UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

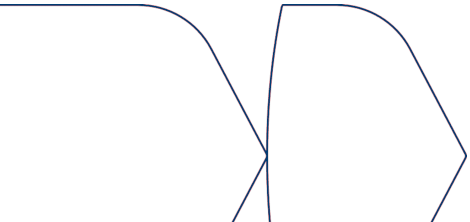
Wind Turbine Maintenance Optimization

Decisional Models and Optimization Project

Professor
Nicoletta Del Buono

Student
Vito Marco Rubino

A.Y. 2025 - 2026



Contents

Chapter 1

Introduction

1.1 Problem Context and Motivation

Wind turbine maintenance is a critical operational task for ensuring the reliability, safety, and long-term efficiency of a wind farm. In this project, we focus on a simplified yet realistic maintenance scheduling problem, whose formal statement is the following.

There are **7 wind turbines**, which all need to be maintained once during the week.

There are **two maintenance teams**: maintenance team 1 and maintenance team 2. There is no difference between the two maintenance teams. The maintenance teams only work on **workdays**, i.e. from Monday to Friday. It takes one maintenance team a full day to maintain one wind turbine.

Due to different locations of each wind turbine and the weather of the date, the maintenance costs are different. The costs are the same for both maintenance teams.

The costs are stated in Table 1.1:

turbine	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**
1	10	11	12	13	14
2	12	14	16	18	20
3	17	18	17	18	17
4	20	19	18	17	16
5	22	22	22	22	33
6	24	23	22	23	23
7	9	6	8	7	9

Table 1.1: Maintenance costs for each turbine and day of the week.

The problem consists in deciding, for each turbine, on which day and with which team the maintenance should be carried out, under the operational constraints described above. The final goal is to formulate an binary integer linear model to minimize the total maintenance cost (and eventually solve it numerically).

1.2 Exploratory Analysis of Maintenance Costs

Figures 1.1 and 1.2 provide an exploratory visual analysis of the maintenance costs reported in Table 1.1.

The **heatmap** highlights the global cost structure, showing how certain turbines systematically present higher costs than others and how cost variations across the week differ substantially from turbine to turbine.

A first observation is that turbines 4, 5, and 6 tend to have consistently higher maintenance costs, with **turbine 5** showing a pronounced **outlier** on Friday (cost = 33). Turbine 7, on the other hand, systematically exhibits the lowest cost values, especially on Tuesday (cost = 6). The heatmap further reveals that Mondays and Tuesdays generally correspond to lower costs for several turbines, while Fridays tend to be more expensive.

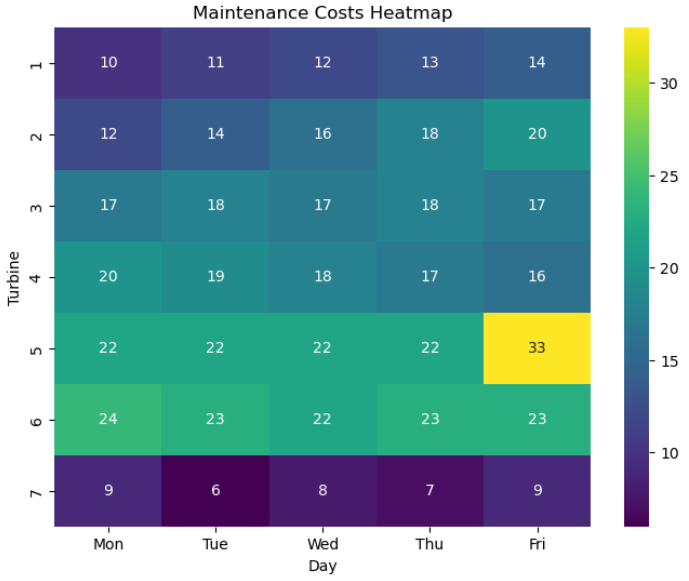


Figure 1.1: Heatmap

The **cost trend chart** in Figure 1.2 complements this analysis by visualising day-by-day cost fluctuations for each turbine. Some turbines (e.g., turbine 1 and turbine 2) follow a monotonic increasing pattern throughout the week, whereas turbine 4 decreases steadily from Monday to Friday. Turbine 3 and turbine 6 exhibit mid-week stability, suggesting that scheduling them on Wednesday or Thursday results in relatively balanced costs. These patterns play a crucial role in the assignment decisions of the optimization model, influencing which days become more attractive for specific turbines.

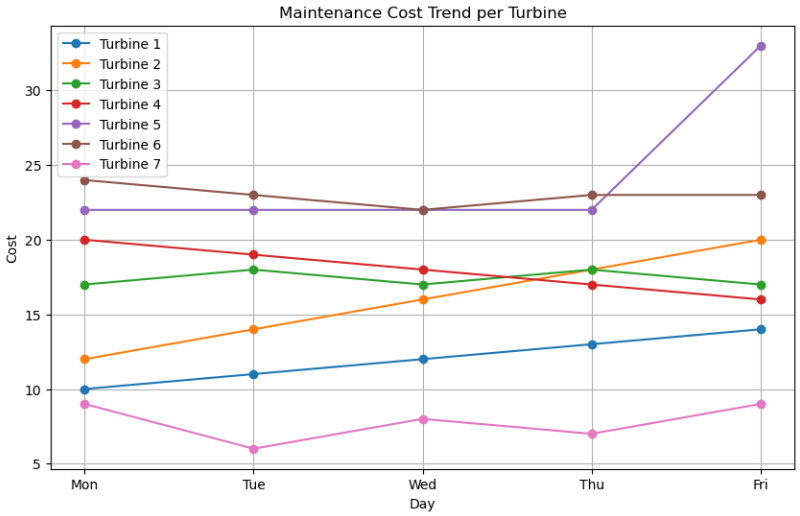


Figure 1.2: Cost trend

1.3 Decision-Making Framework

The methodological foundation of this work is rooted in the **decision-making framework** presented by Herbert **Simon's** and shown in Figure 1.3.

According to Simon's classical model of bounded rationality, decision processes evolve through **iterative** and **structured phases**. In this perspective, optimization constitutes an essential component of rational decision-making, as it provides a systematic tool for evaluating alternatives against a set of objectives and constraints.

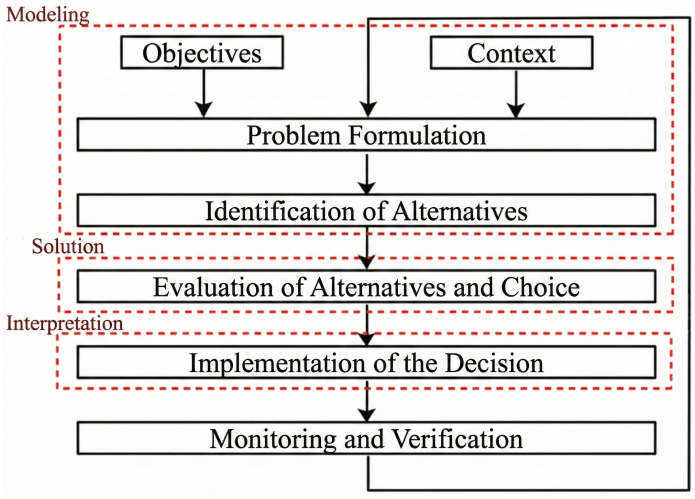


Figure 1.3: Simon's Decision-Making model

The scheduling problem introduced above fits naturally within Simon's decision model:

- **Problem formulation**: definition of decision variables, identification of operational constraints, and construction of a cost function.

- **Algorithmic phase**: selection of a mathematical optimisation paradigm (in this case, Boolean Integer Linear Programming) and numerical resolution through a dedicated solver.
- **Interpretation**: analysis of the optimal schedule, validation of its feasibility, and assessment of its economic and operational significance.

This structured approach ensures rigour, reproducibility, and transparency, and it reflects the modelling methodology adopted in modern operations research and data-driven decision sciences.

1.4 Why Linear and Boolean Integer Optimization?

The maintenance scheduling problem naturally leads to a Boolean Integer Linear Programming (BILP) formulation. The assignment of each turbine to a specific day and team is inherently discrete: each decision variable is binary and represents a yes/no assignment. The operational constraints are linear, as they impose:

- exactly one maintenance operation per turbine per week;
- at most one turbine serviced by each team per day.

The total maintenance cost is a linear combination of these binary assignment variables.

1.5 Goals

The objectives of this project are the following:

- to construct a rigorous **ILP formulation** for the wind turbine maintenance scheduling problem;
- to **implement the optimization model** using the PuLP library [2] in Python;
- to **solve the model numerically** and obtain the optimal weekly maintenance plan;
- to analyse and **interpret** the resulting schedule in terms of feasibility, efficiency, and operational insight.

Chapter 2

Mathematical formulation

This section presents the complete mathematical formulation of the wind turbine maintenance scheduling problem introduced in Chapter 1.

2.1 Sets

We introduce the **index sets** that describe the combinatorial structure of the problem:

$\mathcal{T} = \{1, 2, \ldots, 7\}$	set of wind turbines,
$\mathcal{D} = \{\text{Mon}, \text{Tue}, \text{Wed}, \text{Thu}, \text{Fri}\}$	set of working days,
$\mathcal{K} = \{1, 2\}$	set of available maintenance teams.

The maintenance cost of servicing turbine i on day j is denoted by c_{ij} and is taken from Table 1.1. Costs do not depend on the team index k, since the two teams are identical.

2.2 Decision Variables

The **decision variables** encode the assignment of turbines to days and teams. For every $i \in \mathcal{T}$, $j \in \mathcal{D}$, and $k \in \mathcal{K}$, define:

$$x_{i,j,k} = \begin{cases} 1 & \text{if team } k \text{ performs maintenance on turbine } i \text{ on day } j, \\ 0 & \text{otherwise.} \end{cases}$$

These are binary variables, as typical in assignment and scheduling problems.

2.3 Objective Function

The goal is to **minimize** the total weekly maintenance cost. The objective function is therefore:

$$\min \sum_{i \in \mathcal{T}} \sum_{j \in \mathcal{D}} \sum_{k \in \mathcal{K}} c_{ij}\, x_{i,j,k}.$$

This is a linear function of the binary variables, consistent with the standard BILP form $\min \boldsymbol{c}^\top \boldsymbol{x}$.

2.4 Constraints

2.4.1 Coverage Constraint: One Maintenance per Turbine

Each turbine must receive **exactly one maintenance operation** during the week:

$$\sum_{j\in\mathcal{D}}\sum_{k\in\mathcal{K}} x_{i,j,k} = 1, \qquad \forall i \in \mathcal{T}.$$

2.4.2 Team Capacity Constraints

Each maintenance team can service at most **one turbine per day**:

$$\sum_{i\in\mathcal{T}} x_{i,j,k} \leq 1, \qquad \forall j \in \mathcal{D},\ \forall k \in \mathcal{K}.$$

Remark 2.4.1 (Team index)**.** The model could be simplified by removing the team index k and representing the daily capacity with a single constraint

$$\sum_{i\in\mathcal{T}} x_{ij} \leq 2$$

In this project, however, I wanted to explicitly keep the team dimension to preserve the natural turbine-day-team structure of the assignment problem and to facilitate the interpretation and visualization of the resulting schedule.

2.4.3 Binary Feasibility

All decision variables are **binary**:

$$x_{i,j,k} \in \{0,1\}, \qquad \forall i,j,k.$$

2.5 Complete BILP Model

The complete optimization problem can be written compactly as follows:

$$\begin{aligned}
\min_{x_{i,j,k}} \quad & \sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{D}}\sum_{k\in\mathcal{K}} c_{ij}\, x_{i,j,k} \;\equiv\; \min_{\mathbf{x}} \mathbf{c}^\top \mathbf{x} \\
\text{s.t.} \quad & \sum_{j\in\mathcal{D}}\sum_{k\in\mathcal{K}} x_{i,j,k} = 1 && \forall i \in \mathcal{T}, \\
& \sum_{i\in\mathcal{T}} x_{i,j,k} \leq 1 && \forall j \in \mathcal{D},\ \forall k \in \mathcal{K}, \\
& x_{i,j,k} \in \{0,1\} && \forall i,j,k.
\end{aligned}$$

2.6 Modelling Assumptions and Validity

The formulation relies on the following assumptions:

- both maintenance teams are identical in capability and cost;
- each maintenance operation requires exactly one full working day;
- maintenance costs depend only on turbine and day, not on the team;
- no preemption or splitting of tasks is allowed.

2.7 Relation to Linear Programming Theory

The BILP model presented above is a discrete variant of the classical linear programming structure. If we flatten all binary variables into a single vector $x \in \mathbb{R}^{70}$ and collect all costs into a vector $c \in \mathbb{R}^{70}$, the objective becomes $c^\top x$. The feasible region defined by the constraints corresponds to a 0-1 polytope, i.e. a discrete subset of the convex polyhedron obtained by relaxing the binary constraints.

Chapter 3

Implementation in PuLP

This chapter illustrates how the Boolean Integer Linear Programming (BILP) model is implemented in Python using the **PuLP** optimization library [2], together with the **CBC solver** [1].

3.1 Overview of the Implementation Strategy

The implementation is structured so as to mirror the formal BILP model introduced in Chapter 2. For each mathematical component—sets, parameters, variables, objective and constraints there is a corresponding Python block.

The workflow adopted is the following:

1. import required packages;
2. define index sets that represent $\mathcal{T}$, $\mathcal{D}$, and $\mathcal{K}$;
3. load maintenance cost parameters c_{ij};
4. declare binary decision variables $x_{i,j,k}$;
5. initialize the BILP model as a minimization problem;
6. implement the objective function $\sum c_{ij}x_{i,j,k}$ using a triple summation;
7. implement all linear constraints;
8. solve the model with CBC and extract the optimal schedule.

3.2 Importing required packages

Only **two packages** are required: **NumPy** and **PuLP**. NumPy provides convenient array structures to store parameters such as the cost matrix c_{ij}, while PuLP is the modelling interface we use to construct the BILP model.

Code 1 : Importing required packages

```
!pip install -r requirements.txt
import numpy as np
import pulp as pl
```

3.3 Definition of Index Sets

Code 2 : Definition of Index Sets

```
turbines = range(1, 8)  # From 1 to 7
days = range(1, 6)      # From 1 (Mon) to 5 (Fri)
teams = [1, 2]
```

These Python objects defines structures represent the index sets used in the mathematical formulation:

$$\mathcal{T} = \{1, \ldots, 7\}, \quad \mathcal{D} = \{1, \ldots, 5\}, \quad \mathcal{K} = \{1, 2\}.$$

These sets are implemented as ranges and lists.

Remark 3.3.1. Note that Python ranges are half-open intervals `[start, stop)`, therefore `range(1,8)` produces integers 1 through 7, matching $\mathcal{T}$ exactly.

3.4 Maintenance Costs

The **cost matrix** c_{ij}, defined in Table 1.1, is implemented as a NumPy array with shape $(7, 5)$. The mapping between turbine/day indices and Python indices uses the standard $i - 1$, $j - 1$ shift. This structure allows direct indexing in the objective function.

Code 3 : Maintenance Costs

```
costs = np.array([
    # Mon, Tue, Wed, Thu, Fri
    [10, 11, 12, 13, 14],  # Turbine 1
    [12, 14, 16, 18, 20],  # Turbine 2
    [17, 18, 17, 18, 17],  # Turbine 3
    [20, 19, 18, 17, 16],  # Turbine 4
    [22, 22, 22, 22, 33],  # Turbine 5
    [24, 23, 22, 23, 23],  # Turbine 6
    [9,  6,  8,  7,  9]    # Turbine 7
])

print("Costs matrix shape:", costs.shape)
```

```
Costs matrix shape: (7, 5)
```

3.5 Binary Decision Variables

The decision variables $x_{i,j,k}$ are created using `LpVariable.dicts`, producing a three-dimensional dictionary indexed by turbine, day, and team. The arguments `lowBound = 0`, `upBound = 1`, and `cat=LpBinary` enforce binary feasibility, matching the definition:

$$x_{i,j,k} \in \{0, 1\}.$$

Code 4 : Binary Decision Variables

```
x = pl.LpVariable.dicts(
    "x",
    (turbines, days, teams),
    lowBound = 0,
    upBound = 1,
    cat = pl.LpBinary
)

print("Variable for maintened turbine 1 on Wednesday by Team 2:", x[1][3][2])
print("Total decision variables:", len(x) * len(days) * len(teams))
```

```
Variable for maintened turbine 1 on Wednesday by Team 2: x_1_3_2
Total decision variables: 70
```

PuLP generates variable names automatically based on the tuple (i, j, k), producing identifiers of the form:

$$x\_i\_j\_k \iff x_{i,j,k}.$$

The three-level dictionary access `x[i][j][k]` mirrors the triple-index variable of the BILP. The total number of variables equals:

$$|\mathcal{T}| \cdot |\mathcal{D}| \cdot |\mathcal{K}| = 7 \cdot 5 \cdot 2 = 70.$$

3.6 BILP Model Initialization

The BILP model is initialized as a minimisation problem that corresponds directly to the mathematical structure:

$$\min_x \mathbf{c}^\top \mathbf{x}.$$

Code 5 : BILP Model Initialization

```
lp_prob = pl.LpProblem("Wind_Turbine_Maintenance_Optimization", pl.LpMinimize)
```

At this stage, the model contains no variables and no constraints; these will be added incrementally.

3.7 Objective Function

The objective function is implemented using a triple nested summation that mirrors the mathematical expression

$$\sum_{i\in\mathcal{T}}\sum_{j\in\mathcal{D}}\sum_{k\in\mathcal{K}} c_{ij}\, x_{i,j,k}.$$

Code 6 : Objective Function

```
lp_prob += pl.lpSum(
    costs[i-1, j-1] * x[i][j][k]
    for i in turbines
    for j in days
    for k in teams
)
```

PuLP automatically interprets this expression as a linear combination of decision variables, building the internal constraint matrix in standard LP form.

3.8 Constraints

The two families of constraints introduced in Chapter 2 are directly implemented in Python.

(1) Coverage constraints. For each turbine i, the constraint

$$\sum_{j,k} x_{i,j,k} = 1$$

is implemented via a loop over turbines. This ensures that every turbine is serviced exactly once during the week.

(2) Team capacity constraints. For each day j and team k, the constraint

$$\sum_{i} x_{i,j,k} \leq 1$$

is implemented via nested loops. So, for fixed j and k, we sum over turbines.

Code 7 : Constraints

```
# 1) Each turbine is maintained exactly once during the week
for i in turbines:
    lp_prob += pl.lpSum(x[i][j][k] for j in days for k in teams) == 1,
    f"one_maintenance_turbine_{i}"

# 2) Each team can work on at most one turbine per day
for j in days:
    for k in teams:
```

```
        lp_prob += pl.lpSum(x[i][j][k] for i in turbines) <= 1,
        f"capacity_team_{k}_{j}"

print("Number of constraints:", len(lp_prob.constraints))
lp_prob.constraints
```

```
Number of constraints: 17

{'one_maintenance_turbine_1': 1*x_1_1_1 + ... + 1*x_1_5_2 + -1.0 = 0,
...
 'one_maintenance_turbine_7': 1*x_7_1_1 + .... + 1*x_7_5_2 + -1.0 = 0,
 'capacity_team_1_1': 1*x_1_1_1 + ... + 1*x_7_1_1 + -1.0 <= 0,
 'capacity_team_2_1': 1*x_1_1_2 + ... + 1*x_7_1_2 + -1.0 <= 0,
...
 'capacity_team_1_5': 1*x_1_5_1 + ... + -1.0 <= 0,
 'capacity_team_2_5': 1*x_1_5_2 + ... + 1*x_7_5_2 + -1.0 <= 0}
```

PuLP stores each constraint under a provided name; for example, `"capacity_team_1_3"` corresponds to team $k = 1$ on day $j = 3$.

Total constraints:

$$7 \text{ (coverage)} + 5 \cdot 2 \text{ (capacity)} = 17.$$

3.9 Solving the Model

The model is solved using CBC through the interface `PULP_CBC_CMD()`. CBC enhances the classical Branch-and-Bound method by generating **cutting planes** during the tree search, tightening the linear relaxation and reducing the occurrence of fractional solutions.

Code 8 : Solving the Model

```
lp_prob.solve(pl.PULP_CBC_CMD())
print("Status:", pl.LpStatus[lp_prob.status])
```

```
Status: Optimal
```

Finally, the solver status is printed and it is **Optimal**, meaning that a valid optimal solution has been found.

The most common solver statuses are:

- **Optimal**: a valid optimal solution was found
- **Infeasible**: no feasible solution satisfies all constraints
- **Unbounded**: the model is not bounded below
- **Not Solved**: the solver did not converge due to time limits or numerical instability

While this default configuration is sufficient for small-scale instances such as the one considered in this study, PuLP provides several optional parameters that allow finer control over the solver's behavior, which become particularly relevant for larger or more complex mixed-integer problems.

In particular, the `PULP_CBC_CMD` wrapper allows the user to specify some interesting parameters [1], such as:

- `timeLimit`: sets a maximum solving time (in seconds), providing a safeguard against excessive computation times when the problem size increases (default is None);
- `gapRel` and `gapAbs`: define relative or absolute optimality gap tolerances, allowing the solver to terminate early with a near-optimal solution (default is None);
- `maxNodes`: the max number of nodes during branching (default is None).

Although these parameters were not required for the present instance, their availability highlights the flexibility of PuLP as a modeling interface and its suitability for scaling to more demanding optimization problems.

3.10 Displaying the Optimal Solution

Once the solver terminates, the solution is extracted by scanning all variables and selecting the activated ones with value 1. Since each turbine must be maintained exactly once, exactly seven variable entries will take value 1.

Code 9 : Displaying the optimal solution

```
day_labels = ["Mon", "Tue", "Wed", "Thu", "Fri"]

for i in turbines:
    for j in days:
        for k in teams:
            if pl.value(x[i][j][k]) == 1:
                print(f"Turbine {i} maintained on {day_labels[j-1]} by Team {k}")
print("Total optimal cost", pl.value(lp_prob.objective))
```

```
Turbine 1 maintained on Mon by Team 1
Turbine 2 maintained on Mon by Team 2
Turbine 3 maintained on Wed by Team 2
Turbine 4 maintained on Fri by Team 1
Turbine 5 maintained on Tue by Team 1
Turbine 6 maintained on Wed by Team 1
Turbine 7 maintained on Tue by Team 2
Total optimal cost 105.0
```

[1] All `PULP_CBC_CMD` parameters are listed at https://coin-or.github.io/pulp/technical/solvers.html#pulp.apis.PULP_CBC_CMD

The printed schedule corresponds to the optimal feasible solution of the BILP: each turbine is assigned a unique day and team, and no team works on more than one turbine per day.

3.11 Computational Aspects

From a computational perspective, the model contains **70 binary variables** and **17 linear constraints**. Although this appears to be a small instance, the theoretical combinatorial complexity of the problem is significant.

If we were to consider the search space via **total enumeration** (*Brute Force*), treating each decision variable strictly as an independent binary switch, we would need to evaluate every possible combination of zeros and ones.

Consequently, the size of the unrestricted search space is determined by the total number of binary variables:

$$2^{|\mathcal{T}|\cdot|\mathcal{D}|\cdot|\mathcal{K}|} = 2^{70}$$

This magnitude illustrates the potential combinatorial explosion inherent in binary optimization problems and demonstrates why a naive trial-and-error approach is computationally intractable, rendering the use of specialized optimization algorithms necessary.

Despite this theoretical complexity, the CBC solver computes the optimal solution almost instantly. This efficiency is largely due to the structural regularity of the formulation: the model is assignment-like, the constraints consist solely of simple coverage and capacity relations, and the linear relaxation tends to be relatively tight.

Such characteristics typically enable exact BILP solvers to maintain a small branch-and-bound tree and converge quickly, even when the number of binary variables is moderately large.

Moreover, CBC does not rely on a pure Branch-and-Bound strategy. Instead, it implements a Branch-and-Cut algorithm, which augments the search process with cutting planes. Cutting planes strengthen the linear relaxation by removing fractional LP solutions while preserving all integer-feasible ones. By tightening the relaxation, cuts reduce the need for extensive branching, leading to a shallower search tree and a significant gain in efficiency.

3.11.1 Branch-and-Bound vs. Branch-and-Cut: Theoretical and Practical Differences

Boolean Integer Linear Programming (BILP) solvers enforce integrality through tree-based search methods. CBC, the solver used in this project, implements a *Branch-and-Cut* (B&C) approach, which combines classical Branch-and-Bound (B&B) with cutting-plane techniques. To understand its behaviour, it is useful to contrast the two paradigms.

Branch-and-Bound (B&B)

Branch-and-Bound explores the solution space by recursively dividing it into subproblems. At each node, the LP relaxation is solved. If the optimal solution is:

- **integer**, the node gives a feasible solution;
- **infeasible**, the node is pruned;

- **fractional**, branching is applied.

The branching step splits the domain of a fractional variable, e.g. $x_i = 0$ vs. $x_i = 1$, creating two new subproblems (children in the search tree). Figure 3.1 illustrates this process. This approach is conceptually simple, but may require exploring a large number of nodes when the LP relaxation is weak.

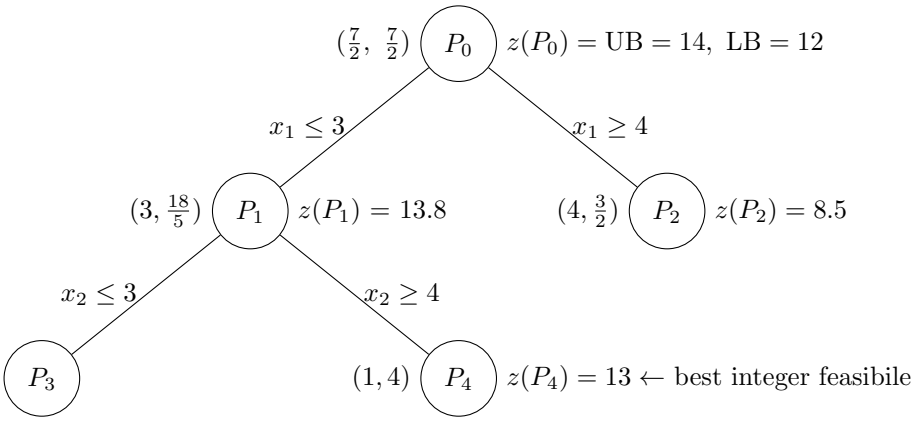


Figure 3.1: Classical Branch-and-Bound tree. Fractional LP solutions drive branching, generating a search tree. Nodes are pruned due to bounds, feasibility, or integrality.

Branch-and-Cut (B&C): Strengthening the Relaxation

Branch-and-Cut augments B&B by adding *cutting planes*: valid inequalities that remove fractional LP solutions without excluding any integer-feasible ones. Cuts tighten the LP relaxation, making it closer to the convex hull of integer points and often reducing the amount of branching required.

Example 3.11.1 (Chvátal-Gomory Cut)**.** Consider the **linear constraint**:

$$2x_1 + 2x_2 \leq 5.$$

with **integer** and **non-negative decision variables** $x_1, x_2 \in \mathbb{Z}_{\geq 0}$. The solver computes the linear relaxation and give as optimal solution:

$$x_1 = \frac{3}{2}, \qquad x_2 = 1.$$

We can verify that this fractional point satisfies the original constraint:

$$2\left(\frac{3}{2}\right) + 2\,(1) = 3 + 2 = 5$$

hence it is feasible for the LP relaxation.
However, it is *not* feasible for the integer problem, because $x_1 = \dfrac{3}{2}$ is not an integer.
Dividing the constraint by 2 yields

$$x_1 + x_2 \leq \frac{5}{2}.$$

Now observe that, since both x_1 and x_2 must be integers, their sum must also be an integer.

If the sum must be at most 2.5, the largest integer it can take is:

$$\lfloor 2.5 \rfloor = 2.$$

This yields the following Chváta-Gomory cutting plane:

$$x_1 + x_2 \leq 2.$$

So the cut preserves all meaningful integer solutions while excluding only the fractional ones. As a result, the LP relaxation becomes tighter and more closely aligned with the convex hull of the integer-feasible set, achieving this refinement reducing the size of the search tree.
Figure 3.2 shows how a cutting plane eliminates a fractional LP optimum while preserving all integer solutions.

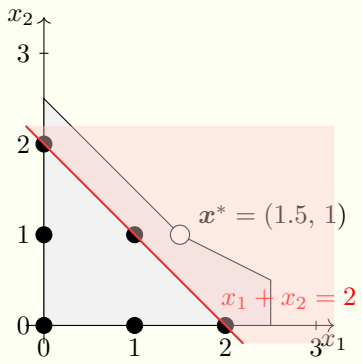


Figure 3.2: Geometric interpretation of the Chvátal–Gomory cut. The cutting plane $x_1 + x_2 \leq 2$ removes the fractional LP optimum while preserving all integer-feasible points.

Impact on the Wind Turbine Maintenance Problem In the present project, CBC generates only a small number of cuts at the root node, which is sufficient to strengthen the relaxation and drastically reduce the number of fractional LP extreme points. Combined with the assignment-like structure of the model, this leads to a very compact search tree and explains why the problem is solved efficiently despite the presence of 70 binary variables.

3.12 Other Optimization Packages

PuLP is a high-level modeling interface specifically designed for linear programming (LP), mixed-integer linear programming (MILP) and convex problems. It is not designed to solve constrained optimization problems.

However, several alternative or complementary optimization packages exist within the Python ecosystem, each characterized by different modeling paradigms and solver capabilities.

Among general-purpose optimization modeling frameworks, **Pyomo** represents a widely adopted alternative to PuLP. It offers a more expressive algebraic modeling language and supports a broad range of problem classes, including large-scale MILPs and nonlinear formulations, provided that suitable solvers are available.

Another relevant library is **python-mip**, which focuses explicitly on mixed-integer programming and provides a lightweight interface to state-of-the-art MIP solvers such as CBC and Gurobi. This package is particularly well-suited for applications where fine-grained control over integer variables and solver performance is required.

For continuous and convex optimization problems, libraries such as **SciPy** offers a collection of numerical optimization routines primarily aimed at nonlinear continuous problems.

The choice of an optimization package therefore depends on the problem structure, the required solver features, and scalability considerations. In the present project, PuLP combined with the CBC solver represents an appropriate and effective choice, given the linear structure and moderate size of the optimization problem.

Chapter 4

Results

This chapter presents and analyses the numerical results obtained by solving the BILP model described in Chapter 2 and implemented in Chapter 3. The CBC solver successfully identified an optimal solution, providing a complete weekly maintenance schedule for all seven turbines and the associated minimum cost.

4.1 Overview of the Numerical Output

The solver terminated with status `Optimal`, indicating that an integer feasible solution satisfying all constraints was found and proven optimal with respect to the objective function. The optimal objective value (total maintenance cost) is:

$$\text{Optimal Cost} = 105.0.$$

The solution consists of a mapping between each turbine and its assigned maintenance day and team.

4.2 Optimal Maintenance Schedule

Table 4.1 reports the optimal maintenance plan returned by the solver. Each turbine is assigned exactly one day and one team, respecting the coverage and capacity constraints.

Day	**Team 1**	**Team 2**
Mon	Turbine 1	Turbine 2
Tue	Turbine 5	Turbine 7
Wed	Turbine 6	Turbine 3
Thu	–	–
Fri	Turbine 4	–

Table 4.1: Optimal weekly maintenance schedule.

Figure 4.1 provides a visual representation of the optimal weekly maintenance schedule. Each bar corresponds to a maintenance task and shows, for every turbine, the assigned day and the responsible

team. The **Gantt chart** therefore offers an intuitive overview of how the solution respects both operational constraints and cost efficiency.

Figure 4.1: Gantt

A first observation is the balanced distribution of tasks across the two teams: on each active day, at most one turbine is assigned to each team, in full accordance with the capacity constraints. No maintenance is scheduled on Thursday, reflecting the fact that all turbines could be optimally assigned to less expensive days earlier in the week or on Friday in the case of turbine 4, whose cost profile decreases during the week.

The chart also highlights how the optimisation model naturally exploits low-cost patterns visible in the cost matrix. **Turbines 1 and 2**, whose maintenance is cheapest on **Monday**, are both scheduled on that day but assigned to different teams to avoid conflicts. **Turbines 5 and 7** are assigned to **Tuesday**, consistent with favourable costs for those turbines. The midweek assignments of **turbines 3 and 6 on Wednesday** match their relatively stable and low midweek cost levels. Finally, **turbine 4** is scheduled **on Friday**, as its cost is minimised at the end of the week.

4.3 Feasibility and Constraint Verification

The optimal solution fully satisfies the structural requirements of the model. Each turbine appears exactly once in the weekly plan, and the team-day assignments respect the daily capacity limit of one task per team.

This is reflected in the solution vector, where exactly seven binary variables take value 1—one for each turbine—confirming that the schedule is both feasible and complete.

4.4 Solver Behaviour and Computational Performance

CBC solves the instance very efficiently, exploring only a limited number of nodes. The problem benefits from:

- a small number of variables and constraints;
- an assignment-like structure;

- strong LP relaxations (few fractional directions);
- automatic addition of cutting planes at the root node.

These factors lead to fast convergence and minimal branching.

Chapter 5

Conclusions

This project addressed a real-world scheduling problem arising in the maintenance of wind turbines. Starting from the operational description of the system, we formulated the task as an Boolean Integer Linear Programming (BILP) model and solved it using the PuLP modelling library and the CBC Branch-and-Cut solver.

The project followed the structured decision-modelling process introduced in the course: *problem formulation*, *algorithmic phase*, and *interpretation of results*. This approach ensured conceptual clarity, methodological rigour, and full alignment with the theoretical framework of linear and integer optimization.

The optimal weekly maintenance plan obtained is cost-efficient, feasible, and interpretable. CBC found the optimal solution rapidly, confirming that the assignment-like structure of the model produces strong linear relaxations and is particularly suitable for ILP solvers. The resulting plan respects all operational constraints and directly reflects the cost patterns observed in the input data.

Despite its effectiveness, the model relies on simplifying assumptions: the maintenance teams are treated as identical, the costs are deterministic, and maintenance tasks always require one full day. While these assumptions are reasonable for a didactic setting, real-world applications often involve heterogeneous skills, variable weather conditions, uncertain durations, or additional logistical constraints.

The ILP modelling approach remains robust and extensible. Potential future developments include:

- incorporating multiple teams with different capabilities;
- introducing time windows, deadlines, or precedence constraints between tasks;
- modelling uncertainty through stochastic or robust optimisation;
- extending the objective to multi-criteria settings (e.g., cost vs. risk or reliability);
- developing dynamic or rolling-horizon scheduling approaches.

Overall, this work demonstrates how optimisation techniques provide powerful tools for decision-making in operational contexts. Integer programming models, when properly formulated and

implemented, offer transparent solutions and actionable insights, making them highly relevant in data-driven decision processes.

Bibliography

[1] John Forrest, Ted Ralphs, Stefan Vigerske, Haroldo Gambini Santos, John Forrest, Lou Hafer, Bjarni Kristjansson, jpfasano, EdwinStraver, Jan-Willem, Miles Lubin, rlougee, a-andre, jpgoncal1, Samuel Brito, h-i-gassmann, Cristina, Matthew Saltzman, tosttost, Bruno Pitrus, Fumiaki MATSUSHIMA, Patrick Vossler, Ron @ SWGY, and to-st. coin-or/Cbc: Release releases/2.10.12, August 2024.

[2] Stuart Mitchell, Michael O'Sullivan, and Iain Dunning. Pulp : A linear programming toolkit for python. 2011.