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

Industrial Informatics Research Center

February 9, 2021

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

Generally, you can follow the instructions in the installation guide.

Instalation guide: https://www.gurobi.com/documentation/quickstart.html

The installation consists of several steps:

- Retrieving a Gurobi licence and downloading the binaries.
- Seting up system variables and the installation of programming interfaces.
- Activation of the Gurobi licence.

For your convinience, this tutorial will guide you through the most critical points of the installation, but you should be able to proceeds with the online guide.

To retrieve a Gurobi licence, 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 for your favorite operating system (GNU/Linux, Mac OS, Windows). You should download either ≥ 7.5 (current version is 9.1.1) or 7.0 version depending on the context:

- you would like to program in Python: for Python 3.5 select 7.0, for Python 3.6 select ≥ 7.5. **Lab computers** have Python 3.6.9 therefore, you need to install Gurobi ≥ 7.5 there if you want to use them. All other versions of Python **ARE NOT SUPPORTED** by Gurobi, so please update your Python environment or use different programming language.

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

- others: we recommend Gurobi 9.1.1 (However, BRUTE uses Gurobi 8.1, therefore, using the same version will give you maximum compatibility, even though API is not changing that much.)

In this document, we will assume that you use Gurobi 9.1.1, so if necessary, modify the following commands according to your version. To install Gurobi, follow the installation guide https://www.gurobi.com/documentation/9.1/quickstart_windows/software_installation_guid.html.

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`

```
$ echo $GUROBI_HOME
/import/users/cimrman/opt/gurobi911/linux64
$ echo $LD_LIBRARY_PATH
:/import/users/cimrman/opt/gurobi911/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`. By default, Gurobi will be installed in `/Library/gurobi911/mac64`.

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:\gurobi911\win64
> echo %PATH%
C:\gurobi911\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 . There, you can read terms of use, etc. When you are log in, you can request the licence via the link
https://www.gurobi.com/downloads/end-user-license-agreement-academic/.

After pushing "I accept these conditions" 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 execute the command while being connected in CTU domain (*eduroam* or local area network). For students at FEE CTU, it is possible to connect to university network via VPN. Please follow instruction at faculty website: https://svti.fel.cvut.cz/en/services/vpn.html.

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://www.gurobi.com/licenses/for-online-courses/. 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.
- 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 Python (see Section 4.1), Java (see Section 4.3), and C++ (see Section 4.2) 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 Python Interface

Officially, only Python 3 version is 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. This can be done via `pip` package manager:

```
python3 -m pip install -i https://pypi.gurobi.com gurobipy
```

You should see `gurobipy` among the installed packages listed after typing `python -m pip list`.

If for some reason you cannot use `pip`, you can install bindings manually:

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

If you do not have the administrator rights (e.g., lab computers), you can install Gurobi with following

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

You can test if everything is well installed by opening the interactive interpret and typing `import gurobipy`. No error message should appear. Listing 1 shows the implementation of the example in Python.

Listing 1: 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")

# 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)
```

To run the example from command line just call (should work in all operating systems)

```
$ python3 example.py
```

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)
```

4.2 C++ Interface

Listing 2 shows the implementation of the example in C++.

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

```
#include <gurobi_c++.h>
using namespace std;

int main(int argc, char *argv[]) {
    // Create new environment.
    GRBEnv env;

    // Create empty optimization model.
    GRBModel model(env);

    // Create variables x, y.
    // addVar(lowerBound, upperBound, objectiveCoeff, variableType, name)
    GRBVar x = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "x");
    GRBVar y = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "y");

    // Set objective: maximize 32x + 25y
    model.setObjective(32*x + 25*y, 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.
    cout << "Optimal objective: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
    cout << "x: " << x.get(GRB_DoubleAttr_X) << " ";
    cout << "y: " << y.get(GRB_DoubleAttr_X) << endl;

    return 0;
}
```

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_g++4.2 -lgurobi91
```

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. The archive contains a Gurobi module finder for CMake.

4.3 Java Interface

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

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.

5 Common Issues

5.1 Windows, Python: ImportError: DLL load failed: %1 is not a valid Win32 application

Occurs when you try to import Gurobi library within a Python script or interpreter. Reason is that you probably installed a 32-bit Python on a 64-bit system You check it by calling python from command line and reading the interpreter's info message whether it contains 64bit. If not, then you need to install 64-bit version of Python.

5.2 Java, Netbeans: package gurobi does not exist

You need to link gurobi.jar to Netbeans project. Right-click on the Libraries of your project in the Projects view and select Add JAR/Folder.... Find the Gurobi JAR file which should be located at $GUROBI_HOME/lib.

5.3 Java, IntelliJ: cannot resolve symbol gurobi

You need to link gurobi.jar to IntelliJ project. Click on File in the menu and select Project Structure ... . Select Libraries tab, click on the green plus sign (New Project Library) and select Java. Find the Gurobi JAR file which should be located at $GUROBI_HOME/lib. Click on the OK button, then click Apply and then OK .

5.4 Java: no GurobiJni80 in java.library.path

Occurs when you try to run application that uses Gurobi. You need to manually specify path to Jni file by providing -Djava.library.path=$GUROBI_HOME/lib to VM (replace $GUROBI_HOME with the absolute path where this environment variable is pointing).

In Netbeans, right-click on the project in the Projects view and select Properties . Find the Run tab and there fill in the VM Options textbox.

In IntelliJ, select Run in the menu and select Edit Configurations ... . In the opened window, select your project in the left view, click on the Configuration tab and fill in VM options textbox.

5.5 Linux, C++: undefined references

If you are using g++ ≥ 5 compiler, please build the C++ binding to Gurobi for your compiler by doing the following

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

5.6 Linux: bash: grbgetkey: command not found ...

You do not have $GUROBI_HOME/bin in your $PATH environment variable. Either add it there or call $GUROBI_HOME/bin/grbgetkey instead.

5.7 C++, CLion: `Could NOT find GUROBI_INCLUDE_DIR` and others...

For some reason, CLion is not picking up your environment variables. Therefore, you need to open the file `modules/FindGUROBI.cmake` in your project directory and replace all the occurrences of `$ENV{GUROBI_HOME}` with the absolute path.

5.8 Windows, CLion

This one is a little bit harder, so we really recommend using either Visual Studio or different language. The reason is that on Windows basically only MSVC compiler is officially supported by Gurobi and CLion on Windows right now works only with Mingw and Cygwin compilers (MSVC support in CLion is experimental and we were not able to make it run).

We will be using Mingw compiler so start by downloading mingw-w64 application and begin its installation (be sure to select the correct architecture according to your system during the installation). Once everything is installed, open CLion, click on `File -> Settings ...` in the menu. Open `Build, Execution, Deployment -> Toolchains` and click on the green plus sign (`Add`). As an environment, select Mingw and CLion should autodetect all the necessary executables. Click on `Apply` button, then `OK`.

Next, add the directory `mingw64/bin` from your installed Mingw directory to `%PATH%` environment variable (make sure to logout so that the changes are applied).

Now, **COMPLETELY** replace the content of the file `$GUROBI_HOME/src/build/Makefile` with the following

```
C++      = g++
C++FLAGS = -m64 -O

C++OBJS = Env.o Model.o Var.o Constr.o LinExpr.o QuadExpr.o \
          Exception.o Callback.o Column.o SOS.o QConstr.o GenConstr.o \
          TempConstr.o

c++: libgurobi_c++.a

%.o: ../cpp/%.cpp ../cpp/%.h
$(C++) $(C++FLAGS) -I../../include -c $<

libgurobi_c++.a: $(C++OBJS)
ar rv libgurobi_c++.a $(C++OBJS)

clean:
rm -f *.o libgurobi_c++.a
```

We need to build the C++ binding for the Mingw compiler, which can be done on the Windows command line as follows

```
> cd %GUROBI_HOME%\src\build
> mingw32-make
> copy "libgurobi_c++.a" ..\..\lib
```

Clear the cmake cache in CLion by selecting `Tools -> CMake -> Reset Cache and Reload Project` in the CLion menu. Now you should be able to build and run your project.