Combinatorial Optimization
Lab No. 1
An introduction to the experimental environment

Industrial Informatics Research Center

March 14, 2017

Abstract

The purpose of this lab is to introduce Gurobi Optimizer, which will be used during the course for solving Linear Programming or Integer Linear Programming models. We also show by example how to use Gurobi with different programming languages, namely C++, Java and Python.

1 Gurobi Optimizer

Gurobi Optimizer[1] is, at the present time, one of the best commercial solvers for a wide range of optimization problems such as Linear Programming (LP), Quadratic Programming (QP), Quadratically Constrained Programming (QCP), Mixed Integer Linear Programming (MILP), Mixed-Integer Quadratic Programming (MIQP), and Mixed-Integer Quadratically Constrained Programming (MIQCP). Moreover, obtaining license for academic purposes is quick and easy, therefore this solver will be used for the purposes of this course.

2 Installation

First, create an account on Gurobi website http://www.gurobi.com/index . As "Account type", select "Academic" and use your CTU email address. After that, download Gurobi 6.5 (unless you use `g++` ≥ 5, see the comment below) for your favorite operating system (GNU/Linux, Mac OS, Windows) and follow the installation guide http://www.gurobi.com/documentation/6.5/.

IMPORTANT: If you are using GNU/Linux system with `g++` ≥ 5, Gurobi 6.5 will not work for C++. In that case, install Gurobi 7.0 and perform the following

```
$ cd $GUROBI_HOME/src/build
$ make
$ cp libgurobi_c++.a $GUROBI_HOME/lib
```

This will build the C++ interface compatible with your `g++` version. However, make sure that you **will not** use API that was introduced in Gurobi 7.0, otherwise your code could not be compiled in UploadSystem.

2.1 GNU/Linux

Make sure that OS environment variable `GUROBI_HOME` is pointing to directory with Gurobi and `LD_LIBRARY_PATH` contains reference to `$GUROBI_HOME/lib`

[1] http://www.gurobi.com/index

```
$ echo $GUROBI_HOME
/home/cimrman/opt/gurobi650/linux64
$ echo $LD_LIBRARY_PATH
:/home/cimrman/opt/gurobi650/linux64/lib
```

If this is not the case, you have to set the environment variables by appending the following into your `~/.bashrc`

```
export GUROBI_HOME=/path-to-gurobi-directory/linux64
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GUROBI_HOME/lib
```

It is possible that you have to logout from your account so that the environment variables are visible to the system.

2.2 Mac OS

Make sure that OS environment variable `GUROBI_HOME` is pointing to directory with Gurobi and `DYLD_LIBRARY_PATH` contains reference to `$GUROBI_HOME/lib`.

2.3 Windows

Make sure that OS environment variable `GUROBI_HOME` is pointing to directory with Gurobi and `PATH` contains reference to `%GUROBI_HOME%\bin`

```
> echo %GUROBI_HOME%
C:\gurobi652\win64
> echo %PATH%
C:\gurobi652\win64\bin;C:\WINDOWS\system32;C:\WINDOWS;
```

These variables should be already set by the Gurobi installer.

3 Obtaining Gurobi License

To use Gurobi, you need Academic License which can be requested at http://user.gurobi.com/download/licenses/free-academic . After pushing "Request License" button, a new page appears with command that you need to copy and paste to your system terminal, e.g. on UNIX command line

```
$ $GUROBI_HOME/bin/grbgetkey license-key
```

where `license-key` is your license key.

IMPORTANT: in order to validate your academic license, you are required to call the command while being connected in CTU domain (*eduroam* or local area network). If you would like to install Gurobi on your desktop computer at home and you do not have a possibility to connect into CTU domain, you may request Online Course License at https://user.gurobi.com/download/licenses/free-online. The difference between the academic and the course licenses is that the latter can solve models with up to 2000 variables and 2000 constraints (should be enough for the purpose of the course).

4 Programming interfaces

Gurobi Optimizer supports a variety of programming and modeling languages including C++, Java, .NET, Python, C, R and MATLAB. In this course we support C++, Java and Python; choose the language according to your preferences.

Let us consider the following LP model:

$$
\begin{aligned}
\max \quad & 32x + 25y \\
\text{s.t.} \quad & 5x + 4y \leq 59 \\
& 4x + 3y \leq 46 \\
& x, y \geq 0 \\
& x, y \in \mathbb{R}
\end{aligned}
$$

Figure 1: LP model.

which has optimal value of 374. The following steps are generally required for implementing the given model in any language:

- Importing the Gurobi functions and classes.
- Creating the environment for the optimization model. The environment represents the configuration of the Gurobi (e.g. logging verbosity, number of used threads).
- Creating an empty optimization model.
- Adding the decision variables to the model with their types and bounds.
- Updating the model. The decision variables are added to the model in lazy fashion, i.e. the effects of the modifications are not seen immediately, hence, it is necessary to manually update the model.
- Setting and adding the objective function and the constraints to the model.
- When all the necessary components are created and set, the model is solved by calling `optimize()`.
- Reporting results. In particular, you can obtain the objective and the values of the decision variables in the current solution.
- Cleaning up the resources associated with the model and environment. This step is optional, garbage collector (Java, Python) or RAII (C++) will eventually clean up the resources.

In the following subsections, we show how to use the Java (see Section 4.2), Python (see Section 4.3) and C++ (see Section 4.1) interfaces to solve the above mentioned LP model.

If you are interested in more examples, check http://www.gurobi.com/documentation/current/examples.pdf or `$GUROBI_HOME/examples` .

4.1 C++ Interface

Listing 1 shows the implementation of the example in C++.

Listing 1: C++ implementation of the model shown in Figure. 1.

```
1  #include <gurobi_c++.h>
2  using namespace std;
3
4  int main(int argc, char *argv[]) {
5      // Create new environment.
6      GRBEnv env;
7
8      // Create empty optimization model.
9      GRBModel model(env);
10
11     // Create variables x, y.
```

```
12      // addVar(lowerBound, upperBound, objectiveCoeff, variableType, name)
13      GRBVar x = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "x");
14      GRBVar y = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "y");
15
16      // Integrate new variables into model.
17      model.update();
18
19      // Set objective: maximize 32x + 25y
20      model.setObjective(32*x + 25*y, GRB_MAXIMIZE);
21
22      // Add constraint: 5x + 4y <= 59
23      model.addConstr(5*x + 4*y <= 59, "cons1");
24
25      // Add constraint: 4x + 3y <= 46
26      model.addConstr(4*x + 3*y <= 46, "cons2");
27
28      // Solve the model.
29      model.optimize();
30
31      // Print the objective
32      // and the values of the decision variables in the solution.
33      cout << "Optimal objective: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
34      cout << "x: " << x.get(GRB_DoubleAttr_X) << " ";
35      cout << "y: " << y.get(GRB_DoubleAttr_X) << endl;
36
37      return 0;
38  }
```

To compile the example, you need to pass include and lib files to your compiler, e.g. for `g++`

```
$ g++ example.cpp -std=c++11 -O2 -march=native -pthread \
-I$GUROBI_HOME/include -L$GUROBI_HOME/lib -lgurobi_c++ -lgurobi65
```

If you are using Windows+Visual Studio, please follow this link http://www.technical-recipes.com/2016/getting-started-with-gurobi-in-microsoft-visual-studio/ . If you prefer building your programs using CMake, check `example.zip` that you will find on CourseWare/Labs page.

4.2 Java Interface

Listing 2 shows the implementation of the example in Java. Unfortunately, Java does not support operator overloading, therefore to create constraints and objective, `GRBLinExpr` has to be used. `GRBLinExpr` represents a linear expression of a form

$$a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n$$

where x_i are variables (instances of `GRBVar` class) and a_i are scalar values. The terms a_ix_i are added to `GRBLinExpr` one-by-one with `addTerm(double a, GRBVar x)` or as a scalar product `addTerms(double[] a, GRBVar[] x)`.

Listing 2: Java implementation of the model shown in Figure. 1.

```
1   import gurobi.*;
2
3   public class Example {
4     public static void main(String[] args) throws Exception {
5       // Create new environment.
6       GRBEnv env = new GRBEnv();
7
8       // Create empty optimization model.
9       GRBModel model = new GRBModel(env);
10
11      // Create variables x, y.
12      // addVar(lowerBound, upperBound, objectiveCoeff, variableType, name)
13      GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
14      GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
```

```
15
16        // Integrate new variables into model.
17        model.update();
18
19        // Set objective: maximize 32x + 25y
20        GRBLinExpr obj = new GRBLinExpr();
21        obj.addTerm(32.0, x);
22        obj.addTerm(25.0, y);
23        model.setObjective(obj, GRB.MAXIMIZE);
24
25        // Add constraint: 5x + 4y <= 59
26        GRBLinExpr cons1 = new GRBLinExpr();
27        cons1.addTerm(5.0, x);
28        cons1.addTerm(4.0, y);
29        // addConstr(leftHandSide, inequalityType, rightHandSide, name)
30        model.addConstr(cons1, GRB.LESS_EQUAL, 59.0, "cons1");
31
32        // Add constraint: 4x + 3y <= 46
33        GRBLinExpr cons2 = new GRBLinExpr();
34        cons2.addTerm(4.0, x);
35        cons2.addTerm(3.0, y);
36        model.addConstr(cons2, GRB.LESS_EQUAL, 46.0, "cons2");
37
38        // Solve the model.
39        model.optimize();
40
41        // Print the objective
42        // and the values of the decision variables in the solution.
43        System.out.println(x.get(GRB.StringAttr.VarName)+ " " +x.get(GRB.DoubleAttr.X)
              );
44        System.out.println(y.get(GRB.StringAttr.VarName) + " " +y.get(GRB.DoubleAttr.X
              ));
45        System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));
46    }
47 }
```

To run the example from UNIX command line, make sure that `$GUROBI_HOME/lib/gurobi.jar` is in your classpath

```
$ javac -cp $GUROBI_HOME/lib/gurobi.jar Example.java
$ java -cp $GUROBI_HOME/lib/gurobi.jar:. Example
```

Similarly, the example can be run from Windows command line as follows (assuming that the Java executables are in your `PATH` environment variable)

```
> javac.exe -cp %GUROBI_HOME%\lib\gurobi.jar Example.java
> java.exe -cp %GUROBI_HOME%\lib\gurobi.jar;. Example
```

If you prefer using IDE, it should be enough to add jarfile `$GUROBI_HOME/lib/gurobi.jar` to your project.

4.3 Python Interface

Both Python 2 and Python 3 versions are supported by Gurobi, just make sure that you use the proper shebang in your scripts (here we will use Python 3). To include Gurobi's module, one has to install it first

```
$ cd $GUROBI_HOME
$ python3 setup.py install
```

If you do not have the administrator rights, you can install the Gurobi with following

```
$ python3 setup.py install --user
```

Listing 3 shows the implementation of the example in Python.

Listing 3: Python implementation of the model shown in Figure. 1.

```
#!/usr/bin/env python3

import gurobipy as g

# Create empty optimization model.
# In Python, only one environment exists and it is created internally
# in the Model() constructor.
model = g.Model()

# Create variables x, y.
x = model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS, name="x")
y = model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS, name="y")

# Integrate new variables into model.
model.update()

# Set objective: maximize 32x + 25y
model.setObjective(32*x + 25*y, sense=g.GRB.MAXIMIZE)

# Add constraint: 5x + 4y <= 59
model.addConstr(5*x + 4*y <= 59, "cons1")

# Add constraint: 4x + 3y <= 46
model.addConstr(4*x + 3*y <= 46, "cons2")

# Solve the model.
model.optimize()

# Print the objective and the values of the decision variables in the solution.
print("Optimal objective:", model.objVal)
print("x:", x.x, "y:", y.x)
```

There is also a neat shortcut `quicksum` for creating a linear expression of a form

$$a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n$$

where x_i are variables and a_i are scalar values. For example, we could do

```
# Create list of variables, x_i.
x = [model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS),
     model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS)]

# Create list of coefficients, a_i.
a = [10, 50]

# Add constraint: 10x_1 + 50x_2 <= 31
model.addConstr(g.quicksum([a_i*x_i
                            for a_i, x_i in zip(a, x)])
                <= 31)
```