Delft University of Technology
Faculty of EEMCS

The potential of stochastic linear programming

BY
J. A. van Delft

to obtain the degree of Master of Science in Applied Mathematics – Stochastics
at the Delft University of Technology

to be defended publicly on 11 April

Thesis Committee:	Dr. H. N. Kekkonen	TU Delft	supervisor
	Dr. C. Kraaikamp	TU Delft	associate professor
	Dr. N. Lindner	Zuse Institute Berlin	external supervisor

Contents

Abstract

Solution methods and computational efficiency of stochastic linear programs have been extensively studied over the years. Despite this, their practical applications remain limited due to persistent computational challenges and the lack of a user-friendly interface for modeling and solving these problems. This thesis revisits those computational challenges and explores the usability of stochastic linear programs in the field of energy system modeling. This is done by implementing two solution methods for solving two-stage stochastic linear programs: the Extensive Form method and the L-Shaped method, and applying both methods on existing input problems. To additionally test the performance on an energy system model, the existing large-scale energy model Oemof-B3 is extended to a stochastic linear program, and the solution methods are applied on this problem too. Since the stochastic linear programming framework can be easily applied, and both methods successfully produce solutions for various input problems, we conclude stochastic linear programming holds significant potential as a modeling tool. However, computational challenges remain, particularly when applying the stochastic programming framework to the energy model Oemof-B3. Additionally, the lack of user-friendly tools and readily available solvers for solving stochastic linear programs limits the practical applicability of these methods.

1 Introduction

Linear programming is a fundamental tool for planning and decision-making. Its applications span multiple fields, including supply chain management, finance, transportation, and energy system modeling. In many real-world optimization problems, however, key parameters such as costs, demands, or supply levels are not known precisely at the time of decision-making. Classical linear programming assumes perfect information, but in practice, uncertainty is inevitable [Birge and Louveaux, 2011].

Typically, uncertainty is exorcised by taking reasonable guesses or by making careful estimates. Additionally, sensitivity analysis is performed to evaluate how changes in coefficients, such as costs or resource availability, impact the optimal solution. In most cases, this approach is satisfactory. However, if the optimal solution depends heavily on the value of some inaccurate data, the uncertainty of the coefficients must be taken into consideration in a more fundamental way. Probability theory offers a natural framework for quantifying uncertainty, making it reasonable to represent these uncertain coefficients as random variables. This perspective forms the foundation of stochastic linear programming [Haneveld et al., 2024].

In energy system modeling, uncertainty is ubiquitous. Energy demand, fuel prices, and policy changes all exhibit significant variability, making it challenging to develop reliable and cost-effective energy plans. Additionally, the increasing share of variable renewable electricity supply, combined with more unpredictable electricity consumption, requires true flexibility of the energy system to ensure that electricity demand is always met [Seljom et al., 2021]. For these reasons, stochastic linear programming has emerged as a powerful tool for modeling energy systems.

Stochastic linear programming has proven particularly useful for modeling investment and operation planning in distributed energy systems. For these types of systems, we often encounter two-stage stochastic linear programs. As we will see later, these models typically consist of:

1. First stage: Investment decisions (e.g., capacity expansion)
2. Second stage: Operational decisions (e.g., power dispatch)

The objective function often aims to minimize the expected total cost of electricity generation while considering constraints such as carbon dioxide emissions. An example of a two-stage stochastic linear program applied to the design of a distributed energy system is presented in [Zhou et al., 2013], where the model is used for planning a distributed energy system for a hotel. Furthermore, Yu et al. [Yu et al., 2019] present a two-stage stochastic programming model for the optimal operation of hybrid renewable systems. Additional works have investigated the sizing of storage systems using stochastic linear programming. In one study, Narayan and Ponnambalam [Narayan and Ponnambalam, 2017] developed a risk-averse stochastic programming model –based on the two-stage stochastic programming framework– to determine the optimal number of renewable energy production facilities, diesel generators, and batteries in a microgrid. Abbey and Joos [Abbey and Joos, 2009] used a stochastic model to analyze the impact of energy storage on wind-diesel hybrid systems, highlighting how storage can mitigate wind variability and improve diesel generator efficiency.

To build intuition on the topic of stochastic linear programming, we now present a simple example of a stochastic linear program in the field of energy system modeling, where demand is uncertain. This example was first introduced by Louveaux and Smeers [Louveaux and Smeers, 1988] and has since become a standard benchmarking example in the literature.

A country wants to invest in two different electricity generation technologies: one being a fossil fuel power plant and the other a renewable power plant. Both technologies are used to produce electricity in three modes: base load, intermediate load and peak load. These three modes correspond to different levels of electricity demand. Base load represents the continuous minimum demand, intermediate load fluctuates between base and peak levels throughout the day, and peak load corresponds to short periods of maximum demand. The fossil fuel power plant is best suited for base load production, as it provides a stable and continuous power supply. In contrast, the renewable power plant is best suited for peak load production due to its intermittent nature, allowing it to meet high demand when

conditions are good.

Two key decisions need to be made. First, the total capacity of the power plants to be built must be determined. Second, it must be decided how much of the total capacity will be allocated to generating electricity in each of the three modes (base load, intermediate load, and peak load). These values, of course, are not determined arbitrarily; they will be the outcomes of the following model, which we refer to as a *linear program*. The model is formulated as follows:

First, we introduce the *decision variables* x_F and x_R, which determine the capacity allocated to each power plant:

$$\begin{aligned} x_F &= \text{ capacity allocated to the fossil fuel based power plant [MW].} \\ x_R &= \text{ capacity allocated to the renewable power plant [MW].} \end{aligned} \tag{1.1}$$

Next, we define six additional decision variables: y_{F1}, y_{F2}, y_{F3}, y_{R1}, y_{R2} and y_{R3}, which represent the amount of capacity devoted to producing electricity in each mode. The modes are base load, intermediate load, and peak load, indexed by $j = 1, 2, 3$.

$$\begin{aligned} y_{Fj} &= \text{ amount of capacity of the fossil fuel based power plant devoted to producing electricity in mode } j,\ j = 1, 2, 3. \\ y_{Rj} &= \text{ amount of capacity of the renewable power plant devoted to producing electricity in mode } j,\ j = 1, 2, 3. \end{aligned} \tag{1.2}$$

The cost per unit of capacity installed for plants F and R are given by c_F and c_R, respectively. Additionally, the operational costs of the power plants F and R in mode j per unit of capacity are denoted f_{Fj} and f_{Rj}. The Total Cost (TC) function can be formulated as:

$$\begin{aligned} TC &= c_F\, x_F + c_R\, x_R + f_{F1}\, y_{F1} + f_{F2}\, y_{F2} + f_{F3}\, y_{F3} + f_{R1}\, y_{R1} + f_{R2}\, y_{R2} + f_{R3}\, y_{R3} \\ &= c_F\, x_F + c_R\, x_R + \sum_{j=1}^{3} \left(f_{Fj}\, y_{Fj} + f_{Rj}\, y_{Rj} \right). \end{aligned} \tag{1.3}$$

Letting $x = (x_F, x_R)$ and $y = (y_{F1}, y_{F2}, y_{F3}, y_{R1}, y_{R2}, y_{R3})$, where x represents the capacity allocated to each power plant and y represents the amount of capacity devoted to each mode for both plants. The goal of our linear program is to minimize the total costs, i.e., to minimize the value of the Total Cost (TC) function. This leads to the *objective function*:

$$\min_{x,y} \quad TC = c_F\, x_F + c_R\, x_R + \sum_{j=1}^{3} \left(f_{Fj}\, y_{Fj} + f_{Rj}\, y_{Rj} \right). \tag{1.4}$$

The outcome of the model that minimizes the total cost would be to set all decision variables equal to zero, resulting in zero total costs. However, this would imply that no power plants are built at all, which is not a realistic solution. To avoid this, we introduce *constraints* into our linear program, which limit the feasible solutions and ensure that only practical and feasible outcomes are considered.

The first constraint we introduce is a capacity constraint, which ensures that the total installed capacity across both power plants is at least m MW. This constraint can be expressed as:

$$x_F + x_R \geq m.$$

Secondly, we introduce a cost constraint, which ensures that the total installation costs do not exceed a budget b. This constraint is formulated as:

$$c_F\, x_F + c_R\, x_R \leq b.$$

Related to the decision variables y, we introduce two capacity constraints to ensure that the total amount of capacity devoted to producing electricity in the three modes does not exceed the installed

capacity of each power plant. These constraints are:

$$\begin{aligned} \sum_{j=1}^{3} y_{Fj} &\leq x_F, \\ \sum_{j=1}^{3} y_{Rj} &\leq x_R. \end{aligned} \tag{1.5}$$

Finally, it is crucial that the electricity produced in each mode meets a certain demand. For each mode $j = 1, 2, 3$, we require that:

$$y_{Rj} + y_{Fj} \geq d_j, \quad j = 1, 2, 3. \tag{1.6}$$

The objective function, together with the constraints, defines our optimization model. Solving this model will yield the values of the decision variables x and y, which minimize the total costs.

Now, we extend our linear program to a stochastic linear program by following these steps. As previously mentioned, the demand d_j is uncertain. Therefore, we replace each demand with the random variable $\tilde{d}_j$, which has a finite number of possible outcomes. The final constraint of our model then becomes:

$$y_{Rj} + y_{Fj} \geq \tilde{d}_j, \quad j = 1, 2, 3. \tag{1.7}$$

As a result of this change, the linear program can no longer be solved in the traditional way. To address this, we split the problem into two stages that occur sequentially. Decisions about the installed capacity must be made now, but the allocation of capacity to each mode can be determined later. Thus, x represents first-stage decisions, and y represents second-stage decisions. Since the second-stage decision variables depend on the resolution of uncertainty, we redefine the total cost function by replacing the operarational costs with expected operational costs:

$$TC = c_F\, x_F + c_R\, x_R + \mathbb{E}\left[\sum_{j=1}^{3} (f_{Fj}\, y_{Fj} + f_{Rj}\, y_{Rj})\right]. \tag{1.8}$$

The complete stochastic linear program is defined as follows:

$$\begin{aligned} \min_{x \geq 0} \quad TC \quad &= c_F\, x_F + c_R\, x_R + \mathbb{E}\left[\sum_{j=1}^{3} (f_{Fj}\, y_{Fj} + f_{Rj}\, y_{Rj})\right] \\ \text{s.t.} \quad & x_F + x_R \geq m, \\ & c_F\, x_F + c_R\, x_R \leq b, \\ & \sum_{j=1}^{3} y_{Fj} \leq x_F, \\ & \sum_{j=1}^{3} y_{Rj} \leq x_R, \\ & y_{Rj} + y_{Fj} \geq \tilde{d}_j, \quad j = 1, 2, 3. \end{aligned} \tag{1.9}$$

Solving problem (1.9) will provide the first-stage solutions x and the corresponding total cost TC. The second-stage decisions y will be determined at a later stage, after the realization of the random variables.

From our example, we observe that the stochastic linear programming (SLP) framework is straightforward to apply: the key step involves introducing random variables and distinguishing between first-stage and second-stage decision variables. However, one significant drawback of SLPs is that their formulations can result in large-scale problems, which present considerable computational challenges. Over the years, much research has focused on developing efficient solution methods, which we'll be

discussing now.

A well-known method for solving stochastic linear programs (SLPs) is the L-Shaped method, introduced by van Slyke and Wets [Van Slyke and Wets, 1969], which offers an analytical framework for tackling SLPs. However, solving SLPs analytically often proves too challenging, prompting researchers to explore sampling methods instead. Sampling methods can be divided into two main categories: interior sampling and exterior sampling. Interior sampling algorithms aim to solve the original stochastic linear program, but resort to sampling whenever the algorithm requires an (approximate) value of the objective function. We mention in particular the L-Shaped method with embedded sampling of [Dantzig and Infanger, 1991], the stochastic decomposition method of [Higle and Sen, 1991], and stochastic quasi-gradient methods [Ermoliev, 1988]. A second fundamental approach to sampling is an "exterior" approach, in which a sample is selected from the set of all possible outcomes and a corresponding approximation to the objective function is defined from this sample. This approximate objective is then minimized using a deterministic optimization algorithm; no further sampling is performed. This approach is known variously as sample path optimization [Gürkan et al., 1994] or the stochastic counterpart method [Rubinstein and Shapiro, 1993].

Despite advancements in solution methods for stochastic linear programs (SLPs), computational challenges remained, preventing the widespread application of the framework. The first notable computational results were presented by [Linderoth et al., 2006], who demonstrated that the Sample Average Approximation (SAA) method could provide high-quality solutions with proven accuracy within a within a reasonable computational time across all test problems. More recently, [Sen and Liu, 2016] demonstrated that their Stochastic Decomposition approach could achieve similar solution quality to SAA but in significantly less time, even when using standard desktop or laptop computers. However, in their first paper, uncertainty was only accounted for in the constraints. In a subsequent study, [Gangammanavar et al., 2020] extended the Stochastic Decomposition implementation to also address uncertainty in the cost coefficients of the objective function.

To summarize, the field of stochastic linear programming, particularly in the application to energy system modeling, holds great promise. This motivates the objective of this thesis: to revisit the computational challenges of solving stochastic linear programs and examine their practical applicability in energy system modeling. This is achieved through the implementation of two solution methods, followed by an evaluation of their accuracy and computational efficiency using test problems from the literature. Ultimately, these methods are applied to a large-scale energy system model to assess their real-world performance.

This thesis is structured as follows. Section 2 introduces the formulation of stochastic linear programs and outlines various solution methods. Section 3 describes the modeling framework 'oemof', and introduces the energy model oemof-B3. Section 4 provides an overview of the methodology, detailing the file formats of the input problems, the preparation of Gurobi models for solution methods, and the algorithmic steps of the Extensive Form method and the L-Shaped method. Section 5 presents the computational results of both solution methods, ultimately comparing their performance. Finally, in section 6, our results are discussed, and a conclusion on the potential of stochastic linear programming based on our findings is drawn.

2 Stochastic linear programs

In this section, we introduce the two-stage stochastic linear program and outline various solution methods for solving it. We start with a brief overview of linear programming before defining the two-stage stochastic model with fixed recourse. We then discuss two different solution methods: the Extensive Form method and the L-Shaped method based on Benders Decomposition. Lastly, we discuss the sampling-based techniques Sample Average Approximation (SAA) and Stochastic Decomposition (SD).

2.1 Linear programming

Before introducing the formal definition of a stochastic linear program, let us shortly discuss some basics about linear programming. A linear program is an optimization problem with a linear objective function that must be maximized or minimized, subject to a set of linear constraints. The general form is:

$$\begin{aligned} \min_x \; z = \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2.1}$$

where x is the vector of decision variables, c is the cost vector, A is the matrix of constraint coefficients, and b is the vector of constraint bounds. Given a problem of type (2.1), precisely one of the following three situations must occur:

1. The problem is *feasible*, and the optimal objective function value is *bounded*.
2. The problem is *feasible*, and the optimal objective function value is *unbounded*.
3. The problem is *infeasible*.

The *feasible region* for x of problem (2.1) is the space $P = \{x \geq 0 \,|\, Ax = b\}$. If the problem is infeasible, $P = \emptyset$. Otherwise, the set P is a *polyhedron*. A point $x \in P$ is called an *extreme point* of P if x cannot be written as a convex combination of two distinct points in P. So, geometrically, an extreme point of P is a 'corner point' of P. An example polyhedron including one of its extreme points is given in figure 2.1.

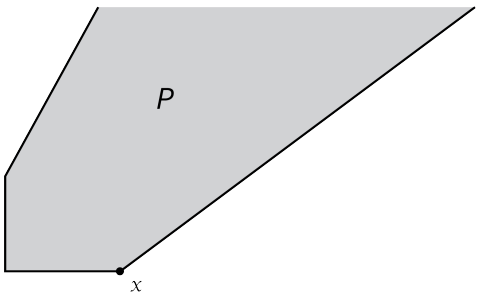

Figure 2.1: Polyhedron P with extreme point x.

Given figure 2.1, we can also give the definition of a *ray* of P. Informally, a ray r of P is a 'direction' that we can follow infinitely far, and still be within P. Additionally, we call a ray r of P an *extreme ray* of P if r cannot be expressed as a convex combination of other rays of P. Polyhedron P with extreme rays r^1 and r^2 is given in figure 2.2.

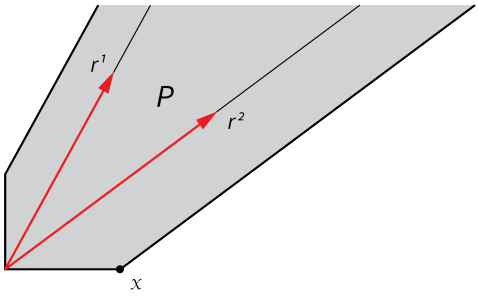

Figure 2.2: Polyhedron P with extreme rays r^1 and r^2.

One of the cornerstone results in linear programming is the fundamental theorem that establishes the connection between the feasibility of solutions and the optimization process. The following theorem formally states this important result:

Theorem 1. Given a linear optimization problem of the form (2.1), where $P \neq \emptyset$. If the objective function is bounded, there exists an optimal solution at an extreme point of P.

This concept is fundamental to several solution techniques, such as the simplex method, which systematically searches for optimal solutions by evaluating the objective value at extreme points. The theorem, which is stated as Theorem 3.2 in [Aardal et al., 2023], can be found along with its proof in that reference.

The final linear programming concept to be introduced is *duality*. Duality establishes a powerful relationship between two optimization problems: the *primal problem* and its associated *dual problem*. The dual problem can be derived from the original (or primal) problem by reinterpreting its constraints and objective in terms of resource values or shadow prices. It has many nice properties and forms the basis for most theory and practical algorithms for solving linear programming problems.

Given the primal problem (2.1), the dual is formulated as:

$$\begin{aligned} \max_{\pi} \; w = \quad & b^T \pi \\ \text{s.t.} \quad & A^T \pi \leq c, \end{aligned} \tag{2.2}$$

where π are the dual variables associated with the equality constraints $Ax = b$. The key relationship between the primal and dual variables is stated in the principle of strong duality.

Theorem 2. If x^* is an optimal solution to the primal problem (2.1), then the dual problem (2.2) has an optimal solution π^* and

$$z(x^*) = w(\pi^*).$$

The result implies that solving the dual problem is an equivalent approach to solving the primal, as both yield the same optimal objective value. This equivalence underpins advanced methods in optimization, such as decomposition techniques, as we will see in section 2.4. The theorem and proof can be found in [Aardal et al., 2023, Theorem 5.3].

2.2 A two-stage stochastic linear program with fixed recourse

Having established the fundamentals of linear programming, we now turn to the two-stage stochastic linear program with fixed recourse (2-SLP). An example of such a problem was already given in (1.9), we now use this example to illustrate the general formulation of a 2-SLP. We have a set of decisions to be taken without full information on some random events. These decisions are called *first-stage* decisions and are usually represented by the vector x. Later, full information is received on the realization of some random vector $\boldsymbol{\xi}$. In the context of our example, $\boldsymbol{\xi}$ represents the complete information regarding the random variables $\tilde{d}_j$. Then, *second-stage* decisions y are taken.

The classical two-stage stochastic linear program with fixed recourse is the problem of finding

$$\begin{aligned} \min_x F(x) = \quad & c^T x + \mathbb{E}_{\boldsymbol{\xi}} \left[Q(x, \boldsymbol{\xi}) \right] \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{2.3}$$

The first-stage decisions are represented by the $n_1 \times 1$ vector x. Corresponding to x are the first-stage problem data c, b, and A, of sizes $n_1 \times 1$, $m_1 \times 1$ and $m_1 \times n_1$ respectively [Birge and Louveaux, 2011].

The function $Q(x, \boldsymbol{\xi})$ is the so-called *recourse function*. We will call an outcome of $\boldsymbol{\xi}$ a *realization* or a *scenario* ξ. For a realization ξ, the recourse function $Q(x, \xi)$ is defined:

$$\begin{aligned} Q(x, \xi) := \min_y \quad & q(\xi)^T y \\ \text{s.t.} \quad & T(\xi)x + Wy = h(\xi), \\ & y \geq 0. \end{aligned} \tag{2.4}$$

The $n_2 \times 1$ vector y represents the second-stage decisions. Matrix W of size $m_2 \times n_2$ is called the *recourse matrix*, which we assume here is fixed (*fixed recourse*). Corresponding to y are the second-stage problem data $q(\xi)$, $h(\xi)$ and $T(\xi)$ of sizes $n_2 \times 1$, $m_2 \times 1$ and $m_2 \times n_1$ respectively. For a given realization ξ, these problem data become known. Each component of $q(\xi)$, $h(\xi)$ and $T(\xi)$ is thus a possible random variable.

For clarity, let us denote

$$q(\xi) = \begin{bmatrix} q_1(\xi) \\ \vdots \\ q_{n_2}(\xi) \end{bmatrix}, \quad h(\xi) = \begin{bmatrix} h_1(\xi) \\ \vdots \\ h_{m_2}(\xi) \end{bmatrix}, \quad T(\xi) = \begin{bmatrix} T_{1\cdot}(\xi) \\ \vdots \\ T_{m_2\cdot}(\xi) \end{bmatrix},$$

where $T_{i\cdot}(\xi) = (T_{i,1}(\xi), \ldots, T_{i,n_1}(\xi))$ is the i-th row of $T(\xi)$. Piecing together the stochastic components of the second-stage data, we obtain the vector $\boldsymbol{\xi}^T = (q(\xi)^T, h(\xi)^T, T_{1\cdot}(\xi), \ldots, T_{m_2\cdot}(\xi))$, with potentially up to $n_2 + m_2 + (m_2 \times n_1)$ components [Birge and Louveaux, 2011].

2.3 Extensive form

In the case that $\boldsymbol{\xi}$ has a finite discrete distribution, it is possible to reformulate the 2-SLP model as a linear programming model. That is, for a finite discrete distribution the expected value in the objective of (2.3) becomes a sum and one set of constraints is introduced for each realization of the random variable $\boldsymbol{\xi}$. The resulting linear program is called the *extensive form* [Birge and Louveaux, 2011].

Let us assume that (q_k, h_k, T_k), $k = 1, \ldots, K$, are the joint realizations of $(q(\xi), h(\xi), T(\xi))$, with corresponding probabilities p_k, $k = 1, \ldots, K$. We can then express the 2-SLP in its *extensive form* (EF) as follows:

$$\begin{aligned} \min_x \ F(x) = \quad & c^T x + \sum_{k=1}^{K} p_k \, q_k^T y_k \\ \text{s.t.} \quad & Ax = b, \\ & T_k x + W y_k = h_k, \quad k = 1, ..., K, \\ & x \geq 0, \ y_k \geq 0, \quad k = 1, ..., K. \end{aligned} \tag{2.5}$$

Rewriting the problem into the form given in (2.5) immediately presents our first solution method. That is, (2.5) is a standard linear program that can be solved using Gurobi or other optimization solvers. However, the fact that such an equivalent LP exists does not imply that all 2-SLPs with a finite discrete distribution can be easily solved. That is, a 2-SLP involving 9 independent random variables has a total number of scenarios:

- $K = 3^9 = 19\,683$ for 3 outcomes for each random variable;
- $K = 5^9 = 1\,953\,125$ for 5 outcomes for each random variable.

This illustrates that the problem rapidly grows with an increasing number of realizations of the components of the random vector and with an increasing number of random variables. The size of the problem may easily grow to an extent where it is impossible to generate the equivalent LP not to speak of solving it [Kall and Mayer, 1998].

Rather than directly solving the extensive form, alternative methods leverage its structure for a more efficient approach. That is, the extensive form has a so-called block angular structure, which becomes evident when considering the structure of the constraints. Problem (2.5) can be rewritten as:

$$\begin{aligned}
\min_{x,y} F(x) = \quad & c^T x \quad + p_1\, q_1^T y_1 \quad + \ \ldots \quad + p_K\, q_K^T y_K \\
\text{s.t.} \quad & Ax & & & = b, \\
& T_1 x \quad + W y_1 & & & = h_1, \\
& \vdots & \ddots & & \vdots \\
& T_K x & & + W y_K & = h_K, \\
& x & & & \geq 0, \\
& & & y_k & \geq 0 \quad k = 1, \ldots, K.
\end{aligned} \tag{2.6}$$

Given this structure, it is intuitive to apply Benders Decomposition, which ultimately leads to the L-Shaped method developed by Van Slyke and Wets [Birge and Louveaux, 2011].

2.4 Benders Decomposition

Before diving into the L-Shaped method, let us discuss Benders Decomposition of the extensive form. Our formulation will be based on the explanation provided in [Rahmaniani et al., 2017]. Assume we have a problem of type (2.5) (or equivalently (2.6)). We can split up problem (2.5) into the *master problem*

$$\begin{aligned}
\min_x \quad & c^T x \\
\text{s.t.} \quad & Ax = b, \\
& x \geq 0,
\end{aligned} \tag{2.7}$$

and the K *subproblems*

$$\begin{aligned}
\min_{y_k} \quad & q_k^T y_k \\
\text{s.t.} \quad & T_k x + W y_k = h_k, \\
& y_k \geq 0,
\end{aligned} \tag{2.8}$$

With the master problem and subproblem formulations, the extensive form (2.5) can be reformulated as:

$$\min_{\bar{x} \in X} F(\bar{x}) = \left[c^T \bar{x} + \sum_{k=1}^{K} p_k \cdot \min_{y_k \geq 0} \{ q_k^T y_k \,|\, T_k \bar{x} + W y_k = h_k \} \right], \tag{2.9}$$

where $X = \{x \,|\, Ax = b,\ x \geq 0\}$. Every subproblem is a continuous linear program that can be dualized by means of dual variables π_k associated with the constraint set $T_k \bar{x} + W y_k = h_k$. That is, for every $k = 1, \ldots, K$, we can write:

$$\max_{\pi_k} \{ \pi_k^T (h_k - T_k \bar{x}) \,|\, \pi_k^T W \leq q_k \}. \tag{2.10}$$

Based on strong duality, the primal and dual formulations can be interchanged to extract the following equivalent formulation:

$$\min_{\bar{x}\in X} F(\bar{x}) = \left[c^T\bar{x} + \sum_{k=1}^{K} p_k \cdot \max_{\pi_k}\{\pi_k^T(h_k - T_k\bar{x}) \,|\, \pi_k^T W \leq q_k\}\right]. \tag{2.11}$$

Let us now focus on subproblem k only. The feasible space of subproblem k, i.e., $F_k = \{\pi_k \,|\, \pi_k^T W \leq q_k\}$, is independent of the choice of $\bar{x}$. Thus, if F_k is not empty, the problem can be either unbounded or feasible for any arbitrary choice of $\bar{x}$. In the case that the dual is unbounded, there is a direction of unboundedness μ_{l_k}, $l_k \in L_k$, for which $\mu_{l_k}^T(h_k - T_k\bar{x}) > 0$; this must be avoided because it indicates the infeasibility of the $\bar{x}$ solution. We add a cut

$$\mu_{l_k}^T(h_k - T_k\bar{x}) \leq 0, \quad l_k \in L_k \tag{2.12}$$

to restrict movement in this direction. In the latter case, the solution of the dual problem is one of the extreme points π_{e_k}, $e_k \in E_k$, where E_k denotes the set of extreme points of the feasible space F_k. Now combining these results, we can reformulate (2.11) as:

$$\begin{aligned} \min_{\bar{x}\in X} F(\bar{x}) = \quad & \left[c^T\bar{x} + \sum_{k=1}^{K} p_k \cdot \max_{e_k}\{\pi_{e_k}^T(h_k - T_k\bar{x})\}\right] \\ \text{s.t.} \quad & \mu_{l_k}^T(h_k - T_k\bar{x}) \leq 0, \quad l_k \in L_k,\ k = 1,\dots,K, \end{aligned} \tag{2.13}$$

where $e_k \in E_k$. This problem can be linearized via a continuous variable η to give the following equivalent formulation to problem (2.14), which we refer to as the *Benders Decomposition*:

$$\begin{aligned} \min_{x,\eta}\ F(x) = \quad & c^T x + \eta \\ \text{s.t.} \quad & Ax = b, \\ & \eta \geq \sum_{k=1}^{K} p_k\, \pi_{e_k}^T\,(h_k - T_k x), \quad e_k \in E_k, \\ & 0 \geq\ \mu_{l_k}^T\,(h_k - T_k x), \quad l_k \in L_k,\ k = 1,\dots,K, \\ & x \geq 0. \end{aligned} \tag{2.14}$$

Instead of using one of the dual solutions $\pi_{e_k} \in F_k$ of corresponding subproblem k in the formulation of our optimality cuts, easier would be to obtain the optimal dual solution π_k^* and use this to define the optimality cuts. The optimality cuts can then be formulated as:

$$\eta \geq \sum_{k=1}^{K} p_k\,(\pi_k^*)^T\,(h_k - T_k x). \tag{2.15}$$

Although Benders Decomposition offers an appealing formulation, fully enumerating all feasibility and optimality cuts is impractical. Instead, the L-Shaped method for solving 2-SLPs with finite discrete $\boldsymbol{\xi}$ adopts an iterative approach, progressively generating feasibility and optimality cuts. This procedure will be detailed in the next section.

2.5 L-Shaped method

The L-Shaped method is an iterative variant of Benders Decomposition to solve problems of type (2.5). Its basic idea is to approximate the term $\sum_{k=1}^{K} p_k\, q_k^T y_k$ in the objective function. A general principle behind this approach is that, because this objective term involves a solution of all second-stage recourse linear programs, we want to avoid numerous function evaluations for it. We introduce the variable η as an approximation for $\sum_{k=1}^{K} p_k\, q_k^T y_k$.

In each iteration of the L-Shaped method the master problem is solved, thereby obtaining solutions for x and η. The master problem in iteration 0 only considers the first-stage constraints, and will

therefore return solutions for x and η that lead to a much lower minimum than the actual minimum of problem (2.5). In each iteration of the L-Shaped method, a cut is added to the master problem, incorporating second-stage information. As the master problem is progressively refined, we expect the value of η to increase. The algorithm continues iterating until a convergence criterion is met, at which point it terminates and returns the optimal solutions for x and η. Each iteration consists of three main steps, which will be referred to as step 1, step 2, and step 3 of the L-Shaped method.

In iteration 0, the master problem is formulated as:

$$\begin{aligned} \min_{x^0,\eta} \quad & c^T x^0 + \eta \\ \text{s.t.} \quad & Ax^0 = b, \\ & x^0 \geq 0, \end{aligned} \tag{2.16}$$

To solve this problem, η is initially set to be $-\infty$. The solution x^0 of this problem is called a *first-stage solution.*

Before presenting the three main steps of the L-Shaped method, we define two feasibility sets. We define the *first-stage feasibility set* S_1 as

$$S_1 = \{x \,|\, Ax = b,\, x \geq 0\}. \tag{2.17}$$

The set S_1 is determined by the first-stage constraints: those that do not depend on the particular realization of the random vector. For any given realization ξ, we may define a so-called *elementary feasibility set* as

$$S_2(\xi) = \{x \,|\, y \geq 0 \text{ exists s.t. } Wy = h(\xi) - T(\xi)x\}. \tag{2.18}$$

As $\boldsymbol{\xi}$ is finite discrete, we may easily define the *second-stage feasibility set*

$$S_2 = \bigcap_{k=1}^{K} S_2(\xi_k). \tag{2.19}$$

Let us emphasize that $S_1 \not\subset S_2$. In other words, a solution x to the master problem does not guarantee the feasibility of each subproblem for this specific solution. This will in fact be the main thing to check in the first step of the L-Shaped method, which we will discuss now.

Given a solution $\bar{x} \in S_1$ of the master problem, step 1 of the L-Shaped method determines whether this solution is also second-stage feasible, i.e. $\bar{x} \in S_2$. That is, for each $k = 1, \ldots, K$, the subproblem

$$\begin{aligned} Q(\bar{x}, \xi^k) := \min_{y} \quad & q_k^T y \\ \text{s.t.} \quad & T_k \bar{x} + Wy = h_k, \\ & y \geq 0, \end{aligned} \tag{2.20}$$

is checked to be feasible. If for some k problem (2.20) is infeasible, then we want to exclude the considered first-stage solution $\bar{x}$ among other x's from the first-stage feasibility set S_1 in subsequent iterations of the L-Shaped method. We do this by generating a *feasibility cut*, and adding this cut to the master problem. Using the dual of (2.20), we calculate the dual extreme ray μ_k^* associated with the subproblem constraints and set

$$(\mu_k^*)^T (h_k - T_k x) \leq 0, \quad \forall x \in S_1. \tag{2.21}$$

By letting $\gamma = (\mu_k^*)^T h_k$ and $\Gamma^T = (\mu_k^*)^T T_k$, we obtain the following inequality:

$$\Gamma^T x \geq \gamma, \quad \forall x \in S_1. \tag{2.22}$$

This inequality is called the *feasibility cut* and cuts off a part of the first-stage feasibility set S_1 to get rid of the x's leading to the infeasibility of the subproblem for scenario k [Ntaimo, 2015].

Since obtaining the dual extreme ray μ_k^* can be challenging, [Birge and Louveaux, 2011] present an alternative approach to generating a feasibility cut as follows: Instead of checking the infeasibility of problems of type (2.20), we can solve linear programs of the following form and check their objective value:

$$\begin{aligned} \min_{y,v^+,v^-} \; w' = \quad & e^T v^+ + e^T v^- \\ \text{s.t.} \quad & Wy + T_k \bar{x} + Iv^+ - Iv^- = h_k, \\ & y \geq 0,\ v^+ \geq 0,\ v^- \geq 0, \end{aligned} \tag{2.23}$$

where $e^T = (1, \ldots, 1)$. If subproblem (2.20) is infeasible, then subproblem (2.23) has an objective value bigger than 0. The same holds the other way around. Therefore, solving subproblems of type (2.23) provides an alternative method for checking first-stage feasibility, that avoids the need to compute the extreme ray μ_k^* for generating the feasibility cut. Instead, we generate the feasibility cut as follows. If for some k, problem (2.23) has an optimal objective value $w' > 0$, then we get the associated dual solution σ_k^* and set

$$(\sigma_k^*)^T (h_k - T_k x) \leq 0, \quad \forall x \in S_1. \tag{2.24}$$

Again, by letting $\gamma = (\sigma_k^*)^T h_k$ and $\Gamma^T = (\sigma_k^*)^T T_k$, we obtain the feasibility cut:

$$\Gamma^T x \geq \gamma, \quad \forall x \in S_1. \tag{2.25}$$

Adding either one of the feasibility cuts to the master problem completes step 1 of the L-Shaped method. If a feasibility cut is added, we skip step 2 and proceed directly to step 3. Otherwise, if no feasibility cut is added because all subproblems are feasible for the first-stage solution $\bar{x}$, we move on to step 2.

In step 2 of the L-Shaped method, an optimality cut is generated. As we may assume all subproblems (2.20) are feasible, we use their dual solutions π_k^* to generate the optimality cut.

By letting $\beta_0 = \sum_{k=1}^K p_k (\pi_k^*)^T h_k$ and $\beta = \sum_{k=1}^K p_k (\pi_k^*)^T T_k$, we obtain the inequality:

$$\beta^T x + \eta \geq \beta_0, \quad \forall x \in S_1. \tag{2.26}$$

This inequality is called an *optimality cut*. Adding this cut to the master problem completes step 2 of the L-Shaped method.

In step 3 of the L-Shaped method, the master problem is solved. In iteration ν of the algorithm, the master problem looks like:

$$\begin{aligned} l^\nu := \min_{x,\eta} \quad & c^T x + \eta \\ \text{s.t.} \quad & Ax = b, \\ & (\beta^t)^T x + \eta \geq \beta_0^t, \quad t \in \Theta_\nu \\ & (\Gamma^t)^T x \geq \gamma^t, \quad t \notin \Theta_\nu \\ & x \geq 0. \end{aligned} \tag{2.27}$$

The set Θ_ν is defined as $\Theta_\nu := \{t \leq \nu \,|\, \text{an optimality cut was added in iteration } t\}$ and is updated in each iteration ν. Solving problem (2.27) gives solution (x^ν, η^ν).

Since the number of optimality and feasibility cuts is finite, the L-Shaped method is guaranteed to converge. To keep track of convergence, two different methods can be applied. We will discuss them separately.

In [Birge and Louveaux, 2011], the following stopping criterion is provided. During iteration ν, specifically in step 2 of the algorithm, before adding an optimality cut to the master problem, the value of

$$w^\nu := \beta_0 - \beta x^{\nu-1} \tag{2.28}$$

is computed. We expect $w^\nu \geq \eta^{\nu-1}$, as the value of η is supposed to increase in every iteration. Therefore, if $w^\nu \leq \eta^{\nu-1}$, we stop and return $x^{\nu-1}$ as optimal solution. Otherwise, we add the optimality cut to the master problem and proceed to step 3. However, this knowledge is not very useful in predicting the progress of the algorithm, since the gap $(w^\nu - \eta^{\nu-1})$ does not necessarily steadily decrease. For this reason, [Ntaimo, 2015] propose using an alternative stopping criterion, which will be discussed now.

The second approach relies on computing an upper and lower bound of the optimal objective value v^* of (2.5) in every iteration. The notion of gradually narrowing the range of the master objective through a non-decreasing lower and a non-increasing upper bound seems rather appealing. In addition, this measure allows us not to calculate a problem to full optimality, but to cut the algorithm when the upper and lower bound is close enough. Computation of the lower bound in iteration ν is straightforward: the objective value l^ν of problem (2.27) serves as a lower bound of v^*, as its constraints are less restrictive than those in (2.5) and both problems are minimization problems. Computation of the upper bound is a bit more involved.

To compute the upper bound, it is assumed that all subproblems are solvable. Consequently, the upper bound is computed only if step 2 of the L-Shaped method is executed. Let $Q(\bar{x}, \xi^k)$ represent the solution of subproblem k of type (2.20). Then, the expression

$$u(\bar{x}) := c^T \bar{x} + \sum_{k=1}^{K} p_k \, Q(\bar{x}, \xi^k) \tag{2.29}$$

is an upper bound of v^*. In iteration ν the upper bound is thus computed as

$$u^\nu := c^T x^{\nu-1} + \sum_{k=1}^{K} p_k \, Q(x^{\nu-1}, \xi^k). \tag{2.30}$$

Even though the lower and upper bound expressions are similar, there is one big difference. When computing the lower bound, one is allowed to vary the first-stage decisions, while simultaneously taking into account the impact of this choice on the first and second stage–although the effect on second stage can only be approximated through the restrictions on η. When computing the upper bound, however, the inputs are fixed. If the inputs happened to be optimal, the lower bound and upper bound coincide. In this case, we stop and return $x^{\nu-1}$ as an optimal solution. However, it is not necessary to wait until $u^\nu = l^\nu$. Instead, we can decide to stop when the gap satisfies $u^\nu - l^\nu \leq \epsilon |u^\nu|$, for some $\epsilon > 0$. In this case, the solution $x^{\nu-1}$ is ϵ-optimal.

2.6 Sampling

So far, we have explored two approaches for solving 2-SLPs: the first involves rewriting the problem in its extensive form and solving it directly using optimization solvers, while the second applies the L-Shaped method to the 2-SLP. However, both methods have limitations. Specifically, they rely on the assumption of a finite discrete distribution of the random vector $\boldsymbol{\xi}$. Furthermore, even with a finite discrete distribution, the methods may become computationally infeasible when the number of scenarios is too large. To address these challenges, we introduce sampling.

Sampling offers several advantages. First it enables us to solve problems with continuous random variables. More importantly, it significantly reduces the size of any 2-SLP, allowing us to tackle a much broader range of problems. Given that uncertainty is inherent in our problems, the use of sampling is both natural and highly practical.

Sampling techniques can be applied in different ways. The first fundamental approach to sampling is called *Sample Average Approximation*: an 'exterior' sampling approach, in which a sample is selected from the underlying distribution of $\boldsymbol{\xi}$ and a corresponding approximation $\hat{F}_N(x)$ to the objective value function $F(x)$ in (2.3) is defined from this sample. $\hat{F}_N(x)$ is called the *sample-average approximation function*. The algorithmic details of Sample Average Approximation in combination with both of our solution methods will be discussed in section 2.6.1.

Secondly, sampling can be used in 'interior' fashion. Sampling algorithms of this kind aim to solve the original 2-SLP (2.3), but resort to sampling whenever the algorithm requires an (approximate) value of $F(x)$ or subgradient information for $F(x)$ at some point x. Typically, a different sample is used each time function or subgradient information is required. An example of an interior sampling method is the *Stochastic Decomposition* method, which will be discussed in section 2.6.2 [Linderoth et al., 2006].

2.6.1 Sample Average Approximation

The method of Sample Average Approximation is taken from [Linderoth et al., 2006]. The first step in Sample Average Approximation is sampling itself. As before, the vector $\boldsymbol{\xi}$ contains complete information on the random variables. Now the most straightforward way to obtain a sample $\xi^1, \xi^2, \ldots, \xi^N \sim P$ of N realizations of the random vector $\boldsymbol{\xi}$ is through Monte Carlo sampling, where all ξ^i are independent.

A second approach to sampling is Latin Hypercube sampling, yielding dependent ξ^i. In this technique, a sample of size N is constructed by dividing the interval $(0, 1)$ into N subintervals of equal size and picking one number randomly from each subinterval. These N numbers are then shuffled, and the resulting sequence is used to generate random variates for a given distribution, possibly by performing an inverse transform. Assuming that the components of the random vector are independently distributed, the procedure is repeated for each component, yielding a stratified sample of size N for that vector. One advantage of Latin Hypercube Sampling over Monte Carlo sampling is that it often yields a significantly lower variance in the approximation function $\hat{F}_N(x)$. However, since $\hat{F}_N(x)$ remains an unbiased estimator of $F(x)$ for both methods, we will use Monte Carlo sampling for the remainder of this thesis.

The second step in Sample Average Approximation is to construct an approximation of the 2-SLP, as defined in (2.3). For clarity, we restate the 2-SLP below in a slightly more compact formulation:

$$\min_{x \in X} F(x) = c^T x + \mathbb{E}_{\boldsymbol{\xi}} \left[Q(x, \boldsymbol{\xi}) \right]. \tag{2.31}$$

In this formulation, $X = \{x \mid Ax = b,\ x \geq 0\}$ is the first-stage feasibility set and $Q(x, \xi)$ is the recourse function, as defined in (2.4). Let v^* denote the optimal value of (2.31) and x^* its corresponding solution. Given a sample $\xi^1, \ldots, \xi^N \sim P$ of N realizations of the vector $\boldsymbol{\xi}$, we can approximate problem (2.31) by replacing $F(x)$ by an approximation based on this sample:

$$\begin{aligned} \min_{x \in X} \hat{F}_N(x) \quad &:= \frac{1}{N} \sum_{i=1}^{N} F(x, \xi^i) \\ &= c^T x + \frac{1}{N} \sum_{i=1}^{N} Q(x, \xi^i) \end{aligned} \tag{2.32}$$

The function $\hat{F}_N(x)$ is a sample-average approximation (SAA) to the objective $F(x)$ of (2.3). Since the realizations ξ^i have the same probability distribution P, it follows that $\hat{F}_N(x)$ is an unbiased estimator of $F(x)$, for any x.

The third and final step in this method is to solve the SAA problem (2.32). From a computational point of view, one can regard the SAA problem as a 2-SLP with a finite number of scenarios $\xi^1, \ldots, \xi^N$, each with equal probability $p_i = N^{-1}$. Therefore, any numerical algorithm suitable for solving the corresponding stochastic problem with a discrete distribution can be applied to the SAA problem. Solving (2.32) yields the optimal value $\hat{v}_N$ and its corresponding solution $\hat{x}_N$. These quantities provide approximations to the optimal value v^* and solution x^* of (2.31) respectively. In particular, the mean of $\hat{v}_N$ serves as a lower bound of v^*, which is why we refer to $\hat{v}_N$ as a *lower bound estimate* of v^*. This

can be proven as follows:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\xi}}[\hat{v}_N] &= \mathbb{E}_{\boldsymbol{\xi}}\left[\min_{x\in X}\hat{F}_N(x)\right] \\ &\leq \min_{x\in X}\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{F}_N(x)\right] \\ &= \min_{x\in X}F(x) \\ &= v^*.\end{aligned} \tag{2.33}$$

In the first step of the proof, we used

$$\begin{aligned}\min_{x\in X}\hat{F}_N(x) \leq \hat{F}_N(\bar{x}) &\implies \mathbb{E}_{\boldsymbol{\xi}}\left[\min_{x\in X}\hat{F}_N(x)\right] \leq \mathbb{E}_{\boldsymbol{\xi}}\left[\hat{F}_N(\bar{x})\right] \\ &\implies \mathbb{E}_{\boldsymbol{\xi}}\left[\min_{x\in X}\hat{F}_N(x)\right] \leq \min_{x\in X}\mathbb{E}_{\boldsymbol{\xi}}\left[\hat{F}_N(x)\right],\end{aligned} \tag{2.34}$$

and in the second step, we used that $\hat{F}_N(x)$ is an unbiased estimator of $F(x)$. Given a first-stage solution $\bar{x}$, one can also show that the mean of $\hat{F}_N(\bar{x})$ is an upper bound of v^*. In particular, $\hat{F}_N(\bar{x})$ serves as an *upper bound estimate*, since

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\xi}}[\hat{F}_N(\bar{x})] &= F(\bar{x}) \\ &\geq \min_{x\in X}F(x) \\ &= v^*.\end{aligned} \tag{2.35}$$

Thus, performing SAA in combination with some numerical algorithm for solving the SAA problem (2.32) returns solution $\hat{x}_N$, optimal value $\hat{v}_N$ serving as a lower bound estimate, and upper bound estimate $\hat{F}_N(\bar{x})$. As SAA is an approximation method, it is usually repeated a number M times, such that the first-stage solutions and the upper and lower bound estimates can be averaged to improve reliability. Additionally, confidence intervals for the average upper and lower bound estimate can be created using the Central Limit Theorem. The steps of finding the average lower bound estimate $L_{M,N}$ and its confidence interval are as follows:

1. Using the lower bound estimates $\{\hat{v}_N^j\}_{j=1}^M$, compute

   $$L_{N,M} := \frac{1}{M}\sum_{j=1}^{M}\hat{v}_N^j.$$

   $L_{N,M}$ is an unbiased estimator of $\mathbb{E}[\hat{v}_N]$.

2. When the M replications are i.i.d., by the Central Limit Theorem it follows that

   $$\sqrt{M}\left(L_{N,M} - \mathbb{E}[\hat{v}_N^j]\right) \xrightarrow{d} \mathcal{N}(0,\sigma_L^2), \quad M\to\infty.$$

   where $\sigma_L^2 = \mathrm{Var}(\hat{v}_N)$. The sample variance estimator of σ_L^2 is

   $$s_L^2(M) := \frac{1}{M-1}\sum_{j=1}^{M}(\hat{v}_N^j - L_{N,M})^2.$$

   Defining z_α to satisfy $\mathbb{P}(\mathcal{N}(0,1)\leq z_\alpha) = 1-\alpha$ and replacing σ_L by $s_L(M)$, we can obtain an approximate $(1-\alpha)$-confidence interval for $\mathbb{E}[\hat{v}_N]$ to be

   $$\left[L_{N,M} - \frac{z_{\alpha/2}s_L(M)}{\sqrt{M}},\ L_{N,M} + \frac{z_{\alpha/2}s_L(M)}{\sqrt{M}}\right].$$

The process for computing the average upper bound estimate $U_{M,N}(\bar{x})$, along with its confidence interval, is similar.

This concludes the method of Sample Average Approximation. We have now established how to construct the approximation function $\hat{F}_N(x)$ and use it to obtain an approximate first-stage solution $\hat{x}_N$, along with lower and upper bound estimates, $\hat{v}_N$ and $\hat{F}_N(\bar{x})$, of v^*. By repeating steps 1-3 a fixed number of M times, we can refine these estimates to achieve greater accuracy.

2.6.2 Stochastic Decomposition

Stochastic Decomposition is an interior sampling approach designed to approximate solutions of the two-stage stochastic linear program (2.31). The first version of the SD algorithm was introduced by Higle and Sen in [Higle and Sen, 1991], and since then, numerous variations have been proposed. This section provides an overview of the fundamental steps of the algorithm, as presented in [Sen and Liu, 2016], but leaves out some mathematical details. For a more in-depth discussion, including mathematical derivations and implementation results, we refer to the source.

Unlike SAA, where a fixed sample size N is chosen in advance, the SD algorithm adaptively determines a sufficiently large sample size during its execution. Rather than optimizing a single sample average function $\hat{F}_N$, SD iteratively constructs successive approximations $\hat{f}_k$ at each iteration k of the algorithm, until a specified stopping criterion is met, at which point it returns a lower bound estimate and an incumbent solution. These approximations satisfy

$$\hat{f}_k(x) \leq \hat{F}_k(x),$$

where $\hat{F}_k$ denotes a sample average function with a sample size of k (as in (2.32)). We refer to the functions $\hat{f}_k$ as Value Functions (VF).

Iteration 0, which initializes the algorithm, obtains the incumbent solution $\hat{x}^0$ by solving the master problem:

$$\hat{x}^0 = \{c^T x \,|\, x \in X\},$$

where X again denotes the first-stage feasiblity set. Additionally, an initial lower bound approximation $\hat{f}_0(x)$ of $F(x)$ is defined.

Iteration k of the algorithm then proceeds as follows:

1. **Obtain candidate decision** x^k: Solve the optimization problem using the current approximation of the VF $\hat{f}_{k-1}(x)$:
$$x^k = \arg\min \left\{ \hat{f}_{k-1}(x) \,|\, x \in X \right\}.$$
Alternatively, a regularized version of the VF approximation can be optimized:
$$x^k \in \arg\min \left\{ \hat{f}_{k-1}(x) + \frac{\rho}{2}\|x - \hat{x}^{k-1}\|^2 \,|\, x \in X \right\}.$$

2. **Update VF approximation:** Add one sampled outcome ξ^k to the available sample and solve the second-stage problem for the current incumbent $\hat{x}^0$. Construct a new approximation $\hat{f}_k(x)$ by adding cutting planes, ensuring that it remains a valid lower bound on $F(x)$.

3. **Update incumbent solution for first-stage solution:** Determine whether the candidate x^k or the previous incumbent $\hat{x}^{k-1}$ provides a better solution. A possible way to do this is to define
$$\Delta_k := \hat{f}_{k-1}(x^k) - \hat{f}_{k-1}(\hat{x}^{k-1})$$
denoting the predicted change under the VF approximation $\hat{f}_{k-1}$. The next incumbent solution $\hat{x}^k$ is then selected as follows, using a fixed parameter $r \in (0, 1)$:
$$\hat{x}^k = \begin{cases} x^k & \text{if } \hat{f}_k(x^k) < \hat{f}_k(\hat{x}^{k-1}) + r\Delta_k, \\ \hat{x}^{k-1} & \text{otherwise.} \end{cases}$$

4. **Check stopping criterion:** If the approximations have stabilized within a given tolerance, the algorithm terminates, returning lower bound estimate $\hat{f}_k(\hat{x}_k)$ and incumbent $\hat{x}_k$.

Just like in the Sample Average Approximation method, the algorithm is typically repeated M times rather than just once to improve the accuracy of the solution and lower bound estimates. In most cases, the final estimates are obtained by averaging the solutions and lower bounds across all repetitions.

3 Oemof

As mentioned in the introduction, we will apply the solution methods described in the previous section to an existing large-scale energy model, oemof-B3. Before proceeding with the application of these methods, we must first understand the model's formulation and extend it to a two-stage stochastic linear programming model. This section focuses on these preparatory steps. We begin by analyzing the oemof modeling framework, which serves as a method for energy system modeling. As we deepen our understanding of the framework, we derive the linear program that represents the oemof-B3 model, based on the graph structure. Finally, we extend this linear program to a two-stage stochastic linear program by splitting the variables and constraints into two stages and incorporating random variables where appropriate.

3.1 The modeling framework

The main concept behind oemof, as presented in [Hilpert et al., 2018], is to represent an energy system as a graph, consisting of nodes N and edges E. The set of nodes N contains the components C and the buses B. Components represent actual sources, sinks, or transformers within the energy system. Examples include a photovoltaic (PV) system (source), the energy demand of a group of households (sink), or a power plant that converts gas into electricity (transformer). Buses, on the other hand, represent the connections between components. For example, an electricity bus in an energy system links the PV system to the demand of a group of households. As an additional requirement, we require buses to be solely connected to components and vice versa. The edges E represent the connections within the energy system, and they are directed to indicate the flow of energy. A schematic illustration of an energy system represented as an oemof network is shown in figure 3.1. For clarity, the buses and components in the energy system have all been given different shapes and colors.

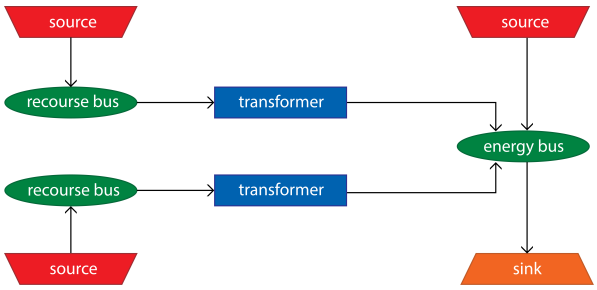

Figure 3.1: Schematic illustration of an energy system represented as an oemof network. Buses are depicted as green ovals, transformer components as blue rectangles, source components as red trapezoids, and sink components as orange trapezoids.

Mathematically speaking, an energy system represented in such a way can be described as a directed bipartite graph G. The formulation of this graph in its general form is given in equation (3.1).

$$\begin{aligned} &G := (N, E) \\ &N := \{B, C\} \\ &E \subseteq B \times C \cup C \times B \\ &C^+ \subseteq C \\ &C^- \subseteq C \\ &Tr \subseteq C \end{aligned} \tag{3.1}$$

As mentioned earlier, the set of components consists of sources C^+, sinks C^- and transformers Tr. Each of these component types has the following characteristics:

- Transformers have inflows and outflows.
- Sinks only have inflows but no outflows.
- Sources have outflows but no inflows.

Furthermore, the set of transformers includes the greatest variety of components, so it is often subdivided into more specific categories to simplify the model formulations.

3.2 Derivation of the linear program

Given a graph G that adheres to the modeling framework described above, the goal is to derive a linear program from this graph, to ultimately be able to determine the optimal energy flow while minimizing overall costs. Figure 3.2 presents the graph of an example energy system featuring 'real-life' components that comply with the modeling framework outlined in section 3.1. This example will serve as the basis for explaining the derivation of the linear program in the rest of the section.

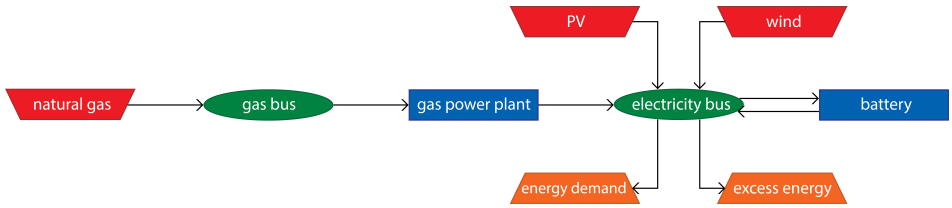

Figure 3.2: Graph of an example energy system complying with the modeling framework in section 3.1.

Energy sources include natural gas, photovoltaic (PV) systems, and wind energy; the sinks consist of demand and excess energy; and the transformers are represented by the gas power plant and the battery. The two transformers are of different types. Namely, the power plant is a converter, whereas the battery is a storage component. This distinction will be important when deriving the LP.

We begin by defining the decision variables and formulating the objective function of the LP. The decision variables represent the energy flow through the edges of a given graph, which must be determined for each hour of a full year. Specifically, we define $f_{(s,e)}(t)$ as the flow through edge (s, e) at time t. Certain flows are predetermined, including the energy input from renewable sources and the required energy flow to meet demand at the sink component. These predetermined values will be incorporated when defining the constraints of the LP.

The model aims to minimize the marginal costs over the course of an entire year. The set T represents all hourly time steps in a year, defined as $T = \{0, 1, \ldots, 8759\}$. The marginal costs, denoted $c_{(s,e)}$, represent the costs per unit of energy flow through edge (s, e). These costs are assumed to be time-independent.

The objective can be formulated as:

$$\min_{f} \sum_{t \in T} \sum_{(s,e) \in E} c_{(s,e)} \cdot f_{(s,e)}(t) \tag{3.2}$$

- $f_{(s,e)}(t)$: flow through edge (s, e) at time t [MW].
- $c_{(s,e)}$: marginal costs of transporting 1 MW through edge (s, e) [Eur/MW].

The first set of constraints derived from a given graph are the bus balance constraints. These ensure the conservation of energy flow at all bus components. In the graph shown in figure 3.2, these components include the gas bus and the electricity bus. The balance condition can be expressed as:

$$\sum_{(s,b) \in E} f_{(s,b)}(t) = \sum_{(b,e) \in E} f_{(b,e)}(t) \quad \forall b \in B, \ \forall t \in T. \tag{3.3}$$

This equation ensures that, at every time step, the total incoming energy flow to a bus equals the total outgoing energy flow.

The second set of constraints represent the balance around converter components, which, in the graph in figure 3.2, only includes the power plant. These constraints ensure the conservation of energy flow while accounting for a conversion rate k_{co}. Since converter components generally have one inflow and one outflow, the constraint can be expressed as:

$$k_{co} \cdot f_{(s,co)}(t) = f_{(co,e)}(t) \quad \forall co \in Co,\ \forall t \in T. \tag{3.4}$$

- k_{co}: conversion rate of $co \in Co$ [no unit].

The final set of constraints derived from the given graph pertains to storage components. In the graph shown in figure 3.2, the only storage component present is the battery. Since the storage content and the storage loss at each time step t must be determined by the linear program, we introduce them as decision variables: $cont_{st}(t)$ for the stored energy and $loss_{st}(t)$ for storage losses, for all $st \in St$. Here, St denotes the set of storage components. The constraints are as follows:

$$\begin{aligned} & loss_{st}(t) = \lambda_{st} \cdot cont_{st}(t) && \forall st \in St,\ \forall t \in T, \\ & k^{in}_{st} \cdot f_{(b,st)}(t) + cont_{st}(t) - loss_{st}(t) = && \tfrac{1}{k^{out}_{st}} \cdot f_{(st,b)}(t) + cont_{st}(t+1), \\ & && \forall st \in St,\ \forall t \in T, \\ & cont_{st}(0) = cont_{st}(8760) && \forall st \in St. \end{aligned} \tag{3.5}$$

- $cont_{st}(t)$: content of $st \in St$ at time t [MWh].
- $loss_{st}(t)$: loss of $st \in St$ at time t [MWh].
- λ_{st}: loss rate of $st \in St$ [no unit].
- k^{in}_{st}: conversion rate for the incoming flow of $st \in St$ [no unit].
- k^{out}_{st}: conversion rate for the outgoing flow of $st \in St$ [no unit].

The first of the three constraints in (3.5) states that the loss of the storage component at time t equals the loss rate times the content of the storage component. The loss rate λ_{st} can take any value $0 \leq \lambda_{st} < 1$, but is generally very small. In the case no loss rate is considered, this constraint can be disregarded.

The second constraint in (3.5) is a constraint of main importance, as it describes the energy balance around the storage component while also linking time step t to time step $t+1$. The constraint states that the storage inflow, plus the stored content minus the loss at time t, must equal the outflow at time t plus the storage content at time $t+1$. The last constraint in (3.5) ensures that the storage component's energy content at time $t = 0$ matches its content at time $t = 8760$.

The bounds on the flow decision variables are straightforward. All flows must be non-negative, i.e., they are constrained to be greater than or equal to zero. Additionally, an upper bound can be imposed on the flow, denoted as $f^{max}_{(s,e)}$, which serves as the maximum allowable value for $f_{(s,e)}(t)$ at every time step t. The flow bounds can then be defined as:

$$0 \leq f_{(s,e)}(t) \leq f^{max}_{(s,e)} \quad \forall f,\ \forall t \in T. \tag{3.6}$$

Similarly, the variables $loss_{st}(t)$ and $cont_{st}(t)$ are defined non-negative for all storage components $st \in St$. Furthermore, the storage content at every time step t is always constrained to be less than or equal to the maximum capacity of the storage component, denoted as cap_{st}. The storage bounds can be defined as:

$$\begin{aligned} 0 \leq \quad & loss_{st}(t) \quad \leq \infty \quad \forall st \in St,\ \forall t \in T, \\ 0 \leq \quad & cont_{st}(t) \quad \leq cap_{st} \quad \forall st \in St,\ \forall t \in T. \end{aligned} \tag{3.7}$$

The objective function (3.2), along with the constraints in (3.3), (3.4), and (3.5), and the bounds in (3.6) and (3.7), together form the complete LP formulation for the graph in figure 3.2. The complete model formulation can additionally be found in appendix A.1.

3.3 Oemof-B3

The oemof-B3 model is an energy model based on the oemof framework, designed to represent the energy system of the Berlin-Brandenburg area. It covers two federal states —Berlin and Brandenburg— which will be referred to as 'regions'. The two regions are connected by a converter component, which represents an electricity transmission line. Through this line, electricity can flow between regions, subject to a conversion factor. The model includes one constraint to account for this. Apart from this constraint, the regions are treated independently.

Similar to the example energy system in figure 3.2, the oemof-B3 model consists of nodes N and edges E. The set of nodes N contains the components C and the buses B. Components represent actual sources, sinks, or transformers within the energy system, while buses represent the connections between these components. Figure 3.3 presents the graphical representation of one region of the oemof-B3 model, depicting all buses and transformers while omitting sources and sinks for simplicity. To clarify the figure further, we make the following three remarks:

- The edges E of the graph, shown as arrows, represent connections within the energy system through which energy flows.
- The complete energy system consists of two distinct regions: Berlin and Brandenburg. As a result, all components and buses appear twice in the complete energy system, including sources and sinks–though the figure only depicts them in one region. The only unique element is the electricity transmission line. The dotted arrows illustrate the connections between the regions but are not actually part of the region shown in the figure.
- All buses and transformers are labeled according to their names in the linear program. For instance, the storage component 'heatcen_storage' corresponds to the model component 'GenericInvestmentStorageBlock_total(BB_heat_central_storage_0)', where BB indicates that this storage component belongs to the Brandenburg region.

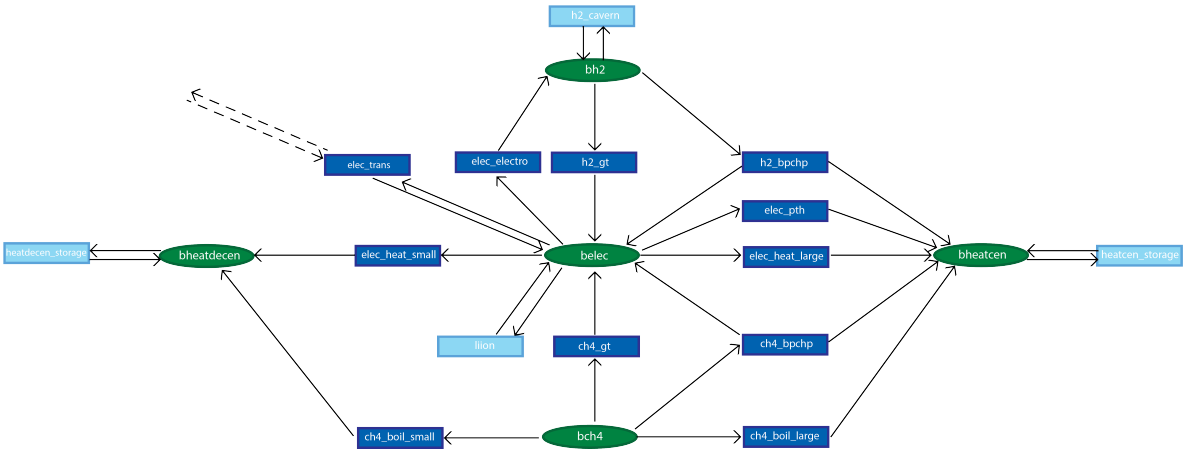

Figure 3.3: Graphical representation of the buses, transformers, and their connections in the oemof-B3 energy system. Buses are shown as green ovals, transformer components as blue rectangles, and storage components (which are a subset of the transformer components) as light blue rectangles.

Given the graphical representation, a linear program can be formulated to determine the optimal energy flow within the model. The full derivation of the LP will is not discussed here, as it involves numerous decision variables and constraints. Instead, we focus on deriving only the 'relevant' parts of the LP, which will later enable us to extend it to a 2-SLP. However, a complete description of the LP is provided in appendix A.2.

Similar to our example energy system, the objective of the linear program is to determine the optimal energy flows while minimizing overall costs. However, in this case, the total costs include not only the marginal costs but also the investment costs of storage components and transmission lines (i.e., the

edges $(s, e) \in E$). As a result, the objective function consists of two parts: the first part minimizes the investment costs, and the second part minimizes the marginal costs. These two parts will be derived separately.

The first part of the objective function, which is time-independent, focuses on determining the capacity investments for storage components and transmission lines, while minimizing the total investment costs. We define decision variables $C_{st}^{storinv,r}$ to represent the capacity investment in storage components, and $C_{(s,e)}^{flowinv,r}$ to represent the capacity investment in transmission lines. Corresponding to these decision variables are the investment costs $c_{st}^{stor,r}$ for storage components and $c_{(s,e)}^{flow,r}$ for transmission lines. The time-independent part of the objective function can then be formulated as:

$$\min_{C} \sum_{r \in R} \sum_{st \in St} c_{st}^{stor,r} \cdot C_{st}^{storinv,r} + \sum_{r \in R} \sum_{(s,e) \in E} c_{(s,e)}^{flow,r} \cdot C_{(s,e)}^{flowinv,r}. \tag{3.8}$$

- $C_{st}^{storinv,r}$: capacity investment in storage component st in region r [MWh].
- $C_{(s,e)}^{flowinv,r}$: capacity investment in transmission line (s, e) in region r [MW].
- $c_{st}^{stor,r}$: investment costs of storage component st in region r [Eur/MWh].
- $c_{(s,e)}^{flow,r}$: investment costs of transmission line (s, e) in region r [Eur/MW].

The second part of the objective function minimizes the marginal costs of the energy system over the entire year. We define the decision variables $f_{(s,e)}^{r}(t)$ to represent the flow through connection (s, e) at time t. Related to the flows are the marginal costs, denoted $c_{(s,e)}^{mar,r}$. The second, time-dependent part of the objective function can be formulated as:

$$\min_{f} \sum_{t \in T} \sum_{r \in R} \sum_{(s,e) \in E} c_{(s,e)}^{mar,r} \cdot f_{(s,e)}^{r}(t). \tag{3.9}$$

- $f_{(s,e)}^{r}(t)$: flow through connection (s, e) at time t in region r [MW].
- $c_{(s,e)}^{mar,r}$: marginal costs of transporting 1 MW through edge (s, e) in region r [Eur/MW].

To summarize, the complete objective function becomes:

$$\min_{C,f} \sum_{r \in R} \sum_{st \in St} c_{st}^{stor,r} \cdot C_{st}^{storinv,r} + \sum_{r \in R} \sum_{(s,e) \in E} c_{(s,e)}^{flow,r} \cdot C_{(s,e)}^{flowinv,r} + \sum_{t \in T} \sum_{r \in R} \sum_{(s,e) \in E} c_{(s,e)}^{mar,r} \cdot f_{(s,e)}^{r}(t). \tag{3.10}$$

Similar as in the objective function, the constraints can be categorized as either time-independent or time-dependent. Naturally, constraints that include the investment decision variables $C_{st}^{storinv,r}$ or $C_{(s,t)}^{flowinv,r}$, but do not include the flow decision variables $f_{(s,e)}^{r}(t)$ are time-independent. Two constraints satisfy this condition:

$$C_{st}^{storinv,r} \geq C_{st}^{init,r} \quad \forall st \in St,\ \forall r \in R, \tag{3.11}$$

which ensures the capacity investment is at least as large as the initial storage content. Here, the initial storage content $C_{st}^{init,r}$ is also a decision variable. The second constraint is given by:

$$C_{(b,st)}^{flowinv,r} = C_{(st,b)}^{flowinv,r} \quad \forall st \in St,\ \forall r \in R, \tag{3.12}$$

which ensures that the capacity of the inflowing and outflowing transmission line of a storage component are equal.

The linear program derived from the graph in figure 3.3 consists of numerous time-dependent constraints due to the complexity of the system and the large number of components involved. Among these, the bus balance constraints are particularly noteworthy. The energy system comprises five buses,

each of which must adhere to the principle of flow conservation. As in (3.3), this requirement can be expressed as:

$$\sum_{(s,b)\in E} f^r_{(s,b)}(t) = \sum_{(b,e)\in E} f^r_{(b,e)}(t) \quad \forall b \in B,\ \forall r \in R,\ \forall t \in T. \tag{3.13}$$

Focusing specifically on the electricity bus, which is centrally located in figure 3.3, we observe that it has numerous inflows and outflows. In addition to its connections to 10 transformer components, it is also linked to 5 source components and 4 sink components, as shown in figure 3.4.

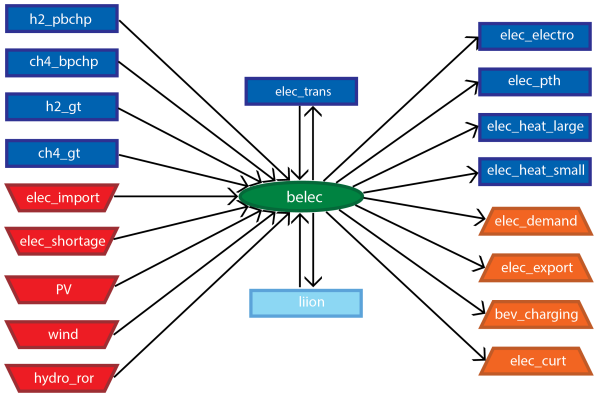

Figure 3.4: In- and outflows of the electricity bus in the oemof-B3 model.

Among these in- and outflows, two are provided as input data:

- The flow $f^r_{(belec,elec\_demand)}(t)$ represents the electricity demand at time t.
- The flow $f^r_{(belec,bev\_charging)}(t)$ represents electric vehicle (BEV) charging at time t.

In a more robust and realistic version of the model, electricity demand does not necessarily have to be predetermined for every time step throughout the entire year, but is subject to uncertainty. In our extension of the LP to a two-stage stochastic linear program (2-SLP), we will explicitly incorporate this uncertainty.

3.4 Extension of the oemof-B3 LP to a 2-SLP

Electricity demand is a key component of the oemof-B3 model, as it represents the largest share of total demand in the system, and heavily influences capacity investment decisions. For this reason, making an estimate for the demand is insufficient. Instead, we apply the stochastic linear programming framework to carefully account for this uncertainty. In the following section, we describe how to extend the oemof-B3 LP to a 2-SLP, allowing us to incorporate uncertainty in electricity demand.

The first step in extending the LP to a 2-SLP is to introduce two stages. Given that the model naturally consists of time-independent variables and time-dependent variables, it is intuitive to use this distinction to define the stages. The first stage comprises the time-independent variables, which include the capacity investment decisions $C^{storinv,r}_{st}$ and $C^{flowinv,r}_{(s,e)}$, as well as the initial storage content decision $C^{init,r}_{st}$. The second stage comprises the time-dependent variables, which include the flow decisions $f^r_{(s,e)}(t)$. Splitting up the variables, along with the objective function and the constraints in

two stages yields the 2-SLP:

$$
\begin{aligned}
\min_{C} \quad & \sum_{r\in R}\sum_{st\in St} c_{st}^{stor,r}\cdot C_{st}^{storinv,r} + \sum_{r\in R}\sum_{(s,e)\in E} c_{(s,e)}^{flow,r}\cdot C_{(s,e)}^{flowinv,r} + \mathbb{E}_{\boldsymbol{\xi}}[Q(C,\boldsymbol{\xi})] \\
\text{s.t.} \quad & C_{st}^{storinv,r} \geq C_{st}^{init,r} \quad \forall st\in St,\ \forall r\in R, \\
& C_{(b,st)}^{flowinv,r} = C_{(st,b)}^{flowinv,r} \quad \forall st\in St,\ \forall r\in R, \\
& C_{st}^{storinv,r},\ C_{st}^{init,r} \geq 0 \quad \forall st\in St,\ \forall r\in R, \\
& C_{(s,e)}^{flowinv,r} \geq 0 \quad \forall (s,e)\in E,\ \forall r\in R.
\end{aligned}
\tag{3.14}
$$

The recourse function $Q(C,\boldsymbol{\xi})$ and the uncertainty vector $\boldsymbol{\xi}$ will be formally defined later. For now, we focus on the master problem, which corresponds to problem (3.14) without the term $\mathbb{E}_{\boldsymbol{\xi}}[Q(C,\boldsymbol{\xi})]$ in the objective function. Since this formulation only involves first-stage decision variables in the constraints, a first-stage solution can be obtained by setting all variables to zero. However, this solution is infeasible in the second stage, as it would fail to satisfy energy demands and other operational constraints.

Now that we adopted our model to the 2-SLP framework, we can introduce uncertainty in the second stage. As previously discussed, we will uncorporate uncertainty in electricity demand. Specifically, for all $r\in\{Be, BB\}$ and all $t\in T$ we define the random variable $\tilde{f}^{r}_{(belec,elec\_demand)}(t)$ as follows:

$$
\tilde{f}^{r}_{(belec,elec\_demand)}(t) = \begin{cases} 0.9\cdot f^{r}_{(belec,elec\_demand)}(t) & p=0.33, \\ f^{r}_{(belec,elec\_demand)}(t) & p=0.33, \\ 1.1\cdot f^{r}_{(belec,elec\_demand)}(t) & p=0.34. \end{cases}
\tag{3.15}
$$

This random variable can be interpreted as follows. Given the original electricity demand, the actual demand fluctuates with a probability 1/3, either decreasing to 90% or increasing to 110% of its original value. Since the model considers two regions, two random variables $\tilde{f}$ are introduced per time step. The full set of these random variables is represented by the uncertainty vector $\boldsymbol{\xi}$, which explains the expectation over $\boldsymbol{\xi}$ in the objective function of (3.14). Because all random variables are independent, the total number of possible realizations of $\boldsymbol{\xi}$ after T time steps is 3^{2T}. For a specific realization ξ of $\boldsymbol{\xi}$, the recourse function $Q(C,\xi)$ is defined:

$$
\begin{aligned}
Q(C,\xi) := \min_{f} \quad & \sum_{t\in T}\sum_{r\in R}\sum_{(s,e)\in E} c_{(s,e)}^{mar,r}\cdot f^{r}_{(s,e)}(t) \\
\text{s.t.} \quad & \sum_{(s,b)\in E} f^{r}_{(s,b)}(t) = \sum_{(b,e)\in E} f^{r}_{(b,e)}(t) \quad \forall b\in B\setminus\{belec\},\ \forall r\in R,\ \forall t\in T, \\
& \sum_{(s,belec)\in E} f^{r}_{(s,belec)}(t) - \sum_{(belec,e)\in E\setminus\{(belec,elec\_demand)\}} f^{r}_{(belec,e)}(t) \\
& = \tilde{f}^{r}_{(belec,elec\_demand)}(t,\xi) \quad \forall r\in R,\ \forall t\in T, \\
& \ \ \vdots \\
& f^{r}_{(s,e)}(t) \geq 0, \quad \forall f,\ \forall r\in R,\ \forall t\in T,
\end{aligned}
\tag{3.16}
$$

where $\tilde{f}(\xi)$ is the outcome of the random variable $\tilde{f}$ in scenario ξ. Let us emphasize that this formulation of the recourse function $Q(C,\xi)$ is an oversimplified version of the true formulation, which, apart from the bus balance constraints, considers all additional second-stage constraints. We have, however, chosen this representation because it clearly introduces the uncertainty in the electricity bus balance constraint. Specifically, instead of using the deterministic value $f^{r}_{(belec,elec\_demand)}(t)$, we incorporate an outcome $\tilde{f}(\xi)$ of the random variable $\tilde{f}$.

4 Method

In this section, we outline the methods used to solve two-stage stochastic linear programs (2-SLPs). Two solution approaches were implemented: the Extensive Form method and the L-Shaped method. The experiments were performed on a MacBook Air (2024) equipped with an Apple M3 chip, featuring an 8-core CPU and a 10-core GPU. The device has 8 GB of unified memory and runs on macOS Sonoma, version 14.5. The programming language used for the implementation was Python 3.12.4, and the optimization solver employed was Gurobi version 11.0.3.

The section begins with an overview of the input data and its corresponding file format. Next, we describe the construction of the oemof-B3 model in this format. Following this, we detail the algorithmic steps of each solution method, emphasizing their key components and implementation. Finally, we discuss the use of high-performance computing to optimize both methods.

4.1 Input data

To evaluate the performance of the implemented methods, a set of diverse test problems was selected: LandS, LandS2, LandS3, baa99, pgp2, 20, SSN, and storm. All problems are two-stage stochastic linear programs with recourse obtained from [Linderoth et al., 2002]. Some of these problems will be briefly described here, for others just the problem data will be presented.

LandS
LandS is a simple problem in electrical investment planning presented in [Louveaux and Smeers, 1988]. In fact, the example problem (1.9) in the introduction is a simplified version of this problem. In the original problem, there are four different technologies, indexed by $i = 1, 2, 3, 4$. For each technology, the amount of capacity devoted to producing electricity in each of the three modes, indexed by $j = 1, 2, 3$, has to be determined. As in our example problem, the second-stage constraints include capacity constraints for each of the four technologies, and the demand constraints, which have the form

$$\sum_{i=1}^{4} y_{ij} \geq \tilde{d}_j, \quad j = 1, 2, 3, \tag{4.1}$$

where $\tilde{d}_1, \tilde{d}_2$ and $\tilde{d}_3$ are the (possibly uncertain) demands. In the problem LandS, $\tilde{d}_2$ and $\tilde{d}_3$ are fixed at 3 and 2, respectively, while $\tilde{d}_1$ takes on three possible values. More specifically, $\tilde{d}_1$ is defined as:

$$\tilde{d}_1 = \begin{cases} 3 & p = 0.3, \\ 5 & p = 0.4, \\ 7 & p = 0.3. \end{cases} \tag{4.2}$$

The total number of scenarios is therefore 3. In LandS2, the problem is modified to allow four different values for each of these random variables, namely, for $j = 1, 2, 3$:

$$\tilde{d}_j = \begin{cases} 0 & p = 0.25, \\ 0.96 & p = 0.25, \\ 2.96 & p = 0.25, \\ 3.96 & p = 0.25. \end{cases} \tag{4.3}$$

The total number of scenarios is therefore 64. Lastly, in LandS3, the problem is modified to allow 100 different values for each of the random variables, namely, for $j = 1, 2, 3$:

$$\tilde{d}_j = 0.04(k - 1), \quad k = 1, \ldots, 100, \tag{4.4}$$

each with $p = 0.01$. The total number of scenarios therefore equals 10^6, each with probability 10^{-6}.

20
This problem is presented in the work of [Mak et al., 1999], which models the operations of a motor freight carrier. In this model, the first-stage variables represent the positions of a fleet of vehicles at the start of the day. The second-stage variables then determine how the fleet moves through a network

to meet point-to-point shipment demands, with penalties for any unsatisfied demands, while ensuring that the fleet ends the day in the same configuration as it began [Linderoth et al., 2006].

SSN
The SSN problem arises in telecommunications network design. The owner of the network sells private-line services between pairs of nodes in the network. When a demand for service is received, a route between the two nodes with sufficient bandwidth must be identified for the time period in question. If no such route is available, the demand cannot be satisfied and the sale is lost. The optimization problem is to decide where to add capacity to the network to minimize the expected rate of unmet requests. In the data set for SSN, there are a total of 89 arcs and 86 point-to-point pairs. That is, the first-stage vector x, which contains the capacity of each arc, has dimension 89, and the dimension of the random vector $\boldsymbol{\xi}$ of demands is 86. Each component of $\boldsymbol{\xi}$ is an independent random variable with a known discrete distribution. Specifically, there are between three and seven possible values for each component of $\boldsymbol{\xi}$, given a total of approximately 10^{70} possible complete demand scenarios [Linderoth et al., 2006].

Oemof-B3
The oemof-B3 2-SLP, as described in section 3.4, includes a total of 3^{2T} possible demand scenarios for a given number of time steps T. Over an entire year, this results in an exceptionally large number of scenarios. Furthermore, the number of second-stage variables and constraints increases linearly with time, leading to a corresponding linear growth in the size of the subproblems. To manage computational complexity in our numerical experiments, we analyze two versions of the oemof-B3 model: one considering only three time steps and another encompassing the full year.

The input problem data for all test problems, along with the specific data for oemof-B3, is summarized in table 1.

Table 1: Input problem data

Name	Application	Random Variables	Scenarios	First-stage (Vars., Cons.)	Second-stage (Vars., Cons.)
lands	Electricity Planning	1	3	$(4, 2)$	$(12, 7)$
lands2	Electricity Planning	3	64	$(4, 2)$	$(12, 7)$
lands3	Electricity Planning	3	10^6	$(4, 2)$	$(12, 7)$
baa99		2	625	$(2, 0)$	$(7, 4)$
pgp2		3	576	$(4, 2)$	$(16, 7)$
20	Vehicle Assignment	40	$1.1 \cdot 10^{12}$	$(63, 3)$	$(764, 124)$
ssn	Telecom Network Design	86	10^{70}	$(89, 1)$	$(706, 175)$
storm	Cargo Flight Scheduling	117	$6 \cdot 10^{81}$	$(121, 185)$	$(1259, 528)$
Oemof-B3 ($T = 3$)	Energy planning	6	729	$(58, 16)$	$(330, 303)$
Oemof-B3 ($T = 8760$)	Energy planning	17 530	$3^{17\,530}$	$(58, 16)$	$(40150, 35779)$

4.2 SMPS file format

All input data sets are given in SMPS format. This is an extension of the MPS file format to the stochastic setting. The SMPS file format comprises three separate files: CORE, TIME and STOCH files. We will now present these files for the test instance LandS described above.

CORE file
The CORE file contains the 'core' problem in MPS file format, which is given as follows:

$$\begin{aligned} \min_{x,y} = \quad & c^T x + q^T y \\ \text{s.t.} \quad & Ax = b, \\ & Tx + Wy = h, \\ & x \geq 0,\ y \geq 0. \end{aligned} \tag{4.5}$$

The core problem contains only deterministic data and no distinction between stages is made yet. However, it defines the ordering of decision variables (columns) x and y as well as the constraints (rows), preparing for the separation into two stages [Ntaimo, 2015]. The CORE file of the test instance LandS is shown in figure 4.1.

```
NAME          LandS
ROWS
 N  OBJ
 G  S1C1
 L  S1C2
 L  S2C1
     .
     .
 G  S2C7
COLUMNS
    X1         OBJ          10.0
    X1         S1C1          1.0
    X1         S1C2         10.0
    X1         S2C1         -1.0
     .          .             .
     .          .             .
    Y43        OBJ           5.5
    Y43        S2C4          1.0
    Y43        S2C7          1.0
RHS
    RHS        S1C1          12.0
    RHS        S1C2          120.0
     .          .             .
     .          .             .
    RHS        S2C5          0.0
    RHS        S2C6          3.0
    RHS        S2C7          2.0
BOUNDS
 LO BND        X1            0.0
     .          .             .
     .          .             .
 LO BND        Y43           0.0
ENDATA
```

Figure 4.1: CORE file of the LandS test instance. The dots have been placed to indicate missing data for better readability.

In the LandS test instance, demand $\tilde{d}_1$ is stochastic, as described by the random variable in equation (4.2). This random variable corresponds to the right-hand side of constraint S2C5 in the CORE file, which is initially set to zero. However, since the randomness is introduced in the STOCH file, this placeholder value is arbitrary and can be disregarded.

TIME file
The TIME file contains the information of the stages. The time file of LandS is given in figure 4.2.

```
TIME          LandS
PERIODS       LP
    X1        S1C1                    ROOT
    Y11       S2C1                    STAGE-2
ENDATA
```

Figure 4.2: TIME file of the LandS test instance.

In this file, it is evident that the first first-stage variable is X1, and the first first-stage constraint is S1C1. All subsequent variables and constraints listed in the CORE file after these will also belong to the first stage. Similarly, Y11 is identified as the first second-stage variable, and S2C1 as the first second-stage constraint. Any variables and constraints appearing after these in the CORE file will be part of the second stage.

STOCH file
As the name implies, the STOCH file contains the information about the random variables, and their distribution. The STOCH file of LandS is given in figure 4.3.

```
STOCH         LandS
INDEP         DISCRETE
    RHS       S2C5          3     0.3
    RHS       S2C5          5     0.4
    RHS       S2C5          7     0.3
ENDATA
```

Figure 4.3: STOCH file of the LandS test instance.

Clearly, the RHS of constraint S2C5 is stochastic, as described by the random variable $\tilde{d}_1$.

4.3 Construction of the SMPS file for the oemof-B3 2-SLP

The SMPS files for all test instances in table 1 are available online. However, since we constructed the 2-SLP for the oemof-B3 model ourselves, we had to construct the corresponding SMPS file accordingly.

The first step in creating the SMPS file is the construction of the CORE file. Since the MPS file follows the same structure as the CORE file, a preliminary CORE file could be easily created by converting the original LP into an MPS file using Gurobi's built-in read and write functions. However, since the variables and constraints in the CORE file must be arranged according to their respective first and second stages, this reordering still had to be done manually.The original LP model of oemof-B3 uses the following names for the first-stage variables and constraints, as listed in table 2.

Table 2: Names of first-stage variables and constraints in oemof-B3 LP file.

Variable	Name in LP file
$C_{st}^{storinv,r}$	GenericInvestmentStorageBlock_invest("r"_"st"_0)
$C_{st}^{init,r}$	GenericInvestmentStorageBlock_init_content("r"_"st")
$C_{(s,e)}^{flowinv,r}$	InvestmentFlowBlock_invest("r"_"s"_"r"_"e"_0)
Constraint	**Name in LP file**
$C_{st}^{storinv,r} \geq C_{st}^{init,r} \quad \forall st \in St,\ \forall r \in R$	c_u_GenericInvestmentStorageBlock_init_content_limit
$C_{(b,st)}^{flowinv,r} = C_{(st,b)}^{flowinv,r} \quad \forall st \in St,\ \forall r \in R$	c_e_GenericInvestmentStorageBlock_power_coupled

Specifically, the first-stage constraints were placed immediately below the objective function in the 'Rows' section, and the first-stage variables were listed first in the 'Columns' section.

The construction of the TIME file was straightforward. The only requirement was identifying the first first-stage variable and constraint, as well as the first second-stage variable and constraint in the CORE file–information we had already determined.

Finally, the STOCH file had to be created. For all $r \in R$ and $t \in T$, we generated row i, where $i = 1, 2, 3$, as follows:

- The first column contained the term 'RHS'.
- The second column listed the constraint name c_e_BusBlock_balance("r"_electricity_0_"t")_.

- The third column held the sum of the random variable outcome $\tilde{f}^r_{(belec,elec\_demand)}(t)$ and the deterministic value $f^r_{(belec,BEV\_charging)}(t)$.
- The fourth column contained the corresponding probability p_i.

The random variable outcomes were computed in Excel, while the STOCH file was generated using Python. The final STOCH file for the oemof-B3 model with is shown in figure 4.4.

```
STOCH   oemofb3_t3
INDEP   DISCRETE
RHS     c_e_BusBlock_balance(BB_electricity_0_0)_      3653.4528731365453      0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_0)_      3806.1154334889115      0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_0)_      3958.7779938412777      0.34
RHS     c_e_BusBlock_balance(BB_electricity_0_1)_      3145.598881135836       0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_1)_      3296.249937572325       0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_1)_      3446.9009940088135      0.34
RHS     c_e_BusBlock_balance(BB_electricity_0_2)_      2967.3494187926863      0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_2)_      3110.695539955726       0.33
RHS     c_e_BusBlock_balance(BB_electricity_0_2)_      3254.041661118766       0.34
RHS     c_e_BusBlock_balance(B_electricity_0_0)_       863.8231126213357       0.33
RHS     c_e_BusBlock_balance(B_electricity_0_0)_       931.3531447589269       0.33
RHS     c_e_BusBlock_balance(B_electricity_0_0)_       998.8831768965181       0.34
RHS     c_e_BusBlock_balance(B_electricity_0_1)_       847.0363803207014       0.33
RHS     c_e_BusBlock_balance(B_electricity_0_1)_       913.6766270140384       0.33
RHS     c_e_BusBlock_balance(B_electricity_0_1)_       980.3168737073756       0.34
RHS     c_e_BusBlock_balance(B_electricity_0_2)_       811.21454421599         0.33
RHS     c_e_BusBlock_balance(B_electricity_0_2)_       874.6234648222995       0.33
RHS     c_e_BusBlock_balance(B_electricity_0_2)_       938.0323854286091       0.34
ENDDATA
```

Figure 4.4: STOCH file of the oemof-B3 2-SLP with three time steps.

4.4 Preparation of Gurobi models for solution methods

Given the input data in SMPS format, it first needed to be read and converted into usable objects to execute our solution methods for solving 2-SLPs. Unfortunately, no optimization solver provides a standard built-in function for reading or solving SMPS files, requiring us to implement this ourselves.

Now, more specifically, the goal was to read the SMPS file format using Python and construct Gurobi models for the master and subproblems of a given 2-SLP. To simplify implementation, we restricted ourselves to 2-SLPs with a finite discrete distribution of $\boldsymbol{\xi}$, where randomness occurs only in the right-hand side of the constraints. That is, all input problems can theoretically be written in extensive form as follows:

$$\begin{aligned} \min_{x,y} \; F(x) = \quad & c^T x + \sum_{k=1}^{K} p_k \, q^T y_k \\ \text{s.t.} \quad & Ax = b, \\ & Tx + W y_k = h_k, \quad k = 1, ..., K, \\ & x \geq 0, \; y_k \geq 0, \quad k = 1, ..., K. \end{aligned} \tag{4.6}$$

This formulation is similar to the one presented in (2.5); however, note that the vector q in the objective function and the matrix T are now fixed.

The GitHub repository readSMPS-Py by siavashtab provided a framework for reading SMPS files. This repository was forked as a starting point, and modifications were made to adapt it to our needs. Specifically, the file decmps_2slp.py was modified to implement the following functions, each constructing a specific Gurobi model:

- The function `create_master`() creates the master problem:

$$\begin{aligned} \min_{x} \quad & c^T x \ + \eta \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{4.7}$$

  where the decision variable η in the objective is included only when the L-Shaped method is used. In the Extensive Form method, this function creates the 'base' model of the extensive form (4.6). The model is not solved standalone. In the L-Shaped method, the function is used to create the master problem in iteration 0 of the algorithm. The model is solved to return initial solution (x^0, η^0), before proceeding to iteration 1.

- The function `add_sub`(ξ^k) creates the k-th subproblem given observation ξ^k and adds the variables and constraints to an existing Gurobi model:

$$\begin{aligned} \min_{y} \quad & q^T y \\ \text{s.t.} \quad & Tx + Wy = h_k, \\ & y \geq 0. \end{aligned} \tag{4.8}$$

  This function is used in the Extensive Form method to create all K subproblems and add them to the base model, which was created using the function `create_master`. The final model corresponds to the extensive form (4.6).

- The function `create_LSsub`($\bar{x}, \xi^k$) creates k-th subproblem given observation ξ^k and first-stage solution $\bar{x}$:

$$\begin{aligned} \min_{y} \quad & q^T y \\ \text{s.t.} \quad & Wy = h_k - T\bar{x}, \\ & y \geq 0. \end{aligned} \tag{4.9}$$

  The subproblems created using `create_LSsub` are slightly different from those created with `add_sub` in that they incorporate the first-stage solution $\bar{x}$ when creating the subproblems. The function `create_LSsub` is used in every iteration of the L-Shaped method to check for first-stage feasibility. Additionally, if all subproblems are feasible, their optimal values $Q(\bar{x}, \xi^k)$ and solutions π_k^* for each k are computed. These values are then used to compute an upper bound and an optimality cut, respectively. Furthermore, `create_LSsub` is also used in the upper bound computation of the Extensive Form method, as this computation requires the objective values $Q(\bar{x}, \xi^k)$. However, this is not crucial, as the method can still operate without explicitly computing the upper bound.

- The function `create_feas_sub`($\bar{x}, \xi^k$) creates the k-th 'feasibility'-subproblem given observation ξ^k and first-stage solution $\bar{x}$:

$$\begin{aligned} \min_{y, v^+, v^-} \ w' = \quad & e^T v^+ + e^T v^- \\ \text{s.t.} \quad & W y_k + I v^+ - I v^- = h_k - T\bar{x}, \\ & y_k \geq 0, \ v^+ \geq 0, \ v^- \geq 0. \end{aligned} \tag{4.10}$$

  This function is used in the L-Shaped method when one of the subproblems created with `create_LSsub` is found to be infeasible. The dual multipliers from this subproblem are then used to compute a feasibility cut.

With these four functions enabling the construction of Gurobi models for linear programs (4.7)-(4.10), we can now proceed to implement our solution methods.

4.5 Algorithmic steps of the Extensive Form method

To implement the Extensive Form method, a new Python file, extensive_form.py, was created and added to the GitHub repository readSMPS-Py by jolijnvandelft. Given the tools to construct the master problem (4.7) and all K subproblems (4.8), the remaining task in the Extensive Form method is to combine these linear programs into the full extensive form (4.6) and solve it using Gurobi.

Before presenting the pseudocode for constructing the extensive form, we first introduce the function `get_obs_probs`, which generates and returns the lists *observations* and *probabilities*, that we will need in both the Extensive Form method and the L-Shaped method. This function takes the object *rand_vars*, a boolean *sampling*, and the number of iterations N as inputs. If *sampling* = `False`, the function creates the list *observations* by enumerating all possible outcomes of the random variables in *rand_vars*, and stores the corresponding probabilities in the *probabilities* list. If *sampling* = `True`, N observations are generated using Monte Carlo sampling, and the corresponding probabilities for each observation are set to $1/N$. The pseudocode for the function `get_obs_probs` is provided in algorithm 1.

Algorithm 1 `obtain_obs_probs`

Input: Object *rand_vars*, boolean *sampling*, number of iterations N.
Output: Lists *observations* and *probabilities*.
Initialize Empty lists *observations* = `[]`, *probabilities* = `[]`.
if *sampling* = `False` **then**
    for all scenarios k, $k = 1, \ldots, K$ **do**
        Generate observation ξ^k with probability p_k.
        Add values to lists *observations* and *probabilities* respectively.
    end for
else
    for all i, $i = 1, \ldots, N$ **do**
        Generate observation ξ^i with probability $1/N$ using Monte-Carlo sampling.
        Add values to lists *observations* and *probabilities* respectively.
    end for
end if
Return Lists *observations* and *probabilities*.

By using the function `get_obs_probs`, along with the previously defined functions `create_master` and `create_sub`, we can formulate the extensive form in the case of no sampling. The pseudocode for this is provided in algorithm 2.

Algorithm 2 Extensive Form method

Input: 2-SLP *model*
Output: First-stage solution x^* and objective value v^*.
Obtain random variables *rand_vars* from *model*.
Obtain lists *observations*, *probabilities* using `create_obs_probs(`*rand_vars*`, sampling = False)`.
Create the Gurobi model *extensive_form* and add the master variables and constraints to it using `create_master`.
for all scenarios k, $k = 1, \ldots, K$ **do**
    Create submodel k of type (4.8) and add its variables and constrains to *extensive_form* using `add_sub`.
end for
Optimize model *extensive_form*.
Return First-stage solution x^* and objective value v^*.

In the case where we want to incorporate sampling, the Extensive Form method is adjusted slightly.

The objective of the extensive form changes to the sample-average approximation function, which is defined as:

$$\min_{x,y} \hat{F}_N(x) := c^T x + \frac{1}{N} \sum_{i=1}^{N} q^T y_i \tag{4.11}$$

as given in (2.32). Specifically, the second-stage variables y will have different objective coefficients. Moreover, instead of returning the optimal objective value v^*, we now return the *lower bound estimate* $\hat{v}_N$ as the optimal value of the extensive form. We also compute an upper bound estimate using $\hat{F}_N(\bar{x})$ for some given near-optimal $\bar{x}$. The updated Extensive Form method is now described in algorithm 3.

Algorithm 3 Extensive form with sampling

1: **Input:** 2-SLP *model*, number of samples N, near-optimal solution $\bar{x}$.
2: **Output:** Approximate first-stage solution $\hat{x}_N$, lower bound estimate $\hat{v}_N$ and upper bound estimate $\hat{F}_N(\bar{x})$ of the true objective value v^*.
3: Obtain random variables *rand_vars* from *model*.
4: Obtain lists *observations*, *probabilities* using `create_obs_probs(`*rand_vars*`, sampling = True,` N`)`.
5: Create the Gurobi model *extensive_form* and add the master variables and constraints to it using `create_master`.
6: **for** all i, $i = 1, \ldots, N$ **do**
7: Create submodel i of type (4.8) and add its variables and constrains to *extensive_form* using `add_sub`.
8: Create subproblem i of type (4.9) given first-stage solution $\bar{x}$ using `create_LSsub`.
9: Optimize subproblem i of type (4.9) to obtain objective value $Q(\bar{x}, \xi^i)$.
10: **end for**
11: Compute upper bound estimate

$$\hat{F}_N(\bar{x}) = c^T \bar{x} + \frac{1}{N} \sum_{i=1}^{N} Q(\bar{x}, \xi^i).$$

12: Optimize model *extensive_form*.
13: **Return** Approximate first-stage solution $\hat{x}_N$, lower bound estimate $\hat{v}_N$ and upper bound estimate $\hat{F}_N(\bar{x})$ of v^*.

One remark is in order. The computation of the upper bound can be time-consuming because it involves creating and optimizing N additional subproblems, as indicated in line 8 of algorithm 3. We will address this further in the results section.

4.6 Algorithmic steps of the L-Shaped method

As explained in section 2.5, the L-Shaped method iteratively generates optimality and feasibility cuts based on the solution of the K subproblems for a given first-stage solution $\bar{x}$. These cuts are then added to the master problem, which is solved in each iteration until the optimal solution x^* is reached. The implementation relies on the function `create_master` to initialize the master model (4.7), while `create_LSsub` and `create_feas_sub` are used to create subproblems (4.9) and (4.10) respectively. Additionally, the function `get_obs_probs`, as defined in algorithm 1, is used to obtain the lists *observations* and *probabilities*. The pseudocode for the L-Shaped method is provided in algorithm 4, and its implementation can be found in the Python file L-Shaped.py in the GitHub repository readSMPS-Py by jolijnvandelft.

Algorithm 4 L-Shaped method

1: **Input:** 2-SLP *model*
2: **Output:** First-stage solution x^* and objective value v^*.
3: **Initialize.** $\nu = 1$, $l = -\infty$, $u = \infty$, $\epsilon = 10^{-6}$, *convergence* = `False`.

Step 0: Initialize.
4: Obtain random variables *rand_vars* from *model*.
5: Obtain lists *observations*, *probabilities* using `create_obs_probs(`*rand_vars*`,` *sampling* `= False)`.
6: Create the Gurobi model *master_* and add the master variables and constraints to it using `create_master`.
7: Optimize model *master_* to obtain initial first-stage solution x^0.
8: **while** *convergence* = `False` **do**
Step 1: Check feasibility of subproblems for $x^{\nu-1}$.
9: **for** all scenarios k, $k = 1, \dots, K$ **do**
10: Create subproblem k of type (4.9) given first-stage solution $x^{\nu-1}$ using `create_LSsub`.
11: Optimize subproblem k to obtain objective value $Q(x^{\nu-1}, \xi^k)$.
12: **if** Subproblem k is infeasible **then**
13: Create subproblem k of type (4.10) given solution $x^{\nu-1}$ using `create_feas_sub`.
14: Generate a feasibility cut using the dual solution π_k^* of (4.10) and add to model *master_*.
15: Break and skip to step 3.
16: **end if**
17: **end for**
Step 2: Compute upper bound and generate optimality cut.
18: Compute upper bound

$$u^\nu = c^T x^{\nu-1} + \sum_{k=1}^{K} p_k \, Q(x^{\nu-1}, \xi^k).$$

19: Set $u = \min\{u^\nu, u\}$
20: **if** u is updated **then**
21: Set $x^* = x^{\nu-1}$
22: **end if**
23: Generate an optimimality cut using the dual solutions $\{\pi_k^* \,|\, k = 1, \dots, K\}$ of (4.9) and add to model *master_*.
Step 3: Solve the master problem and check for convergence.
24: Optimize model *master_* to obtain solution (x^ν, η^ν) and objective value l^ν.
25: Set $l = \max\{l^\nu, l\}$
26: **if** $u - l \leq \epsilon |u|$ **then**
27: *convergence* = `True`.
28: **Return** Solution x^* and objective value l.
29: **end if**
30: Set $\nu = \nu + 1$
31: **end while**

Let us now make three important remarks. First, in the case of sampling, the algorithmic steps of the L-Shaped method remain unchanged. The only modification is to the inputs of the function `get_obs_probs`, which is responsible for generating the observations and probabilities. Specifically, line 5 would be updated to:

observations, *probabilities* `= create_obs_probs(`*rand_vars*`,` *sampling* `= True,` N`)`.

Second, in each iteration of the while-loop, and for each scenario $k = 1, \dots, K$, subproblem k of type 4.9 is created and solved. However, the only difference between the subproblems lies in the vector h_k and the first-stage solution $\bar{x}$. In the next section (4.7), we will demonstrate how to leverage the structure of a single subproblem to efficiently build all the others.

Finally, in section 2.5, we introduced two possible stopping criteria for the L-Shaped method. In

the current pseudocode, we have used the second stopping criterion, $u - l \leq \epsilon|u|$. However, the first stopping criterion could also be used, which is given by:

$$w^\nu \leq \eta^{\nu-1},$$

where w^ν was defined in (2.28). In this case, steps 2 and 3 of the L-Shaped method would be modified as outlined in algorithm 5.

Algorithm 5 L-Shaped method with alternative stopping criterion

1: **while** *convergence* = `False` **do**
   ⋮
   Step 2: Generate optimality cut and check for convergence.
2: Generate an optimality cut $\eta \geq \beta^0 - \beta^T x$ using the dual solutions $\{\pi_k^* \,|\, k = 1, \ldots, K\}$ of (4.9) and add to model *master_*.
3: Compute $w^\nu = \beta^0 - \beta^T x^{\nu-1}$.
4: **if** $w^\nu \leq \eta^{\nu-1}$ **then**
5: *convergence* = `True`.
6: **Return** Solution $x^{\nu-1}$ and objective value $l^{\nu-1}$.
7: **end if**
   Step 3: Solve the master problem.
8: Optimize model *master_* to obtain solution (x^ν, η^ν) and objective value l^ν.
9: Set $\nu = \nu + 1$.
10: **end while**

We will explore the impact of the chosen stopping criterion on the convergence behavior of the L-Shaped method in the results section.

4.7 High-performance computing for optimization

With our solution methods implemented, we are able to compute solutions for 2-SLPs with finite $\boldsymbol{\xi}$ and randomness in the right-hand side of the constraints. However, as the size of our input problems increases, we are likely to encounter significant computational challenges. In the L-Shaped method, the most time-consuming part of the algorithm is the creation of all K subproblems, which is done in every iteration of the algorithm. To reduce computation time, we can leverage the structure of the previously built subproblems and make small adjustments, rather than building each one from scratch. The same adjustment can be applied to the Extensive Form method, to reduce computation time of upper bound calculations. Beyond this, the Extensive Form method isn't particularly suited for optimization within the algorithm. However, when applied in combination with sampling, optimization can be performed across the different replications of the algorithm. Like all subproblems in the L-Shaped method, the extensive forms created in all replications share a similar structure, differing only in the right-hand side of the second-stage constraints. Therefore, by reusing the structure of the extensive form created in iteration 0 in subsequent iterations, we can save considerable computation time. These strategies are expected to greatly reduce the overall computational cost for both methods, which we will discuss in more detail below.

We will begin by optimizing the L-Shaped method. Before proceeding, we will revisit the k-th subproblem for observation ξ^k of $\boldsymbol{\xi}$ and solution $\bar{x}$, as defined in (4.9), for clarity.

$$\begin{aligned} \min_y \quad & q^T y \\ \text{s.t.} \quad & Wy = h_k - T\bar{x}, \\ & y \geq 0. \end{aligned}$$

Line 10 of the L-Shaped method creates the above subproblem in each iteration of the while loop and for every observation ξ^k:

Create subproblem k of type (4.9) given first-stage solution $x^{\nu-1}$ using `create_LSsub`.

However, since only the vector h_k and the first-stage solution $\bar{x}$ change, it is unnecessary to rebuild problem (4.9) from scratch in every iteration for each observation. To optimize this, we defined the function `change_LSsub`, which updates an existing LP of type (4.9) instead of recreating it from scratch. This function takes as inputs an LP (4.9), an observation ξ^k, and a first-stage solution $\bar{x}$. The pseudocode for `change_LSsub` is as follows:

Algorithm 6 `change_LSsub`

1: **Input:** LP (4.9) for arbitrary ξ and x; observation ξ^k, first-stage solution $\bar{x}$.
2: **Output:** LP (4.9) for observation ξ^k and first-stage solution $\bar{x}$.
3: **for** all (i, c) in `enumerate(`*constraints*`)` **do**
4: Reset right-hand side of c to `rhs` $= 0$.
5: Add $h_k[i]$ to `rhs`.
6: Substract $T[i]\bar{x}$ from `rhs`.
7: Reset right-hand side of c to `rhs`.
8: **end for**

In this pseudocode, $h_k[i]$ denotes the i-th element of vector h_k, which varies with the observation ξ^k. Additionally, $T[i]$ represents the i-th row of matrix T.

This function can be integrated into algorithm 4 by calling `create_LSsub` once before entering the while-loop. Subsequently, `change_LSsub` replaces `create_LSsub` in line 10. This modification will be incorporated when analyzing the results.

In the Extensive Form method with sampling, this adjustment is also applied to reduce computation time during upper bound calculations. Additionally, a function named `change_ef` is introduced to reuse the existing structure of the extensive form when performing multiple replications of the algorithm. The extensive form with sample-average approximation $\hat{F}_N(x)$ for given samples $\xi^1, \ldots, \xi^N$ of $\boldsymbol{\xi}$ is formulated as follows:

$$\begin{aligned} \min_{x,y} \ \hat{F}_N(x) = \quad & c^T x + \frac{1}{N} \sum_{i=1}^{N} q^T y_i \\ \text{s.t.} \quad & Ax = b, \\ & Tx + Wy_i = h_i, \quad i = 1, ..., N, \\ & x \geq 0, \ y_i \geq 0, \quad i = 1, ..., N. \end{aligned} \tag{4.12}$$

The idea behind `change_ef` is similar to that of `change_LSsub`. Given the extensive form (4.12), only the right-hand side vectors $h_1, \ldots, h_N$ are affected by the samples $\xi^1, \ldots, \xi^N$. The corresponding pseudocode is as follows:

Algorithm 7 `change_ef`

1: **Input:** Extensive form (4.12) for arbitrary N samples of $\boldsymbol{\xi}$; samples $\{\xi^1, \ldots, \xi^N\}$.
2: **Output:** Extensive form (4.12) for samples $\{\xi^1, \ldots, \xi^N\}$.
3: `no_sub_cons = len(`*constraints*`)`$/N$.
4: **for** all (j, c) in `enumerate(`*constraints*`)` **do**
5: Reset right-hand side of c to `rhs` $= 0$.
6: `index` $= j$ `mod no_sub_cons`.
7: Add h_i[`index`] to `rhs`.
8: Reset right-hand side of c to `rhs`.
9: **end for**

Note that, since we are working with the extensive form, the total number of constraints is N times the number of sub-constraints. Given that each vector h_i has length `no_sub_cons`, we use `index` $= j$ `mod no_sub_cons` to ensure the index stays within bounds. The resulting extensive form incorporates

each vector h_i, which varies with the observation ξ^i, as the right-hand side. The function `change_ef` can be invoked during any replication of algorithm 3 after the first one. This modification will be incorporated when analyzing the results.

5 Results

In this section, we present the results obtained from applying the Extensive Form method and the L-Shaped method to solve the input problems. The analysis is structured into two main parts: the performance of the Extensive Form method and the performance of the L-Shaped method. For each method, we evaluate the solutions obtained, computation time, and solution quality. Additionally, we analyze the runtime characteristics and, in the case of the L-Shaped method, its convergence behavior. Finally, we compare the two solution methods based on their computation time, solution quality, and potential.

5.1 Extensive Form method: solutions and computation time

The input problems listed in the first column of table 3 were solved exactly using the Extensive Form method. Table 3 presents their first-stage solutions, objective values, and computation times.

Table 3: First-stage solutions, objective values and computation times obtained using the Extensive Form method for listed input problems.

Name	Solution	Objective value	Time (s)
lands	$x^* = (2.67, 4.00, 3.33, 2.00)$	381.85	0.01
lands2	$x^* = (2.0, 3.96, 0.96, 5.08)$	227.60	0.02
baa99	$x^* = (159.49, 111.38)$	-238.78	0.10
pgp2	$x^* = (1.5, 5.5, 5.0, 5.5)$	447.32	0.15
Oemof-B3 ($T = 3$)	$x^* = (214.58, 4.86, \ldots, 0.00)$	660 117 808.21	60.74

For the remaining input problems, obtaining an exact solution within a reasonable time was not feasible. As a result, sampling was performed using a selected number of samples N, initially with only one replication. Additionally, sampling was applied to the input problems that had already been solved, allowing for comparison between the estimated lower and upper bounds and the true objective values, as presented in table 3. For each input problem, an appropriate sample size was selected based on the number of possible outcomes of the vector $\boldsymbol{\xi}$. Specifically, for problems with a small number of outcomes, a sample size of $N = 20$ was used, whereas for problems with a large number of outcomes, a sample size of $N = 100$ or even $N = 1000$ was chosen. The results are presented in table 4.

Table 4: First-stage solutions, lower bound estimates, upper bound estimates and computation times obtained using the Extensive Form method with number of samples N and replications $M = 1$ for listed input problems.

Name	N	M	First-stage solution	LB Estimate	UB Estimate	Time (s)
lands	20	1	$\hat{x}_N = (1.17, 5.00, 3.83, 2.00)$	377.33	377.35	0.02
lands2	20	1	$\hat{x}_N = (0.96, 5.00, 0.96, 5.08)$	218.87	219.02	0.02
baa99	100	1	$\hat{x}_N = (151.82, 106.28)$	-232.08	-214.23	0.02
pgp2	100	1	$\hat{x}_N = (1.5, 5.5, 5.0, 5.5)$	437.68	438.65	0.05
Oemof-B3 ($T = 3$)	100	1	$\hat{x}_N = (225.27, 4.86, \ldots, 0.00)$	658 486 529.50	659 751 411.80	8.83
lands3	1000	1	$\hat{x}_N = (0.80, 3.48, 1.92, 5.80)$	226.42	226.45	0.03
20	1000	1	$\hat{x}_N = (346.5, 10.0, \ldots, 36.0)$	253 938.76	253 943.63	156.29
ssn	1000	1	$\hat{x}_N = (0.00, 4.91, \ldots, 0.00)$	9.51	9.99	340.36
storm	1000	1	$\hat{x}_N = (0.00, 0.00, \ldots, 0.00)$	15 489 620.52	15 489 636.08	776.2

Lastly, sampling was performed using a smaller number of samples N for the largest input problems, and a fixed number of $M = 100$ iterations. Results are presented in table 5.

Table 5: First-stage solutions, lower bound estimates, upper bound estimates and computation times obtained using the Extensive Form method with number of samples N and replications $M = 100$ for listed input problems.

Name	N	M	First-stage solution	LB Estimate	UB Estimate	Time (s)
lands	20	100	$\tilde{x} = (2.13, 4.29, 3.52, 2.06)$	382.07	382.14	0.18
lands2	20	100	$\tilde{x} = (1.28, 4.08, 1.38, 5.29)$	226.45	227.67	0.18
baa99	100	100	$\tilde{x} = (156.23, 113.25)$	-244.47	-228.59	0.51
pgp2	100	100	$\tilde{x} = (1.42, 5.17, 5.08, 5.46)$	444.64	450.96	0.84
Oemof-B3 ($T = 3$)	100	100	$\tilde{x} = (260.35, 24.06, \ldots, 0.00)$	659 216 149.56	742 380 590.71	94.78
lands3	100	100	$\tilde{x} = (0.83, 3.35, 1.86, 5.95)$	224.01	224.14	1.02
20	100	100	$\tilde{x} = (0.88, 17.20, \ldots, 2.46)$	254 505.56	254 597.53	74.61
ssn	100	100	$\tilde{x} = (0.00, 15.32, \ldots, 0.06)$	6.81	11.33	372.33
storm	100	100	$\tilde{x} = (0.00, 0.00, \ldots, 0.00)$	15 496 474.70	15 496 913.20	512.66

5.2 Extensive Form method: solution quality analysis

This section analyzes the quality of solutions and objective estimates obtained through the Extensive Form method with sampling. We compare the average solutions and objective estimates obtained from this method, presented in tables 4 and 5, to the true solutions and objective values of the input problems. When no true solution and objective value could be computed, we compare the results to known values from external sources. Table 6 presents the solution and objective (estimates) of input problems lands3, 20, ssn and storm, obtained from [Sen and Liu, 2016].

Table 6: Solution and objective (estimates) of input problems lands3, 20, ssn and storm, obtained from [Sen and Liu, 2016].

Name	Solution	Objective value
lands3	$x^* = (0.84, 3.40, 1.88, 5.88)$	225.63
20		$\approx$ 254 515.48
ssn		$\approx$ 9.93
storm		$\approx$ 15 481 852.29

Comparing our estimates from table 4 and 5 to the values in table 3 and 6, we observe that our method generally performs very well. For instance, in the case of input problem lands3, with $(N, M) = (1000, 1)$, we obtained the approximate solution $\hat{x}_N = (0.80, 3.48, 1.92, 5.80)$ and a lower bound estimate of 226.42. These values are close to the true solution $x^* = (0.84, 3.40, 1.88, 5.88)$ and the true objective value of 225.63. Furthermore, we find that repeating the algorithm yields even more accurate approximations. That is, in the case of lands3 with $(N, M) = (100, 100)$, the approximate solution improved to $\hat{x}_N = (0.83, 3.35, 1.86, 5.95)$, with a lower bound estimate of 224.01.

In each replication, we compute a lower and upper bound estimate. As described in section 2.6.1, these values are averaged to compute the final estimates. These estimates are presented in table 5. For several input problems, including lands2, baa99, pgp2, oemof-B3 ($T = 3$), 20, and ssn, the true objective value falls within the lower and upper bound estimate. However, for lands, lands3, and storm, this was not the case. Interestingly, despite the high accuracy of our estimates for lands and lands3, the true objective was not captured within the interval. This likely occurs because our method solves these problems with such precision that the lower and upper bounds are extremely close together, as is also the case for the average estimates in table 2.6.1. For Oemof-B3 ($T = 3$), the true objective was captured in 57 out of 100 replications, while for ssn, this occurred most frequently, with 72 out of 100 replications capturing the true objective value.

Lastly, as additionally described in section 2.6.1, confidence intervals can be computed for the lower and upper bound estimates in table 5. The confidence intervals indicate that there is a 95% probability that the lower bound estimate lies within its corresponding lower bound confidence interval and that the upper bound estimate lies within its corresponding upper bound confidence interval. Therefore, we do not expect the true objective value to fall within these intervals. Table 7 presents the lower

bound and upper bound estimates from table 5 along with their respective 95% confidence intervals.

Table 7: Objective estimates from table 5 along with their respective 95% confidence intervals.

Name	LB Estimate	LB Conf. Interval	UB Estimate	UB Conf. Interval
lands	382.07	(340.05, 424.10)	382.14	(340.49, 423.78)
lands2	226.45	(165.17, 287.73)	227.67	(171.39, 283.96)
baa99	-244.47	(−445.67, −43.28)	-228.59	(−491.48, 34.29)
pgp2	444.64	(425.10, 464.18)	450.96	(392.76, 509.15)
Oemof-B3 ($T = 3$)	659 216 149.56	(−77 067 495 543.98, 78 385 927 843.10)	742 380 590.71	(−606 317 594 669 132.00, 606 319 079 430 314.00)
lands3	224.01	(218.44, 229.59)	224.14	(218.73, 229.55)
20	254 505.56	(12 179.81, 496 831.31)	254 597.53	(32 924.73, 476 270.34)
ssn	6.81	(6.26, 7.36)	11.33	(10.70, 11.96)
storm	15496474.70	(−194 697 269.07, 225 690 218.47)	15 496 913.20	(−194 351 705.50, 225 345 531.90)

From the values in table 7, we conclude that the intervals are too wide to provide meaningful insights. In particular, for ill-conditioned problems like Oemof-B3 ($T = 3$), the confidence intervals are completely nonsense.

5.3 Extensive Form method: runtime analysis

For the results obtained through the Extensive Form method with sampling, as presented in table 5, we analyzed the computation time for the first five replications. In replication 0, the extensive form is constructed and solved to compute a near-optimal solution $\bar{x}$. Since this involves building the model from scratch, we expect replication 0 to take the longest. In replications 1-5, this extensive form is reconstructed using the function `change_ef`, where the right-hand side of the stochastic constraints is replaced with samples $\xi^{1,j}, \ldots, \xi^{N,j}$, with $j = 1, \ldots, M$ denoting the replication number. As a result, we expect the building time of the extensive form to significantly decrease compared to the building time in replication 0. Additionally, we expect the solving time to decrease, as Gurobi retains information from the previously solved model and its solution. Because the replications 1-5 are assumed to be independent, we expect similar computation times across those replications. The results for the input problems pgp2, 20, ssn, and storm are shown in figure 5.1.

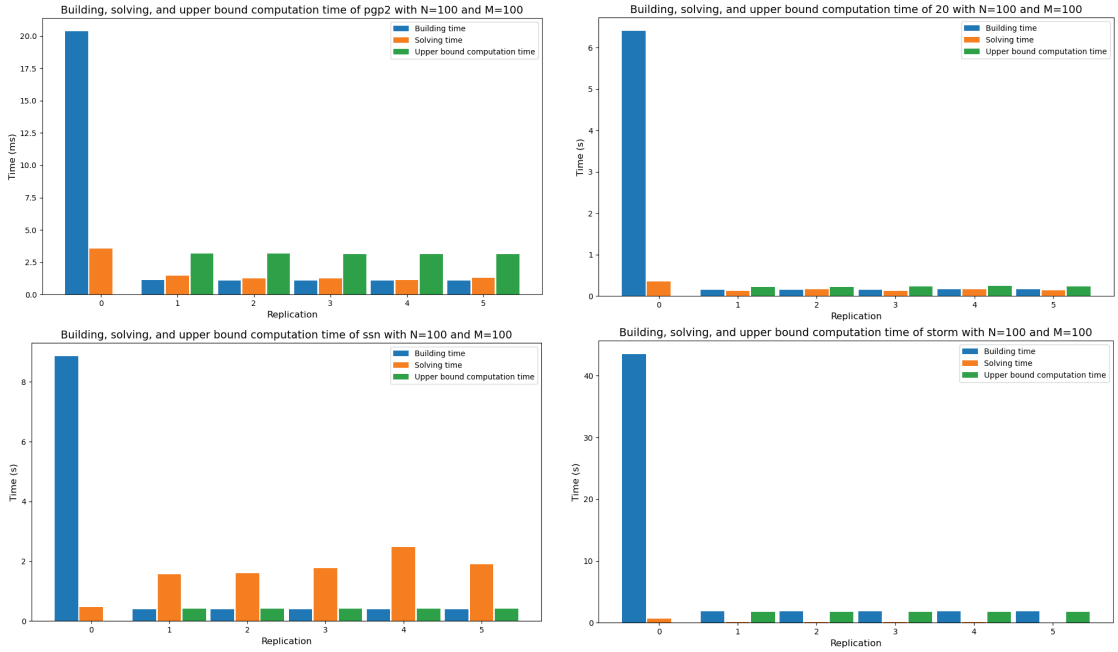

Figure 5.1: Building, solving, and upper bound computation times of replications 0-5 using the Extensive Form method with number of samples $N = 100$ for different input problems.

Key observations from figure 5.3 are as follows:

- For all four datasets, the majority of the computation time is spent building the extensive form in replication 0. Since the `change_ef` function is used in replications 1–5, the building time is significantly reduced in those replications.
- Upper bound computation is time-consuming, particularly in the case of pgp2.
- As expected, the solving times in replications 1–5 decrease compared to replication 0 for the input problems pgp2, 20, and storm. However, for the ssn dataset, the solving time in replications 1–5 is higher than in replication 0. Additionally, we observe an increasing trend across these replications, which is unexpected given that they are assumed to be independent. This behavior will be investigated further.

To verify whether the observed increase in solution time across replications 1–5 is valid, we conducted an additional experiment using the ssn input problem. Specifically, we solved it with $N = 1000$ samples and analyzed the runtime of the first 10 replications. The results are shown in figure 5.2.

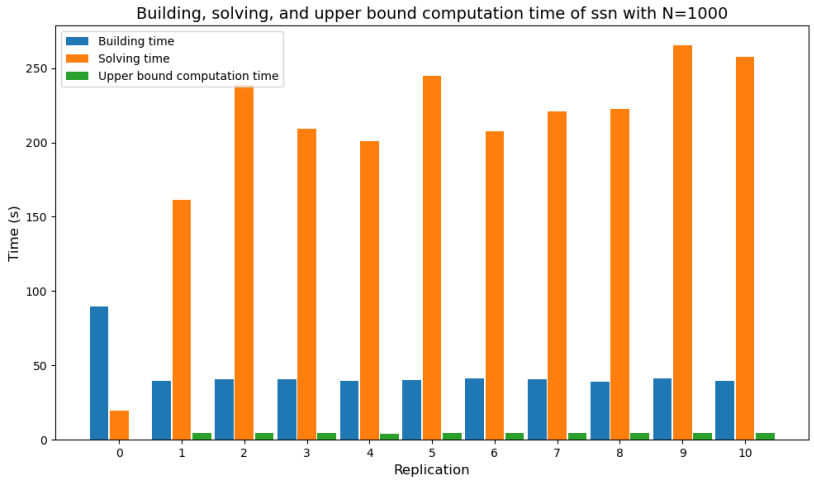

Figure 5.2: Building, solving, and upper bound computation times of replications 0-10 using the Extensive Form method with number of samples $N = 1000$ for ssn.

Once again, we observe that the solving times for replications 1–10 are higher than that of replication 0. Additionally, there appears to be a gradual increase in solving time across replications, a pattern we cannot yet explain. To address this issue, we may consider fully rebuilding the extensive form in each replication, thereby making the replications 1-M independent of replication 0.

5.4 L-Shaped method: solutions and computation time

This section will present the results of solving our input problems with the L-Shaped method. For the instances lands, lands, baa99, and pgp2, an exact solution and objective value were obtained, matching those found with the Extensive Form method, as expected. These results, along with the corresponding computation times, are presented in table 5.

Table 8: First-stage solutions, objective values and computation times obtained using the L-Shaped method for listed input problems.

Name	Solution	Objective value	Time (s)
lands	$x^* = (2.67, 4.00, 3.33, 2.00)$	381.85	0.02
lands2	$x^* = (2.0, 3.96, 0.96, 5.08)$	227.60	0.05
baa99	$x^* = (159.49, 111.38)$	-238.78	0.37
pgp2	$x^* = (1.5, 5.5, 5.0, 5.5)$	447.32	0.57

The input problem Oemof-B3 ($T = 3$) is not included in this table, as numerical issues prevented it from solving within a reasonable time. This issue will be explored further in the runtime analysis and the examination of the L-Shaped method's convergence behavior.

For the remaining problems, an exact solution could not be obtained, prompting us to use sampling. To evaluate solution quality, sampling was also applied to the problems that had already been solved exactly. The chosen sample sizes were the same as those used in the Extensive Form method. However, for the largest input problems–20, ssn, and storm–we used a sample size of $N = 100$ instead of $N = 1000$ to keep computation times manageable. Table 9 presents the approximations, computation times, and sample size N for the listed input problems.

Table 9: First-stage solutions, objective values and computation times obtained using the L-Shaped method with number of samples N for listed input problems.

Name	N	First-stage solution	Objective value	Time (s)
lands	20	$\hat{x}_N = (1.17, 5.00, 3.83, 2.00)$	376.9	0.03
lands2	20	$\hat{x}_N = (2.96, 3.00, 0.96, 5.08)$	240.07	0.05
baa99	100	$\hat{x}_N = (161.33, 101.25)$	-227.02	0.07
pgp2	100	$\hat{x}_N = (1.5, 5.5, 4.0, 5.5)$	436.55	0.11
lands3	1000	$\hat{x}_N = (0.81, 3.34, 1.91, 5.94)$	225.55	1.03
20	100	$\hat{x}_N = (210.95, 25.00, \ldots, 27.19)$	254 623.34	620.70
ssn	100	$\hat{x}_N = (0.00, 0.00, \ldots, 0.00)$	7.45	3730.04
storm	100	$\hat{x}_N = (0.00, 0.00, \ldots, 0.00)$	15 507 768.78	193.08

Once again, Oemof-B3 ($T = 3$) is not included.

5.5 L-Shaped method: solution quality analysis

As before, we evaluate the quality of our approximate solutions and objective values against the true solutions and objective values. For input problems that could not be solved exactly, we compare our approximations to those presented in table 6. In most cases, our method provides accurate estimates of the true objective value. For example, for the input problem 20, we obtain an approximate objective value of 254 623.34, while the true value, according to [Sen and Liu, 2016], is around 254 515.48. However, when comparing our approximate first-stage solution $\hat{x}_N = (210.95, 25.00, \ldots, 27.19)$ to earlier solutions–$\hat{x}_N = (346.5, 10.0, \ldots, 36.0)$ (from table 4) and $\tilde{x} = (0.88, 17.20, \ldots, 2.46)$ (from table 5)–we observe that even though we only saved three out of 63 indices, the solutions differ significantly. This discrepancy is most likely to be blamed on the ill-conditioning of the problem, which causes small changes in the linear program to lead to large differences in the solutions, but is still concerning. That is, the first-stage decisions represent capacity investments that must be robust to small changes.

5.6 L-Shaped method: runtime analysis

We also analyzed the computation time of the L-Shaped method, but with a different approach compared to the extensive form. For each solution presented in table 8 and table 9, we examined the time spent in each step of the L-Shaped method–steps 0, 1, 2, and 3, as outlined in algorithm 4. The computation time per step, corresponding to the solutions presented in table 8 and table 9, is summarized in table 10. This time, the table includes Oemof-B3 ($T = 3$); however, the algorithm was terminated after reaching the maximum number of $\nu = 1000$ iterations, which occurred before convergence was achieved.

Table 10: Computation time per step in the L-Shaped method for listed input problems.

Name	N	main	Step 0	Step 1	Step 2	Step 3
lands	-	0.0215	0.0015	0.0011	0.0004	0.0003
lands2	-	0.0525	0.0032	0.0280	0.0005	0.0006
baa99	-	0.3666	0.0088	0.2634	0.0008	0.0008
pgp2	-	0.5652	0.0135	0.4261	0.0011	0.0011
lands3	1000	1.0274	0.0362	0.7891	0.0005	0.0009
20	100	620.70	0.29	517.76	0.67	47.74
ssn	100	3730.04	0.52	2551.11	14.58	189.67
storm	100	193.08	2.35	157.71	0.36	0.17
Oemof-B3 ($T = 3$)	20	194.01	0.19	171.74	1.02	1.22

To visualize the data presented in table 10, we created a bar chart that shows the relative computation time for each step of the L-Shaped method across different input problems. Each bar shows the relative time spent in steps 0, 1, 2, and 3 of the L-Shaped method, along with the 'remaining' time used by Python. The number displayed at the top of each bar indicates the total computation time of each input problem. The barchart is shown in figure 5.3.

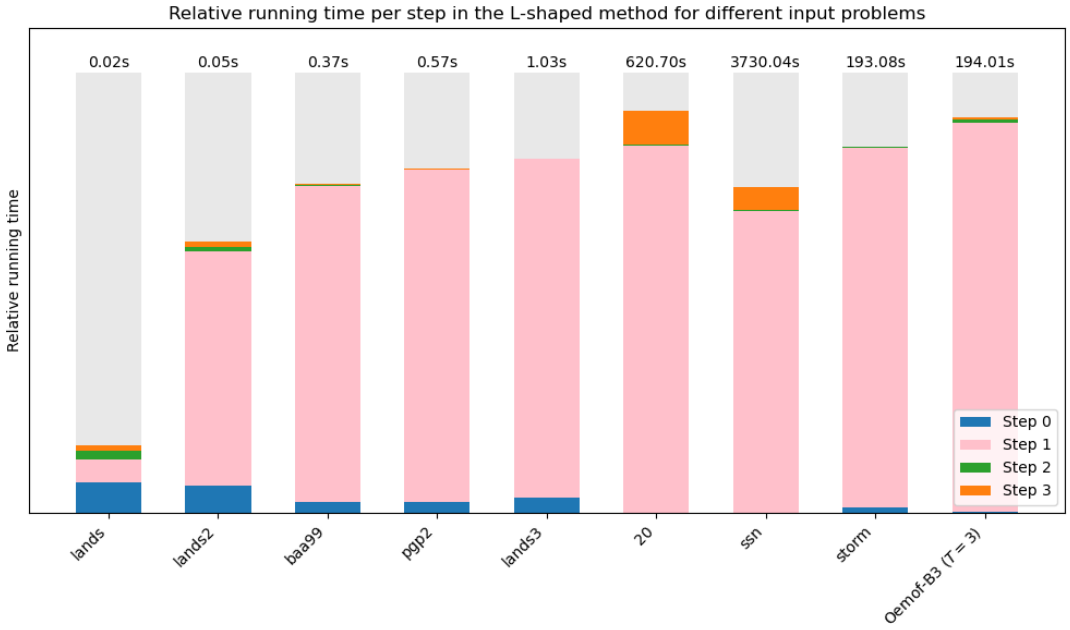

Figure 5.3: Relative computation time per step in the L-Shaped method. Each bar shows the relative time spent in steps 0, 1, 2, and 3 of the L-Shaped method, along with the 'remaining' time used by Python. The number displayed at the top of each bar indicates the total computation time of each input problem.

We can now observe a few key points from the graph:

- First, as expected, the majority of the time is spent in step 1 of the L-Shaped method, as this is where the subproblems are created and solved during each iteration of the algorithm. Had we not created the function `change_sub`, which leverages the structure of an initialized subproblem, this time would be even higher.
- Step 3 of the algorithm, in which the master problem is solved, doesn't take much time. However, for more challenging problems, such as 20 and ssn, which require more iterations, the size of the master problem increases. As a result, step 3 requires relatively more computation time for these cases.
- The light grey portion of the bar represents the remaining time used by Python. This time is relatively large, though its exact cause cannot be explained. Notably, it is particularly large for

the input problem lands. However, the computation time for this input problem is very short, which may result in inaccuracies in the measurements.

5.7 L-Shaped method: convergence behavior

Lastly, we analyzed the convergence behavior of the L-Shaped method. As outlined in section 4.6, we use two different stopping criteria. First, we will present the convergence graph according to our initial stopping criterion, as described in algorithm 4. Then, we will examine the convergence behavior using an alternative stopping criterion, described in algorithm 5. In both cases, particular attention will be given to the convergence graphs of Oemof-B3 ($T = 3$).

For clarity, let us restate the first stopping criterion. The lower bound l^ν is the solution of the master problem in iteration ν (as given in equation (2.27)), while the upper bound u^ν consists of the first-stage cost, plus the probability-weighted sum of optimal responses in the second stage, given solution $x^{\nu-1}$ (as given in equation (2.30)). Since the master problem is a minimization problem, and it becomes more restrictive with the addition of cuts in each iteration, the lower bound l^ν will either increase or remain constant throughout the process. This continues until all second-stage information is included and the true minimum of the optimization problem is found.

Given the upper bound u^ν and the lower bound l^ν, the overall upper and lower bounds are updated as follows: $u = \min\{u^\nu, u\}$ (ensuring that u remains non-increasing) and $l = \max\{l^\nu, l\}$, respectively. While updating l is technically unnecessary since l^ν is naturally non-decreasing, it is included for consistency.

The algorithm converges when $u - l \leq \epsilon|u|$, with a fixed tolerance of $\epsilon = 10^{-6}$. The choice of ϵ is quite small, ensuring accurate solutions. However, as we will see in the convergence graphs later, a tolerance of $\epsilon = 10^{-4}$ would have been sufficient for approximating the objective values. Ultimately, the size of ϵ did not determine whether an input problem was solved or not. Therefore, we chose a smaller ϵ to gain more detailed insights into the solutions.

Figure 5.4 shows the lower and upper bound values at each iteration ν for input problems lands3, 20, ssn and storm. The red point at the right of each graph indicates where convergence was reached, with a fixed $\epsilon = 10^{-6}$. The coordinates next to the red point show the iteration number ν at which convergence occurred, along with the lower bound l, which serves as the objective value estimate. Additionally, the leftmost dot on each graph represents where convergence would have occurred if $\epsilon = 10^{-4}$ had been chosen, again with the corresponding coordinates (ν, l).

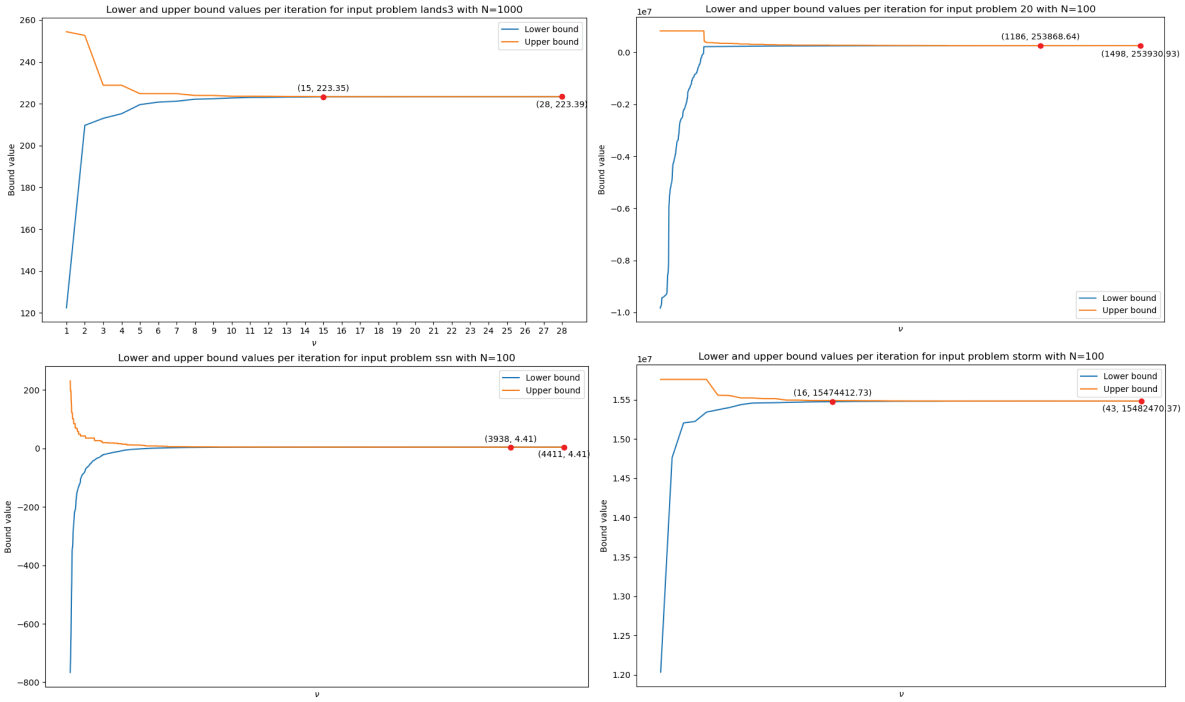

Figure 5.4: Lower and upper bound values at each iteration ν for input problems lands3, 20, ssn and storm. The red point on the right side of each graph indicates where convergence was reached, with a fixed $\epsilon = 10^{-6}$. The leftmost point on each graph represents where convergence would have occurred if $\epsilon = 10^{-4}$ had been chosen.

In figure 5.4, we observe that the non-decreasing lower bound and non-increasing upper bound gradually converge to each other for all input problems, indicating that the L-Shaped method with this stopping criterion demonstrates steady convergence behavior. However, it is clear that $\epsilon = 10^{-4}$ would have sufficed as a choice for ϵ, or perhaps even larger, as the objective value approximations obtained with $\epsilon = 10^{-4}$ are only slightly smaller or equal to those obtained with $\epsilon = 10^{-6}$. Choosing a larger ϵ would have significantly reduced computational time.

Figure 5.5 shows the convergence graph of Oemof-B3 ($T = 3$), executed within a 10 000 second time limit.

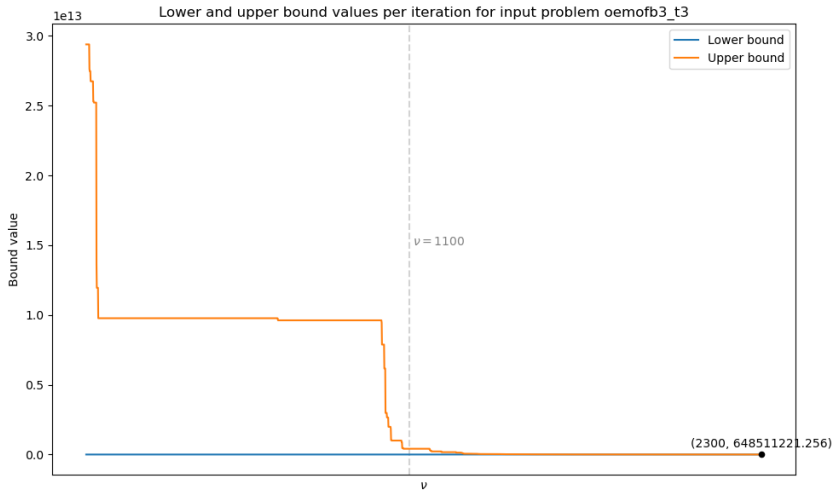

Figure 5.5: Lower and upper bound values at each iteration ν for the input problem Oemof-B3 ($T = 3$), obtained within a time limit of 10 000 seconds. The black point on the right side of the graph indicates the last iteration number and its corresponding lower bound estimate.

The graph differs significantly from those in figure 5.4, due to the large scale on the y-axis. Although the lower bound curve appears straight, it actually ranges from 0 to 648 511 221.26. To gain deeper insights into the convergence behavior, we replot the graph with a logarithmic scale on the y-axis and focus on iterations starting from $\nu = 1100$. The updated graph is presented in figure 5.6.

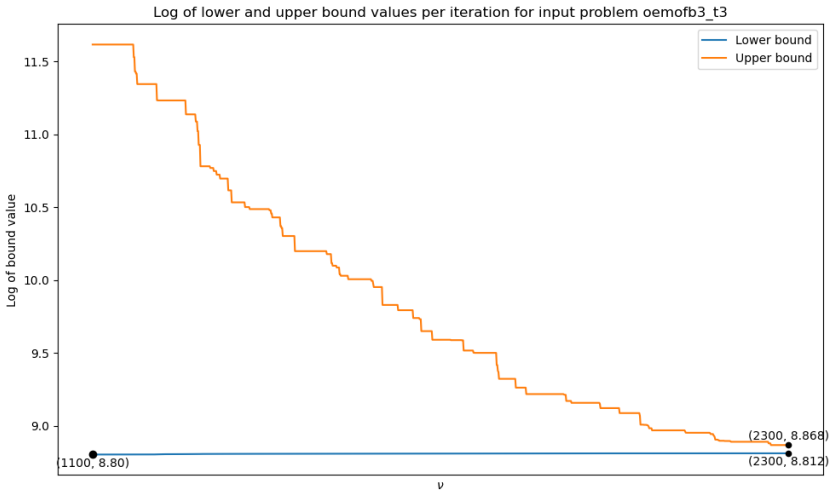

Figure 5.6: Lower and upper bound values at each iteration ν for the input problem Oemof-B3 $T = 3$, obtained within a time limit of 10,000 seconds. The y-axis is displayed on a logarithmic scale, and the graph starts from iteration $\nu = 1100$ to better illustrate the convergence behavior. The black point on the left represents $(\nu, \log(l^\nu))$ at $\nu = 1100$, while the two black points on the right correspond to $(\nu, \log(l^\nu))$ and $(\nu, \log(u^\nu))$ at the final iteration $\nu = 2300$, respectively.

From the graph in figure 5.6, we can observe the following. The figure demonstrates steady convergence behavior and even suggests that convergence was nearly reached. However, $\log(8.812)$ corresponds to a lower bound value $l^{2300} = 648\,511\,221.26$, while $\log(8.868)$ corresponds to an upper bound value of $u^{2300} = 738\,604\,720.26$, leaving a gap of $90\,093\,499.00$ between the two. From this, we can conclude that convergence would have been achieved within the time limit if $\epsilon > 0.13$, though such a threshold would be too large. Despite not fully converging, the lower bound obtained in the final iteration, $l^{2300} = 648\,511\,221.26$, is relatively close to the true objective value of $660\,117\,808.21$. This suggests that the method remains effective, though numerical issues arising from the model formulation may have impacted its performance.

Let us now present the alternative stopping criterion, as described in algorithm 5. In this version of the L-Shaped method, w^ν is defined as $w^\nu = \beta^0 - \beta^T x^{\nu-1}$, where β^0 and β are defined to compute optimality cuts. The algorithm converges if $w^\nu \leq \eta^{\nu-1}$, where $\eta^{\nu-1}$ corresponds to the optimal objective of the master problem in iteration $\nu - 1$.

Figure 5.7 shows the values of $\eta^{\nu-1}$ and w^ν at each iteration ν, for input problems lands3, 20, ssn and storm. The red point at the end of the graph indicates where convergence was reached, along with the coordinates $(\nu, \eta^{\nu-1})$. Note that $\eta^{\nu-1}$ approximates the second term in the objective function, and not the optimal objective value. To obtain the objective value estimate $\hat{v}_N$, we compute $\hat{v}_N = c^T x^{\nu-1} + \eta^{\nu-1}$, given the using first-stage solution $x^{\nu-1}$.

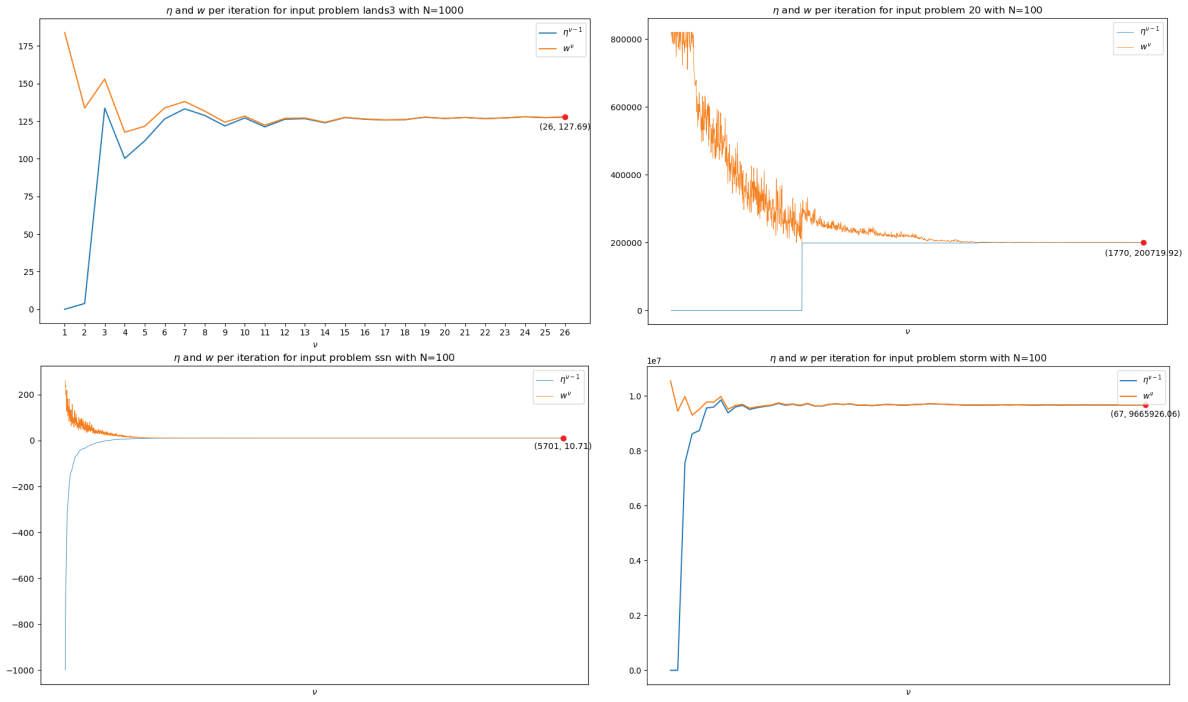

Figure 5.7: $\eta^{\nu-1}$ and w^{ν} at each iteration ν for input problems lands3, 20, ssn and storm. The red point at the end of the graph indicates where convergence was reached, along with the coordinates $(\nu, \eta^{\nu-1})$.

In figure 5.7, we see that the convergence behavior is much less steady compared to the convergence behavior of the L-Shaped method with the first stopping criterion. Additionally, in three out of four cases analyzed, convergence is reached at a later iteration than with the previous stopping criterion. However, we also observe that long before the algorithm reaches convergence–well before $w^{\nu} \leq \eta^{\nu-1}$–the value of η is already very close to the final objective estimate $\eta^{\nu-1}$ obtained in the last iteration. This suggests that if this algorithm were to be adopted, it would be beneficial to introduce a small ϵ and modify the stopping criterion to $w^{\nu} - \eta^{\nu-1} \leq \epsilon |w^{\nu}|$, ensuring an earlier yet reliable termination.

Lastly, let us have a look at the convergence graph for Oemof-B3 ($T = 3$) obtained using the second stopping criterion. The graph is shown in figure 5.8.

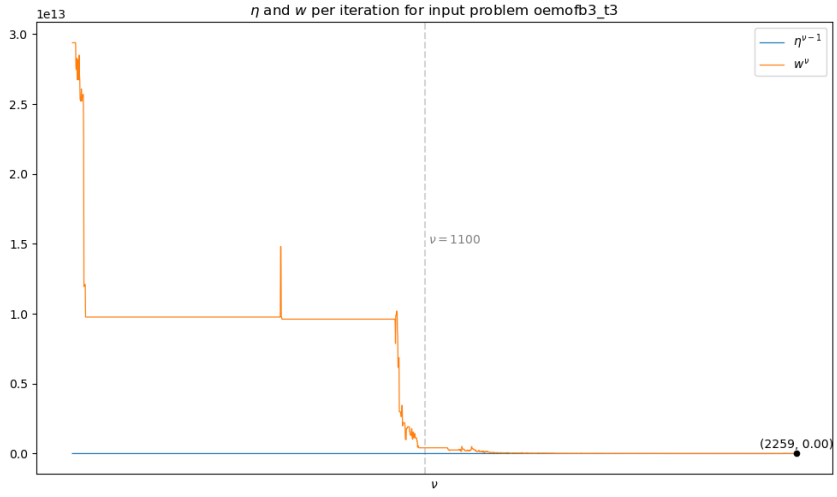

Figure 5.8: $\eta^{\nu-1}$ and w^{ν} at each iteration ν for input problem Oemof-B3 ($T = 3$), obtained within a time limit of 10 000 seconds. The black point on the right side of the graph indicates the last iteration number and its corresponding value $\eta^{\nu-1}$.

Strangely, the graph closely resembles the one in figure 5.5, despite the fact that w is computed differently from the upper bound u, and η is computed differently from the lower bound l. Another unusual observation is that η remains zero throughout all iterations. To further examine the convergence behavior beyond $\nu = 1100$, we have replotted the graph starting from this iteration, but plotting $\log(w)$ instead of w. The revised graph is presented in figure 5.9.

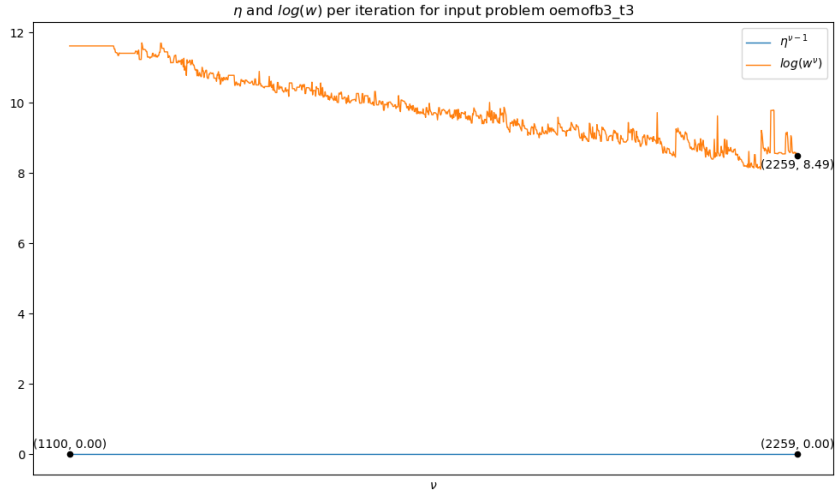

Figure 5.9: $\eta^{\nu-1}$ and $\log(w^\nu)$ at each iteration ν for input problem Oemof-B3 ($T = 3$), obtained within a time limit of 10 000 seconds. The graph starts from $\nu = 1100$ to better illustrate the convergence behavior. The black point on the left represents $(\nu, \eta^{\nu-1})$ at $\nu = 1100$, while the two black points on the right correspond to $(\nu, \eta^{\nu-1})$ and $(\nu, \log(w^\nu))$ at the final iteration $\nu = 2259$, respectively.

The significant gap between $\log(w)$ and $\eta^{\nu-1}$ suggests that this algorithm is unlikely to ever satisfy the convergence criterion $w^\nu \leq \eta^{\nu-1}$. However, on a positive note, the objective value estimate obtained after $\nu = 2259$ iterations is 648 484 342.34, which is a good estimate of the true objective value 660 117 808.21. Interestingly, since η remains zero in the final iteration, it is implied that all costs are attributed to the first-stage variables. While this aligns with the expectation that investment costs are significantly higher than marginal costs, we do not fully understand this outcome. Therefore, we will revisit the issue in the conclusion and discussion section.

5.8 Comparison of solution methods

Let us first compare computation time of both methods. Table 11 summarizes the previously presented results, with the new values representing the computation times for 20, ssn, and storm obtained using the Extensive Form method with number of samples $N = 100$. The Extensive Form method is not repeated in the case of sampling.

Table 11: Computation times obtained using different solution methods with optional number of samples N.

Name	N	Time EF method (s)	Time L-Shaped method (s)
lands	-	0.01	0.02
lands2	-	0.02	0.05
baa99	-	0.10	0.37
pgp2	-	0.15	0.57
Oemof-B3 ($T = 3$)	-	60.74	-
lands3	1000	0.03	1.03
20	100	7.24	620.70
ssn	100	11.08	3730.04
storm	100	47.40	193.08

Clearly, in terms of computation time, the Extensive Form method is the better choice.

In terms of solution quality, both methods perform similarly on the test problems. However, for the model Oemof-B3 ($T = 3$), the L-Shaped method fails to return a solution within the 10 000-second time limit. Despite this, as we have observed, the objective value estimates returned after 10 000 seconds are comparable to those obtained with the Extensive Form method. The numerical difficulties within the model seem to prevent the L-Shaped method from converging. This raises the question: to what extent is the issue with the model itself, and to what extent with the method? We will explore this further in the conclusion and discussion.

6 Conclusion and discussion

In this section, we reflect on our findings, examine the key challenges encountered, and draw a conclusion from our research. Finally, we provide suggestions for future work.

We begin by discussing the results. The most notable outcome is that we were unable to successfully compute any solutions for the model Oemof-B3 ($T = 8760$), as significant challenges had already emerged with Oemof-B3 ($T = 3$). This suggests that the difficulties encountered stem from both the complexity of the model and the limitations of the methods used. It is likely that the challenges are a result of a combination of both factors.

To critically assess our approach, we now turn to several limitations of the method and our result analysis. First, while we conclude that both methods perform well in estimating first-stage solutions and objective values, our assessment approach differs between small and large problems. For small input problems, such as lands, baa99, and pgp2, we validate our findings by directly comparing both the approximate solution and the objective value to the true solution and true objective. However, for larger problems like 20, ssn, and Oemof-B3 ($T = 3$), our evaluation if primarily based on comparing objective function values, as only three indices of the first-stage solution vector are known.

As demonstrated in the case of input problem 20, even though we computed three objective value estimates–$\hat{v}_N = 353\,938.76$ (table 4), $\tilde{v} = 254\,505.56$ (table 5), and $\hat{v}_N = 254\,623.34$ (table 9)–that were all relatively close to the objective value estimate $v^* = 254\,515.48$ (table 6), the corresponding first-stage solutions differed significantly. The three first-stage solutions obtained were $\hat{x}_N = (346.5, 10.0, \ldots, 36.0)$, $\tilde{x} = (0.88, 17.20, \ldots, 2.46)$, and $\hat{x}_N = (210.95, 25.00, \ldots, 27.19)$. Such drastic differences are concerning, as the first-stage solution represents critical decision variables. Even if problem 20 is ill-conditioned, as seems likely, discrepancies of this magnitude should not occur. Therefore, in evaluating method performance, a more rigorous approach would have been to measure not only the accuracy of objective values but also the deviation in first-stage solution vectors compared to either the true solution or a reliable estimate.

A second point of criticism we would like to highlight is our inability to explain the runtime behavior of the input problem ssn when solved with the Extensive Form method, as presented in figure 5.2. Specifically, cannot explain the increase in runtime in replications 1–10 compared to replication 0, nor can we account for the increasing trend in runtime across replications 1–10. Although this is not a major concern, we should be able to fully understand the behavior of the algorithm.

Lastly, the L-shaped method can result in very large computation times. For the input problem ssn, with only $N = 100$ samples, it took 3730.04 seconds to compute our solution, as shown in table 9, whereas the Extensive Form method with the same number of samples took only 11.08 seconds (table 11). Moreover, for input problems like ssn, which involves a total number of $\approx 10^{70}$ scenarios, the sample size of $N = 100$ is actually too small to provide accurate results.

Now, we turn to discussing the Oemof-B3 model and the challenges we encountered when attempting to solve it. One key area for self-reflection is that we should have explored uncertainty modeling in energy systems more thoroughly before deciding to use the stochastic linear programming framework to include uncertainty in this model. On the surface, given the linear nature of Oemof-B3 and its role in modeling investment and operational planning in the Berlin-Brandenburg area, it seemed well-suited to this framework. An additional reason for choosing this approach was the fact that similar energy models, such as those discussed in [Zhou et al., 2013] and [Yu et al., 2019], were also formulated as stochastic linear programs to include uncertainty. However, as we will explain below, there were several reasons why this approach may not have been the most appropriate.

First of all, Oemof-B3 is an exceptionally large model, containing numerous demand variables, all of which we chose to model as stochastic. While this is an inherent feature of the model that we cannot change, it is something we should have considered more carefully before applying the stochastic linear programming framework.

Secondly, we did not perform a sensitivity analysis to understand the relative importance of the variables in the model. We introduced random variables for demand volatility, but it's possible that these variables don't influence the model outcome as drastically as we initially assumed. If that were the case, including randomness in these areas may not have been necessary.

Furthermore, no comparison with a deterministic model was conducted. A sufficient approach might have been to replace the random variables with their mean values or another reliable estimate, and then solve the resulting linear program.

Lastly, and most importantly, there is a fundamental lack of understanding of the model's interpretation. It is unclear what the outcomes of the decision variables and the resulting objective value actually represent in a real-world context. Moreover, it was only after analyzing the convergence behavior of the L-Shaped method with the second stopping criterion, as shown in figure 5.8, that we realized that (nearly) all costs were coming from the first-stage decision variables. Whether this is realistic remains uncertain, and it is a critical issue that requires further investigation.

Having discussed the results and research limitations, we will now draw a conclusion from our research. The objective of this thesis was to revisit the computational challenges of solving stochastic linear programs and to evaluate their usability, with the aim of assessing their potential. To achieve this, we implemented two solution methods for solving two-stage stochastic linear programs: the Extensive Form method and the L-Shaped method. We then assessed their performance on various 2-SLP test problems and applied them to the energy model Oemof-B3.

Mathematically speaking, stochastic linear programs offer great potential. The 2-SLP framework can be applied to any linear program involving a time component, naturally extending it to a 2-SLP that accounts for uncertainty in the second stage. Furthermore, because these problems inherently involve uncertainty, they are well-suited for various sampling techniques, which can provide significant computational advantages. However, in practice, solving 2-SLPs presents challenges, with the primary difficulty being the high computational cost. This is primarily driven by the size of the random vector $\boldsymbol{\xi}$, which grows exponentially with the number of random variables. In the context of energy system modeling, there is definite potential in applying stochastic linear programming, but both the initial model and the number of random variables should be of manageable size to ensure computational feasibility. A second challenge we encountered when solving 2-SLPs was the lack of user-friendly tools and readily available methods. Every step in both the Extensive Form method and the L-Shaped method, including the reading of SMPS files, had to be implemented manually. If optimization solvers like Gurobi provided built-in solution methods for 2-SLPs, or if reliable open-source implementations were available, much less time would have been spent on the implementation process. This would have allowed for a deeper exploration of the true potential of stochastic linear programs. Due to this lack of accessibility, we conclude that while stochastic linear programs hold great promise, their practical usability remains limited.

Based on the challenges and limitations encountered in this research, we propose two directions for future work. One promising direction for optimizing the L-Shaped method is the implementation of parallel computing. The for-loop that iterates over all scenarios is inherently suited for parallelization, and it would be interesting to see how much this could accelerate the method's performance.

Generally, rather than focusing on developing additional solution methods, the priority should be on developing more efficient and user-friendly optimization solvers for stochastic linear programs. By providing more intuitive interfaces and optimizing solver performance, it would become easier for users to implement and solve stochastic linear programs. As a result, stochastic optimization could reach a wider audience, encouraging practical application in various fields.

A Appendix

A.1 Example energy system model formulation

Objective function:

$$\min_{f} \sum_{t \in T} \sum_{(s,e) \in E} c_{(s,e)} \cdot f_{(s,e)}(t). \tag{A.1}$$

Bus balance constraints:

$$\sum_{(s,b) \in E} f_{(s,b)}(t) = \sum_{(b,e) \in E} f_{(b,e)}(t) \quad \forall b \in B,\ \forall t \in T. \tag{A.2}$$

Converter balance constraints:

$$k_{co} \cdot f_{(s,co)}(t) = f_{(co,e)}(t) \quad \forall co \in Co,\ \forall t \in T. \tag{A.3}$$

Storage constraints:

$$\begin{aligned} & loss_{st}(t) = \lambda_{st} \cdot cont_{st}(t) && \forall st \in St,\ \forall t \in T, \\ & k^{in}_{st} \cdot f_{(b,st)}(t) + cont_{st}(t) - loss_{st}(t) = && \tfrac{1}{k^{out}_{st}} \cdot f_{(st,b)}(t) + cont_{st}(t+1), \\ & && \forall st \in St,\ \forall t \in T, \\ & cont_{st}(0) = cont_{st}(8760) && \forall st \in St. \end{aligned} \tag{A.4}$$

Flow bounds:

$$0 \leq f_{(s,e)}(t) \leq f^{max}_{(s,e)} \quad \forall f,\ \forall t \in T. \tag{A.5}$$

Storage bounds:

$$\begin{aligned} 0 \leq \quad & loss_{st}(t) \quad \leq \infty \quad \forall st \in St,\ \forall t \in T, \\ 0 \leq \quad & cont_{st}(t) \quad \leq cap_{st} \quad \forall st \in St,\ \forall t \in T. \end{aligned} \tag{A.6}$$

A.2 Oemof-B3 model formulation

Objective function:

$$\min_{C,f} \sum_{r \in R} \sum_{st \in St} c^{stor,r}_{st} \cdot C^{storinv,r}_{st} + \sum_{r \in R} \sum_{(s,e) \in E} c^{flow,r}_{(s,e)} \cdot C^{flowinv,r}_{(s,e)} + \sum_{t \in T} \sum_{r \in R} \sum_{(s,e) \in E} c^{mar,r}_{(s,e)} \cdot f^{r}_{(s,e)}(t). \tag{A.7}$$

Initial storage content constraints:

$$C^{storinv,r}_{st} \geq C^{init,r}_{st} \quad \forall st \in St,\ \forall r \in R, \tag{A.8}$$

Storage capacity balance constraints:

$$C^{flowinv,r}_{(b,st)} = C^{flowinv,r}_{(st,b)} \quad \forall st \in St,\ \forall r \in R, \tag{A.9}$$

Bus balance constraints:

$$\sum_{(s,b) \in E} f^{r}_{(s,b)}(t) = \sum_{(b,e) \in E} f^{r}_{(b,e)}(t) \quad \forall b \in B,\ \forall r \in R,\ \forall t \in T. \tag{A.10}$$

Converter balance constraints:

$$k^{r}_{co} \cdot f^{r}_{(s,co)}(t) = \quad f^{r}_{(co,e)}(t) \quad \forall co \in Co,\ \forall r \in R,\ \forall t \in T. \tag{A.11}$$

Maximum emission constraint:

$$\sum_{t\in T}\sum_{r\in R}\varepsilon^{r}_{ch4\_import}\cdot f^{r}_{(ch4\_import,bch4)}(t)+\varepsilon^{r}_{h2\_import}\cdot f^{r}_{(h2\_import,bh2)}(t)+\sum_{t\in T}\varepsilon^{BB}_{elec\_import}\cdot f^{BB}_{(elec\_import,belec)}(t)\leq 0. \tag{A.12}$$

Electricity-gas relations:

$$\begin{aligned}&k^{r}_{bheatcen}\cdot f^{r}_{(ch4\_boil\_large,bheatcen)}(t)=f^{r}_{(elec\_heat\_large,bheatcen)}(t)+f^{r}_{(elec\_pth,bheatcen)}(t)\\&\forall r\in R,\,\forall t\in T.\\&k^{r}_{bheatdecen}\cdot f^{r}_{(ch4\_boil\_small,bheatdecen)}(t)=f^{r}_{(elec\_heat\_heat\_small,bheatdecen)}(t)\\&\forall r\in R,\,\forall t\in T.\end{aligned} \tag{A.13}$$

Renewable generation flow constraints:

$$\begin{aligned}f^{r}_{(PV,belec)}(t)=&\quad p^{r}_{PV}(t)\cdot C^{flowinv,r}_{(PV,belec)}\quad\forall r\in R,\,\forall t\in T,\\f^{r}_{(wind,belec)}(t)=&\quad p^{r}_{wind}(t)\cdot C^{flowinv,r}_{(wind,belec)}\quad\forall r\in R,\,\forall t\in T,\\f^{r}_{(hydro\_ror,belec)}(t)=&\quad p^{r}_{hydro\_ror}(t)\cdot C^{flowinv,r}_{(hydro\_ror,belec)}\quad\forall r\in R,\,\forall t\in T.\end{aligned} \tag{A.14}$$

Maximum converter flow constraints:

$$f^{r}_{(co,e)}(t)\leq\quad C^{flowinv,r}_{(co,e)}\quad\forall co\in Co,\,\forall r\in R,\,\forall t\in T. \tag{A.15}$$

Maximum storage flow constraints:

$$\begin{aligned}f^{r}_{(s,st)}(t)\leq&\quad C^{flowinv,r}_{(s,st)}\quad\forall st\in St,\,\forall r\in R,\,\forall t\in T,\\f^{r}_{(st,e)}(t)\leq&\quad C^{flowinv,r}_{(st,e)}\quad\forall st\in St,\,\forall r\in R,\,\forall t\in T.\end{aligned} \tag{A.16}$$

Storage balance constraints at $t=0$:

$$k_{st}\cdot f^{r}_{(s,st)}(0)+(1-\delta_{st})\cdot C^{init,r}_{st}=\tfrac{1}{k_{st}}\cdot f^{r}_{(st,e)}(0)+C^{stor,r}_{st}(0)\quad\forall st\in St,\,\forall r\in R. \tag{A.17}$$

Storage balance constraints:

$$\begin{aligned}&k_{st}\cdot f^{r}_{(s,st)}(t)+(1-\delta_{st})\cdot C^{stor,r}_{st}(t-1)=\tfrac{1}{k_{st}}\cdot f^{r}_{(st,e)}(t)+C^{stor,r}_{st}(t)\\&\forall st\in St,\,\forall r\in R,\,\forall t\in T\setminus\{0\}.\end{aligned} \tag{A.18}$$

Storage balance constraints for $t=0$ and $t=8760$:

$$C^{init,r}_{st}=C^{stor,r}_{st}(8760)\quad\forall st\in St,\,\forall r\in R. \tag{A.19}$$

Maximum storage content contraints:

$$C^{stor,r}_{st}(t)\leq\quad C^{storinv,r}_{st}\quad\forall st\in St,\,\forall r\in R,\,\forall t\in T. \tag{A.20}$$

Link-block relations:

$$\begin{aligned}&\frac{1}{k_{elec\_trans}}\cdot f^{BB}_{(elec\_trans,belec)}(t)=f^{B}_{(belec,elec\_trans)}(t)\quad\forall t\in T,\\&\frac{1}{k_{elec\_trans}}\cdot f^{B}_{(elec\_trans,belec)}(t)=f^{BB}_{(belec,elec\_trans)}(t)\quad\forall t\in T.\end{aligned} \tag{A.21}$$

Flow bounds:

$$0 \leq f_{(s,e)}(t) \leq f_{(s,e)}^{max} \quad \forall f,\ \forall t \in T. \tag{A.22}$$

Initial storage bounds:

$$0 \leq C_{st}^{init,r} \leq \infty \quad \forall st \in St,\ \forall r \in R. \tag{A.23}$$

Storage content bounds:

$$0 \leq C_{st}^{stor,r}(t) \leq \infty \quad \forall st \in St,\ \forall r \in R,\ \forall t \in T. \tag{A.24}$$

Generic investment storage bounds:

$$0 \leq C_{st}^{storinv,r} \leq cap_{st}^{r} \quad \forall st \in St,\ \forall r \in R. \tag{A.25}$$

Generic investment flow bounds:

$$\begin{aligned} 0 \leq C_{(s,st)}^{flowinv,r} &\leq flowcap_{st}^{r} \quad \forall st \in St,\ \forall r \in R, \\ 0 \leq C_{(st,e)}^{flowinv,r} &\leq flowcap_{st}^{r} \quad \forall st \in St,\ \forall r \in R. \end{aligned} \tag{A.26}$$

References

[Aardal et al., 2023] Aardal, K., van Iersel, L., and Janssen, R. (2023). Lecture notes am2020/in4344 optimization.

[Abbey and Joos, 2009] Abbey, C. and Joos, G. (2009). A stochastic optimization approach to rating of energy storage systems in wind-diesel isolated grids. *IEEE Transactions on Power Systems*, 24(1):418–426.

[Birge and Louveaux, 2011] Birge, J. R. and Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer-Verlag, New York, NY, USA.

[Dantzig and Infanger, 1991] Dantzig, G. B. and Infanger, G. (1991). Large-scale stochastic linear programs—importance sampling and benders' decomposition. In Brezinski, C. and Kulisch, U., editors, *Computational and Applied Mathematics I*, pages 111–120. North-Holland, Amsterdam.

[Ermoliev, 1988] Ermoliev, Y. (1988). Stochastic quasigradient methods. In Ermoliev, Y. and Wets, R.-B., editors, *Numerical Techniques for Stochastic Optimization Problems*. Springer-Verlag, Berlin.

[Gangammanavar et al., 2020] Gangammanavar, H., Liu, Y., and Sen, S. (2020). Stochastic decomposition for two-stage stochastic linear programs with random cost coefficients. *INFORMS Journal on Computing*, 33(1):51–71.

[Gürkan et al., 1994] Gürkan, G., Özge, A. Y., and Robinson, S. M. (1994). Sample-path optimization in simulation. In *Proceedings of the Winter Simulation Conference*, pages 247–254.

[Haneveld et al., 2024] Haneveld, W. K. K., van der Vlerk, M. H., and Romeijnders, W. (2024). *Stochastic Programming: Modeling Decision Problems*. Graduate Texts in Operations Research. Springer, Enschede, The Netherlands.

[Higle and Sen, 1991] Higle, J. L. and Sen, S. (1991). Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Math. Oper. Res.*, 16:650–669.

[Hilpert et al., 2018] Hilpert, S., Kaldemeyer, C., Krien, U., Günther, S., Wingenbach, C., and Plessmann, G. (2018). The open energy modelling framework (oemof) - a new approach to facilitate open science in energy system modelling. *Energy Strategy Reviews*, 22:16–25.

[Kall and Mayer, 1998] Kall, P. and Mayer, J. (1998). On solving stochastic linear programming problems. In Marti, K. and Kall, P., editors, *Stochastic Programming Methods and Technical Applications*, pages 329–344. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Linderoth et al., 2006] Linderoth, J., Shapiro, A., and Wright, S. (2006). The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142(1):215–241.

[Linderoth et al., 2002] Linderoth, J. T., Shapiro, A., and Wright, S. J. (2002). The empirical behavior of sampling methods for stochastic programming. Technical Report Optimization Technical Report 02-01, Computer Science Department, University of Wisconsin-Madison. Revised October, 2002.

[Louveaux and Smeers, 1988] Louveaux, F. V. and Smeers, Y. (1988). Optimal investments for electricity generation: A stochastic model and a test problem. In Ermoliev, Y. and Wets, R., editors, *Numerical Techniques for Stochastic Optimization Problems*, pages 445–452. Springer-Verlag, Berlin.

[Mak et al., 1999] Mak, W.-K., Morton, D. P., and Wood, R. (1999). Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(1):47–56.

[Narayan and Ponnambalam, 2017] Narayan, A. and Ponnambalam, K. (2017). Risk-averse stochastic programming approach for microgrid planning under uncertainty. *Renewable Energy*, 101:399–408.

[Ntaimo, 2015] Ntaimo, L. (2015). *Computational Stochastic Programming*, volume 774 of *Springer Optimization and Its Applications*. Springer, New York.

[Rahmaniani et al., 2017] Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.

[Rubinstein and Shapiro, 1993] Rubinstein, R. Y. and Shapiro, A. (1993). *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*. Wiley, New York.

[Seljom et al., 2021] Seljom, P., Kvalbein, L., Hellemo, L., Kaut, M., and Ortiz, M. M. (2021). Stochastic modelling of variable renewables in long-term energy models: Dataset, scenario generation and quality of results. *Energy*, 236:121415.

[Sen and Liu, 2016] Sen, S. and Liu, Y. (2016). Mitigating uncertainty via compromise decisions in two-stage stochastic linear programming: Variance reduction. *Operations Research*, 64(6):1422–1437.

[Van Slyke and Wets, 1969] Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):630–644.

[Yu et al., 2019] Yu, J., Ryu, J.-H., and beum Lee, I. (2019). A stochastic optimization approach to the design and operation planning of a hybrid renewable energy system. *Applied Energy*, 247:212–220.

[Zhou et al., 2013] Zhou, Z., Zhang, J., Liu, P., Li, Z., Georgiadis, M. C., and Pistikopoulos, E. N. (2013). A two-stage stochastic programming model for the optimal design of distributed energy systems. *Applied Energy*, 103:135–144.