Combinatorial Algorithms: Lab No. 1
An introduction to the experimental environment

Industrial Informatics Department
Czech Technical University in Prague
https://industrialinformatics.fel.cvut.cz/

February 14, 2025

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++ 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 a 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 license and downloading the binaries.
- Setting up system variables and the installation of programming interfaces.
- Activation of the Gurobi license.

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

To retrieve a Gurobi license, first, create an account on the Gurobi website https://pages.gurobi.com/registration. 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 the current version which is 12.0.1.

In this document, we will assume that you use Gurobi 12.0.1, so if necessary, modify the following commands according to your version. To install Gurobi, follow the installation guide https://support.gurobi.com/hc/en-us/articles/4534161999889-How-do-I-install-Gurobi-Optimizer. We recommend to perform the full installation.

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

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/gurobi1201/linux64
$ echo $LD_LIBRARY_PATH
:/import/users/cimrman/opt/gurobi1201/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 log out 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 the directory with Gurobi, and `DYLD_LIBRARY_PATH` contains reference to `$GUROBI_HOME/lib`. By default, Gurobi will be installed in `/Library/gurobi1201/macos_universal2`.

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:\gurobi1201\win64
> echo %PATH%
C:\gurobi1201\win64\bin;C:\WINDOWS\system32;C:\WINDOWS;
```

These variables should already be set by the Gurobi installer.

2.4 Google Colab (Python only)

Consider this as an emergency option for how to use Gurobi; thus, this should not be your primary way how to access it. Nevertheless, sometimes it might be useful to use a hosted VM in the cloud with the Gurobi installed there. Google Colab (https://colab.research.google.com/) offers that since Gurobi automatically provides free limited licenses to their hosted VMs. However, there is a possibility to use your full academic license even inside the Colab environment or other cloud/container environments. For more information, visit https://www.gurobi.com/features/web-license-service/ and Section 3.1.

To access the Gurobi solver from Google Colab interface, log in a create a new hosted Python notebook. In the first line of the notebook, type:

```
!pip install gurobipy
```

and run the command by pressing shift+enter. The full example can be accessed here:
https://colab.research.google.com/drive/1Dt6awK5b3b8ghMrXSNehaunnfGsUsa-a?usp=sharing.

3 Obtaining Gurobi License

To use Gurobi, you need Academic License, which can be requested at https://portal.gurobi.com/iam/licenses/list/ . There, you can read terms of use, etc. When you are logged in, you can request the license via the link https://portal.gurobi.com/iam/licenses/request. You will choose "Named-User Academic" license.

After clicking "Generate Now!" button, a new page appears where you will accept EULA. Then, a command appears 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 generate and validate your academic license, you are required to execute the command while being connected in the CTU domain (*eduroam* or local area network). For students at FEE CTU, it is possible to connect to the university network via VPN. Please follow the instructions at the 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 the possibility to connect to the CTU domain, you may request Online Course License. The difference between the academic and the course licenses is that the latter can solve models with up to 2000 variables and 2000 constraints (which should be enough for the purpose of the course).

3.1 WSL licence in Google Colab/container environment

Named-User Academic licence is bound to the specific hardware where it is being activated (i.e., CPU footprint and Ethernet interface MAC address, etc.). Hence, it does not work in a virtual/container environment. For these purposes, Gurobi offers WSL license, which requires Internet access to check its validity every time the optimization model is constructed. To get WSL licence, follow instructions at https://portal.gurobi.com/iam/licenses/request. You will obtain three secret tokens: `WLSACCESSID`, `WLSSECRET`, and `LICENSEID`. To use the WSL license, e.g., in Google Colab notebook, there are two possible ways.

3.1.1 Hard-coded parameters

Initialize Gurobi environment first by.

```
import gurobipy as grb
params = {
"WLSACCESSID": 'secret-hash',
"WLSSECRET": 'secret-hash',
"LICENSEID": secret-licence-id,
}
env = grb.Env(params=params)
```

After that, you can construct optimization model by passing the environment into the constructor:

```
model = grb.Model(env=env)
```

3.1.2 Linked license file

You can link the license file by modifying the environment variable before the import statement:

```
os.environ['GRB_LICENSE_FILE'] = "/path_to_license/gurobi.lic"

import gurobipy as grb
```

To do this securely in shared Google Colab, you can do it by uploading license at the start of every runtime and moving it to the "tmp" folder, which is independent for each running instance of the same Colab notebook:

```
from google.colab import files
import shutil
import os
uploaded = files.upload()  # Upload gurobi.lic manually

# Move the license to a temporary folder
shutil.move("/content/gurobi.lic", "/tmp/gurobi.lic")
os.environ['GRB_LICENSE_FILE'] = "/tmp/gurobi.lic"
```

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++ 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), 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 (for Gurobi 12, python 3.8+) 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:

```
pip install gurobipy
```

Please check in which Python environment the `pip` command has installed the `gurobipy` module. On some platforms, you may need to use `pip3` and `python3` commands.

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 administrator rights (e.g., lab computers), you can install Gurobi with following

```
$ python3 -m pip gurobipy --user
```

or

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

Optional: some IDEs do not support auto-completion for Gurobi out-of-box (or it is very limited). You can try installing the official `gurobipy-stubs` package for better auto-completion

```
$ python3 -m pip gurobipy-stubs --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.

```
1  #!/usr/bin/env python3
2
3  import gurobipy as g
4
5  # Create empty optimization model.
6  # In Python, only one environment exists and it is created internally
7  # in the Model() constructor.
8  model = g.Model()
9
10 # Create variables x, y.
11 x = model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS, name="x")
12 y = model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS, name="y")
13
14 # Set objective: maximize 32x + 25y
15 model.setObjective(32*x + 25*y, sense=g.GRB.MAXIMIZE)
```

```
16
17 # Add constraint: 5x + 4y <= 59
18 model.addConstr(5*x + 4*y <= 59, "cons1")
19
20 # Add constraint: 4x + 3y <= 46
21 model.addConstr(4*x + 3*y <= 46, "cons2")
22
23 # Solve the model.
24 model.optimize()
25
26 # Print the objective and the values of the decision variables in the solution.
27 print("Optimal objective:", model.objVal)
28 print("x:", x.x, "y:", y.x)
```

To run the example from the 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

```
1  # Create list of variables, x_i.
2  x = [model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS),
3       model.addVar(lb=0, ub=g.GRB.INFINITY, vtype=g.GRB.CONTINUOUS)]
4
5  # Create list of coefficients, a_i.
6  a = [10, 50]
7
8  # Add constraint: 10x_1 + 50x_2 <= 31
9  model.addConstr(g.quicksum([a_i*x_i
10                              for a_i, x_i in zip(a, x)])
11                  <= 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.

```
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     // Set objective: maximize 32x + 25y
17     model.setObjective(32*x + 25*y, GRB_MAXIMIZE);
18
19     // Add constraint: 5x + 4y <= 59
20     model.addConstr(5*x + 4*y <= 59, "cons1");
21
22     // Add constraint: 4x + 3y <= 46
23     model.addConstr(4*x + 3*y <= 46, "cons2");
24
25     // Solve the model.
```

```
26      model.optimize();
27
28      // Print the objective
29      // and the values of the decision variables in the solution.
30      cout << "Optimal objective: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
31      cout << "x: " << x.get(GRB_DoubleAttr_X) << " ";
32      cout << "y: " << y.get(GRB_DoubleAttr_X) << endl;
33
34      return 0;
35  }
```

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++ -lgurobi120
```

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.

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 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
```

5.3 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.

5.4 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.

5.5 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.