

Hands-on Large Scale Optimization in Python

From Beginning to Giving Up

Kunlei Lian

2023-02-18

Table of contents

Preface	3
1 Environment Setup	4
1.1 Install Homebrew	4
1.2 Install Anaconda	4
1.3 Create a Conda Environment	5
1.4 Install Google OR-Tools	6
2 Introduction	8
3 Environment Setup	9
3.1 Install Homebrew	9
3.2 Install Anaconda	9
3.3 Create a Conda Environment	10
3.4 Install Google OR-Tools	11
I Benders Decomposition	13
4 Benders Decomposition	14
4.1 The Decomposition Logic	14
4.2 A linear programming example	17
4.2.1 The original problem and its optimal solution	17
4.2.2 Benders decomposition	19
4.2.3 Solving the problem step by step	20
4.2.4 Putting it together	27
4.2.5 A generic solver	30
Implementation with callbacks	30

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 `conda 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 environment 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
abs1-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 `conda 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 environment 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
abs1-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 reasoning 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.

$$(\mathbf{P}) \quad \min. \quad \mathbf{c}^T \mathbf{x} + \mathbf{f}^T \mathbf{y} \quad (4.1)$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{b} \quad (4.2)$$

$$\mathbf{x} \geq 0, \mathbf{y} \geq 0 \quad (4.3)$$

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 \mathbf{P} as the following form:

$$\min. \quad \mathbf{f}^T \mathbf{y} + g(\mathbf{y}) \quad (4.4)$$

$$\text{s.t.} \quad \mathbf{y} \geq 0 \quad (4.5)$$

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

$$(\mathbf{SP}) \quad \min. \quad \mathbf{c}^T \mathbf{x} \quad (4.6)$$

$$\text{s.t.} \quad \mathbf{Ax} = \mathbf{b} - \mathbf{By} \quad (4.7)$$

$$\mathbf{x} \geq 0 \quad (4.8)$$

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 \mathbf{SP} , \mathbf{DSP} , is given below.

$$(\mathbf{DSP}) \quad \max. \quad (\mathbf{b} - \mathbf{By})^T \mathbf{u} \quad (4.9)$$

$$\text{s.t.} \quad \mathbf{A}^T \mathbf{u} \leq \mathbf{c} \quad (4.10)$$

$$\mathbf{u} \text{ unrestricted} \quad (4.11)$$

A key characteristic of the above \mathbf{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

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

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 \mathbf{DSP} is equivalent to

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

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

$$\lambda_j \geq 0, \quad \forall j \in \mathbf{J} \quad (4.15)$$

$$\mu_k \geq 0, \quad \forall k \in \mathbf{K} \quad (4.16)$$

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:

$$\min. \quad g \quad (4.17)$$

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

$$(\mathbf{b} - \mathbf{B}\mathbf{y})^T \mathbf{u}_j^{point} \leq g, \quad \forall j \in \mathbf{J} \quad (4.19)$$

$$j \in \mathbf{J}, k \in \mathbf{K} \quad (4.20)$$

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:

$$(\mathbf{BMP}) \quad \min. \quad \mathbf{f}^T \mathbf{y} + g \quad (4.21)$$

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

$$(\mathbf{b} - \mathbf{B}\mathbf{y})^T \mathbf{u}_j^{point} \leq g, \quad \forall j \in \mathbf{J} \quad (4.23)$$

$$j \in \mathbf{J}, k \in \mathbf{K}, \mathbf{y} \geq 0 \quad (4.24)$$

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**):

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

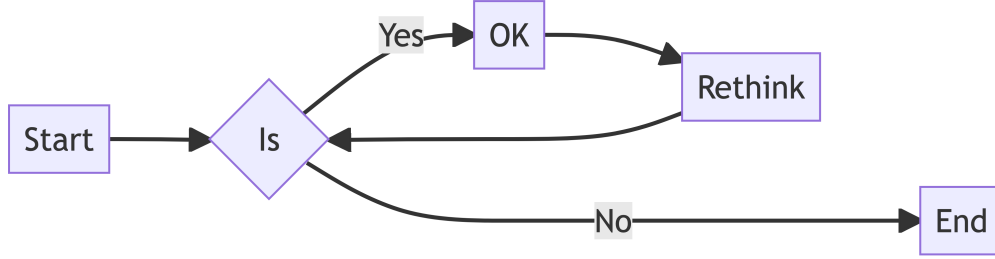


Figure 4.1: Benders decomposition workflow

4.2 A linear programming example

In this section, we will first present a small linear programming problem and solve it directly using the Gurobi API in Python - *gurobipy*. Then we will demonstrate the Benders decomposition approach on this artificial problem. Lastly, we will provide an implementation to solve this problem in *gurobipy*.

4.2.1 The original problem and its optimal solution

The linear program we examine here is devoid of any practical meaning and is solely used to demonstrate the solution process of Benders decomposition. The problem is stated below, in which $\mathbf{x} = (x_1, x_2, x_3)$ and $\mathbf{y} = (y_1, y_2)$ are the decision variables. We assume that \mathbf{y} is the complicating variable.

$$\begin{aligned}
\text{min.} \quad & 8x_1 + 12x_2 + 10x_3 + 15y_1 + 18y_2 \\
\text{s.t.} \quad & 2x_1 + 3x_2 + 2x_3 + 4y_1 + 5y_2 = 300 \\
& 4x_1 + 2x_2 + 3x_3 + 2y_1 + 3y_2 = 228.75 \\
& 1x_1 + 2x_2 + 1x_3 + 1.5y_1 + 2y_2 = 150 \\
& 3x_1 + 2x_2 + 2x_3 + 1y_1 + 2y_2 = 180 \\
& x_i \geq 0, \quad \forall i = 1, \dots, 3 \\
& y_i \geq 0, \quad \forall j = 1, 2
\end{aligned}$$

In this example, $\mathbf{c}^T = (8, 12, 10)$, $\mathbf{f}^T = (15, 18)$ and $\mathbf{b}^T = (300, 228.75, 150, 180)$. In addition,

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 2 \\ 4 & 2 & 3 \\ 1 & 2 & 1 \\ 3 & 2 & 2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 4 & 5 \\ 2 & 3 \\ 1.5 & 2 \\ 1 & 2 \end{bmatrix}$$

We first use Gurobi to identify its optimal solution.

```

import gurobipy as gp
from gurobipy import GRB

# Create a new model
env = gp.Env(empty=True)
env.setParam('OutputFlag', 0)
env.start()
model = gp.Model(env=env, name="original_problem")

# Decision variables
x1 = model.addVar(vtype=GRB.CONTINUOUS, name='x1')
x2 = model.addVar(vtype=GRB.CONTINUOUS, name='x2')
x3 = model.addVar(vtype=GRB.CONTINUOUS, name='x3')
y1 = model.addVar(vtype=GRB.CONTINUOUS, name='y1')
y2 = model.addVar(vtype=GRB.CONTINUOUS, name='y2')

# Objective function
model.setObjective(8*x1 + 12*x2 + 10*x3 + 15*y1 + 18*y2,
                  GRB.MINIMIZE)

# Constraints
model.addConstr(2*x1 + 3*x2 + 2*x3 + 4*y1 + 5*y2 == 300)

```

```

model.addConstr(4*x1 + 2*x2 + 3*x3 + 2*y1 + 3*y2 == 220)
# model.addConstr(1*x1 + 2*x2 + 1*x3 + 1.5*y1 + 2*y2 == 150)
# model.addConstr(3*x1 + 2*x2 + 2*x3 + 1*y1 + 2*y2 == 180)

# Optimize the model
model.optimize()

# Print the results
if model.status == GRB.OPTIMAL:
    print("Optimal solution found!")
    print(f'x1 = {x1.X:.2f}')
    print(f'x2 = {x2.X:.2f}')
    print(f'x3 = {x3.X:.2f}')
    print(f'y1 = {y1.X:.2f}')
    print(f'y2 = {y2.X:.2f}')
    print(f"Total cost: {model.objVal:.2f}")
else:
    print("No solution found.")

# Close the Gurobi environment
model.dispose()
env.dispose()

```

The optimal solution and objective value are as follows.

```

```{python}
Optimal solution found!
x1 = 14.29
x2 = 0.00
x3 = 0.00
y1 = 0.00
y2 = 54.29
Total cost: 1091.43
```

```

4.2.2 Benders decomposition

We first state the subproblem as follows:

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

We define two dual variables u_1 and u_2 to associate with the two constraints in the subproblem. The dual subproblem could then be stated as follows:

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

The *RBMP* can be stated as:

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

4.2.3 Solving the problem step by step

In this section, we will solve the linear program step by step using Gurobi. To this end, we first import the necessary libraries and create an environment `env`.

```
# output: false
import numpy as np
import gurobipy as gp
from gurobipy import GRB

env = gp.Env('benders')
env.setParam('OutputFlag', 0)
```

Next, we initialize several algorithm parameters, specifically, we use `lb` and `ub` to represent the lower and upper bounds of the solution. The `eps` is defined as a small number to decide whether the searching process should stop.

The remaining codes aim to create the restricted master Benders problem indicated by `rbmp`. Note that it only has the y and g variables and the objective function, there is no constraint added to the model yet.

```
# parameters
lb = -GRB.INFINITY
ub = GRB.INFINITY
eps = 1.0e-5

# create restricted Benders master problem
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)
```

We then define the model in Gurobi to solve the dual subproblem, represented by `dsp`. It consists of two decision variables $u1$ and $u2$. The constraints are created in lines 12 - 14.

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

# create decision variables
u1 = dsp.addVar(vtype=GRB.CONTINUOUS, name='u1')
u2 = dsp.addVar(vtype=GRB.CONTINUOUS, 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 the very first iteration, we solve the **RBMP**, as shown in the following code snippet.

```

rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'optimal solution found!')

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

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

```

```

optimal solution found!
optimal obj: 0.00
y1 = 0.00
y2 = 0.00
g = 0.00
lb=0.0, ub=1200.0

```

Now we have obtained an optimal solution $(\bar{y}_1, \bar{y}_2, \bar{g}) = (0, 0, 0)$, which also provides a new lower bound to our problem. We now feed the values of \bar{y}_1 and \bar{y}_2 into the Benders subproblem (SP):

```

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 = u1.X
    u2_opt = u2.X

    print(f'optimal obj = {dsp.objVal:.2f}')

```

```

    print(f'u1 = {u1_opt:.2f}')
    print(f'u2 = {u2_opt:.2f}')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb}, ub={ub}')
elif dsp.Status == GRB.UNBOUNDED:
    # add feasibility cut
    pass
else:
    pass

```

```

optimal obj = 1200.00
u1 = 4.00
u2 = 0.00
lb=0.0, ub=1200.0

```

We see that the dual subproblem has an optimal solution. Note that in line 15, the upper bound of the problem is 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.

```

rbmp.addConstr((300-4*y2-5*y2)*u1_opt
               + (220-2*y1-3*y2)*u2_opt <= g,
               name='c3')
rbmp.update()
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'optimal solution found!')

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

    print(f'optimal obj: {rbmp.objVal:.2f}')
    print(f'y1 = {y1_opt:.2f}')
    print(f'y2 = {y2_opt:.2f}')
    print(f'g = {g_opt:.2f}')

```

```

        print(f'lb={lb}, ub={ub}')
    elif rbmp.status == GRB.INFEASIBLE:
        print(f'original problem is infeasible!')

```

```

optimal solution found!
optimal obj: 600.00
y1 = 0.00
y2 = 33.33
g = 0.00
lb=600.0, ub=1200.0

```

Now we solve the subproblem again with the newly obtained solution $(\bar{y}_1, \bar{y}_2, \bar{g}) = (0, 33.33, 0)$.

```

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 = u1.X
    u2_opt = u2.X

    print(f'optimal obj = {dsp.objVal:.2f}')
    print(f'u1 = {u1_opt:.2f}')
    print(f'u2 = {u2_opt:.2f}')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb}, ub={ub}')
elif dsp.status == GRB.UNBOUNDED:
    print(f'dual subproblem is unbounded!')

```

```

optimal obj = 533.33
u1 = 4.00
u2 = 0.00
lb=600.0, ub=1133.3333333333333

```

Since the optimal objective value of the subproblem, 533.33, is still bigger than $\bar{g} = 0$, an optimality cut is needed. In the below code snippet, we add the new cut and solve the restricted Benders master problem again.


```

rbmp.addConstr((300 - 4*y1 - 5*y2) * u1_opt + (220 - 2*y1 - 3*y2) * u2_opt <= g, name='c3')
rbmp.update()
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'optimal solution found!')

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

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

```

```

optimal solution found!
optimal obj: 1080.00
y1 = 0.00
y2 = 60.00
g = 0.00
lb=1080.0, ub=1133.3333333333333

```

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.

```

dsp.setObjective((300 - 4*y1_opt - 5*y2_opt) * u1 + (220 - 2*y1_opt - 3*y2_opt) * u2, GRB.OPTIMIZE)
dsp.update()
dsp.optimize()

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

    print(f'optimal obj = {dsp.objVal:.2f}')
    print(f'u1 = {u1_opt:.2f}')
    print(f'u2 = {u2_opt:.2f}')

```

```

        ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
        print(f'lb={lb}, ub={ub}')
    elif dsp.status == GRB.UNBOUNDED:
        print(f'dual subproblem is unbounded!')

```

```

optimal obj = 80.00
u1 = 0.00
u2 = 2.00
lb=1080.0, ub=1133.3333333333333

```

Now the upper bound is reduced to 1133.33, but the subproblem optimal solution is still bigger than the value of $\bar{g} = 0$.

```

rbmp.addConstr((300 - 4*y1 - 5*y2) * u1_opt + (220 - 2*y1 - 3*y2) * u2_opt <= g, name='c3')
rbmp.update()
rbmp.optimize()

if rbmp.status == GRB.OPTIMAL:
    print(f'optimal solution found!')

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

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

```

```

optimal solution found!
optimal obj: 1091.43
y1 = 0.00
y2 = 54.29
g = 114.29
lb=1091.4285714285713, ub=1133.3333333333333

```

```

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

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

    print(f'optimal obj = {dsp.objVal:.2f}')
    print(f'u1 = {u1_opt:.2f}')
    print(f'u2 = {u2_opt:.2f}')
    ub = np.min([ub, 15*y1_opt + 18*y2_opt + dsp.objVal])
    print(f'lb={lb}, ub={ub}')
elif dsp.status == GRB.UNBOUNDED:
    print(f'dual subproblem is unbounded!')

```

```

optimal obj = 114.29
u1 = 0.00
u2 = 2.00
lb=1091.4285714285713, ub=1091.4285714285713

```

Now the gap between the lower bound and upper bound is reduced to 0, the problem completes.

4.2.4 Putting it together

Certainly we don't want to manually control the interaction between the master problem and subproblem to find the optimal solution. Therefore, in this section, we will put every together to come up with a control flow to help us identify the optimal solution automatically.

```

from enum import Enum

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

class MasterSolver:

```

```

def __init__(self):
    self._env = gp.Env('MasterProbEnv')
    self._model = gp.Model(env=self._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:
    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
    elif self._model.status == GRB.INFEASIBLE:
        opt_status = OptStatus.INFEASIBLE
    else:
        opt_status = OptStatus.ERROR

    return opt_status

def add_feasibility_cut(self) -> None:
    pass

def add_optimality_cut(self) -> None:
    pass

```

```

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

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

@property
def opt_y1(self):
    return self._y1

@property
def opt_y2(self):
    return self._y2

@property
def opt_g(self):
    return self._g

class DualSubprobSolver:

    def __init__(self):

        pass

    def solve(self):

        pass

    def update_objective(self):

        pass

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

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

    while True:
        # 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])

        # solve subproblem
        self._dual_subprob_solver.solve()

        # update upper bound

        if ub - lb <= eps:
            break

```

4.2.5 A generic solver

Implementation with callbacks

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