((x.sum('*',j) == 1 for j in J), 'jobs')
# create resources constraints
m.addConstrs((x.sum(e,'*') <= 1 for e in E), 'employees')
```

Let $x_{e,j}$, is the optimized value and the matching score $ms_{e,j}$ is the performance of an employee in a particular job.

The objective function is to maximize the performance:

$$Max \sum_{e \in E} \sum_{j \in J} ms_{e,j} x_{e,j}, \tag{6}$$

The objective function in Python:

```
# The objective is to maximize total matching score of the assignments
m.setObjective(x.prod(ms), GRB.MAXIMIZE)
```

The mathematical formulation of the problem is completed so the optimization can be run.

```
# Find the optimal solution
m.optimize()
```

Finally, we can output the optimal results and the optimal value of the objective function to the console. At the same time, the numerical value must be assigned to the excel table for the decision variables.

```
def print_solution(variables1, variables2, result, m):
  if m.status == GRB.status.OPTIMAL:
    print('\nSolverResult:', m.objVal)
    print('\nflow:')
    for var1 in variables1:
      for var2 in variables2:
        if result[var1,var2].x > 1e-6:
          print('result(%s.%s)' % (var1, var2), result[var1,var2].x)
        result[var1,var2]=result[var1,var2].x
  else:
    print('No solution')

print_solution(E, J, x, m)
```

If everything went well, the result should be displayed in Excel, see Figure 10.

10	Result			
11		Tester	Java Deve	Architect
12	Carlos	1	0	0
13	Joe	0	0	1
14	Monika	0	1	0
15				

Fig. 10 Result.

7 Conclusion

The author was able to verify the possibility of using Gurobi Optimizer together with Excel. This paper can serve as a guide for someone who would like to use a similar approach. The paper includes two Excel examples that can be found at https://github.com/lukassykora/gurobi. At the same location there is also an example file from SolverStudio modified for use with Python 3.

References

[1] Jablonský, J.: Operační Výzkum. Vysoká škola ekonomická, Fakulta informatiky a statistiky, Praha (1998)

[2] Pano, S.: Tutorial: Mixed-integer linear programming. Gurobi (2022)

[3] Gurobi: Gurobi optimization product brochure 2021. Gurobi (2021)

[4] Lagová, M., Jablonský, J.: Lineární Modely. Vysoká škola ekonomická, Fakulta informatiky a statistiky, Praha (2014)

[5] Jablonský, J.: Programy Pro Matematické Modelování. Vysoká škola ekonomická, Fakulta informatiky a statistiky, Praha (2007)