Hands-on Large Scale Optimization in Python

From Beginning to Giving Up

Kunlei Lian

2023-02-18

Table of contents

Preface

1 Environment Setup

In this chapter, we explain the steps needed to set up Python and Google OR-Tools. All the steps below are based on MacBook Air with M1 chip and macOS Ventura 13.1.

1.1 Install Homebrew

The first tool we need is Homebrew, ‘the Missing Package Manager for macOS (or Linux)’, and it can be accessed at https://brew.sh/. To install Homebrew, just copy the command below and run it in the Terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
```

We can then use the `brew --version` command to check the installed version. On my system, it shows the info below.

```
 ~/ brew --version
Homebrew 3.6.20
Homebrew/homebrew-core (git revision 5f1582e4d55; last commit 2023-02-05)
Homebrew/homebrew-cask (git revision fa3b8a669d; last commit 2023-02-05)
```

1.2 Install Anaconda

Since there are several Python versions available for our use and we may end up having multiple Python versions installed on our machine, it is important to use a consistent environment to work on our project in. Anaconda is a package and environment manager for Python and it provides easy-to-use tools to facilitate our data science needs. To install Anaconda, run the below command in the Terminal.

```
 ~/ brew install anaconda
```

After the installation is done, we can use `conda --version` to verify whether it is available on our machine or not.

```
  ~/ conda --version
conda 23.1.0
```

1.3 Create a Conda Environment

Now we will create a Conda environment named 'ortools'. Execute the below command in the Terminal, which effectively creates the required environment with Python version 3.10.

```
  ~/ conda create -n ortools python=3.10
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/homebrew/anaconda3/envs/test

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    setuptools-67.4.0          |     pyhd8ed1ab_0         567 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         567 KB

The following NEW packages will be INSTALLED:

  bzip2              conda-forge/osx-arm64::bzip2-1.0.8-h3422bc3_4
  ca-certificates    conda-forge/osx-arm64::ca-certificates-2022.12.7-h4653dfc_0
  libffi             conda-forge/osx-arm64::libffi-3.4.2-h3422bc3_5
  libsqlite          conda-forge/osx-arm64::libsqlite-3.40.0-h76d750c_0
  libzlib            conda-forge/osx-arm64::libzlib-1.2.13-h03a7124_4
  ncurses            conda-forge/osx-arm64::ncurses-6.3-h07bb92c_1
  openssl            conda-forge/osx-arm64::openssl-3.0.8-h03a7124_0
  pip                conda-forge/noarch::pip-23.0.1-pyhd8ed1ab_0
  python             conda-forge/osx-arm64::python-3.10.9-h3ba56d0_0_cpython
  readline           conda-forge/osx-arm64::readline-8.1.2-h46ed386_0
```

```
  setuptools         conda-forge/noarch::setuptools-67.4.0-pyhd8ed1ab_0
  tk                 conda-forge/osx-arm64::tk-8.6.12-he1e0b03_0
  tzdata             conda-forge/noarch::tzdata-2022g-h191b570_0
  wheel              conda-forge/noarch::wheel-0.38.4-pyhd8ed1ab_0
  xz                 conda-forge/osx-arm64::xz-5.2.6-h57fd34a_0


Proceed ([y]/n)?
```

Type 'y' to proceed and Conda will create the environment for us. We can use `cnoda env list` to show all the created environments on our machine:

```
  ~/ conda env list
# conda environments:
#
base                     /opt/homebrew/anaconda3
ortools                   /opt/homebrew/anaconda3/envs/ortools
```

Note that we need to manually activate an environemnt in order to use it: `conda activate ortools`. On my machine, the activated environment `ortools` will appear in the beginning of my prompt.

```
  ~/ conda activate ortools
(ortools)   ~/
```

1.4 Install Google OR-Tools

As of this writing, the latest version of Google OR-Tools is 9.5.2237, and we can install it in our newly created environment using the command `pip install ortools==9.5.2237`. We can use `conda list` to verify whether it is available in our environment.

```
(ortools)   ~/ conda list
# packages in environment at /opt/homebrew/anaconda3/envs/ortools:
#
# Name                    Version                   Build  Channel
absl-py                   1.4.0                    pypi_0    pypi
bzip2                     1.0.8                h3422bc3_4    conda-forge
ca-certificates           2022.12.7            h4653dfc_0    conda-forge
libffi                    3.4.2                h3422bc3_5    conda-forge
libsqlite                 3.40.0               h76d750c_0    conda-forge
```

```
libzlib                   1.2.13               h03a7124_4    conda-forge
ncurses                   6.3                  h07bb92c_1    conda-forge
numpy                     1.24.2                   pypi_0    pypi
openssl                   3.0.8                h03a7124_0    conda-forge
ortools                   9.5.2237                 pypi_0    pypi
pip                       23.0.1             pyhd8ed1ab_0    conda-forge
protobuf                  4.22.0                   pypi_0    pypi
python                    3.10.9          h3ba56d0_0_cpython    conda-forge
readline                  8.1.2                h46ed386_0    conda-forge
setuptools                67.4.0             pyhd8ed1ab_0    conda-forge
tk                        8.6.12               he1e0b03_0    conda-forge
tzdata                    2022g                h191b570_0    conda-forge
wheel                     0.38.4             pyhd8ed1ab_0    conda-forge
xz                        5.2.6                h57fd34a_0    conda-forge
```

Now we have Python and Google OR-Tools ready, we can start our next journey.

2 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

3 Environment Setup

In this chapter, we explain the steps needed to set up Python and Google OR-Tools. All the steps below are based on MacBook Air with M1 chip and macOS Ventura 13.1.

3.1 Install Homebrew

The first tool we need is Homebrew, 'the Missing Package Manager for macOS (or Linux)', and it can be accessed at https://brew.sh/. To install Homebrew, just copy the command below and run it in the Terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
```

We can then use the `brew --version` command to check the installed version. On my system, it shows the info below.

```
  ~/ brew --version
Homebrew 3.6.20
Homebrew/homebrew-core (git revision 5f1582e4d55; last commit 2023-02-05)
Homebrew/homebrew-cask (git revision fa3b8a669d; last commit 2023-02-05)
```

3.2 Install Anaconda

Since there are several Python versions available for our use and we may end up having multiple Python versions installed on our machine, it is important to use a consistent environment to work on our project in. Anaconda is a package and environment manager for Python and it provides easy-to-use tools to facilitate our data science needs. To install Anaconda, run the below command in the Terminal.

```
  ~/ brew install anaconda
```

After the installation is done, we can use `conda --version` to verify whether it is available on our machine or not.

```
  ~/ conda --version
conda 23.1.0
```

3.3 Create a Conda Environment

Now we will create a Conda environment named ‘ortools’. Execute the below command in the Terminal, which effectively creates the required environment with Python version 3.10.

```
  ~/ conda create -n ortools python=3.10
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/homebrew/anaconda3/envs/test

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    setuptools-67.4.0          |     pyhd8ed1ab_0         567 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         567 KB

The following NEW packages will be INSTALLED:

  bzip2              conda-forge/osx-arm64::bzip2-1.0.8-h3422bc3_4
  ca-certificates    conda-forge/osx-arm64::ca-certificates-2022.12.7-h4653dfc_0
  libffi             conda-forge/osx-arm64::libffi-3.4.2-h3422bc3_5
  libsqlite          conda-forge/osx-arm64::libsqlite-3.40.0-h76d750c_0
  libzlib            conda-forge/osx-arm64::libzlib-1.2.13-h03a7124_4
  ncurses            conda-forge/osx-arm64::ncurses-6.3-h07bb92c_1
  openssl            conda-forge/osx-arm64::openssl-3.0.8-h03a7124_0
  pip                conda-forge/noarch::pip-23.0.1-pyhd8ed1ab_0
  python             conda-forge/osx-arm64::python-3.10.9-h3ba56d0_0_cpython
  readline           conda-forge/osx-arm64::readline-8.1.2-h46ed386_0
```

```
  setuptools         conda-forge/noarch::setuptools-67.4.0-pyhd8ed1ab_0
  tk                 conda-forge/osx-arm64::tk-8.6.12-he1e0b03_0
  tzdata             conda-forge/noarch::tzdata-2022g-h191b570_0
  wheel              conda-forge/noarch::wheel-0.38.4-pyhd8ed1ab_0
  xz                 conda-forge/osx-arm64::xz-5.2.6-h57fd34a_0


Proceed ([y]/n)?
```

Type 'y' to proceed and Conda will create the environment for us. We can use `cnoda env list` to show all the created environments on our machine:

```
  ~/ conda env list
# conda environments:
#
base                     /opt/homebrew/anaconda3
ortools                   /opt/homebrew/anaconda3/envs/ortools
```

Note that we need to manually activate an environemnt in order to use it: `conda activate ortools`. On my machine, the activated environment `ortools` will appear in the beginning of my prompt.

```
  ~/ conda activate ortools
(ortools)   ~/
```

3.4 Install Google OR-Tools

As of this writing, the latest version of Google OR-Tools is 9.5.2237, and we can install it in our newly created environment using the command `pip install ortools==9.5.2237`. We can use `conda list` to verify whether it is available in our environment.

```
(ortools)   ~/ conda list
# packages in environment at /opt/homebrew/anaconda3/envs/ortools:
#
# Name                    Version                   Build  Channel
absl-py                   1.4.0                    pypi_0    pypi
bzip2                     1.0.8                h3422bc3_4    conda-forge
ca-certificates           2022.12.7            h4653dfc_0    conda-forge
libffi                    3.4.2                h3422bc3_5    conda-forge
libsqlite                 3.40.0               h76d750c_0    conda-forge
```

```
libzlib                   1.2.13               h03a7124_4    conda-forge
ncurses                   6.3                  h07bb92c_1    conda-forge
numpy                     1.24.2                   pypi_0    pypi
openssl                   3.0.8                h03a7124_0    conda-forge
ortools                   9.5.2237                 pypi_0    pypi
pip                       23.0.1             pyhd8ed1ab_0    conda-forge
protobuf                  4.22.0                   pypi_0    pypi
python                    3.10.9          h3ba56d0_0_cpython    conda-forge
readline                  8.1.2                h46ed386_0    conda-forge
setuptools                67.4.0             pyhd8ed1ab_0    conda-forge
tk                        8.6.12               he1e0b03_0    conda-forge
tzdata                    2022g                h191b570_0    conda-forge
wheel                     0.38.4             pyhd8ed1ab_0    conda-forge
xz                        5.2.6                h57fd34a_0    conda-forge
```

Now we have Python and Google OR-Tools ready, we can start our next journey.

Part I

Benders Decomposition

4 Benders Decomposition

In this chapter, we will explain the theories behind Benders decomposition and demonstrate its usage on a trial linear programming problem. Keep in mind that Benders decomposition is not limited to solving linear programming problems. In fact, it is one of the most powerful techniques to solve some large-scale mixed-integer linear programming problems.

In the following sections, we will go through the critical steps during the decomposition process when applying the algorithm on optimization problems represented in standard forms. This is important as it helps build up the intuition of when we should consider applying Benders decomposition to a problem at hand. Often times, recognizing the applicability of Benders decomposition is the most important and challenging step when solving an optimization problem. Once we know that the problem structure is suitable to solve via Benders decomposition, it is straightforward to follow the decomposition steps and put it into work.

Generally speaking, Benders decomposition is a good solution candidate when the resulting problem is much easier to solve if some of the variables in the original problem are fixed. We will illustrate this point using an example in the following sections. In the optimization world, the first candidate that should come to mind when we say a problem is easy to solve is a linear programming formulation, which is indeed the case in Benders decomposition applications.

4.1 The Decomposition Logic

To explain the workings of Benders decomposition, let us look at the standard form of linear programming problems that involve two vector variables, $\mathbf{x}$ and $\mathbf{y}$. Let p and q indicate the dimensions of $\mathbf{x}$ and $\mathbf{y}$, respectively. Below is the original problem ($\mathbf{P}$) we intend to solve.

$$\begin{aligned} \text{min.} \quad & \mathbf{c}^T\mathbf{x} + \mathbf{f}^T\mathbf{y} && (4.1)\\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{b} && (4.2)\\ & \mathbf{x} \geq 0, \mathbf{y} \geq 0 && (4.3) \end{aligned}$$

In this formulation, $\mathbf{c}$ and $\mathbf{f}$ in the objective function represent the cost coefficients associated with decision variables $\mathbf{x}$ and $\mathbf{y}$, respectively. Both of them are column vectors of corresponding dimensions. In the constraints, matrix $\mathbf{A}$ is of dimension $m \times p$, and matrix $\mathbf{B}$ is of dimension $m \times q$. $\mathbf{b}$ is a column vector of dimension m.

Suppose the variable $\mathbf{y}$ is a complicating variable in the sense that the resulting problem is substantially easier to solve if the value of $\mathbf{y}$ is fixed. In this case, we could rewrite problem **P** as the following form:

$$\begin{aligned} \text{min.} \quad & \mathbf{f}^T\mathbf{y} + g(\mathbf{y}) && (4.4) \\ \text{s.t.} \quad & \mathbf{y} \geq 0 && (4.5) \end{aligned}$$

where $g(\mathbf{y})$ is a function of $\mathbf{y}$ and is defined as the subproblem **SP** of the form below:

$$\begin{aligned} \text{min.} \quad & \mathbf{c}^T\mathbf{x} && (4.6) \\ \text{s.t.} \quad & \mathbf{Ax} = \mathbf{b} - \mathbf{By} && (4.7) \\ & \mathbf{x} \geq 0 && (4.8) \end{aligned}$$

Note that the $\mathbf{y}$ in constraint (4.7) takes on some known values when the problem is solved and the only decision variable in the above formulation is $\mathbf{x}$. The dual problem of **SP**, **DSP**, is given below.

$$\begin{aligned} \text{max.} \quad & (\mathbf{b} - \mathbf{By})^T\mathbf{u} && (4.9) \\ \text{s.t.} \quad & \mathbf{A}^T\mathbf{u} \leq \mathbf{c} && (4.10) \\ & \mathbf{u} \text{ unrestricted} && (4.11) \end{aligned}$$

A key characteristic of the above **DSP** is that its solution space does not depend on the value of $\mathbf{y}$, which only affects the objective function. According to the Minkowski's representation theorem, any $\bar{\mathbf{u}}$ satisfying the constraints (4.10) can be expressed as

$$\bar{\mathbf{u}} = \sum_{j \in \mathbf{J}} \lambda_j \mathbf{u}_j^{point} + \sum_{k \in \mathbf{K}} \mu_k \mathbf{u}_k^{ray} \tag{4.12}$$

where $\mathbf{u}_j^{point}$ and $\mathbf{u}_k^{ray}$ represent an extreme point and extreme ray, respectively. In addition, $\lambda_j \geq 0$ for all $j \in \mathbf{J}$ and $\sum_{j \in \mathbf{J}} \lambda_j = 1$, and $\mu_k \geq 0$ for all $k \in \mathbf{K}$. It follows that the **DSP** is equivalent to

$$\max. \quad (\mathbf{b}-\mathbf{B}\mathbf{y})^T(\sum_{j\in\mathbf{J}}\lambda_j\mathbf{u}_j^{point}+\sum_{k\in\mathbf{K}}\mu_k\mathbf{u}_k^{ray}) \tag{4.13}$$

$$\text{s.t.} \quad \sum_{j\in\mathbf{J}}\lambda_j = 1 \tag{4.14}$$

$$\lambda_j \geq 0, \ \forall j \in \mathbf{J} \tag{4.15}$$

$$\mu_k \geq 0, \ \forall k \in \mathbf{K} \tag{4.16}$$

We can therefore conclude that

- The **DSP** becomes unbounded if any $\mathbf{u}_k^{ray}$ exists such that $(\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_k^{ray} > 0$. Note that an unbounded **DSP** implies an infeasible **SP** and to prevent this from happening, we have to ensure that $(\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_k^{ray} \leq 0$ for all $k \in \mathbf{K}$.
- If an optimal solution to **DSP** exists, it must occur at one of the extreme points. Let g denote the optimal objective value, it follows that $(\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_j^{point} \leq g$ for all $j \in \mathbf{J}$.

Based on this idea, the **DSP** can be reformulated as follows:

$$\min. \quad g \tag{4.17}$$

$$\text{s.t.} \quad (\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_k^{ray} \leq 0, \ \forall j \in \mathbf{J} \tag{4.18}$$

$$(\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_j^{point} \leq g, \ \forall k \in \mathbf{K} \tag{4.19}$$

$$j \in \mathbf{J}, k \in \mathbf{K} \tag{4.20}$$

Constraints (4.18) are called **Benders feasibility cuts**, while constraints (4.19) are called **Benders optimality cuts**. Now we are ready to define the Benders Master Problem (**BMP**) as follows:

$$\min. \quad \mathbf{f}^T\mathbf{y} + g \tag{4.21}$$

$$\text{s.t.} \quad (\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_k^{ray} \leq 0, \ \forall j \in \mathbf{J} \tag{4.22}$$

$$(\mathbf{b}-\mathbf{B}\mathbf{y})^T\mathbf{u}_j^{point} \leq g, \ \forall k \in \mathbf{K} \tag{4.23}$$

$$j \in \mathbf{J}, k \in \mathbf{K}, \mathbf{y} \geq 0 \tag{4.24}$$

Typically J and K are too large to enumerate upfront and we have to work with subsets of them, denoted by J_s and K_s, respectively. Hence we have the following Restricted Benders Master Problem (**RBMP**):

$$
\begin{aligned}
\text{min.} \quad & \mathbf{f}^T\mathbf{y} + g && (4.25)\\
\text{s.t.} \quad & (\mathbf{b} - \mathbf{B}\mathbf{y})^T\mathbf{u}_k^{ray} \leq 0,\ \forall j \in \mathbf{J}_s && (4.26)\\
& (\mathbf{b} - \mathbf{B}\mathbf{y})^T\mathbf{u}_j^{point} \leq g,\ \forall k \in \mathbf{K}_s && (4.27)\\
& j \in \mathbf{J}, k \in \mathbf{K}, \mathbf{y} \geq 0 && (4.28)
\end{aligned}
$$

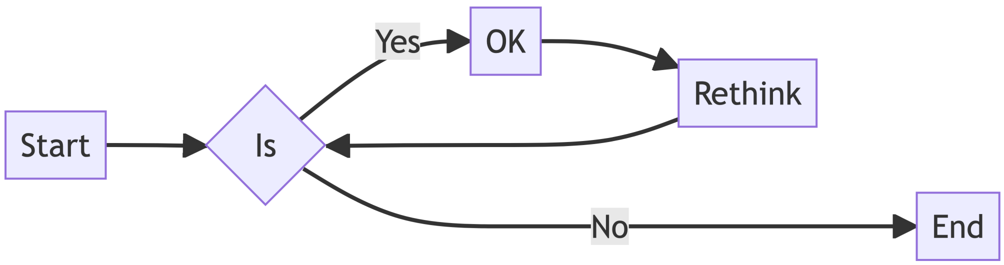

Figure 4.1: Benders decomposition workflow

4.2 Solving Linear Programming Problems with Benders Decomposition

In this section, we use Benders decomposition to solve several linear programming (LP) problems in order to demonstrate the decomposition logic, especially how the restricted Benders master problem interacts with the subproblem in an iterative approach to reach final optimality. Most linear programs could be solved efficiently nowadays by either open source or commercial solvers without resorting to any decomposition approaches. However, by working through the example problems in the following sections, we aim to showcase the implementation details when applying Benders decomposition algorithm on real problems, which helps solidify our understanding of Benders decomposition. Hopefully, by the end of this chapter, we will build up enough intuition as well as hands-on experience such that we are ready to tackle most involved problems in the following chapters.

In the following sections, we employ several steps to illustrate the problem solving process of Benders decomposition.

- We will first create two linear programming solvers based on Gurobi and SCIP that can solve any linear programs defined in the standard form. They are used in later section to validate the correctness of the solutions produced by Benders decomposition.
- Next, we use a specific linear program and give the corresponding RBMP and DSP to prepare for the implementations.

- Then, we will solve the example linear program step by step by examining the outputs of the RBMP and DSP to decided the next set of actions.
- Futhermore, a holistic Benders decomposition implementation is then developed to solve the example linear program.
- Following the previous step, a more generic Benders decomposition implementation is created.
- Then, we will examine an alternative implementation using Gurobi callback functions.
- We will also provide an implementation based on SCIP.
- In the final section, we will do several benchmarking testing.

4.2.1 LP solvers based on Gurobi and SCIP

We aim to use Benders decomposition to solve several linear programming problems in the following sections. To do that, we intentionally decompose the LP problem under consideration into two sets, one set of *complicating* variables and the other set containing the remaining variables. In order to validate the correctness of the results obtained by Benders decomposition, we implement two additional ways of solving the target linear programming problems directly. The first option is based on the Gurobi API in python and the other is based on the open source solve SCIP. The two implementations defined here assume the LP problems under consideration follow the below standard form:

$$\begin{aligned} \text{min.} \quad & \mathbf{c}^T\mathbf{x} && (4.29)\\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} && (4.30)\\ & \mathbf{x} \geq 0 && (4.31) \end{aligned}$$

Listing 4.1 defines a solver for LP problems using Gurobi. It takes three constructor parameters:

- `obj_coeff`: this corresponds to the objective coefficients $\mathbf{c}$.
- `constr_mat`: this refers to the constraint matrix $\mathbf{A}$.
- `rhs`: this is the right-hand side $\mathbf{b}$.

Inside the constructor `__init__()`, a solver environment `_env` is first created and then used to initialize a model object `_model`. The input parameters are then used to create decision variables `_vars`, constraints `_constrs` and objective function respectively. The `optimize()` function simply solves the problem and shows the solving status. Finally, the `clean_up()` function frees up the computing resources.

Listing 4.2 presents an LP solver implementation in class `LpSolverSCIP` using SCIP. The constructor requires the same of parameters as defined in `LpSolverGurobi`. The model building process is similar with minor changes when required to create decision variables, constraints and the objective function.

Listing 4.3 generates a LP problem with 20 decision variables and 5 constraints.

Listing 4.4 solves the generated LP using `LpSolverGurobi` and the solver output shows that an optimal solution was found with objective value of 36.90.

```
Optimal solution found!
Optimal objective = 36.90
```

Listing 4.5 solves the same LP problem using SCIP and not surprisingly, the same optimal objective value was found. This is not exciting, as it only indicates that the two solvers agree on the optimal solution on such a small LP problem as expected. However, they will become more useful in the following sections when we use them to validate our Benders decomposition results.

```
Optimal solution found!
Optimal objective = 36.90
```

4.2.2 A serious LP problem that cannot wait to be decomposed!

With the validation tools available for use, we are ready to solve some serious LP problems using Benders decomposition! What we have below is a LP problem with five decision variables, three of which are denoted by $\mathbf{x} = (x_1, x_2, x_3)$ and the remaining two variables are denoted by $\mathbf{y} = (y_1, y_2)$. We assume that $\mathbf{y}$ are the complicating variables.

$$
\begin{aligned}
\text{min.} \quad & 8x_1 + 12x_2 + 10x_3 + 15y_1 + 18y_2 \\
\text{s.t.} \quad & 2x_1 + 3x_2 + 2x_3 + 4y_1 + 5y_2 = 300 \\
& 4x_1 + 2x_2 + 3x_3 + 2y_1 + 3y_2 = 220 \\
& x_i \geq 0, \ \forall i = 1, \cdots, 3 \\
& y_i \geq 0, \ \forall j = 1, 2
\end{aligned}
$$

According to the standard LP form presented in the previous section, $\mathbf{c}^T = (8, 12, 10)$, $\mathbf{f}^T = (15, 18)$ and $\mathbf{b}^T = (300, 220)$. In addition,

$$
\mathbf{A} = \begin{bmatrix} 2 & 3 & 2 \\ 4 & 2 & 3 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 4 & 5 \\ 2 & 3 \end{bmatrix}
$$

Listing 4.6 solves the LP problem using the two solvers define above. Both solvers confirm the optimal objective value is 1091.43. With this information in mind, we will apply Benders decomposition to see if the same optimal solution could be identified or not.

```
Optimal solution found!
Optimal objective = 1091.43
Optimal solution found!
Optimal objective = 1091.43
```

4.2.3 Benders decomposition formulations

With **y** being the complicating variable, we state the Benders subproblem (**SP**) below for the **y** assuming fixed values $\bar{\mathbf{y}} = (\bar{y_1}, \bar{y_2})$:

$$\begin{aligned} \text{min.} \quad & 8x_1 + 12x_2 + 10x_3 \\ \text{s.t.} \quad & 2x_1 + 3x_2 + 2x_3 = 300 - 4\bar{y_1} - 5\bar{y_2} \\ & 4x_1 + 2x_2 + 3x_3 = 220 - 2\bar{y_1} - 3\bar{y_2} \\ & x_i \geq 0, \ \forall i = 1, \cdots, 3 \end{aligned}$$

We define the dual variable $\mathbf{u} = (u_1, u_2)$ to associate with the constraints in the (**SP**). The dual subproblem (**DSP**) could then be stated as follows:

$$\begin{aligned} \text{max.} \quad & (300 - 4\bar{y_1} - 5\bar{y_2})u_1 + (220 - 2\bar{y_1} - 3\bar{y_2})u_2 \\ \text{s.t.} \quad & 2u_1 + 4u_2 \leq 8 \\ & 3u_1 + 2u_2 \leq 12 \\ & 2u_1 + 3u_2 \leq 10 \\ & u_1, u_2 \text{ unrestricted} \end{aligned}$$

The (**RBMP**) can be stated as below. Note that $\mathbf{u} = (0, 0)$ is a feasible solution to (**DSP**) and the corresponding objective value is 0, which is the reason we restrict the variable g to be nonnegative.

$$\begin{aligned} \text{min.} \quad & 15y_1 + 18y_2 + g \\ \text{s.t.} \quad & y_1, y_2 \geq 0 \\ & g \geq 0 \end{aligned}$$

4.2.4 Benders decomposition step by step

Benders decomposition defines a problem solving process in which the restricted Benders master problem and the dual subproblem interact iteratively to identify the optimal solution or conclude infeasibility/unboundedness. To facilitate our understanding of the process, we demonstrate in this section the workings of Benders decomposition by solving the target LP problem step by step.

Listing 4.7 shows the codes that initialize the lower bound `lb`, upper bound `ub` and threshold value `eps`. Furthermore, the restricted Benders master problem `rbmp` is created with three variables and a minimizing objective function. Note that no constraints are yet included in the model at this moment.

?@lst-bd-dsp initializes the Benders subproblem with two decision variables and three constraints.

```
#| lst-label: lst-bd-dsp
#| lst-cap: Dual subproblem initialization

# create dual subproblem
dsp = gp.Model(env=env, name='DSP')

# create decision variables
u1 = dsp.addVar(vtype=GRB.CONTINUOUS,
                lb=-GRB.INFINITY,
                ub=GRB.INFINITY,
                name='u1')
u2 = dsp.addVar(vtype=GRB.CONTINUOUS,
                lb=-GRB.INFINITY,
                ub=GRB.INFINITY,
                name='u2')

# create objective function
dsp.setObjective(300*u1 + 220*u2)

# create constraints
dsp.addConstr(2*u1 + 4*u2 <= 8, name='c1')
dsp.addConstr(3*u1 + 2*u2 <= 12, name='c2')
dsp.addConstr(2*u1 + 3*u2 <= 10, name='c3')

dsp.update()
```

In Listing 4.8, we solve the (**RBMP**) for the first time. It has an optimal solution with $(\bar{y_1}, \bar{y_2}, \bar{g}) = (0, 0, 0)$ and optimal objective value of 0. This is expected as all the variables assume their minimal possible values in order to minimize the objective function. This objective value also serves as the new lower bound.

```
Optimal solution found! Objective value = 0.00
(y1, y2, g) = (0.00, 0.00, 0.00)
lb=0.0, ub=1e+100
```

Given that $(\bar{y_1}, \bar{y_2}, \bar{g}) = (0, 0, 0)$, we now feed the values of $\bar{y_1}$ and $\bar{y_2}$ into the Benders dual subproblem **(DSP)** by updating its objective function, as shown in Listing 4.9:

```
Optimal objective = 1200.00
(u1, u2) = (4.00, 0.00)
lb=0.0, ub=1200.0
```

We see that the dual subproblem has an optimal solution. The upper bound `ub` is also updated. Since the optimal objective value of the subproblem turns out to be 1200 and is greater than $\bar{g} = 0$, which implies that an optimality cut is needed to make sure that the variable g in the restricted Benders master problem reflects this newly obtained information from the subproblem.

In Listing 4.10, the new optimality cut is added to the (**RBMP**), which is then solved to optimality.

```
Optimal solution found! Objective value = 1080.00
(y1, y2, g) = (0.00, 60.00, 0.00)
lb=1080.0, ub=1200.0
```

Armed with the optimal solution $(\bar{y_1}, \bar{y_2}, \bar{g}) = (0, 60, 0)$, **?@lst-bd-iter2-dsp** updates the objective function of the (**DSP**) and obtains its optimal solution.

```
#| lst-label: lst-bd-iter2-dsp
#| lst-cap: Iteration 2 - solving the dual subproblem

# update objective function
dsp.setObjective((300-4*y1_opt-5*y2_opt)*u1
                 + (220-2*y1_opt-3*y2_opt)*u2,
                 GRB.MAXIMIZE)
dsp.update()
```

```python
dsp.optimize()

if dsp.status == GRB.OPTIMAL:
    u1_opt, u2_opt = u1.X, u2.X

    print(f'Optimal objective = {dsp.objVal:.2f}')
    print(f'(u1, u2) = ({u1_opt:.2f}, {u2_opt:.2f})')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb}, ub={ub:.2f}')
elif dsp.Status == GRB.UNBOUNDED:
    print(f'DSP is unbounded!')
    u1_ray = u1.UnbdRay
    u2_ray = u2.UnbdRay
    print(f'retrieve extreme ray (u1, u2) = ({u1_ray}, {u2_ray})')
else:
    print(f'DSP solve error')
```

```
DSP is unbounded!
retrieve extreme ray (u1, u2) = (-2.0, 1.0)
```

Since the dual subproblem is unbounded, a feasibility cut is further needed. In Listing 4.11, we add the new cut and solve the restricted Benders master problem again.

```
Optimal solution found! Objective value = 1091.43
(y1, y2, g) = (0.00, 54.29, 114.29)
lb=1091.43, ub=1200.00
```

Note that a new lower bound is obtained after solving the master problem. Since there is still a large gap between the lower bound and upper bound, we continue solving the subproblem in Listing 4.12.

```
Optimal objective = 114.29
(u1, u2) = (4.00, 0.00)
lb=1091.43, ub=1091.43
```

Now that the difference between `lb` and `ub` is less than the preset threshold `eps`, we conclude that an optimal solution is reached and the computation resources are freed up.

```
# release resources
rbmp.dispose()
dsp.dispose()
env.dispose()
```

4.2.5 Putting it together

It typically takes Benders decomposition many iterations to reach optimality or conclude infeasibility/unboundedness. In this section, we put everything we have learned from the manual approach above into an automatic workflow.

Listing 4.13 defines an `Enum` class that specifies four possible optimization statuses. The meanings of these statuses are self-explanatory from their corresponding names and further explanations are omitted here.

Listing 4.14 defines a `MasterSolver` class that models the (**RBMP**). Its constructor contains the variable and objective function definitions. The ability to take in either feasibility or optimality cuts is implemented in separate functions `add_feasibility_cut()` and `add_optimality_cut()`, respectively. The `solve()` function is responsible for optimizing the model and retrieve the optimal solution if any.

?@lst-bd-whole-dsp defines the Benders dual subproblem in a similar fashion. Notice that the `update_objective()` function is used to set an updated objective function based on the optimal solution identified in the restricted Benders master problem.

```
#| lst-label: lst-bd-whole-dsp
#| lst-cap: Dual subproblem model

class DualSubprobSolver:

    def __init__(self, env):
        self._model = gp.Model(env=env, name='DSP')

        # create decision variables
        self._u1 = self._model.addVar(vtype=GRB.CONTINUOUS,
                                      lb=-GRB.INFINITY,
                                      ub=GRB.INFINITY,
                                      name='u1')
        self._u2 = self._model.addVar(vtype=GRB.CONTINUOUS,
                                      lb=-GRB.INFINITY,
                                      ub=GRB.INFINITY,
                                      name='u2')
```

```python
        # create constraints
        self._model.addConstr(2*self._u1+4*self._u2 <= 8, name='c1')
        self._model.addConstr(3*self._u1+2*self._u2 <= 12, name='c2')
        self._model.addConstr(2*self._u1+3*self._u2 <= 10, name='c3')

        self._model.setObjective(1, GRB.MAXIMIZE)
        self._model.update()

        self._opt_obj = None
        self._opt_u1 = None
        self._opt_u2 = None
        self._ray_u1 = None
        self._ray_u2 = None

    def solve(self):
        print('-' * 50)
        print(f'Start solving dual subproblem.')
        self._model.optimize()

        status = None
        if self._model.status == GRB.OPTIMAL:
            self._opt_obj = self._model.objVal
            self._opt_u1 = self._u1.X
            self._opt_u2 = self._u2.X
            status = OptStatus.OPTIMAL
            print(f'\tdual subproblem is optimal.')
            print(f'\topt_obj={self._opt_obj:.2f}')
            print(f'\topt_y1={self._opt_u1:.2f}, opt_y2={self._opt_u2:.2f}')
        elif self._model.status == GRB.UNBOUNDED:
            status = OptStatus.UNBOUNDED
            print(f'\tdual subproblem is unbounded!')
            self._ray_u1 = self._u1.UnbdRay
            self._ray_u2 = self._u2.UnbdRay
            print(f'\textreme ray (u1, u2) = ({self._ray_u1}, {self._ray_u2})')
        else:
            status = OptStatus.ERROR

        print(f'Finish solving dual subproblem.')
        print('-' * 50)
        return status

    def update_objective(self, opt_y1, opt_y2):
```

```
        self._model.setObjective(
            (300-4*opt_y1-5*opt_y2)*self._u1 +
            (220-2*opt_y1-3*opt_y2)*self._u2,
            GRB.MAXIMIZE)
        print(f'dual subproblem objective updated!')

    def clean_up(self):
        self._model.dispose()

    @property
    def opt_obj(self):
        return self._opt_obj

    @property
    def opt_u1(self):
        return self._opt_u1

    @property
    def opt_u2(self):
        return self._opt_u2

    @property
    def ray_u1(self):
        return self._ray_u1

    @property
    def ray_u2(self):
        return self._ray_u2
```

?@lst-bd-whole-workflow shows the control flow of Benders decomposition. The main logic is stated as a `while` loop, in which the master problem and dual subproblem are solved sequentially within each iteration. Depending on whether the subproblem is optimal or unbounded, an optimality or feasibility cut is added to the master problem. The process continues until the gap between the lower bound and the upper bound is within a certain threshold.

```
#| lst-label: lst-bd-whole-workflow
#| lst-cap: Benders decomposition control flow

class BendersDecomposition:

    def __init__(self, master_solver, dual_subprob_solver):
```

```
        self._master_solver = master_solver
        self._dual_subprob_solver = dual_subprob_solver


    def optimize(self) -> OptStatus:
        eps = 1.0e-5
        lb = -np.inf
        ub = np.inf

        iter = 1
        while True:
            print(f"\nIteration: {iter}")
            iter += 1
            # solve master problem
            master_status = self._master_solver.solve()
            if master_status == OptStatus.INFEASIBLE:
                return OptStatus.INFEASIBLE

            # update lower bound
            lb = np.max([lb, self._master_solver.opt_obj])
            print(f'Bounds: lb={lb:.2f}, ub={ub:.2f}')

            opt_y1 = self._master_solver.opt_y1
            opt_y2 = self._master_solver.opt_y2

            # solve subproblem
            self._dual_subprob_solver.update_objective(opt_y1, opt_y2)
            dsp_status = self._dual_subprob_solver.solve()

            if dsp_status == OptStatus.OPTIMAL:
                # update upper bound
                opt_obj = self._dual_subprob_solver.opt_obj
                ub = np.min([ub, 15*opt_y1 + 18*opt_y2 + opt_obj])
                print(f'Bounds: lb={lb:.2f}, ub={ub:.2f}')

                if ub - lb <= eps:
                    break

                opt_u1 = self._dual_subprob_solver.opt_u1
                opt_u2 = self._dual_subprob_solver.opt_u2
                self._master_solver.add_optimality_cut(opt_u1, opt_u2)
            elif dsp_status == OptStatus.UNBOUNDED:
```

```
                ray_u1 = self._dual_subprob_solver.ray_u1
                ray_u2 = self._dual_subprob_solver.ray_u2
                self._master_solver.add_feasibility_cut(ray_u1, ray_u2)
```

Listing 4.15 solves the LP problem using the wholesome Benders solver. The optimal solution agrees with the solution obtained in the manual approach.

```
Iteration: 1
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=0.00
    opt_y1=0.00, opt_y2=0.00
    opt_g=0.00
Finish solving master problem.
--------------------------------------------------
Bounds: lb=0.00, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
    dual subproblem is optimal.
    opt_obj=1200.00
    opt_y1=4.00, opt_y2=0.00
Finish solving dual subproblem.
--------------------------------------------------
Bounds: lb=0.00, ub=1200.00
Benders optimality cut added!

Iteration: 2
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=1080.00
    opt_y1=0.00, opt_y2=60.00
    opt_g=0.00
Finish solving master problem.
--------------------------------------------------
Bounds: lb=1080.00, ub=1200.00
dual subproblem objective updated!
--------------------------------------------------
```

```
Start solving dual subproblem.
    dual subproblem is unbounded!
    extreme ray (u1, u2) = (-2.0, 1.0)
Finish solving dual subproblem.
--------------------------------------------------
Benders feasibility cut added!

Iteration: 3
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=1091.43
    opt_y1=0.00, opt_y2=54.29
    opt_g=114.29
Finish solving master problem.
--------------------------------------------------
Bounds: lb=1091.43, ub=1200.00
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
    dual subproblem is optimal.
    opt_obj=114.29
    opt_y1=4.00, opt_y2=0.00
Finish solving dual subproblem.
--------------------------------------------------
Bounds: lb=1091.43, ub=1091.43
```

4.2.6 A generic LP Benders decomposition solver

The Benders decomposition solver we put together in the last section is a big step toward automating the interaction between the master solver and the dual subproblem solver. However, it is still rather limited in that both the master and subproblem solvers are tied to a specific problem. In this section, we aim to develop a more generic solver that is able to solver any LP problem following a specific form.

Listing 4.16 implements a LP solver class named `GenericLpMasterSolver` for (**RBMP**) characterized by the cost coefficient `f`, the constraint matrix `B` and right-hand side `b`. The constructor initializes the complicating variables `_y` and the dummy variable `_g`. The `solve()` function solves the problem and retrieves the optimal solution if exists. Moreover, the solver provides functions to add optimality and feasibility cuts to the existing model.

Listing 4.17 presents a solver for (**DSP**) that's defined by the objective function coefficient `c` and constraint matrix `A`. The model objective function could be updated by `update_objective()` with the latest value of `y`. Notice that, the optimal solution is saved to `_opt_u` if the underlying problem is optimal. Otherwise, an extreme ray is retrieved and stored in `_extreme_ray`.

Listing 4.18 generalizes the Benders decomposition control flow in **?@lst-bd-whole-workflow**.

(**generic-benders-test-1?**) solves the LP problem that we have been tackling in the previous sections. The output confirms that the same optimal solution is identified as in the last section.

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

c = np.array([8, 12, 10])
f = np.array([15, 18])
A = np.array([
    [2, 3, 2],
    [4, 2, 3]
])
B = np.array([
    [4, 5],
    [2, 3],
])
b = np.array([300, 220])

master_solver = GenericLpMasterSolver(f, B, b)
dual_subprob_solver = GenericLpSubprobSolver(A, c, B, b)

benders_solver = GenericBendersSolver(master_solver, dual_subprob_solver)
benders_solver.optimize()
```

```
Iteration: 1
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=0.00
    opt_g=0.00
    opt_y0=0.0
    opt_y1=0.0
Finish solving master problem.
```

```
--------------------------------------------------
Bounds: lb=0.00, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
    dual subproblem is optimal.
    opt_obj=1200.00
    opt_u0=4.0
    opt_u1=0.0
Finish solving dual subproblem.
--------------------------------------------------
DSP is optimal!
Bounds: lb=0.00, ub=1200.00
Benders optimality cut added!

Iteration: 2
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=1080.00
    opt_g=0.00
    opt_y0=0.0
    opt_y1=60.0
Finish solving master problem.
--------------------------------------------------
Bounds: lb=1080.00, ub=1200.00
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
{0: -2.0, 1: 1.0}
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 3
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=1091.43
    opt_g=114.29
    opt_y0=0.0
```

```
    opt_y1=54.285714285714285
Finish solving master problem.
--------------------------------------------------
Bounds: lb=1091.43, ub=1200.00
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
    dual subproblem is optimal.
    opt_obj=114.29
    opt_u0=4.0
    opt_u1=0.0
Finish solving dual subproblem.
--------------------------------------------------
DSP is optimal!
Bounds: lb=1091.43, ub=1091.43
```

4.2.6.1 Detect infeasibility

```python
import numpy as np

np.random.seed(142)
c = np.random.randint(2, 6, size=20)
f = np.random.randint(1, 15, size=10)
A = np.random.randint(2, 6, size=(20, 20))
B = np.random.randint(2, 26, size=(20, 10))
b = np.random.randint(20, 50, size=20)

obj_coeff = np.concatenate([c, f])
constr_mat = np.concatenate([A, B], axis=1)
rhs = b
```

```python
gurobi_solver = LpSolverGurobi(obj_coeff, constr_mat, rhs)
gurobi_solver.save_model('problem2.lp')
gurobi_solver.optimize()
```

```
Model is infeasible!
```

```python
scip_solver = LpSolverSCIP(obj_coeff, constr_mat, rhs)
scip_solver.optimize()
```

```
Model is infeasible!
```

```
master_solver = GenericLpMasterSolver(f, B, b)
dual_subprob_solver = GenericLpSubprobSolver(A, c, B, b)

benders_solver = GenericBendersSolver(master_solver, dual_subprob_solver)
benders_solver.optimize()
```

```
Iteration: 1
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=0.00
    opt_g=0.00
    opt_y0=0.0
    opt_y1=0.0
    opt_y2=0.0
    opt_y3=0.0
    opt_y4=0.0
    opt_y5=0.0
    opt_y6=0.0
    opt_y7=0.0
    opt_y8=0.0
    opt_y9=0.0
Finish solving master problem.
--------------------------------------------------
Bounds: lb=0.00, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 2
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=1.36
```

```
    opt_g=0.00
    opt_y0=0.0
    opt_y1=0.0
    opt_y2=0.0
    opt_y3=0.0
    opt_y4=0.0
    opt_y5=0.27102077358389676
    opt_y6=0.0
    opt_y7=0.0
    opt_y8=0.0
    opt_y9=0.0
Finish solving master problem.
--------------------------------------------------
Bounds: lb=1.36, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 3
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=5.66
    opt_g=0.00
    opt_y0=0.0
    opt_y1=0.0
    opt_y2=0.0
    opt_y3=0.0
    opt_y4=0.6681871194041935
    opt_y5=0.7309128358300898
    opt_y6=0.0
    opt_y7=0.0
    opt_y8=0.0
    opt_y9=0.0
Finish solving master problem.
--------------------------------------------------
Bounds: lb=5.66, ub=inf
dual subproblem objective updated!
```

```
--------------------------------------------------
Start solving dual subproblem.
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 4
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=10.13
    opt_g=0.00
    opt_y0=0.0
    opt_y1=0.0
    opt_y2=0.0
    opt_y3=0.0
    opt_y4=0.0
    opt_y5=1.109793111809558
    opt_y6=0.0
    opt_y7=0.0
    opt_y8=0.0
    opt_y9=1.5284312730742948
Finish solving master problem.
--------------------------------------------------
Bounds: lb=10.13, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 5
--------------------------------------------------
Start solving master problem.
    master problem is optimal.
    opt_obj=11.19
    opt_g=0.00
    opt_y0=0.0
```

```
    opt_y1=0.0
    opt_y2=0.06379053971371684
    opt_y3=0.0
    opt_y4=0.0
    opt_y5=0.7474948667377971
    opt_y6=0.0
    opt_y7=1.350013829440734
    opt_y8=0.0
    opt_y9=0.5128787352623962
Finish solving master problem.
--------------------------------------------------
Bounds: lb=11.19, ub=inf
dual subproblem objective updated!
--------------------------------------------------
Start solving dual subproblem.
dual subproblem is unbounded
Finish solving dual subproblem.
--------------------------------------------------
DSP is unbounded!!!
Benders feasibility cut added!

Iteration: 6
--------------------------------------------------
Start solving master problem.
    master problem is infeasible.
Finish solving master problem.
--------------------------------------------------
Model is infeasible!

<OptStatus.INFEASIBLE: 2>
```

4.2.7 Implementation with callbacks

4.2.8 Implementation with SCIP

```
from pyscipopt import Model
from pyscipopt import quicksum
from pyscipopt import SCIP_PARAMSETTING

# Create a model
```

```python
model = Model("simple_lp")

# Define variables
x1 = model.addVar(lb=0, vtype="C", name="x1")
x2 = model.addVar(lb=0, vtype="C", name="x2")

# Set objective function
model.setObjective(x1 + x2, "maximize")

# Add constraints
# model.addCons(2 * x1 + x2 >= 1, "constraint1")
model.addCons(x1 + x2 >= 2, "constraint2")

# Solve the model
model.setPresolve(SCIP_PARAMSETTING.OFF)
model.setHeuristics(SCIP_PARAMSETTING.OFF)
model.disablePropagation()
model.optimize()

# Print results
status = model.getStatus()
print(f'status = {status}')
if model.getStatus() == "optimal":
    print("Optimal solution found.")
    print(f"x1: {model.getVal(x1):.2f}")
    print(f"x2: {model.getVal(x2):.2f}")
    print(f"Objective value: {model.getObjVal():.2f}")
    hasRay = model.hasPrimaryRay()
    print(hasRay)
elif model.getStatus() == 'unbounded':
    hasRay = model.hasPrimalRay()
    print(f'hasRay={hasRay}')
    ray = model.getPrimalRay()
    print(f'ray={ray}')

else:
    print("Model could not be solved.")
```

```
status = unbounded
hasRay=True
ray=[0.5, 0.5]
presolving:
```

```
   (0.0s) symmetry computation started: requiring (bin +, int +, cont +), (fixed: bin -, int
   (0.0s) symmetry computation finished: 1 generators found (max: 1500, log10 of symmetry gro
   (0.0s) no symmetry on binary variables present.
presolving (0 rounds: 0 fast, 0 medium, 0 exhaustive):
 0 deleted vars, 0 deleted constraints, 0 added constraints, 0 tightened bounds, 0 added hole
 0 implications, 0 cliques
presolved problem has 2 variables (0 bin, 0 int, 0 impl, 2 cont) and 2 constraints
      2 constraints of type <linear>
Presolving Time: 0.00

 time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons |rows |cuts |sepa|confs|strb
* 0.0s|     1 |     0 |     2 |     - |    LP  |   0 |   2 |   2 |   2 |   1 |  0 |   0 |   0

SCIP Status        : problem is solved [unbounded]
Solving Time (sec) : 0.00
Solving Nodes      : 1
Primal Bound       : +1.00000000000000e+20 (1 solutions)
Dual Bound         : +1.00000000000000e+20
Gap                : 0.00 %
```

Testing

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Listing 4.1 A LP solver based on Gurobi

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

class LpSolverGurobi:

    def __init__(self, obj_coeff, constr_mat, rhs, verbose=False):
        # initialize environment and model
        self._env = gp.Env('GurobiEnv', empty=True)
        # self._env.setParam('LogToConsole', 1 if verbose else 0)
        self._env.setParam('OutputFlag', 1 if verbose else 0)
        self._env.start()
        self._model = gp.Model(env=self._env, name='GurobiLpSolver')

        # prepare data
        self._obj_coeff = obj_coeff
        # print(self._obj_coeff)
        self._constr_mat = constr_mat
        # print(self._constr_mat)
        self._rhs = rhs
        self._num_vars = len(self._obj_coeff)
        self._num_constrs = len(self._rhs)

        # create decision variables
        self._vars = self._model.addMVar(self._num_vars,
                                         vtype=GRB.CONTINUOUS,
                                         lb=0)

        # create constraints
        self._constrs = self._model.addConstr(
            self._constr_mat@self._vars == self._rhs
        )

        # create objective
        self._model.setObjective(self._obj_coeff@self._vars,
                                 GRB.MINIMIZE)

    def optimize(self, verbose=False):
        self._model.optimize()
        if self._model.status == GRB.OPTIMAL:
            print(f'Optimal solution found!')
            print(f'Optimal objective = {self._model.objVal:.2f}')
        elif self._model.status == GRB.UNBOUNDED:
            print(f'Model is unbounded!')
        elif self._model.status == GRB.INFEASIBLE:
            print(f'Model is infeasible!')
        else:
            print(f'Unknown error occurred!')

    def save_model(self, filename):
        self._model.write(filename)
```

Listing 4.2 A LP solver based on SCIP

```python
import pyscipopt as scip
from pyscipopt import SCIP_PARAMSETTING

class LpSolverSCIP:

    def __init__(self, obj_coeff, constr_mat, rhs, verbose=False):
        self._model = scip.Model('LpModel')
        if not verbose:
            self._model.hideOutput()

        # create variables
        self._vars = {
            i: self._model.addVar(lb=0, vtype='C')
            for i in range(len(obj_coeff))
        }

        # create constraints
        for c in range(len(rhs)):
            expr = [
                constr_mat[c][j] * self._vars.get(j)
                for j in range(len(obj_coeff))
            ]
            self._model.addCons(scip.quicksum(expr) == rhs[c])

        # create objective
        obj_expr = [
            obj_coeff[i] * self._vars.get(i)
            for i in range(len(obj_coeff))
        ]
        self._model.setObjective(scip.quicksum(obj_expr), "minimize")

    def optimize(self):
        self._model.optimize()
        status = self._model.getStatus()
        if status == "optimal":
            print(f'Optimal solution found!')
            print(f'Optimal objective = {self._model.getObjVal():.2f}')
        elif status == "unbounded":
            print(f'Model is unbounded!')
        elif status == "infeasible":
            print(f'Model is infeasible!')
        else:
            print(f'Unknown error occurred!')
```

Listing 4.3 A randomly generated LP problem

```
import numpy as np

np.random.seed(42)
c = np.random.randint(1, 6, size=20)
A = np.random.randint(-10, 12, size=(5, 20))
b = np.random.randint(20, 100, size=5)
```

Listing 4.4 Solving the generated LP with Gurobi

```
lpsolver_gurobi = LpSolverGurobi(obj_coeff=c, constr_mat=A, rhs=b) # <1>
lpsolver_gurobi.optimize()
```

Listing 4.5 Solving the generated LP with SCIP

```
lpsolver_scip = LpSolverSCIP(obj_coeff=c, constr_mat=A, rhs=b)
lpsolver_scip.optimize()
```

Listing 4.6 Solve the LP problem with Gurobi and SCIP

```
c = np.array([8, 12, 10])
f = np.array([15, 18])
obj_coeff = np.concatenate([c, f])

A = np.array([[2, 3, 2],
     [4, 2, 3]])
B = np.array([[4, 5],
     [2, 3]])
constr_mat = np.concatenate([A, B], axis=1)

rhs = np.array([300, 220])

lpsolver_gurobi = LpSolverGurobi(obj_coeff, constr_mat, rhs, verbose=False)
lpsolver_gurobi.optimize()
# Optimal objective = 1091.43

lpsolver_scip = LpSolverSCIP(obj_coeff, constr_mat, rhs, verbose=False)
lpsolver_scip.optimize()
# Optimal objective = 1091.43
```

Listing 4.7 Gurobi solver setup and restricted master problem initialization

```python
import numpy as np
import gurobipy as gp
from gurobipy import GRB

# initialize lower/upper bounds and threshold value
lb = -GRB.INFINITY
ub = GRB.INFINITY
eps = 1.0e-5

# create restricted Benders master problem
env = gp.Env('benders', empty=True) # <1>
env.setParam('OutputFlag', 0)
env.start()
rbmp = gp.Model(env=env, name='RBMP')

# create decision variables
y1 = rbmp.addVar(vtype=GRB.CONTINUOUS, lb=0, name='y1')
y2 = rbmp.addVar(vtype=GRB.CONTINUOUS, lb=0, name='y2')
g = rbmp.addVar(vtype=GRB.CONTINUOUS, lb=0, name='g')

# create objective
rbmp.setObjective(15*y1 + 18*y2 + g, GRB.MINIMIZE)
```

Listing 4.8 Iteration 1 - solving the restricted Benders master problem

```python
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'Optimal solution found! Objective value = {rbmp.objVal:.2f}')

    y1_opt, y2_opt, g_opt = y1.X, y2.X, g.X
    lb = np.max([lb, rbmp.objVal])

    print(f'(y1, y2, g) = ({y1_opt:.2f}, {y2_opt:.2f}, {g_opt:.2f})')
    print(f'lb={lb}, ub={ub}')
elif rbmp.status == GRB.INFEASIBLE:
    print(f'original problem is infeasible!')
```

Listing 4.9 Iteration 1 - solving the dual subproblem

```
# update objective function
dsp.setObjective((300-4*y1_opt-5*y2_opt)*u1 +
                 (220-2*y1_opt-3*y2_opt)*u2,
                 GRB.MAXIMIZE)
dsp.update()
dsp.optimize()

if dsp.status == GRB.OPTIMAL:
    u1_opt, u2_opt = u1.X, u2.X

    print(f'Optimal objective = {dsp.objVal:.2f}')
    print(f'(u1, u2) = ({u1_opt:.2f}, {u2_opt:.2f})')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb}, ub={ub}')
```

Listing 4.10 Iteration 2 - solving the restricted Benders master problem

```
# add optimality cut
rbmp.addConstr((300-4*y1-5*y2)*u1_opt
               + (220-2*y1-3*y2)*u2_opt <= g,
               name='c1')
rbmp.update()
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'Optimal solution found! Objective value = {rbmp.objVal:.2f}')

    y1_opt, y2_opt, g_opt = y1.X, y2.X, g.X
    lb = np.max([lb, rbmp.objVal])

    print(f'(y1, y2, g) = ({y1_opt:.2f}, {y2_opt:.2f}, {g_opt:.2f})')
    print(f'lb={lb}, ub={ub}')
elif rbmp.status == GRB.INFEASIBLE:
    print(f'original problem is infeasible!')
```

Listing 4.11 Iteration 3 - solving the restricted Benders master problem

```
# add optimality cut
rbmp.addConstr((300-4*y1-5*y2)*u1_ray
               + (220-2*y1-3*y2)*u2_ray <= 0,
               name='c2')
rbmp.update()
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'Optimal solution found! Objective value = {rbmp.objVal:.2f}')

    y1_opt, y2_opt, g_opt = y1.X, y2.X, g.X
    lb = np.max([lb, rbmp.objVal])

    print(f'(y1, y2, g) = ({y1_opt:.2f}, {y2_opt:.2f}, {g_opt:.2f})')
    print(f'lb={lb:.2f}, ub={ub:.2f}')
elif rbmp.status == GRB.INFEASIBLE:
    print(f'original problem is infeasible!')
```

Listing 4.12 Iteration 3 - solving the dual subproblem

```
# update objective function
dsp.setObjective((300-4*y1_opt-5*y2_opt)*u1
                 + (220-2*y1_opt-3*y2_opt)*u2,
                 GRB.MAXIMIZE)
dsp.update()
dsp.optimize()

if dsp.status == GRB.OPTIMAL:
    u1_opt, u2_opt = u1.X, u2.X

    print(f'Optimal objective = {dsp.objVal:.2f}')
    print(f'(u1, u2) = ({u1_opt:.2f}, {u2_opt:.2f})')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb:.2f}, ub={ub:.2f}')
```

Listing 4.13 Optimization status

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np
from enum import Enum

class OptStatus(Enum):
    OPTIMAL = 0
    UNBOUNDED = 1
    INFEASIBLE = 2
    ERROR = 3
```

Listing 4.14 Restricted Benders master model

```
class MasterSolver:

    def __init__(self, env):
        self._model = gp.Model(env=env, name='RBMP')

        # create decision variables
        self._y1 = self._model.addVar(vtype=GRB.CONTINUOUS,
                                      lb=0, name='y1')
        self._y2 = self._model.addVar(vtype=GRB.CONTINUOUS,
                                      lb=0, name='y2')
        self._g = self._model.addVar(vtype=GRB.CONTINUOUS,
                                     lb=0, name='g')

        # create objective
        self._model.setObjective(15*self._y1+18*self._y2+self._g,
                                 GRB.MINIMIZE)

        self._opt_obj = None
        self._opt_y1 = None
        self._opt_y2 = None
        self._opt_g = None

    def solve(self) -> OptStatus:
        print('-' * 50)
        print(f'Start solving master problem.')
        self._model.optimize()

        opt_status = None
        if self._model.status == GRB.OPTIMAL:
            opt_status = OptStatus.OPTIMAL
            self._opt_obj = self._model.objVal
            self._opt_y1 = self._y1.X
            self._opt_y2 = self._y2.X
            self._opt_g = self._g.X
            print(f'\tmaster problem is optimal.')
            print(f'\topt_obj={self._opt_obj:.2f}')
            print(f'\topt_y1={self._opt_y1:.2f}, opt_y2={self._opt_y2:.2f}')
            print(f'\topt_g={self._opt_g:.2f}')
        elif self._model.status == GRB.INFEASIBLE:
            print(f'\tmaster problem is infeasible.')
            opt_status = OptStatus.INFEASIBLE
        else:
            print(f'\tmaster problem encountered error.')
            opt_status = OptStatus.ERROR
```



```
        print(f'Finish solving master problem.')
        print('-' * 50)
        return opt_status

    def add_feasibility_cut(self, ray_u1, ray_u2) -> None:
        self._model.addConstr((300-4*self._y1-5*self._y2)*ray_u1 +
```

Listing 4.15 Solving the LP problem using Benders decomposition

```
env = gp.Env('benders', empty=True)
env.setParam("OutputFlag",0)
env.start()
master_solver = MasterSolver(env)
dual_subprob_solver = DualSubprobSolver(env)

benders_decomposition = BendersDecomposition(master_solver, dual_subprob_solver)
benders_decomposition.optimize()
```

Listing 4.16 Generic Benders master problem solver for LP problems

```
class GenericLpMasterSolver:

    def __init__(self, f: np.array, B: np.array, b: np.array):
        # save data
        self._f = f
        self._B = B
        self._b = b

        # env and model
        self._env = gp.Env('MasterEnv', empty=True)
        self._env.setParam("OutputFlag",0)
        self._env.start()
        self._model = gp.Model(env=self._env, name='MasterSolver')

        # create variables
        self._num_y_vars = len(f)
        self._y = self._model.addVars(self._num_y_vars,
                                       lb=0,
                                       vtype=GRB.CONTINUOUS,
                                       name='y')
        self._g = self._model.addVar(vtype=GRB.CONTINUOUS,
                                      lb=0,
                                      name='g')

        # create objective
        self._model.setObjective(
            gp.quicksum(self._f[i] * self._y.get(i)
                        for i in range(self._num_y_vars)) +
            self._g,
            GRB.MINIMIZE)
        self._model.update()

        self._opt_obj = None
        self._opt_obj_y = None
        self._opt_y = None
        self._opt_g = None

    def solve(self) -> OptStatus:
        print('-' * 50)
        print(f'Start solving master problem.')
        self._model.optimize()

        opt_status = None
        if self._model.status == GRB.OPTIMAL:
            opt_status = OptStatus.OPTIMAL
            self._opt_obj = self._model.objVal
            self._opt_y = {
                i: self._y.get(i).X
                for i in range(self._num_y_vars)
            }
            self._opt_g = self._g.X
```

Listing 4.17 Generic Benders dual subproblem solver for LP problems

```
class GenericLpSubprobSolver:

    def __init__(self, A: np.array, c: np.array, B: np.array, b: np.array):
        # save data
        self._A = A
        self._c = c
        self._b = b
        self._B = B

        # env and model
        self._env = gp.Env('SubprobEnv', empty=True)
        self._env.setParam("OutputFlag",0)
        self._env.start()
        self._model = gp.Model(env=self._env, name='SubprobSolver')

        # create variables
        self._num_vars = len(b)
        self._u = self._model.addVars(self._num_vars,
                                      vtype=GRB.CONTINUOUS,
                                      lb=-GRB.INFINITY,
                                      ub=GRB.INFINITY,
                                      name='u')

        # create constraints
        for c_idx in range(len(c)):
            self._model.addConstr(
                gp.quicksum(A[:,c_idx][i] * self._u.get(i)
                            for i in range(len(b))) <= c[c_idx]
            )

        self._opt_obj = None
        self._opt_u = None
        self._ray_u = None

    def solve(self):
        print('-' * 50)
        print(f'Start solving dual subproblem.')
        self._model.setParam(GRB.Param.DualReductions, 0)
        self._model.setParam(GRB.Param.InfUnbdInfo, 1)
        self._model.optimize()

        status = None
        if self._model.status == GRB.OPTIMAL:
            self._opt_obj = self._model.objVal
            self._opt_u = {
                i: self._u.get(i).X
                for i in range(self._num_vars)
            }
            status = OptStatus.OPTIMAL
            print(f'\tdual subproblem is optimal.')
            print(f'\topt_obj={self._opt_obj:.2f}')
```

Listing 4.18 Generic Benders decomposition for LP problems

```
class GenericBendersSolver:

    def __init__(self, master_solver, dual_subprob_solver):
        self._master_solver = master_solver
        self._dual_subprob_solver = dual_subprob_solver

    def optimize(self,) -> OptStatus:
        eps = 1.0e-5
        lb = -np.inf
        ub = np.inf

        iter = 0
        while True:
            iter += 1
            print(f'\nIteration: {iter}')
            # solve master problem
            self._master_solver.save_model(f'master_iter{iter}.lp')
            master_status = self._master_solver.solve()
            if master_status == OptStatus.INFEASIBLE:
                print(f'Model is infeasible!')
                return OptStatus.INFEASIBLE

            # update lower bound
            lb = np.max([lb, self._master_solver.opt_obj])
            print(f'Bounds: lb={lb:.2f}, ub={ub:.2f}')

            opt_y = self._master_solver.opt_y

            # solve subproblem
            self._dual_subprob_solver.update_objective(opt_y)
            self._dual_subprob_solver.save_model(f'dsp_iter{iter}.lp')
            dsp_status = self._dual_subprob_solver.solve()

            if dsp_status == OptStatus.OPTIMAL:
                print(f'DSP is optimal!')
                # update upper bound
                opt_obj = self._dual_subprob_solver.opt_obj
                opt_obj_y = self._master_solver.opt_obj_y
                ub = np.min([ub, opt_obj_y + opt_obj])
                print(f'Bounds: lb={lb:.2f}, ub={ub:.2f}')

                if ub - lb <= eps:
                    break

                opt_u = self._dual_subprob_solver.opt_u
                self._master_solver.add_optimality_cut(opt_u)
            elif dsp_status == OptStatus.UNBOUNDED:
                print(f'DSP is unbounded!!!')
                ray_u = self._dual_subprob_solver.ray_u
                self._master_solver.add_feasibility_cut(ray_u)
            else:
```