Dissertation

Advanced acceleration techniques for Nested Benders decomposition in Stochastic Programming

Christian Wolf, M.Sc.

Schriftliche Arbeit zur Erlangung des akademischen Grades
doctor rerum politicarum (dr. rer. pol.)
im Fach Wirtschaftsinformatik

eingereicht an der
Fakultät für Wirtschaftswissenschaften der
Universität Paderborn

Gutachter:
1. Prof. Dr. Leena Suhl
2. Prof. Dr. Csaba I. Fábián

Paderborn, im Oktober 2013

Acknowledgements

This thesis is the result of working for nearly four years at the Decision Support & Operations Research (DS&OR) Lab at the University of Paderborn. I would like to thank those people whose support helped me in writing my dissertation.

First of all, I thank my supervisor Leena Suhl for giving me the possibility to pursue a thesis by offering me a job at her research group in the first place. Her support and guidance over the years helped me tremendously in finishing the dissertation. I also thank Achim Koberstein for introducing me to the field of Operations Research through the lecture "Operations Research A" and the opportunity to do research as a student in the field of stochastic programming. It is due to his insistence that I decided to write my computer science master's thesis at the DS&OR Lab. I am very grateful to have Csaba Fábián as my second advisor. He not only gave valuable advice but also pointed me towards the on-demand accuracy concept. Collaborating with him has been both straightforward and effective.

My present and past colleagues at the DS&OR Lab deserve a deep thank you for the enjoyable time I had at the research group in the last years. Discussions on scientific topics were as insightful as our non-work related discussions were humorous. I would like to thank Corinna Hallmann, Stefan Kramkowski, Daniel Rudolph, and Franz Wesselmann for their time and patience in discussing implementation details and other problems that have appeared out of nowhere during the implementation of solver software. I thank my office colleague Boris Amberg for funny and interesting conversations across two monitors and for leaving the office to me until lunch.

Last but not least, I thank my family and friends, especially my parents, who believed in me and supported me. Special thanks go to my wife Pia. Finishing the thesis without her would have been much more difficult.

Paderborn, October 2013 Christian Wolf

Contents

1. Introduction

> *"Our new Constitution is now established, and has an appearance that promises permanency; but in this world nothing can be said to be certain, except death and taxes."*
>
> — Benjamin Franklin, *Letter to Jean-Baptiste Leroy*

Real-world optimization problems are often modeled with traditional mathematical programming techniques. The implicit assumption, when using these tools, is that the underlying real-world problem is deterministic. Many real-world problems usually include uncertainty, such as uncertainty about future events, lack of reliable data, etc. The model could therefore be subject to uncertainty in its parameters or in itself, because the model is an approximation of the real-world problem and thus the optimal solution of the model may not be the optimal solution of the modeled problem.

Attempts to investigate the effects of uncertainty with traditional methods like sensitivity analysis of optimal solutions or scenario analysis, e.g., solving a deterministic model with different parameters, do not suffice to take the effect of uncertainty into account (e.g., (Wallace, 2000) and (King & Wallace, 2012, p. 2ff)). Thus to determine if uncertainty is of importance for a particular model, it has to be checked by incorporating uncertainty into the optimization problem.

Stochastic programming is a mathematical programming field that provides techniques to handle optimization under uncertainty. It is due to the early work of Dantzig (1955), Beale (1955) and Charnes & Cooper (1959). A common concept is that a decision has to be made here and now, and the uncertain future will reveal itself after that. A recourse decision can then be taken to react upon the new information.

The key questions are (King & Wallace, 2012, p. 1)

- What are the important uncertainties?
- How can we handle them?
- Can we deliver valuable solutions and insights?

It is one of the main difficulties for many practical problems to deliver a solution *at all*, because incorporating uncertainty makes a model usually larger and harder to solve. The usage of specialized solution techniques, e.g., Benders decomposition, Progressive Hedging, Stochastic decomposition, Sample Average Approximation, etc., make practical decision problems tractable. The resulting solutions can then be examined for valuable insights. The theoretical development and practical implementation of solution techniques is therefore important to get people to use stochastic programming in the first place and thus improve their decision making capabilities. In addition, modeling tools that aid operation research

practitioners in modeling and analyzing stochastic programs are necessary to make the transition from modeling linear programs to stochastic programs possible.

The importance and widespread applicability of stochastic programming is demonstrated by its variety of application areas, including electricity, finance, supply-chain, production, telecommunications and others (see the collection edited by Wallace & Ziemba (2005)). Two recent examples demonstrate that the usage of stochastic programming leads to better decisions.

A strategic gas portfolio planning problem (Koberstein et al., 2011) determines the parameters of baseload and open contracts for gas delivery for the next gas year, where recourse actions are necessary to cover the demand during a year by using storages, open contracts and the spot market. The uncertainty of the problem is the demand, which correlates with the weather conditions. As gas is widely used for heating, colder winters generate more demand than warmer winters. Incorporating this uncertainty into the model results in an expected advantage of 5.9 million euro of the stochastic solution compared with the solution from the deterministic model. The expected solution value of the stochastic program is 182.2 million euro.

A company that owns wind power plants and hydro power plants has to schedule the plants operationally. The goal is to optimize the profit of the company, while satisfying customer demand (Vespucci et al., 2012). Excess energy generated from wind power plants can be used to pump water into higher reservoirs that can later be used by the hydro power plants. The wind depends on uncertain weather conditions and thus the power generated from wind power plants is subject to uncertainty. Vespucci et al. (2012) analyze a stochastic programming model that takes weather forecast uncertainty into account and contrast this with a deterministic model where the forecast is taken at face value. The stochastic programming model results in significant savings compared to the deterministic model.

Implementing these and other problems is easier with algebraic modeling languages that are capable of modeling stochastic programs directly. Using specialized solution methods directly after specifying the model results in either computing time savings or opens up the possibility to solve the resulting problems in the first place. Supporting and easing this process is the topic of this thesis.

The thesis is structured in three parts. The first part deals with the fundamentals. It gives an understanding of stochastic programming in Chapter 2, along with mathematical properties of these problems from which solution methods can be derived. We introduce different basic solution methods for stochastic programs with recourse in Chapter 3. In particular, we introduce the deterministic equivalent, Benders decomposition, Lagrangean relaxation, and approximative solution methods.

The second part of the thesis reviews the current state-of-the-art with respect to Benders decomposition and modeling languages for stochastic programs. Chapter 4 details acceleration techniques for Benders decomposition, in particular for two-stage problems. Acceleration techniques for multi-stage problems, where Benders decomposition is applied in a nested fashion, are described in Chapter 5. An overview about challenges and developments in the area of algebraic modeling languages for stochastic programs is given in Chapter 6. Given the state-of-the-art, we derive the goals of our research in Chapter 7.

Part III describes advanced acceleration techniques for the nested Benders decomposition algorithm and gives computational results to evaluate their effectiveness. Techniques like cut consolidation, dynamic sequencing, parallelization, different level decomposition projection problems, and on-demand accuracy are detailed in Chapter 8. Our extension of the algebraic modeling language FlopC++ to stochastic programs is described in Chapter 9. Chapter 10 contains a description of the algorithm implementation and gives extensive evaluations of the developed and implemented acceleration techniques. We conclude the contributions of this thesis in Chapter 11 and give directions for future research.

Part I.

Fundamentals

2. Stochastic Programming Preliminaries

Stochastic programs and the needed preliminaries are introduced in this chapter. We start with mathematical programs, in particular linear and mixed-integer programs in Section 2.1. We then give some basic results in polyhedral theory that are necessary for the explanation of Benders decomposition. After that we introduce stochastic programming in Section 2.2, together with basic probability theory. Introductory texts especially for linear programming are, among others, (Vanderbei, 1997; Chvátal, 1983; Nering & Tucker, 1993). A more theoretically oriented textbook is written by Schrijver (1998). A detailed introduction to the implementation of the simplex algorithm, the main solution algorithm of linear programs used in Benders decomposition, can be found in (Maros, 2003).

2.1. Mathematical Programs

A mathematical program is an optimization problem of the following form,

$$\begin{aligned} \min \quad & f(x) \\ & g(x) \geq \mathbf{0} \\ & x \in \mathcal{X}, \end{aligned}$$

where $\mathbf{0}$ (written in boldfont) denotes a column vector of zeroes with appropriate dimension. The function f maps from $\mathcal{R}^n$ to $\mathcal{R}$ and g maps from $\mathcal{R}^n$ to $\mathcal{R}^m$. The set $\mathcal{X} \subseteq \mathcal{R}^n$ together with the constraint $g(x) \geq \mathbf{0}$ defines the feasibility set $\mathcal{F}$ of the mathematical program. The function f is the objective function of the mathematical program. We assume throughout this thesis that the default optimization direction is minimization if not stated otherwise.

A point $x \in \mathcal{R}^n$ is called a solution. A solution x is feasible if $x \in \mathcal{X}$ and the constraints $g(x) \geq \mathbf{0}$ hold, i.e., $x \in \mathcal{F}$. Otherwise the solution is called infeasible. A solution $x^* \in \mathcal{F}$ is optimal if $f(x^*) \leq f(x), \forall x \in \mathcal{F}$. Note that an optimal solution does not have to be unique. A mathematical program is infeasible if the feasibility set $\mathcal{F}$ is empty. A mathematical program is unbounded if for every number $M \in \mathcal{R}$ there is a solution $x \in \mathcal{F}$, such that $f(x) < M$.

Mathematical programming problems are classified according to properties of the functions f and g and the set $\mathcal{X}$. Two important categories are linear programming and mixed-integer linear programming. A linear program (LP) is a mathematical program with linear functions f and g, where $\mathcal{X}$ is continuous. A mixed-integer program (MIP) is a mathematical program with linear functions f and g, where $\mathcal{X}$ is partly continuous and partly discrete. A pure integer program (IP) is a mathematical program with linear functions f and g, where $\mathcal{X}$ is discrete. A convex non-linear program has non-linear functions f and g, where $\mathcal{X}$ is convex. Quadratic programs (QPs) are an example for convex non-linear programs with a quadratic objective function f, but linear function g, where

$\mathcal{X}$ is continuous. The hardness of the problems differs depending on the functions and $\mathcal{X}$. LP problems are in $\mathcal{P}$, together with QP problems that have a positive semidefinite quadratic coefficient matrix in the objective function. General IP, MIP and QP problems are $\mathcal{NP}-$hard.

We write the linear program $P1$ in the following matrix notation standard form

$$\begin{array}{lll} & \min & c^T x \\ (P1) & \text{s.t.} & Ax \geq b \\ & & x \geq \mathbf{0}, \end{array} \tag{2.1}$$

with right hand side vector $b \in \mathcal{R}^m$, objective function coefficients vector $c \in \mathcal{R}^n$, decision variables vector $x \in \mathcal{R}^n$ and the constraint matrix $A \in \mathcal{R}^{m \times n}$. To alleviate notation, in the remainder of this thesis we will not specify which vectors we transpose, but we assume that the vectors have appropriate dimensions and are used in the transposed form if necessary. It helps in keeping the presentation clear, but concise.

A LP can also be written in the summation notation given by equation (2.2), where every decision variable $x_i, i = 1, \ldots, n$ is stated explicitly.

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{n} c_i x_i & \\ \text{s.t.} & \displaystyle\sum_{i=1}^{n} a_{ij} x_i \geq b_j & j = 1, \ldots, m \\ & x_i \geq 0 & i = 1, \ldots, n. \end{array} \tag{2.2}$$

The matrix entry a_{ij} is the coefficient of the constraint matrix A in column i and row j. As both of the forms (2.1) and (2.2) are equivalent and differ only in notation, we use the form which is best suited to explain different concepts later in this thesis.

A more general, but equivalent form of LP (2.2) is formulation (2.3)

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{n} c_i x_i & \\ \text{s.t.} & \displaystyle\sum_{i=1}^{n} a_{ij} x_i + x_{n+j} = b_j & j = 1, \ldots, m \\ & l_i \leq x_i \leq u_i & i = 1, \ldots, n+m. \end{array} \tag{2.3}$$

In formulation (2.3) every decision variable has a lower bound l_i and an upper bound u_i. The variables $x_{n+j}, j = 1, \ldots, m$ are called slack variables, because they take up the slack between $\sum_{i=1}^{n} a_{ij} x_i$ and b_j, as we have only equalities as constraints. The three different constraint types $\geq, \leq$ and $=$ are modeled via the bounds on the slack variables. When the slack variable x_{n+j} has the bounds $l_j = 0$ and $u_j = \infty$, it is a $\geq$ constraint. With the bounds $l_j = -\infty$ and $u_j = 0$ it is a $\leq$ constraint. An equality is achieved with the bounds $l_j = 0$ and $u_j = 0$, i.e., the slack variable is fixed to zero. The coefficient matrix A is necessarily of full rank, due to the slack variables. It is possible to create an equivalent LP in standard form with additional variables and/or constraints (Chvátal, 1983).

An important concept that can be applied to linear programs is duality theory. Every linear program has a corresponding dual linear program; both together form a primal/dual pair. The original LP is also called the primal problem. The dual LP of the dual problem to a primal problem is again the primal problem. The dual LP $D1$ of the primal LP $P1$ (2.1) is

$$\begin{array}{lll} & \max & by \\ (D1) & \text{s.t.} & A^T y \leq c \\ & & y \geq \mathbf{0}. \end{array} \tag{2.4}$$

A dual problem can be used to give a lower bound to the primal problem as well as the primal problem can be used to give an upper bound to the dual problem (Vanderbei, 1997, p. 51ff). We note that every feasible solution for a primal LP is at the same time an upper bound for this problem.

The following basic, but important, theorems and their proofs can be found in every LP textbook, e.g. (Vanderbei, 1997, p. 53-64). The Weak Duality Theorem (2.1) states that every feasible solution for the dual is a lower bound for the primal problem.

Theorem 2.1. *Let x be a feasible solution for a primal LP P1 and y be a feasible solution for the corresponding dual LP D1. Then it holds that $c^T x \geq by$.*

The Strong Duality Theorem 2.2 states that if a primal problem has an optimal solution, the corresponding dual problem also has an optimal solution, such that the objective values coincide.

Theorem 2.2. *Let x^* be an optimal solution for a primal LP P1. Then the corresponding dual LP D1 has an optimal solution y^* such that $c^T x^* = b^T y^*$.*

Together with the Complementary Slackness Theorem (2.3), it is possible to construct these solutions from one another (Vanderbei, 1997, p. 63f).

Theorem 2.3. *Let $(x_1, \ldots, x_n)$ be a primal feasible solution for a primal LP P. Let $(y_1, \ldots, y_m)$ be a dual feasible solution for the corresponding dual LP D. Let $(w_1, \ldots, w_m)$ denote the primal slack variables and $(z_1, \ldots, z_n)$ the dual slack variables. Then x and y are optimal for their respective problem if and only if*

$$\begin{aligned} x_i z_i &= 0, \qquad i = 1, \ldots, n \\ y_j w_j &= 0, \qquad j = 1, \ldots, m. \end{aligned}$$

Every LP in standard form (2.1) has an associated polyhedron $P := \{x \in \mathcal{R}^n \mid Ax \geq b, x \geq 0\}$. For the following definitions and theorems the constraint matrix is assumed to be of full rank and $P \neq \emptyset$ (see (Nemhauser & Wolsey, 1999, p. 92-98) for the definitions and proofs of the theorems, (Schrijver, 1998, p. 85-107) is an alternative source). The feasible region $\mathcal{F}$ of an LP can be described by a finite number of extreme points and extreme rays that we define next.

Definition 2.4. *A point $x \in P$ is called an extreme point of P, if there do not exist points $x_1, x_2 \in P, x_1 \neq x_2$, such that $x = \lambda x_1 + (1 - \lambda) x_2, 0 < \lambda < 1$.*

Definition 2.5. *Let $P^0 := \{r \in \mathcal{R}^n \mid Ar \leq 0\}$. Any $r \in P^0 \setminus \{0\}$ is called a ray of P.*

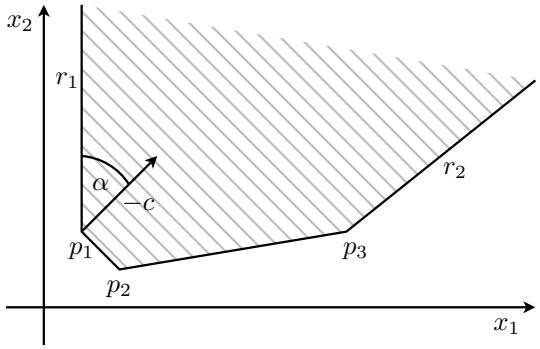

Figure 2.1. A polyhedron with extreme points and extreme rays.

Definition 2.6. *A ray $r \in P$ is an extreme ray if there do not exists rays $r_1, r_2 \in P^0, r_1 \neq \lambda r_2$ for any $\lambda > 0$, such that $r = \mu r_1 + (1-\mu) r_2, 0 < \mu < 1$.*

A polyhedron with extreme points p_1, p_2 and p_3 and extreme rays r_1 and r_2 is shown in Figure 2.1. The optimization problem associated with the polyhedron is a minimization problem, thus the objective function vector is followed in its opposed direction, namely $-c$. The optimization direction is depicted in Figure 2.1 by the vector $-c$. The angle α between r_1 and $-c$ is acute, therefore $-c \cdot r_1$ is greater than zero and $c \cdot r_1$ is less than zero, due to the equation $\cos(\alpha) = \frac{a \cdot b}{|a| \cdot |b|}$, with $a, b \in \mathcal{R}^n \setminus \{0\}$ and α being the angle between them.

Theorem (2.7) (Nemhauser & Wolsey, 1999, p. 95), which we will use in the explanation of Benders decomposition, states that an unbounded maximization problem has an extreme ray that makes an acute angle with the objective function vector.

Theorem 2.7. *If $\max\{cx \mid x \in P\}$ is unbounded P has an extreme ray r^* with $cr^* > 0$.*

The decomposition theorem for polyhedra (also called Minkowski-Weyl's theorem) states that polyhedra can be represented by convex combinations of their extreme points and extreme rays (Nemhauser & Wolsey, 1999, p.96).

Theorem 2.8 (Decomposition theorem for polyhedra)**.** *The polyhedron P can be represented as*

$$P = \left\{ x \in \mathcal{R}^n \mid x = \sum_{i \in I} \lambda_i x_i + \sum_{j \in J} \mu_j r_j \text{ with } \sum_{i \in I} \lambda_i = 1 \right. \\ \left. , \lambda_i \geq 0 \; \forall i \in I, \mu_j \geq 0 \; \forall j \in J \right\}. \quad (2.5)$$

where $\{x_i\}_{i \in I}$ is the set of extreme points and $\{r_j\}_{j \in J}$ is the set of extreme rays of P.

The Decomposition Theorem will be used in the explanation of Benders decomposition method together with the fact that every full-dimensional polyhedron has a finite number of extreme points and extreme rays.

The Minkowski-Weyl decomposition theorem can also be stated for general polyhedra, i.e., $P = \{Ax \leq b\}$ and $rank(A) \leq n$, but for that we need some more definitions (Schrijver, 1998, p. 87f).

Definition 2.9. *A nonempty set of points C in Euclidean space is called a cone if* $\lambda x+\mu y \in C, \forall x, y \in C$ *and* $\lambda, \mu \geq 0$.

Definition 2.10. *A cone C is polyhedral, if* $C = \{x \mid Ax \leq 0\}$.

The cone generated by the vectors $x_1, \ldots, x_m \in \mathcal{R}^n$ is the set

$$cone\{x_1, \ldots, x_m\} := \{\lambda_1 x_1 + \ldots + \lambda_m x_m, \lambda_1, \ldots, \lambda_m \geq 0\}, \tag{2.6}$$

and is called finitely generated (Schrijver, 1998, p. 87).

Theorem 2.11 (Farkas-Minkowski-Weyl theorem)**.** *A convex cone is polyhedral if and only if it is finitely generated.*

If the polyhedron has at least one extreme point, it is called pointed. A polyhedron is bounded if and only if the characteristic cone has dimension zero, i.e., $char.cone = \{\mathbf{0}\}$ (Schrijver, 1998, p. 100f). The characteristic cone is defined as $char.cone(P) = \{r|Ar \leq \mathbf{0}\}$.

Definition 2.12. *F is a face of P if and only if there is a vector c for which F is the set of vectors attaining* $\min\{cx|x \in P\}$*, provided that this minimum is finite*

Finally, the Minkowski-Weyl decomposition theorem for general polyhedra is stated as follows (Schrijver, 1998, p. 88)

Theorem 2.13 (Decomposition theorem for general polyhedra)**.** *A set P of vectors in Euclidean space is a polyhedron if and only if* $P = Q + C$ *for some polytope Q and some polyhedral cone C.*

In particular, the polyhedral cone C in Theorem (2.13) is the characteristic cone $char.cone(P) = \{r|Ar \leq \mathbf{0}\}$ (Schrijver, 1998, p. 100). Regarding the polytope Q, it can be described with the help of the minimal faces of P, as follows.

Let $F_1, \ldots, F_r$ be the minimal faces of the polyhedron P, and choose an element x_i from F_i, for $i = 1, \ldots, r$. Then (Schrijver, 1998, p. 106)

$$P = conv.hull\{x_1, \ldots, x_r\} + char.cone(P). \tag{2.7}$$

Thus the polyhedron P can be described by a finite set of vectors $\{x_1, \ldots, x_r\}$ and its characteristic cone, which is also finitely generated.

The simplex method The well-known simplex algorithm invented by Dantzig in 1947 is one of the main solution techniques for linear programming problems. The simplex algorithm can work on the primal problem as the primal simplex or the dual problem as the dual simplex. The simplex method is an iterative method that improves a starting solution until it reaches optimality or finds that the problem is unbounded. If no starting solution can be found, the problem is infeasible. A detailed introduction to the simplex method can be found in several textbooks, e.g., (Schrijver, 1998; Vanderbei, 1997; Chvátal, 1983; Maros, 2003). Another successful approach to solve LPs is the use of interior-point methods (see the textbooks (Ye, 1997),(Vanderbei, 1997),(Schrijver, 1998), among others).

2.2. Stochastic Programs

Mathematical programs that contain uncertainties can be modeled with the use of stochastic programming techniques. Several textbooks give a good introduction into stochastic programming, both theoretical and practical (Birge & Louveaux, 2011; Kall & Wallace, 1994; Ruszczyński & Shapiro, 2003; Kall & Mayer, 2010; Shapiro et al., 2009). An overview about the application of stochastic programming is given in the volume edited by Wallace & Ziemba (2005). A book about modeling stochastic programs was recently published (King & Wallace, 2012).

In this thesis we restrict ourselves to recourse problems whereas stochastic programming in general also handles problems with chance-constraints as well as distribution problems and combinations of these. A survey by Gassmann (2007) shows that the majority of stochastic problems are recourse problems. We present the taxonomy given by Gassmann & Ireland (1996) in Figure 2.2.

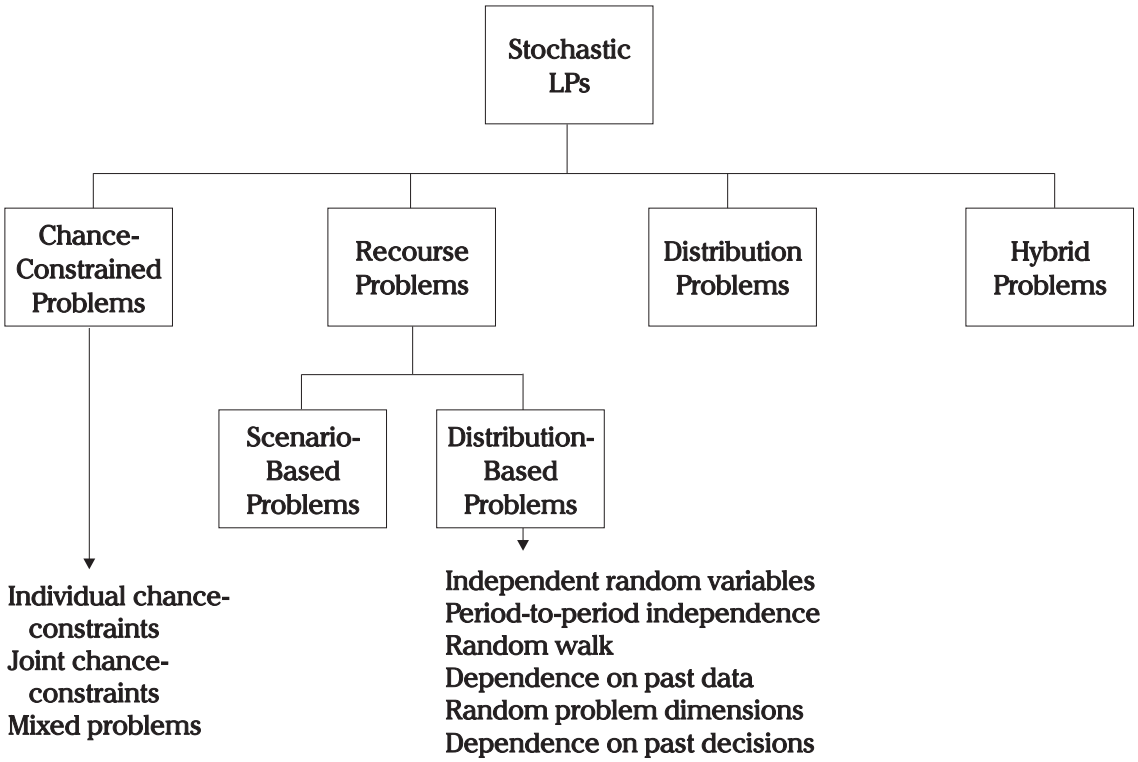

Figure 2.2. A taxonomy of stochastic LPs (Figure 3 in (Gassmann & Ireland, 1996))

A recourse problem is divided in several stages, where every stage except the first marks the realization of some uncertain parameters, but where the decision maker knows the distribution of the uncertain parameters. The first-stage marks decisions that have to be taken before any parameters become known to the decision maker. At the beginning of the second stage, some uncertain parameters are revealed and the decision maker is faced with this outcome and his former first-stage decision. The decision maker reacts with a second-stage or recourse decision to the revealed outcome. If the problem is a multi-stage

problem, this process is repeated until the last stage. The goal of the decision maker is to minimize the objective function value of the first-stage decision and the expected value of the objective function value of the second-stage decision. We stress that the decision maker has knowledge about the distribution of the uncertain parameters but does not know the concrete realization when he has to make his decision. We assume that the distribution of the uncertain parameters is independent of the decisions we take.

In Section 2.2.1 we describe the necessary preliminaries of probability theory to understand the two-stage stochastic problems explained in Section 2.2.2 and multi-stage stochastic problems described in Section 2.2.3. We end this section with basic properties of stochastic problems in Section 2.2.4.

2.2.1. Basic Probability Theory

Stochastic programs deal with uncertainty. Probability theory is an area of mathematics that formalizes uncertainty. As stochastic programming uses concepts defined in probability theory, we shortly describe the necessary ones. For a further introduction into probability theory the reader is referred to the literature (Bauer, 1991; Ross, 2004).

To formalize uncertainty, we use the mathematical concept of a probability space, which is a triplet $(\Omega, \mathcal{A}, P)$ (Birge & Louveaux, 2011, p. 56). Ω is the set of all possible outcomes ω of some random experiments, $\mathcal{A}$ is the set of all events over Ω and P is the associated set of probabilities. The probability of an event $P(A), A \in \mathcal{A}$ is always between zero and one. It holds that $0 \leq P(A) \leq 1, P(\emptyset) = 0$ and $P(\Omega) = 1$. The set of all events A is a σ-algebra.

The mathematical concept *filtration* defined on a measurable space $(\Omega, \mathcal{A})$ is an increasing family $\mathcal{A}_t \subseteq \mathcal{A}, t \geq 0$ of sub-σ-algebras of $\mathcal{A}$, i.e., $\mathcal{A}_t \subset \mathcal{A}_{t+1}$ (Revuz & Yor, 2004). Defined in this way, $\mathcal{A}_t$ is the collection of events that can occur before or at stage t.

The function $X : \Omega \to \mathcal{R}$ is called a random variable if

$$\{\omega \mid X(\omega) \leq r\} \in \mathcal{A}, \quad \forall r \in \mathcal{R}.$$

The cumulative distribution function $F(x)$ is defined as

$$F(x) = P\{X \leq x\}, \quad \forall x \in \mathcal{R}.$$

The probability mass function $p(x)$ is used to describe the probability of X taking the value x, so $p(x) = P\{X = x\}$. A random variable X is discrete if it can only take a countable number of values. A random variable X is continuous if it can take an uncountable number of values. We say that a random variable X is distributed according to a random distribution, specified by $F(x)$. Examples for random distributions are the Binomial distribution, Poisson distribution, Exponential distribution, Normal distribution, etc.

The expectation of a random variable X is denoted as $E[X]$. For a discrete random variable it can be written as $E[X] = \sum_{\omega \in \Omega} \omega P\{X = \omega\}$. For a continuous random variable, it is defined as the integral $E[X] = \int_{-\infty}^{\infty} x f(x) dx$, with $f(\cdot) = \frac{d}{dx} F(x)$ being the probability density function (Ross, 2004).

2.2.2. Two-Stage Stochastic Programs

The general two-stage stochastic program with recourse minimizes the cost of the first-stage decision and the expected cost of the second-stage decision. It is stated as follows

$$\begin{aligned} z = \min \quad & cx + E_\xi \left[\min q(\omega) y(\omega)\right] && (2.8)\\ \text{s.t.} \quad & Ax = b && (2.9)\\ & T(\omega)x + W(\omega)y(\omega) = h(\omega) && (2.10)\\ & x, y(\omega) \geq 0 && (2.11) \end{aligned}$$

The first-stage objective function coefficients $c \in \mathcal{R}^{n_1}$, constraint matrix $A \in \mathcal{R}^{m_1 \times n_1}$ and right hand side $b \in \mathcal{R}^{m_1}$ are deterministic and not subject to uncertainty. Every different outcome $\omega \in \Omega$ is called a scenario or realization. For any scenario ω some values in the technology matrix $T \in \mathcal{R}^{m_2 \times n_1}$, the recourse matrix $W \in \mathcal{R}^{m_2 \times n_2}$, the right hand side $h \in \mathcal{R}^{m_2}$ or the objective function $q \in \mathcal{R}^{n_2}$ may change. We can see every component of $T(\omega), W(\omega), h(\omega), q(\omega)$ as random variables that are influenced by the scenario ω. We can write $\xi(\omega)$ as a set of random vectors

$$\xi(\omega) = (T_1(\omega), \ldots, T_{n_1}(\omega), W_1(\omega), \ldots, W_{n_2}(\omega), h(\omega), q(\omega)),$$

where A_i denotes the $i-$th column of matrix A. The constraints (2.10) and (2.11) hold almost surely with respect to the scenario probabilities, i.e., for all ω with a probability greater than zero. It is possible to extend this formulation, e.g., by introducing integer requirements on the first- and/or second-stage. This can be done by replacing the non-negativity constraint (2.11) with the general form $x \in X, y(\omega) \in Y$ with $X = \mathcal{Z}_+^{n_1}, Y = \mathcal{Z}_+^{n_2}$.

Once we chose a realization ω and a first-stage solution x, we know the second-stage data via $\xi(\omega)$. Then, the second-stage variables or recourse variables $y(\omega)$ have to be chosen, according to objective function and constraints. The name recourse variables derives from the observation that the $y(\omega)$ variables react to the chosen first-stage variables x and the scenario dependent second-stage data $T(\omega), W(\omega), h(\omega)$ and $q(\omega)$. It is usually the case that most parts of T, W, h and q are deterministic or scenario independent and only some data is scenario dependent.

A reformulation of problem (2.8) is the deterministic equivalent model (DEM)

$$\begin{aligned} z = \min \quad & cx + \mathcal{Q}(x) \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned} \qquad (2.12)$$

with expected second stage value function $\mathcal{Q}(x) = E_\xi \left[Q(x, \omega)\right]$ and $Q(x, \omega)$

$$\begin{aligned} Q(x,\omega) = \min \quad & q(\omega)y(\omega) \\ \text{s.t.} \quad & T(\omega)x + W(\omega)y(\omega) = h(\omega) \\ & y(\omega) \geq 0. \end{aligned} \qquad (2.13)$$

As long as the random variables are discrete and finite, it is possible to formulate the two-stage stochastic problem with recourse (2.8) as the DEM, because the expected second

stage value function can be replaced by a summation, as further described in section 3.2. The second stage value function $Q(x, \omega)$ is also called *recourse function* and $\mathcal{Q}(x)$ is consequently called *expected recourse function*. The recourse function is defined to be $-\infty$ if the problem (2.13) is unbounded, and ∞ if it is infeasible, as usual. For the expected recourse function, we adhere to the convention that $\infty + (-\infty) = \infty$. In words it means that if any subproblem is infeasible, the expected recourse function takes the value ∞. This can be interpreted as a conservative approach by regarding the "bad" outcomes, i.e., $Q(x, \omega) = \infty$ that result from choosing x as more important (Walkup & Wets, 1967) than the "good" outcomes, i.e., $Q(x, \omega) = -\infty$.

For the rest of this thesis we assume that we have discrete and finite random variables, as otherwise the solution methods for which we propose enhancements can not work, as multi-dimensional integration would be required. We emphasize that problems with continuous or discrete random variables can be approximated by problems with discrete and finite random variables and thus be solved approximately with approximation methods described in this thesis in Section 3.5.

2.2.3. Multi-Stage Stochastic Programs

A two-stage stochastic program is a special case of the more general multi-stage stochastic program (see (Dupačová, 1995) for an introduction). A multi-stage stochastic program has a fixed number of stages in which uncertainty can be revealed, denoted by T. Therefore a first-stage decision x_1 is taken before uncertainty via a random vector ξ_2 is revealed. The next step is to react upon this with a recourse decision x_2. Then the uncertainty ξ_3 is revealed, where upon a recourse decision x_3 can be taken. This is repeated until the last stage T is reached, uncertainty ξ_T is revealed, and a final recourse decision x_T is taken. The notion of stage and decision is well-defined as we talk only about stochastic programs with recourse (see (Gassmann & Prékopa, 2005) for a discussion).

The general multi-stage stochastic linear program for a canonical probability space $(\Omega, \mathcal{A}, P)$ can be formulated as

$$
\begin{array}{lllll}
\min\limits_{x_1} \left[c_1 x_1 + \mathbf{E}_{\xi_2} \left[\min\limits_{x_2} c_2(\omega_2) x_2(\omega_2) + \ldots + \mathbf{E}_{\xi_T} \left[\min\limits_{x_T} c_T(\omega_T) x_T(\omega_T) \right] \ldots \right] \right] & & & & \\
\text{s.t.} \quad T_1 x_1 & & & = h_1 & \\
T_2(\omega_2) x_1 + W_2(\omega_2) x_2(\omega_2) & & & = h_2(\omega_2) & \\
\ddots & \ddots & & \vdots & \\
T_T(\omega_T) x_{T-1}(\omega_{T-1}) + W_T(\omega_T) x_T(\omega_T) & & & = h_T(\omega_T) & \\
l_t(\omega_t) \leq x_t(\omega_t) \leq u_t(\omega_t) & & & t = 1, \ldots, T. &
\end{array}
\tag{2.14}
$$

The ξ_t are random vectors consisting of

$$
(T_{1,t}(\omega), \ldots, T_{n_t,t}(\omega), W_{1,t}(\omega), \ldots, W_{n_t,t}(\omega), h_t(\omega), q_t(\omega))
$$

for $t = 1, \ldots, T$ defined on a probability space $(\Omega, \mathcal{A}_t, P)$ such that $\mathcal{A}_t \subset \mathcal{A}, t = 1, \ldots, T$ and $\mathcal{A}_t \subset \mathcal{A}_{t+1}, t = 1, \ldots, T-1$. The underlying stochastic process on $(\Omega, \mathcal{A})$ is adapted to the filtration $\mathcal{F} = \{\mathcal{A}_1, \ldots, \mathcal{A}_T\}$ with $\mathcal{A}_1 = \{\Omega, \emptyset\}$, because the first stage contains no

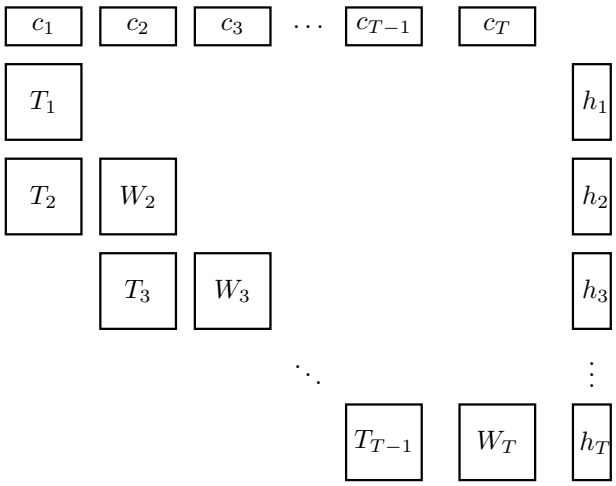

Figure 2.3. Staircase structure of program (2.14).

uncertainty. The decisions taken at stage t thus depend only on outcomes known before or at stage t, they are non-anticipative with respect to the outcomes at stages greater than t.

Program (2.14) is already in *staircase* format, because only adjacent stages are linked via the constraints. The term staircase format follows from the graphical representation of the problem, see Figure 2.3. It may be desirable to form constraints like

$$T_{t-1}(\omega_t)x_{t-2} + T_t(\omega_t)x_{t-1} + W_t(\omega_t)x_t(\omega_t) = h_t(\omega_t), \tag{2.15}$$

to rely not just on decisions taken at stage $t-1$ but also on those taken at earlier stages. This can be done by transforming the non-staircase constraint (2.15) into a staircase constraint with additional columns and rows as follows

$$x_{t-2} = \hat{x}_{t-1} \tag{2.16}$$

$$T_{t-1}(\omega_t)\hat{x}_{t-1} + T_t(\omega_t)x_{t-1} + W_t(\omega_t)x_t(\omega_t) = h_t(\omega_t). \tag{2.17}$$

In a first step, new stage $t-1$ variables are created, $\hat{x}_{t-1}$. These are linked via the constraint (2.16) to take the values of x_{t-2}. The original constraint (2.15) is changed to not include the original variables x_{t-2} but the new variables $\hat{x}_{t-1}$ instead.

It is possible to transform every problem in staircase format with this procedure. Every variable of a stage less than $t-1$ that is present in a non-staircase constraint at stage t needs a stage $t-1$ representation to replace it, see equation (2.17). Every new representation needs a constraint of type (2.16), so that the copy takes the value of the original variable. The number of variables and constraints of the transformed staircase-problem compared to the non-staircase problem is thus increased by at most $\sum_{t=1}^{T-2} n_t \cdot (T-1-t)$.

We stress here that time periods and stages can, but do not have to coincide. It is possible and due to computational considerations probably advisable that a problem with for example 24 time periods is split into six stages with four time periods belonging to each stage.

2.2.4. Basic Properties

In this section we list some basic properties of stochastic programs, in particular properties that are important for developing solution methods.

Stochastic programs can be classified according to which elements of T, W, h and q are fixed, i.e., are the same for every scenario ω. It is possible to exploit the specialized structure in a solution algorithm (Birge & Louveaux, 2011, p. 181ff). The feasibility set of the first stage K_1 is defined as $\{x \mid Ax = b, x \geq 0\}$. The feasibility set of the second stage K_2 is defined as $\{x \mid \mathcal{Q}(x) < \infty\}$. Thus it is possible to reformulate the two-stage stochastic problem given by equations (2.8)-(2.11) in terms of its feasibility sets as

$$\begin{aligned} z = \min \quad & cx + \mathcal{Q}(x) \\ \text{s.t.} \quad & x \in K_1 \cap K_2. \end{aligned} \tag{2.18}$$

The recourse function $Q(\cdot, \xi)$ is convex. It is also polyhedral if there exists a $\bar{x} \in K_1 \cap K_2$, i.e., $Q(\bar{x}, \xi)$ is finite (Shapiro et al., 2009). This is true for both continuous and discrete distributions. The expected recourse function $\mathcal{Q}(x)$ is polyhedral if there exists a $\bar{x} \in K_1 \cap K_2$, i.e., it has a finite value for at least one $x \in K_1$. This result holds under the assumption of finite and discrete distributions. Therefore the DEM (2.12) is a convex problem (see (Walkup & Wets, 1967) for an original proof). These results extend into the multi-stage case (Dupačová, 1995).

A program is said to have *complete recourse* when there exists $y(\omega) \geq 0$, such that $W(\omega)y(\omega) = t$, for every vector $t, t \in \mathcal{R}^{m_2}$. Thus it is guaranteed that a solution can be found for the second stage problem regardless of the actual value of x. A program has *relatively complete recourse* if there is a $y(\omega) \geq 0$, such that $W(\omega)y(\omega) = h(\omega) - T(\omega)x$ for all $x \in K_1$. A program has *fixed recourse*, when $W = W(\omega), \forall\omega$, is deterministic. The recourse function $Q(x, \omega)$ is piecewise-linear and convex for fixed recourse problems, regardless of the distributions (Birge & Louveaux, 2011, p. 109ff).

A question that arises for every decision problem, where the introduction of uncertainty is considered, is the influence and importance of uncertainty for the problem. It has to be kept in mind that "it is extremely difficult to know if randomness is important before we have solved the problem and checked the results" (Kall & Wallace, 1994).

The measures expected value of perfect information (EVPI) and value of the stochastic solution (VSS) give some guidance towards answering the question (cf. (Birge & Louveaux, 2011, p. 163-177)). These measures are based on the solution of several different problems that we introduce first. Let

$$\begin{aligned} z(x, \omega) = \min \quad & cx + \min q(\omega)y(\omega) \\ \text{s.t.} \quad & Ax = b \\ & T(\omega)x + W(\omega)y(\omega) = h(\omega) \\ & x, y(\omega) \geq 0 \end{aligned} \tag{2.19}$$

be the optimization problem associated with one particular outcome ω (Birge & Louveaux, 2011, p. 163f). Let $\bar{x}(\omega)$ denote the optimal solution of problem (2.19), for outcome ω.

The Here-and-Now (HN) problem is another name for the two-stage stochastic program with recourse (2.8) that we can also state as

$$\mathrm{HN} = \min_x E_\xi \left[z(x, \omega) \right]. \tag{2.20}$$

The name derives from the observation that the decision maker, tasked with making a first-stage decision here and now, has to make this decision without knowing how the future will unfold, i.e., which scenario will actually take place. In contrast, the Wait-and-See (WS) problem is the hypothetical problem that the decision maker can make a first-stage decision with perfect foresight. Thus the decision maker can wait and see what happens and make the perfect decision for the revealed uncertainty. The WS problem is defined as

$$\mathrm{WS} = E_\xi \left[\min_x z(x, \omega) \right] = E_\xi \left[z(x(\omega), \omega) \right]. \tag{2.21}$$

Definition 2.14. *The expected value of perfect information is the difference between the objective value of the Wait-and-See and the Here-and-Now problem.*

The EVPI states the maximal amount you should pay a good forecaster on average so that you can adapt your first stage decision to the specific forecast. It measures how much you could gain by possessing perfect information about the future, compared with the solution of the stochastic problem. As it is usually not possible to make good forecasts all the time, the WS solution approach is not implementable in practice.

Solving the corresponding mean value problem instead of the possibly complex stochastic program is an option that could be considered by a decision maker, but that can also come with a cost. The scenario where all random parameters $\xi(\omega)$ are replaced by their expectation is called the expected value scenario and is denoted with $\overline{\omega}$

$$\mathrm{EV} = \min_x z(x, \overline{\omega}), \tag{2.22}$$

where $\bar{x}(\overline{\omega})$ denotes the optimal solution. The solution to this problem is called expected value problem solution (EV solution). This is an implementable solution because it satisfies the first-stage constraints, and it is possible to evaluate it with respect to its second stage cost by optimizing the corresponding second stage problems (2.13). This is called expected result of using the EV solution and is defined as

$$\mathrm{EEV} = E_\xi \left[z(\bar{x}(\overline{\omega}), \omega) \right]. \tag{2.23}$$

Definition 2.15. *The value of the stochastic solution is the difference between the objective value of the Here-and-Now problem and the expected result of using the EV solution.*

The VSS measures the cost of sticking to a deterministic model if stochastic data is available. Of course, to compute the value of the stochastic solution the stochastic problem has to be built and solved first. The relation between WS, HN and EEV is as follows (Birge & Louveaux, 2011, p. 166)

$$WS \leq HN \leq EEV. \tag{2.24}$$

This is intuitively clear, as in the WS problem the optimal first stage decision was taken for every outcome ω. This must be at least as good as the optimal first-stage decision of the stochastic program, i.e., finding a solution where all scenarios are considered together. The HN solution is at least as good as any other feasible first-stage solution for the stochastic program, in particular the EV solution, because it is optimal. The EVPI and the VSS are either equal or greater than zero. This follows from their respective definitions and relation (2.24) (Birge & Louveaux, 2011, p. 167f).

3. Solution Methods

In this chapter we present basic solution methods for stochastic programming problems as defined in the last chapter. To be able to do this, we introduce the notion of scenario trees in Section 3.1. The deterministic equivalent model, which can be solved by traditional LP and MIP optimization software, is then introduced in Section 3.2. We explain the main solution algorithm used in this thesis, Benders decomposition, in depth in Section 3.3. Solution methods based on an alternative decomposition approach, Lagrangean relaxation, are presented in Section 3.4. We finish this chapter with an overview about approximative solution methods and some remarks about scenario generation in Section 3.5. An introduction as well as an in-depth treatment about the different types of decomposition and direct solution methods can be found in the literature, e.g., (Birge & Louveaux, 2011; Kall & Mayer, 2010).

3.1. Scenario Tree

A stochastic program is specified by the deterministic structure like the number of columns and constraints, the objective function coefficients, the matrix coefficients, the right hand side and the variable bounds as well as the stochastic data. The deterministic model is also called the core model. For every scenario the stochastic data consists of coefficients that replace the respective coefficients stored in the core model. A tree structure is well suited to store the stochastic data that is different for every scenario. The scenario tree has a depth equal to the number of stages minus one. The number of leaf nodes is equal to the number of scenarios. The root node of the tree contains no stochastic data, because it represents the first stage. The probability that a certain scenario is realized is stored within the leaf node that corresponds to the scenario.

Every tree node is labeled with its stage $t \in 1, \ldots, T$, with T being the number of stages, and a number from one to K_t, with K_t being the number of nodes in that stage. T also denotes the stage set $\{1, \ldots, T\}$. Every node except the root node has a parent node, denoted by $a(t,i), t \in 1, \ldots, T, i \in 1, \ldots, K_t$. Every node except the leaf nodes has a set of child nodes, denoted by $d(t,i) \subseteq V_{t+1}, t = 1, \ldots, T-1$, with V being the set of all nodes of the tree and V_t the set of nodes at stage t. The *path probability* p_t^i of a node is the sum of the probabilities of its child nodes. For a valid scenario tree the sum of all node probabilities at the same stage must be one. A node has also a conditional probability cp_t^i. It is defined as $cp_t^i = \frac{p_t^i}{p_{t-1}^{a(t,i)}}$, i.e., the probability of node (t,i) given its parent node $(t-1, a(t,i)$ was chosen. For convenience, we denote by $s(t,i) \subseteq S$ the set of scenarios whose path of nodes from the root node to their respective leaf node contains the node (t,i). An example for a scenario tree for a problem with three stages and six scenarios is depicted in Figure 3.1.

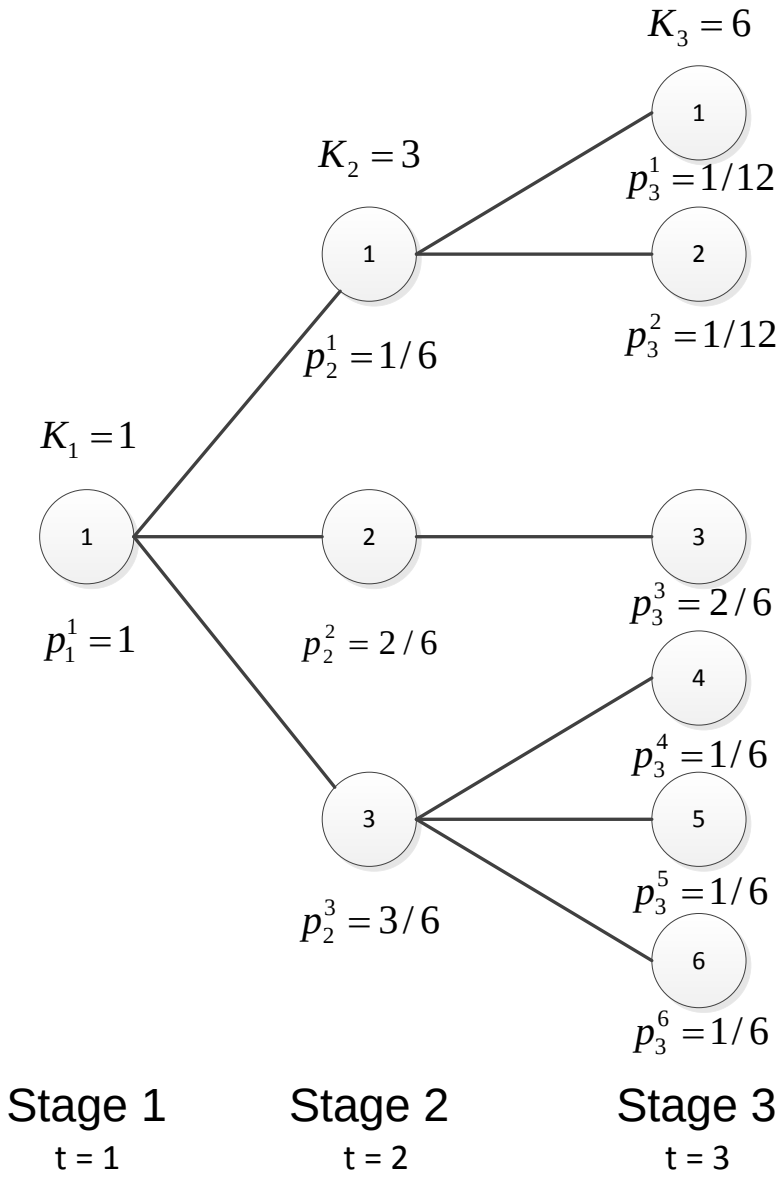

Figure 3.1. Scenario tree with six scenarios and three stages.

3.2. Deterministic Equivalent

The two-stage stochastic problem with recourse (2.8) can be written as a normal deterministic LP called the deterministic equivalent model (DEM) (2.12) or extensive form (EF). The expectation in (2.12) can be replaced by a probability-weighted sum when the random variables have discrete and finite distributions, what we assume throughout this thesis. The set of scenarios is denoted with S which is also the number of scenarios. We can then formulate the DEM as the following large-scale LP

$$\begin{aligned} z = \min_{x,y^1,\dots,y^S} \quad & cx + p^1 q^1 y^1 + \dots + p^S q^S y^S \\ \text{s.t.} \quad & Ax = b \\ & T^1 x + W^1 y^1 = h^1 \\ & \vdots \qquad\qquad \ddots \qquad\quad \vdots \\ & T^S x \qquad\qquad + W^S y^S = h^S \\ & x, y^1, \dots, y^S \geq 0. \end{aligned} \tag{3.1}$$

The LP (3.1) can be solved with state-of-the-art LP solvers, like CPLEX, Gurobi, and others. The widespread availability of modern LP solvers makes this solution approach available without resorting to special purpose software designed especially for stochastic programs. The drawback of this approach is that the special structure of stochastic programs is not used during the solution process. There are existing simplex and interior-point method (IPM) based direct solution methods that work directly with the DEM (see the introduction of Birge & Louveaux (2011, p. 222-236)). These are, to our knowledge, not implemented in commercial LP solvers.

For multi-stage stochastic programs with recourse the deterministic LP formulation is the following

$$\begin{aligned} \min \quad & c_1 x_1 + \sum_{i=1}^{K_2} p_2^i c_2^i x_2^i + \dots + \sum_{i=1}^{K_T} p_T^i c_T^i x_T^i \\ \text{s.t.} \quad & T_1 x_1 = h_1 \\ & T_2^{k_2} x_1 + W_2^{k_2} x_2^{k_2} = h_2^{k_2} \quad k_2 = 1, \dots, K_2 \\ & \ddots \qquad\qquad \ddots \qquad\quad \vdots \\ & T_T^{k_T} x_{T-1}^{a(k_T,T)} + W_T^{k_T} x_T^{k_T} = h_T^{k_T} \quad k_T = 1, \dots, K_t \\ & l_t^{k_t} \leq x_t^{k_t} \leq u_t^{k_t} \qquad k_t = 1, \dots, K_t, t = 1, \dots, T. \end{aligned} \tag{3.2}$$

As can be seen from formulation (3.2) the corresponding columns and constraints from every scenario tree node are added to the deterministic LP. The resulting LP (3.2) is therefore a rather large scale LP which may not even be constructable due to main memory constraints.

The number of variables is equal to

$$\sum_{t=1}^{T} K_t \cdot n_t,$$

whereas the number of constraints is equal to

$$\sum_{t=1}^{T} K_t \cdot m_t.$$

For a two-stage problem with $1,000$ columns (200 first-stage and 800 second stage) and 500 constraints (100 first-stage and 400 second-stage) and 1000 scenarios, the DEM has $800,200$ columns and $400,100$ constraints.

The DEM can be formulated recursively as

$$\begin{aligned} z_t^i\left(x_{t-1}^{a(t,i)}\right) = \min \quad & c_t^i x_t^i + \mathcal{Q}_t^i(x_t^i) \\ \text{s.t.} \quad & T_t^i x_{t-1}^{a(t,i)} + W_t^i x_t^i = h_t^i \\ & l_t^i \leq x_t^i \leq u_t^i, \end{aligned} \tag{3.3}$$

with expected recourse function $\mathcal{Q}_t^i(x) = \sum_{j \in d(t,i)} cp_{t+1}^j Q_{t+1}^j(x_t^i)$ and

$$\begin{aligned} Q_{t+1}^j(x_t^i) = \min \quad & c_{t+1}^j x_{t+1}^j + \mathcal{Q}_{t+1}^j(x_{t+1}^j) \\ \text{s.t.} \quad & T_{t+1}^j x_t^i + W_{t+1}^j x_{t+1}^j = h_{t+1}^j \\ & l_{t+1}^j \leq x_{t+t}^j \leq u_{t+1}^j, \end{aligned} \tag{3.4}$$

and the terminal condition $\mathcal{Q}_T^j(\cdot) = 0, \forall j \in 1, \ldots, K_T$. Problem $z_1^1(\cdot)$ is equivalent to problem (3.2), i.e., starting from the root node.

The formulations (3.1) and (3.2) are also called implicit DEM. The non-anticipativity condition is implicitly satisfied by the variables and constraints of the problem.

Another way to model the DEM is the explicit or split-variable approach. In the explicit DEM a copy of the whole deterministic model is created for every scenario, with the objective function coefficients multiplied by the respective scenario probability. This alone does not suffice, as the different copies have no link to each other, so that all the first-stage variables anticipate their respective scenario and thus yield an optimal decision for this scenario. The solution of this model yields the Wait-and-See solution. To ensure that the scenario copies of the first-stage variables do not anticipate their respective scenario, the non-anticipativity constraints

$$x_t^1 = x_t^i, \forall i \in S \tag{3.5}$$

must be added to the formulation.

When the recourse program is a multi-stage problem, the non-anticipativity constraints must be inserted at every stage, according to the scenario tree structure. For notational convenience, we denote the set of adjacent pairs of child nodes of a node (t,i) with $N(t,i) =$

$\{(s_1, s_2)|s_1, s_2 \in s(t,i) \wedge s_1 + 1 = s_2\}$. For every node of the tree, except the leaf nodes, the following constraints are added to the explicit DEM formulation

$$x_t^{s_1} = x_t^{s_2}, \quad \forall t \in \{1, \ldots, T-1\}, \forall i \in \{1, \ldots, K_t\}, \forall (s_1, s_2) \in N(t,i). \tag{3.6}$$

They ensure that all decisions belonging to nodes with the same parent node take the same value. Of course it is also possible to model these constraints differently, e.g., by using one scenario as the reference scenario (Fourer & Lopes, 2006), as we did for the two-stage case.

For the exemplary scenario tree in Figure 3.1, the following non-anticipativity constraints given by equation (3.6) would be present in the explicit DEM

$$x_1^1 = x_1^2, x_1^2 = x_1^3, x_1^3 = x_1^4, x_1^4 = x_1^5, x_1^5 = x_1^6$$
$$x_2^1 = x_2^2, x_2^4 = x_2^5, x_2^5 = x_2^6.$$

The explicit DEM formulation

$$\begin{aligned}
\min \quad & \sum_{t=1}^{T} \sum_{s=1}^{S} p_s c_t^s x_t^s & & \\
\text{s.t.} \quad & T_1^s x_1^s & = h_1^s & \quad s = 1, \ldots, S \\
& T_2^s x_1^s + W_2^s x_2^s & = h_2^s & \quad s = 1, \ldots, S \\
& \ddots \qquad \ddots & \vdots & \\
& T_T^s x_{T-1}^s + W_T^s x_T^s & = h_T^s & \quad s = 1, \ldots, S \\
x_t^{s_1} = x_t^{s_2}, & t = 1, \ldots, T-1, i = 1, \ldots, K_t, \forall (s_1, s_2) \in N(t,i) & & \\
& l_t^s \leq x_t^s \leq u_t^s & \quad s = 1, \ldots, S, t = 1, \ldots, T,
\end{aligned} \tag{3.7}$$

has even more constraints and variables than the implicit DEM. The number of variables is equal to

$$S \sum_{t=1}^{T} n_t,$$

whereas the number of constraints is equal to

$$\sum_{t=1}^{T} \sum_{i=1}^{K_t} (|s(t,i)| - 1) + S \sum_{t=1}^{T} m_t.$$

3.3. Benders Decomposition

A well known solution method for two-stage stochastic linear programs with recourse is the L-shaped method by (Van Slyke & Wets, 1969), an adaption of Benders decomposition (Benders, 1962) to stochastic problems. The main idea is to approximate the recourse function by an iteratively refined outer linearization via cutting planes. It is also possible to perform an inner linearization via Dantzig-Wolfe decomposition (Dantzig & Wolfe, 1961) that works on the dual problem (see e.g., (Birge & Louveaux, 2011, p. 237-242) for an introduction).

The problem is decomposed by stage into a first-stage master problem and several second-stage subproblems. The first-stage master problem approximates the recourse function with a linear term and delivers an optimal solution for the current approximation. The second-stage subproblems evaluate the chosen first-stage solution for every scenario. With the dual information, the linear approximation is refined and the master problem is resolved. This process repeats until the original problem is solved to optimality. The following detailed description of the Bender's decomposition method applied to stochastic program is based on the work of Freund (2004). It explains the multi-cut form (Birge & Louveaux, 1988) of the algorithm.

The algorithm is used to solve the two-stage stochastic problems with recourse (2.8). The deterministic equivalent formulation (3.1) of the problem can also be written as problem (2.12) with the second-stage problems (2.13) $Q(x,s)$, with $s \in S$. We denote the dual of problem $Q(x,s)$ as $D(x,s)$

$$\begin{aligned} D(x,s) := z(x,s) = \max \quad & \pi^s \left(h^s - T^s x\right) \\ \text{s.t.} \quad & \left(W^s\right)^T \pi^s \leq q^s. \end{aligned} \tag{3.8}$$

The feasible region of $D(\cdot, s)$ is the set

$$\mathcal{D}_s := \left\{ \pi^s \mid (W^s)^T \pi^s \leq q^s \right\},$$

which is independent of x. If the polyhedron is full-dimensional, the extreme points and extreme rays of the feasible region $\mathcal{D}_s$ can be enumerated with $\pi^{s,1}, \ldots, \pi^{s,I_s}$ as extreme points and $r^{s,1}, \ldots, r^{s,J_s}$ as extreme rays (Freund, 2004). If the polyhedron is not full dimensional, it does not have extreme points, but rather "extreme hyperplanes". The polyhedron can still be finitely generated by a set of vectors, where each vector belongs to a different minimal face of $\mathcal{D}_s$, and a set of its extreme rays, as described by Equation (2.7) in Section 2.1.

By the addition of a slack vector to the constraint $(W^s)^T pt^s \leq q^s$, two finite sets of vectors can be defined that fulfill the same goal as the sets of extreme points and extreme rays, namely that they are finite and that the polyhedron can be decomposed into these two sets according to the Minkowski-Weyl decomposition theorem (2.13). The two sets are the set of basic feasible solution of $D(\cdot, s)$ and the set of feasible rays composed of minimal dependent sets of the matrix columns of $W^T I$, see (Zverovich et al., 2012) for details. As the LP solver converts the problem internally into a full-dimensional problem by the addition of slack variables (e.g., (Maros, 2003, p. 4-18)), we continue with the assumption of the full-dimensional case.

If problem $D(x,s)$ is solved, it can either be unbounded or optimal. If the problem is optimal, we get an extreme point of the feasible region as a solution $\bar{\pi}^s = \pi^{s,i}, i \in \{1, \ldots, I_s\}$. As this solution is optimal it holds that

$$z(x,s) = \bar{\pi}^s \left(h^s - T^s x\right) = \max_{k=1,\ldots,I_s} \pi^{s,k} \left(h^s - T^s x\right),$$

and the solution value $z(x, s)$ is thus greater equal than

$$\pi^{s,i}\left(h^s - T^s x\right), i \in \{1, \ldots, I_s\}.$$

If the problem is unbounded, the solver returns an extreme ray $\bar{r}^s = r^{s,j}, j \in \{1, \ldots, J_s\}$. The solution value z^s is therefore ∞ and thus $\bar{r}^s\left(h^s - T^s x\right) > 0$. As long as it holds for any extreme ray $r^{s,j}$ that $r^{s,j}\left(h^s - T^s x\right) > 0$, the second-stage problem $D(x, s)$ is unbounded. Therefore the solution x must be chosen differently to be feasible, as h and T are fixed and determined by the scenario s.

With these two observations we can rewrite $D(x, s)$ in terms of the extreme points and extreme rays of its feasible region $\mathcal{D}_s$ as

$$\begin{aligned} D2(x,s) := z(x,s) = \min \quad & z^s & & \\ s.t. \quad & \pi^{s,i}\left(h^s - T^s x\right) \leq z^s & \quad & i = 1, \ldots, I_s \\ & r^{s,j}\left(h^s - T^s x\right) \leq 0 & & j = 1, \ldots, J_s. \end{aligned} \tag{3.9}$$

A solution $\bar{x}$ that would lead to problem $D(x, s)$ being unbounded is not feasible for problem $D2(x, s)$. Thus $D(\bar{x}, s) = \infty = D2(\bar{x}, s)$ as $D(\cdot, s)$ is a maximization problem. Therefore we can replace $Q(x, s)$ in problem (2.12) by $D2(x, s)$. If we also replace the expectation by the probability weighted sum, we can write this problem, also known as the full master problem (FMP) (Freund, 2004), as

$$\begin{aligned} z = \min_{x, z^1, \ldots, z^S} \quad & c^T x + \sum_{s=1}^{S} p_s z^s & & \\ \text{s.t.} \quad & Ax = b & & \\ & x \geq 0 & & \\ & \pi^{s,i}\left(h^s - T^s x\right) \leq z^s & \quad & i = 1, \ldots, I_s, s = 1, \ldots, S \\ & r^{s,j}\left(h^s - T^s x\right) \leq 0 & & j = 1, \ldots, J_s, s = 1, \ldots, S. \end{aligned} \tag{3.10}$$

If we compare this reformulation with problem (3.1), we see that we removed the second stage variables y_s and the corresponding constraints from the problem, we added S many scalar variables, and a really huge number of constraints. This approach is generally not computationally feasible due to the large number of extreme points and extreme rays of the feasible regions of every second-stage dual subproblem and the resulting number of constraints. The idea is to start with a restricted master problem without any additional constraints and generate the missing constraints when we notice that a not yet added

constraint was violated. The restricted master problem at a given iteration it is formulated like this

$$\begin{aligned} z^{it} = \min_{x,z^1,\dots,z^S} \; & c^T x + \sum_{s=1}^{S} p_s z^s \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \\ & \pi^{s,i}\left(h^s - T^s x\right) \leq z^s \;\; \text{for some } i \text{ and } s \\ & r^{s,j}\left(h^s - T^s x\right) \leq 0 \;\; \text{for some } i \text{ and } s. \end{aligned} \tag{3.11}$$

A solution to problem (3.11) gives us an optimal first-stage solution $\bar{x}, \bar{z}^1, \dots, \bar{z}^S$. The solution value z^{it} is a lower bound to the optimal solution value z of the FMP, as the RMP^{it} misses some constraints that the FMP already has. We now need to check if the given solution is optimal for the FMP. We do this by solving problem (3.8) for $\bar{x}$ and every s. As already described above, if $D(\bar{x}, s)$ has an optimal solution we get an extreme point $\pi^{s,i}$. If $D(\bar{x}, s)$ is unbounded, we get an extreme ray $r^{s,j}$. If it holds that the objective function value of $Q(\bar{x}, s) = D(\bar{x}, s)$, namely $z(\bar{x}, s)$ is greater than the approximating variable $\bar{z}^s$

$$z(\bar{x}, s) = \pi^{s,i}\left(h^s - T^s \bar{x}\right) > \bar{z}^s,$$

than the solution $\bar{x}, \bar{z}^s$ violated the constraint $\pi^{s,i}\left(h^s - T^s x\right) \leq z^s$, and this constraint is then added to RMP^{it}. This constraint is called optimality cut and is usually rearranged to take the form

$$\pi^{s,i} T^s x + z^s \geq \pi^{s,i} h^s.$$

If $D(\bar{x}, s)$ is unbounded, we have an extreme ray $r^{s,j}$. The inequality $r^{s,j}\left(h^s - T^s x\right) > 0$ holds for this extreme ray. The constraint $r^{s,j}\left(h^s - T^s x\right) \leq 0$ is therefore violated and is added to RMP^{it}. This constraint is called feasibility cut.

If all the problems $D(\bar{x}, s)$ have a finite optimal solution, it is possible to compute an upper bound for the original problem (2.12) because every feasible solution $\bar{x}, \bar{y}^1, \dots, \bar{y}^S$ is always greater or equal than the optimal solution. The objective function value of the solution x can be computed as $cx + \sum_{s=1}^{S} p^s q^s y^s$. If this value is lower than the current upper bound, the upper bound can be updated and the solution $\bar{x}, \bar{y}^1, \dots, \bar{y}^S$ can be stored as the incumbent solution. This is repeated until the stopping condition is met. There are three stopping conditions that can be checked. We can stop the algorithm, if no violated constraint could be found, i.e., no cut was added to the RMP^{it}. This means that the found solution is optimal for the FMP and therefore for the original problem. In theory this stopping condition would suffice, but because of numerical inaccuracies in real world computations, it might not be possible to achieve this condition. The algorithm can also be stopped if the gap between the upper and lower bound, $\Delta = UB - LB$, is smaller than a small tolerance $\epsilon_{optimality}$.[1] The third stopping criterion is not an absolute but a relative stopping criterion. When the fraction $\frac{|UB-LB|}{|LB|+10^{-10}}$ is smaller than an $\epsilon_{optimality}$, the

[1] Typical values for $\epsilon_{optimality}$ lie between 10^{-8} and 10^{-5}.

algorithm can also be stopped.[2] This is a common stopping criterion for solvers that work with convergence between upper and lower bounds, e.g., MIP solver.

The algorithm can also be applied in a nested fashion to multi-stage stochastic problems in staircase format (2.14) (Birge, 1985). Every problem associated to an inner node of the scenario tree is then viewed as a subproblem to its ancestor and as a master problem to its descendants. We present a formal description of the nested L-shaped method in Chapter 5.

3.4. Lagrangean Relaxation

A well-known metaheuristic to achieve valid lower bounds for minimization problems with hard and easy constraints is Lagrangean relaxation (see, e.g., (Beasley, 1993; Fisher, 1985) for an introduction). The description of this metaheuristic is relatively simple:

1. Attach Lagrange multipliers to the hard constraints
2. Put these constraints into the objective function
3. Solve the resulting problem
4. Find a feasible solution to the original problem

To apply this metaheuristic to real problems the hard constraints must be identified as well as the values of the Lagrange multipliers. If the obtained solution is not feasible for the original problem, there should be a way to deduce a feasible solution from the solution to the Lagrangean problem (Fisher, 1985). A general method exists to choose values for the Lagrange multipliers, the subgradient method (Held et al., 1974; Fisher, 1981), but choosing better values than those from the subgradient method is possible but problem dependent (Beasley, 1993, p. 280).

The Lagrangean relaxation problem provides a lower bound on the original problem. In case of MIPs, the Lagrangean relaxation lower bound is greater or equal to the lower bound that can be obtained by solving the LP relaxation (Beasley, 1993, p. 253ff). The method can be applied repeatedly with updated Lagrangean multipliers to find the largest bound for the original problem. The problem of finding a set of multipliers that maximize the lower bound obtained by the Lagrangean relaxation problem is called the Lagrangean dual problem (Beasley, 1993, p. 249).

[2] See also the instructions for the POSTS test set (Holmes, 1995).

When we look at the extensive form of the DEM (3.7) we see that it would be possible to decompose this problem into independent scenario subproblems $P_s, \forall s \in S$, if the non-anticipativity constraints (3.6) were not present.

$$\begin{aligned} P_s := \min \quad & \sum_{t=1}^{T} c_t^s x_t^s \\ \text{s.t.} \quad & T_1^s x_1^s & &= h_1^s \\ & T_2^s x_1^s + W_2^s x_2^s & &= h_2^s \\ & \ddots \qquad\qquad \ddots & &\vdots \\ & T_T^s x_{T-1}^s + W_T^s x_T^s &&= h_T^s \\ & l_t^s \le x_t^s \le u_t^s \quad t = 1, \ldots, T. \end{aligned} \tag{3.12}$$

A solution to a particular scenario subproblem P_s (3.12) is relatively easy to find, as this problem is a single scenario problem and as hard to solve as the deterministic version of the problem, thus the hard constraints of problem (3.7) are the non-anticipativity constraints. By putting the non-anticipativity constraints into the objective function with some Lagrange multipliers λ, we get the Lagrangean relaxation of the extensive DEM

$$\begin{aligned} \min \quad & \sum_{t=1}^{T}\sum_{s=1}^{S} p_s c_t^s x_t^s + \sum_{t=1}^{T-1}\sum_{i=1}^{K_t} \sum_{(s_1,s_2)\in N(t,i)} \lambda_{t,s_1,s_2}\left(x_t^{s_1} - x_t^{s_2}\right) \\ \text{s.t.} \quad & T_1^s x_1^s && = h_1^s \quad s = 1, \ldots, S \\ & T_2^s x_1^s + W_2^s x_2^s && = h_2^s \quad s = 1, \ldots, S \\ & \ddots \qquad\qquad \ddots && \vdots \\ & T_T^s x_{T-1}^s + W_T^s x_T^s &&= h_T^s \quad s = 1, \ldots, S \\ & x_t^{s_1} = x_t^{s_2}, t = 1, \ldots, T-1, i = 1, \ldots, K_t, \forall (s_1, s_2) \in N(t,i) \\ & l_t^s \le x_t^s \le u_t^s \quad s = 1, \ldots, S, t = 1, \ldots, T. \end{aligned} \tag{3.13}$$

The Lagrangean relaxation (3.13) of the extensive DEM can be split up into independent subproblems and solved separately. We denote these separate Lagrangean relaxation subproblems as LRP_s

$$\begin{aligned} LRP_s := \min \quad & \sum_{t=1}^{T} c_t^s x_t^s + \lambda_{t,s,s'} x_t^s - \lambda_{t,s',s} x_t^s \\ \text{s.t.} \quad & T_1^s x_1^s & &= h_1^s \\ & T_2^s x_1^s + W_2^s x_2^s & &= h_2^s \\ & \ddots \qquad\qquad \ddots & &\vdots \\ & T_T^s x_{T-1}^s + W_T^s x_T^s &&= h_T^s \\ & l_t^s \le x_t^s \le u_t^s \quad t = 1, \ldots, T, \end{aligned} \tag{3.14}$$

where we set λ_{t,s_1,s_2} to zero, if $(s_1, s_2) \notin N(t, i(t,s))$, with $i(t,s)$ denoting the node index at stage t for scenario s. A naive solution algorithm for the extensive DEM utilizing the

Lagrangean relaxation consists in finding, in an iterative process, Lagrangean multipliers, such that the non-anticipativity constraints holds, i.e., solving the Lagrangean dual.

Due to limited numerical usefulness, the sole use of Lagrangean relaxation does not suffice (Rockafellar & Wets, 1991). The augmented Lagrangean approach (see (Bertsekas, 1982; Rockafellar, 1976a) or (Luenberger & Ye, 2008, Ch. 14)) combines multiplier and penalty methods, where the penalty function is usually a quadratic penalty function, such that the resulting objective function for the Lagrangean relaxed problem of the extensive DEM is

$$\sum_{t=1}^{T}\sum_{s=1}^{S} p_s c_t^s x_t^s + \sum_{t=1}^{T-1}\sum_{i=1}^{K_t} \sum_{(s_1,s_2)\in N(t,i)} \lambda_{t,s_1,s_2}\left(x_t^{s_1} - x_t^{s_2}\right) + \frac{1}{2}\rho \left| \sum_{t=1}^{T-1}\sum_{i=1}^{K_t} \sum_{(s_1,s_2)\in N(t,i)} \left(x_t^{s_1} - x_t^{s_2}\right) \right|^2, \quad (3.15)$$

where the penalty parameter ρ is usually updated during the course of the algorithm. The augmented Lagrangean has the difficulty that the problem formulation cannot be readily decomposed into individual scenario problems, due to the quadratic penalty term in the objective (3.15). This difficulty is addressed in different methods that are based on the augmented Lagrangean, for example Progressive Hedging (PH) (Rockafellar & Wets, 1991), or diagonal quadratic approximation (Mulvey & Ruszczyński, 1992).

A dual decomposition method for stochastic integer problems (Carøe & Schultz, 1999) is based on solving the Lagrangean dual. An introduction to these scenario-based decomposition approaches can be found in the literature, e.g., (Birge & Louveaux, 2011, p. 253ff), (Ruszczyński, 2003, p. 187ff). For the description of a recent implementation of PH see (Watson et al., 2012). For a comparison of different scenario-based decomposition approaches see (Escudero et al., 2012). A recent comparison of self-implemented versions of stage-based and scenario-based decomposition algorithms favors the stage-based decomposition approaches (Parpas & Rustem, 2007).

3.5. Approximative Solution Methods

Continuous distributions for random variables pose a problem for solving stochastic programs, as for an exact solution, integration over multiple dimensions would be required. Discrete distributions for random variables that lead to a large number of realizations can also pose problems, as the number of scenarios can get too large to be handled with the already mentioned solution techniques. In these cases, it is preferable to solve an approximated problem with discrete and finite distributions instead of not solving the problem at all. Usually Monte Carlo sampling techniques are employed in the approximation techniques. For an introduction see (Birge & Louveaux, 2011, p. 389ff) and the references therein.

Approximate solution techniques can be broadly divided into two parts: techniques that employ sampling outside an optimization algorithm and techniques that employ sampling within an optimization algorithm. We look first at the “exterior” methods (Shapiro, 2003).

3.5.1. Exterior Sampling

Sample average approximation (SAA) is one technique to solve stochastic programs with continuous or discrete distributions approximately. The implementation of the method can differ in details regarding the sampling strategy or statistical bound computation, but the overall idea is the following. A set of N realizations for the random vector ξ is sampled, from which we can then deduce a scenario tree. The resulting SAA-problem is an estimator of the expected recourse function of the original problem. Its solution converges to the solution of the true problem, as $N \to \infty$, under mild conditions (cf. Shapiro (2003)). Although the theoretical bound on the sample size is typically too conservative to be of use in practical applications, the sample size required to solve the true problem within a given tolerance probability α depends only logarithmically both on the size of the feasible set and α, for the case of finite feasible sets (Shapiro, 2003, p. 374ff), i.e., when the problem is bounded. When this is not the case, the bound depends linearly on the dimension n of the decision vector x.

The SAA-problem is solved and its solution and objective function value is stored. This process is repeated M times, or until some statistical check is satisfied. After this, the "best" solution is taken and evaluated with a bigger number of scenarios $\bar{N}$. Finally, a confidence interval is computed to specify how "good" the solution is in a statistical sense. Several authors propose improvements to this basic scheme, mostly by altering the way in which the random variates are sampled. Mak et al. (1999) employ common random numbers to achieve variance reduction. Linderoth et al. (2006) use the SAA approach on a set of test problems in a parallel environment and find that latin hypercube sampling is superior to crude monte carlo sampling in terms of solution quality. A good overview about available techniques and an extended introduction is given by Shapiro (2003).

3.5.2. Interior Sampling

"Interior" methods stand in contrast to "exterior" methods in that they do not sample a scenario tree which can then be used by an existing stochastic programming solver. Instead they sample random variates according to the given distribution inside the optimization algorithm. Methods that are based on the L-shaped method are Stochastic Decomposition, proposed by Higle & Sen (1991) and Importance Sampling, proposed by Dantzig & Glynn (1990). For an overview about other procedures, e.g., stochastic quasi-gradient methods, see the literature and the references therein (Birge & Louveaux, 2011, p. 399ff).

Stochastic Decomposition In Stochastic Decomposition Higle & Sen (1991) a new scenario is sampled in every iteration, thus the number of subproblems grows by one with every iteration. All generated scenario subproblems are then solved to optimality. As in the regular L-shaped method it is possible to compute an optimality cut. In addition the cut coefficients of the already present optimality cuts are updated to ensure that no valid optimal point is accidentally cut off. After generating the cut, the master problem is resolved to get a new iterate x. The procedure stops when a statistical stopping condition is satisfied, either by error bounds or optimality conditions. A complete explanation of the

method with implementation considerations and stopping conditions is given by the same authors (Higle & Sen, 1996).

Importance Sampling Importance Sampling is used to reduce the variance of the approximate recourse function value for stochastic programs with random variables with discrete and finite distributions. In contrast to stochastic decomposition, where only one new scenario is sampled at every iteration, here a new scenario tree is sampled at every iteration to get an estimate for the expected recourse function $\mathcal{Q}(\cdot)$ at the current first-stage solution $\hat{x}$. The difference to the crude Monte-Carlo approach is that the scenario tree is not sampled according to the original distribution, but to an altered distribution based on previous results of the recourse function approximation. For h different random variables with respective support $\Omega_i, i = 1, \ldots, h, \Omega = \Omega_1 \times \ldots \times \Omega_h$ a total sampling size N is determined, such that $\sum_{i=1}^{h} N_i = N$, where N_i is the sampling size for the $i-$th random variable. For every random variable i, N_i scenarios are created, such that a random variate is sampled for every random variable $j \neq i, j = 1, \ldots, h$ according to its original marginal distribution, and the $i-$th component is sampled according to its importance distribution. After the creation of the scenarios all subproblems are solved to generate a new optimality cut. The master problem is resolved to get a new lower bound. If the bounds are not indistinguishable by a statistical test, the next set of scenarios is generated. A detailed explanation and analysis of the methods as well as computational results are given by Dantzig & Infanger (1991) and Infanger (1992). An extension of the method was investigated by Dempster & Thompson (1999) by the use of EVPI-based importance sampling.

3.5.3. Scenario Tree Generation

Instead of sampling a scenario tree by variations of Monte Carlo sampling, a scenario tree can also be constructed with other methods. Scenario tree generation is a research area that is important for both theoretical and practical considerations, but peripheral to this thesis. We refer the reader to several surveys (Dupačová et al., 2000; Römisch, 2011) and the good overview of Kaut & Wallace (2003)[3] about techniques that are used to generate scenario trees from either empirical data or distribution information and to the more recent introduction given in Chapter 4 in the book by King & Wallace (2012), in collaboration with Michal Kaut. In (King & Wallace, 2012, p. 83ff) it is also shown how to use out-of-sample and in-sample tests to measure the quality of scenario trees. It is important to distinguish between approximating the distributions of the scenario tree and the decisions that result from such a tree, because a good approximation may not lead to good results. "*We are not concerned about how well the distribution is approximated, as long as the scenario tree leads to a 'good' decision*"(emphasis in original) (Kaut & Wallace, 2003).

Techniques that reduce a given scenario tree in size while staying close to the original distribution of the tree are called tree reduction techniques. They where originally proposed

[3]The overview is missing in the published paper (Kaut & Wallace, 2007).

by Dupačová et al. (2003) and developed further by Heitsch & Römisch (2003), see (Heitsch & Römisch, 2011) for an overview and introduction.

A different approach is proposed by Mirkov & Pflug (2007), where the original distributions are approximated by simpler, discrete distributions but the resulting problem and its result is compared with the original formulation. The distance between the original probability model and its discrete approximation is measured by a conditional transportation distance (Mirkov & Pflug, 2007).

Part II.

State-of-the-Art

4. Benders Decomposition

In this chapter we give an introduction into additions and enhancements to Benders decomposition as it is used in stochastic programming, termed L-shaped method. We explain these concepts for the two-stage case, before we extend the L-shaped method to the multi-stage case in Chapter 5. First, we reformulate the problem introduced in Section 3.3 to a common notation, thereby allowing general LP formulations. In Section 4.2 the concept of cut aggregation is introduced. We present techniques to hinder the typical zig-zagging behavior of cutting plane methods in Section 4.3. Regularized decomposition, level decomposition and trust-region methods are three different methods to achieve that goal. There are different ways to generate the cuts, which we present in Section 4.4. We end this chapter with ways to solve similar subproblems efficiently in Section 4.5.

4.1. Notational Reconcilation

We introduced the L-shaped method in Section 3.3 in the multi-cut variant (Birge & Louveaux, 1988). To be consistent with the literature we replace the aggregate variables z_s we used previously with the commonly used θ^s. In addition we move the probabilities from the objective function to the constraint level, by multiplying all the cut coefficients and the right-hand-side with the scenario probability p^s. We extend the description by allowing general LP formulations (2.3) instead of the standard form (2.1). Due to these changes, we reformulate the master and subproblem formulation given in Section 3.3.

$$\begin{aligned} z^k = \min_{x,\theta^1,\dots,\theta^S} \quad & c^T x + \sum_{s=1}^{S} \theta^s \\ \text{s.t.} \quad & Ax = b \\ & E^{s,i}x + \theta^s \geq e^{s,i} && \forall i \in \mathcal{I}_k \\ & D^{s,j}x \geq d^{s,j} && \forall j \in \mathcal{J}_k, \forall s \in S(j) \\ & l \leq x \leq u. \end{aligned} \tag{4.1}$$

We denote the current solution of z^k as $\bar{x}, \bar{\theta}^1, \dots, \bar{\theta}^S$. At iteration k, the set $\mathcal{I}_k$ consists of all iterations where all subproblems were feasible, whereas the set $\mathcal{J}_k$ consists of all iterations where at least one subproblem was infeasible. Thus the union of $\mathcal{I}_k \cup \mathcal{J}_k$ contains the iteration numbers $1, \dots, k-1$ and $\mathcal{I}_k \cap \mathcal{J}_k = \emptyset$. When the set $\mathcal{I}_k$ is empty, i.e., no optimality cuts are present in the master problem, the variables $\theta^1, \dots, \theta^S$ are ignored in the computation. $S(j)$ denotes the set of scenarios for which feasibility cuts were generated at iteration j.

The subproblem at iteration k for scenario s is formulated as

$$\begin{aligned} Q(\bar{x}, s) = \min_{y^{s,k}} \quad & q^s y^{s,k} \\ \text{s.t.} \quad & W^s y^{s,k} = h^s - T^s \bar{x} \\ & l^s \leq y^{s,k} \leq u^s, \end{aligned} \tag{4.2}$$

whose dual $D(\bar{x}, s)$ is

$$\begin{aligned} D(\bar{x}, s) = \min_{\pi^{s,k}} \quad & \pi^{s,k}(h^s - T^s \bar{x}) + \lambda^{s,k} l^s + \mu^{s,k} u^s \\ \text{s.t.} \quad & (W^s)^T \pi^{s,k} + \lambda^{s,k} + \mu^{s,k} = q^s \\ & \lambda^{s,k} \geq 0, \mu^{s,k} \leq 0. \end{aligned} \tag{4.3}$$

The optimality cut coefficients are computed as

$$E^{s,k} = p^s \pi^{s,k} T^s, \quad e^{s,k} = p^s \pi^{s,k} h^s + p^s \lambda^{s,k} l^s + p^s \mu^{s,k} u^s, \tag{4.4}$$

where $\pi^{s,k}, \lambda^{s,k}$ and $\mu^{s,k}$ denote an optimal solution of problem (4.3) at iteration k. If the current recourse function approximation $\bar{\theta}^s$ was insufficient

$$\bar{\theta}^s < e^{s,k} - E^{s,k} \bar{x},$$

add the optimality cut $E^{s,k} x + \theta^s \geq e^{s,k}$ to the problem, for all $s \in S$ (Birge & Louveaux, 1988).

The feasibility cut coefficients are computed as

$$D^{s,k} = \pi^{s,k} T^s, \quad d^{s,k} = \pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s, \tag{4.5}$$

where $\pi^{s,k}, \lambda^{s,k}$ and $\mu^{s,k}$ denote an extreme ray of problem (4.3). The feasibility cut is then generated as $D^{s,k} x \geq d^{s,k}$ (Birge & Louveaux, 2011, p. 191f). The formalized algorithm can be seen in Algorithm 1.

4.2. Aggregates

The L-shaped method was originally introduced by Van Slyke & Wets (1969) in the single-cut variant. In this variant, when all subproblems are feasible, only one optimality cut is added to the master problem. This cut is computed by summing up all the cut coefficients generated for every scenario at the current iteration i

$$E^i = \sum_{s=1}^{S} E^{s,i}, \quad e^i = \sum_{s=1}^{S} e^{s,i}.$$

The cut is then generated as

$$E^i x + \theta \geq e^i.$$

Initialization;
while $UB - LB > \epsilon_{optimality}$ **do**
  Solve Master problem (4.1) and store solution $x^k, \theta^{1,k}, \ldots, \theta^{S,k}$;
  Set $LB \leftarrow cx^k + \sum_{s=1}^{S} \theta^{s,k}$;
  if *Master problem infeasible* **then**
    return Problem infeasible;
  for *every scenario* $s \in S$ **do**
    Solve second-stage problem $Q(x^k, s)$ (4.2) for scenario s;
    if *Subproblem feasible* **then**
      Let $y^{s,k}$ be the primal solution and $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual solution of $Q(x^k, s)$;
      Generate optimality cut coefficients and right-hand side;
      $E^{s,k} = p^s \pi^{s,k} T^s$;
      $e^{s,k} = p^s \left[\pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s\right]$;
      Form optimality cut $E^{s,k} x + \theta^s \geq e^{s,k}$;
    if *Subproblem infeasible* **then**
      Let $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual extreme ray;
      Generate feasibility cut and add it to Master problem;
      $\pi^{s,k} T^s x \geq \pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s$;
  if *No subproblem was infeasible* **then**
    for *every scenario* $s \in S$ **do**
      if $\theta^{s,k} < e^s - E^s x^k$ **then**
        Add generated optimality cut to Master problem;
  Compute current solution value $z^k = cx^k + \sum_{s=1}^{S} p^s q^s y^{s,k}$;
  if $UB > z^k$ **then**
    Set $UB \leftarrow z^k$ and store incumbent solution $x^k, y^{1,k}, \ldots, y^{S,k}$;
  Set $k \leftarrow k + 1$;
Return incumbent solution;

Algorithm 1: Multi-cut L-shaped method

Birge & Louveaux (1988) come to the conclusion that the multi-cut method typically leads to a considerable reduction in major iterations compared with the single-cut method. They observe that the multi-cut method is more effective than the single-cut method when the number of scenarios is not considerably larger than the number of first-stage constraints. This simple heuristic does not hold for all problems though (Dohle, 2010). Birge & Louveaux (1988) suggest to explore the computational experience with an aggregation level between single- and multi-cut, termed *hybrid approach*. Vladimirou (1998) implements the hybrid approach to decrease communication overhead in a parallel implementation of the algorithm. Linderoth & Wright (2003) also implement the hybrid approach for the same reasons in their asynchronous version of the L-shaped method. Both implementations were not analyzed with respect to the number of aggregates. Recently, Trukhanov et al. (2010) implemented and tested this idea and find that it is indeed superior to single- and multi-cut, at least on their test problems. The set of scenarios S is partitioned into a total of A partitions, also termed *aggregates*, $S^a \subseteq S, a = 1, \ldots, A$ such that $S^i \cap S^j = \emptyset, \forall i \neq j$ and $S^1 \cup \ldots \cup S^A = S$. Every aggregate a is associated with a corresponding free variable θ^a that approximates the expected recourse function for the scenarios in S^a. Thus the optimality cut coefficients are computed as

$$E^{a,i} = \sum_{s \in S^a} E^{s,i}, \quad e^{a,i} = \sum_{s \in S^a} e^{s,i}. \tag{4.6}$$

A cut is then generated as

$$E^{a,i}x + \theta^a \geq e^{a,i},$$

for every aggregate $a = 1, \ldots, A$, if the current recourse function approximation is insufficient, i.e., $\bar{\theta}^a < e^{a,i} - E^{a,i}\bar{x}$. It is easy to see that the single-cut method is an extreme case with only one aggregate $S^1 = S$, and the multi-cut method is the other extreme with S aggregates, where every aggregate S^a consists of exactly one scenario. For completeness we state the master problem for an arbitrary level of cut aggregation A

$$\begin{aligned} z^k = \min_{x,\theta^1,\ldots,\theta^A} \quad & c^T x + \sum_{a=1}^{A} \theta^a \\ \text{s.t.} \quad & Ax = b \\ & E^{a,i}x + \theta^a \geq e^{a,i} \quad a = 1, \ldots, A, \forall i \in \mathcal{I}_k \\ & D^{s,j}x \geq d^{s,j} \quad \forall j \in \mathcal{J}_k, \forall s \in S(j) \\ & l \leq x \leq u. \end{aligned} \tag{4.7}$$

The hybrid-cut method partitions the scenario set, so a partitioning method must be chosen. Trukhanov et al. (2010) propose two different methods to partition the scenario set. The first is the static method, where a fixed number of aggregates A is determined a priori. The second is an adaptive approach that we discuss in Section 8.4.

In the static method, the scenarios are distributed evenly on all aggregates (Trukhanov et al., 2010), in the following way

$$S^a = \left\{ a, A + a, 2 \cdot A + a, \ldots, \left\lfloor \frac{S}{A} \right\rfloor + a \right\}, \forall a = \{1, \ldots, A\}. \tag{4.8}$$

It is also possible to partition the scenarios by dividing the scenario set by the number of aggregates, such that

$$S^a = \left\{ (a-1) \cdot \left\lfloor \frac{S}{A} \right\rfloor + 1, \cdots, a \cdot \left\lfloor \frac{S}{A} \right\rfloor \right\}, \forall a = \{1, \ldots, A\}. \tag{4.9}$$

Should $\frac{S}{A}$ be non-integer, in the first case the remaining $S - \lfloor \frac{S}{A} \rfloor \cdot A$ scenarios are added to the first $S - \lfloor \frac{S}{A} \rfloor \cdot A$ aggregates, whereas in the second case, the last aggregate gets all of the remaining scenarios.

A random partitioning of the scenarios is also possible. This can be done in two different ways. Either a random aggregate can be chosen for every scenario. Or for every aggregate, the belonging scenarios can be chosen at random. The first case can lead to aggregates that are not evenly distributed, although this does not mean that it is in general a bad idea to have uneven partitions.

Brandes (2011) compares the second method with the static approaches above on a small set of test problems and shows that a static selection is in most cases superior to a random partitioning. He also uses clustering methods, in particular k-means and hierarchical clustering, and shows that the number of major iterations can be reduced compared with the random approaches, although the overall running time increases due to the time spend to cluster the scenario set. These results show that the partitioning of the scenarios is a pivotal element in the hybrid-method and should be subject to further research.

4.3. Stabilizing the master problem

Cutting plane methods (Kelley, 1960), like Benders decomposition, have some widely recognized disadvantages (see, e.g., (Vanderbeck & Wolsey, 2010, p. 454f)):

- ineffective initial iterations,
- slow convergence in the end of the algorithm,
- primal solutions can behave erratically, i.e., zigzagging (Zverovich et al., 2012),
- upper bounds can remain stuck for successive iterations, due to multiple solutions.

Some remedies have been proposed to overcome these disadvantages. Techniques to combat zigzagging are described in this section. Ineffective initial iterations are dealt with in Section 5.4, because those techniques are also applied to multi-stage problems.

The series of iterates generated by Benders decomposition can zigzag, which is typical for methods based on single-point linearizations (Birge & Louveaux, 2011, p. 255). This can lead to slow convergence of the algorithm. One aim is therefore to prevent the typical zigzagging generated by Benders decomposition. This can be achieved by additional constraints or an altered objective for the master problem. We will look at three different methods that combat the zigzagging by stabilizing the sequence of iterates, namely regularized decomposition, level decomposition, and the trust region method. These methods belong to the broader class of proximal point (Rockafellar, 1976b) or bundle-type methods (Lemaréchal, 1978; Kiwiel, 1985).

4.3.1. Regularized Decomposition

The regularized decomposition (RD) method was developed by Ruszczyński (1986) and further extended by Ruszczyński & Świętanowski (1997). It uses a quadratic objective function with a penalty term to keep solutions close to the current reference solution $\hat{x}$. In addition, the method keeps only a limited number of cuts in the problem (see (Ruszczyński, 2003) for a detailed description and proof of convergence). Regularized decomposition uses the multi-cut method by default, but it can be adapted to an arbitrary number of aggregates. The master problem has the following form

$$\begin{aligned} \min_{x,\theta^1,\dots,\theta^S} \quad & c^T x + \sum_{s=1}^{S} \theta^s + \frac{1}{2\sigma} \parallel \hat{x}^k - x \parallel^2 \\ \text{s.t.} \quad & Ax = b \\ & E^{s,i} x + \theta^s \geq e^{s,i} \qquad \forall i \in \mathcal{I}_k \\ & D^{s,j} x \geq d^{s,j} \quad \forall j \in \mathcal{J}_k, \forall s \in S(j) \\ & l \leq x \leq u. \end{aligned} \tag{4.10}$$

To describe the update of σ and $\hat{x}^k$ we let $F^k = c^T x + \sum_{s=1}^{S} \theta^s$ be the objective function value of the approximating master problem at iteration k and $F(x) = c^T x + \sum_{s=1}^{S} p_s Q(x, s)$ be the objective function value of the stochastic program at point x. The reference solution for the next iteration $\hat{x}^{k+1}$ is updated during the algorithm with the current solution x^k, if

$$F(x^k) = F^k \text{ or } F(x^k) < F(\hat{x}^k),$$

otherwise it stays the same, i.e., $\hat{x}^{k+1} = \hat{x}^k$. In addition to the second condition, exactly $n + S$ cuts must be active (Ruszczyński & Świętanowski, 1997). The penalty parameter σ is initialized with one and adjusted during the course of the algorithm. It is doubled, when

$$F(x^k) < (1 - \gamma) F(\hat{x}^k) + \gamma F^k$$

and halved, if

$$F(x^k) > \gamma F(\hat{x}^k) + (1 - \gamma) F^k,$$

with $\gamma \in (0, 1)$ being a fixed parameter, usually set to 0.9. When σ is doubled, the quadratic penalty term becomes less important, because the coefficient is $\frac{1}{2\sigma}$, thus the algorithm is allowed to find solutions that are further away from the current reference point $\hat{x}^k$.

The algorithm needs a different stopping criterion than the original L-shaped method, because the algorithm does not provide a global lower bound, thus the convergence of the bounds does not work. Instead the algorithm stops, when $F^k = F(\hat{x}^k)$, i.e., the approximation is equal to the solution value at point $\hat{x}^k$ and thus $\hat{x}^k$ is an optimal solution. This might be challenging for problems with numerical difficulties, see (Zverovich et al., 2012) for a remark. The equality test can be implemented as $\frac{|F(x^k) - F^k|}{|F(x^k)| + \epsilon}$ for a small ϵ, e.g., $\epsilon = 10^{-10}$, see (Ellison et al., 2012, p. 34).

Computational results can be found in (Ruszczyński & Świętanowski, 1997), a comparison with other methods was done by (Zverovich et al., 2012). The method can be extended

to the multi-stage case (Ruszczyński, 1993a). Regularized decomposition was also implemented by Vladimirou (1998), who experimented with different values for the parameter γ. He finds other values for the update of the penalty parameter σ more appropriate than doubling or halving.

4.3.2. Level Decomposition

Level decomposition is another method to stabilize the sequence of iterates, first proposed as the Level method by Lemaréchal et al. (1995) in the general context of nonsmooth convex optimization and adapted to stochastic programming as Level decomposition by Fábián & Szőke (2007). A favorable computational assessment of the method in comparison to Benders decomposition, regularized decomposition and the trust-region method was done recently (Zverovich et al., 2012). The method uses the single-cut variant in its default description, but an arbitrary number of aggregates is possible. Oliveira & Sagastizábal (2012) classify different versions of the bundle, level, and proximal point methods in a unifying framework, and propose the on-demand accuracy approach. This approach allows to skip solving the second stage problem by adding on-demand accuracy cuts out of stored dual information. Fábián (2013) presents the on-demand accuracy approach in a form which shows that this approach, if applied to two-stage stochastic programming problems, combines the advantages of the disaggregate and the aggregate models.

The purpose of the level method is to dampen the zigzagging effect of the L-shaped method with respect to first-stage solutions. This is achieved by projecting the current iterate, denoted by x^k, to a certain level set of the model function and by solving the subproblems with the projected solution instead of the current solution. This requires a starting solution x^0 in the first iteration, e.g, the EV solution. The current incumbent solution is denoted with x^*, i.e., the solution that achieved the current upper bound. The projection problem is formed as follows

$$\begin{aligned}
\min_{x} \quad & \| x - x^k \|^2 & & \\
\text{s.t.} \quad & Ax = b & & \\
& E^i x + \theta \geq e^i & & \forall i \in \mathcal{I}_k \\
& D^{s,j} x \geq d^{s,j} & & \forall j \in \mathcal{J}_k, \forall s \in S(j) \\
& c^T x + \theta \leq (1-\lambda) F^k + \lambda F(x^*) & & \\
& l \leq x \leq u. & &
\end{aligned} \tag{4.11}$$

It minimizes the euclidean distance of its solution x to the current iterate x^k, while ensuring that the approximated solution value $c^T x + \theta$ is not greater than a convex combination of the current lower and upper bound, i.e., a certain level set of the model function. This condition is ensured via the constraint $c^T x + \theta \leq (1-\lambda) F^k + \lambda F(x^*)$. The parameter $\lambda \in (0,1)$ is fixed a priori and kept constant throughout the algorithm. Note that for $\lambda = 0$ the Level method behaves like the original L-shaped method. The solution x is used as the next iterate, x^{k+1}. The subproblems (4.2) are then solved and a new cut is added to the master problem. The algorithm stops, when the global lower and upper bound have converged.

4.3.3. Trust-Region Method

The trust-region method of Linderoth & Wright (2003), build upon bundle-trust-region approaches (see the references in (Linderoth & Wright, 2003)), uses the l_∞ norm to bound available solutions. Thus it can be thought of as a hypercube around a reference solution x^k in which the next iterate is to be found, respecting the usual L-shaped constraints. The method divides the iterations into major and minor iterations, where the reference solution x^k stays the same for all minor iterations that follow a major iteration. It is defined for the multi-cut variant, but it can be adapted to an arbitrary number of aggregates.

The problem solved at minor iteration l and major iteration k is

$$\begin{aligned}
\min_{x^{k,l}} \quad & c^T x^{k,l} + \sum_{s=1}^{S} \theta^{s,k,l} \\
\text{s.t.} \quad & Ax^{k,l} = b \\
& E^{s,i} x^{k,l} + \theta^s \geq e^{s,i} && \forall i \in \mathcal{I}_k \\
& D^{s,j} x^{k,l} \geq d^{s,j} && \forall j \in \mathcal{J}_k, \forall s \in S(j) \\
& -\Delta^{k,l}\mathbf{1} \leq x^{k,i} - x^k \leq \Delta^{k,l}\mathbf{1} \\
& l \leq x \leq u,
\end{aligned} \tag{4.12}$$

where $\Delta^{k,l}$ is the current trust-region bound and $\mathbf{1}$ is a vector of appropriately many ones. The constraint $-\Delta^{k,l}\mathbf{1} \leq x^{k,l} - x^k \leq \Delta^{k,l}\mathbf{1}$ is the LP formulation of $\| x^{k,l} - x^k \|_\infty \leq \Delta^{k,l}$. A minor iteration leads to a new major iteration, when the solution value $F(x^{k,l})$ was reduced enough, i.e., $F(x^{k,l}) \leq F(x^k) - \epsilon(F(x^k) - F^{k,l})$, for $\epsilon \in (0, \frac{1}{2})$. The solution $x^{k,l}$ is then used as the next reference solution x^{k+1}. The trust-region bound $\Delta^{k,l}$ is also updated during the algorithm. The goal is to allow wide steps, when it helps in finding good solutions in a wider vicinity of the current reference point, and to restrict the set of available solutions with a tighter bound, when no good solutions can be found within the current bound.

The trust-region method increases the bound, when the solution value $F(x^{k,l})$ was reduced sufficiently, e.g., $F(x^{k,l}) \leq F(x^k) - 0.5(F(x^k) - F^{k,l})$ in a minor iteration. It is gradually reduced, when the solution value $F(x^{k,l})$ is not reduced for some consecutive minor iterations. For implementation details, computational results and a convergence proof see (Linderoth & Wright, 2003). The authors also propose a cut deletion scheme and an asynchronous version for more effective parallelization.

4.4. Cut Generation

The textbook approach to the L-shaped method cut generation is to generate only one feasibility cut, whenever a subproblem is infeasible, and then resolve the master problem (Birge & Louveaux, 2011, p. 191). This can be altered by generating more than one feasibility cut in the same iteration, i.e., by solving all subproblems. It is also possible to generate both optimality and feasibility cuts at the same time, when some subproblems are infeasible and some can be solved to optimality, but the effects have not been evaluated for stochastic programming problems.

In the context of integer programming, Fischetti et al. (2008), propose new cut selection rules for feasibility cuts by finding a minimal infeasible subsystem. It is refined and compared favorably with the classical cut selection by Benders decomposition, with a unified framework for feasibility and optimality cuts (Fischetti et al., 2010).

The use of inexact optimality cuts, generated from a feasible but not yet optimal solution to the dual subproblem (4.3), was also proposed for Benders decomposition (Zakeri et al., 2000). Fábián (2000) suggested a similar idea related to the level method. A dual feasible solution to the primal subproblem results in a valid but possibly not very strong cut for the master problem. For the correct computation of the upper bound, all problems must be solved to optimality, e.g., to test if the bounds have converged.

To solve a problem within a given accuracy of the optimal solution, some kind of primal-dual optimization algorithm is necessary, as the simplex method does not provide a way of estimating the gap between the current solution and the optimum. As of today, no commercial interior-point solver has the possibility of an efficient warm start procedure (see (Gondzio, 1998) for warmstarting IPM), thus the usage of an IPM method to solve subproblems repeatedly seems unwarranted. Therefore the usage of inexact cuts, as proposed by Zakeri et al. (2000), is not applicable for simplex-based implementations of the L-shaped method. The inexact level decomposition algorithm developed by Fábián & Szőke (2007) refines the distribution approximation[1] during the course of the algorithm. If the distribution is refined, more scenario subproblems are present. Even if all subproblems are solved exactly, Fábián & Szőke observe that the generated cuts from previous iterations are inexact with respect to the new information. Fábián & Szőke (2007) also allow for the subproblem solution runs to be inexact.

Closely connected with the process of cut generation is the removal of inactive cuts. Inactive cuts make the master problem larger than it has to be, which has implications for solution time and needed memory. It is noticed that there is no reliable way to delete cuts from the master problem (Ruszczyński, 2003), which is common for cutting-plane methods. Cut deletion techniques are nevertheless important to prevent proliferation of unnecessary cuts, especially in the multi-cut case. Regularized decomposition allows to delete cuts and keep the overall number of cuts bounded, the trust-region method also allows to delete cuts from the master problem, but in a rather heuristic approach. Level decomposition also allows to remove cuts after certain iterations (see (Fábián & Szőke, 2007)).

4.5. Solving Similar Subproblems

In the L-shaped method a lot of subproblems have to be solved in every iteration. Depending upon which components of (T, W, q, h) are stochastic, the problems are very similar to one another. This can be exploited by the used solution algorithm, usually the simplex method. The most important technique is the simplex warm start capability. When we recall the dual subproblem formulation (4.3), we see that the feasibility set depends on W and q. If these two are deterministic, every basic solution of (4.3) is feasible for every other dual subproblem. The simplex method can warm start from another basic solution

[1] Fábián & Szőke (2007) observe that this strategy is possible when the number of random parameters is not large.

and find the optimum in just a few iterations. Thus when the dual simplex is used to solve problem (4.2), a dual feasible basic solution can be used to warm start the dual simplex method on that problem. Reasonable candidates are the basic solution of the last iteration of that subproblem, or another basic solution of some other subproblem solved at the same iteration. Which method works best seems to be problem dependent (Dohle, 2010), although Vladimirou (1998) argues on empirical grounds that the use of the optimal basis of the last iteration seems to be the better choice.

Bunching methods are used (see (Ermoliev, 1988, p. 83ff) for a detailed introduction) to find bases that are optimal for several subproblems. It is implemented in a variant that can also handle random W and q in the multi-stage solver MSLiP (Gassmann, 1990). The sifting procedure (Garstka & Rutenberg, 1973) is used in another multi-stage code (Birge, 1985), but it can only handle random right-hand-side h. These techniques require more involved implementation than just passing a stored basis into an LP solver (see (Birge & Louveaux, 2011, p. 217ff)). Morton (1996) finds that a heuristic selection of bases, depending on knowledge of the underlying problem, can be advantageous compared with simple basis reuse. The parallelization of the second stage computations is done with a good speedup factor, as all subproblems are independent of one another. This is demonstrated by empirical evaluations (Ariyawansa & Hudson, 1991; Wolf & Koberstein, 2013).

5. Nested Benders Decomposition

The Benders decomposition method, developed for two-stage problems, can be extended to handle multi-stage problems (Birge, 1985). A nested application of the method, where two adjacent stages are seen as the first and second stage respectively, is possible. Every problem associated to an inner node in the scenario tree is at the same time a master problem towards its child nodes and a subproblem towards its parent node. All the master/subproblems can therefore contain feasibility and optimality cuts, and these must be considered in the cut generation process.

In contrast to the two-stage case, the algorithm has to decide upon a *direction*. After all problems at stage $t, 1 < t < T$, have been solved, which stage should be solved next? Is it better to go *forward* towards the leaf nodes, i.e., push primal information in the form of solutions down the tree, or go *backward* towards the root node, i.e., push dual information in the form of cuts up the tree? Sequencing protocols, which decide this question, are described in Section 5.2. We give an overview about parallelization approaches for the nested application of Benders decomposition in Section 5.3, as these vary considerably more than those for the two-stage case.

Advanced start procedures try to take advantage of existing solutions. They are described in Section 5.4. In Section 5.5 we introduce stage aggregation techniques. They are used to transform an existing problem into a problem with less stages, by combining adjacent stages. The goal is to reduce the computing time. But first, we describe the nested L-shaped method and the differences to the two-stage case in the following section (see (Birge & Louveaux, 2011, p. 266-276) for a detailed explanation).

5.1. Nested L-shaped method

The problem $P_t^v(x_{t-1}^{a(t,v)})$ that has to be solved at iteration k for a given node v at stage t is formulated as

$$
\begin{aligned}
z_t^{v,k} = \min_{x_t^v, \theta_t^{v,1}, \ldots, \theta_t^{v,A_t^v}} \quad & c_t^v x_t^v + \sum_{a=1}^{A_t^v} \theta_t^{v,a} \\
\text{s.t.} \quad & T_t^v x_{t-1}^{a(t,v)} + W_t^v x_t^v = h_t^v \\
& E_t^{v,a,i} x_t^v + \theta_t^{v,a} \geq e_t^{v,a,i} \qquad \forall i \in \mathcal{I}_{t,k}^v, \forall a \in A_t^v \\
& D_t^{v,s,j} x_t^v \geq d_t^{v,s,j} \quad \forall j \in \mathcal{J}_{t,k}^v, \forall s \in S_t^v(j), \\
& l_t^v \leq x_t^v \leq u_t^v.
\end{aligned}
\tag{5.1}
$$

The first-stage problem is denoted as P_1^1, but it has no prior solution and thus no $T_0^{a(1,1)}$ matrix. As in the two-stage case, the θ variables are only considered in computation, when

a corresponding optimality cut has been added to the problem. The problems associated with the leaf nodes at stage T have neither optimality nor feasibility cuts. Thus they have no θ variables. The dual problem $D_t^v(x_{t-1}^{a(t,v)})$ to $P_t^v(x_{t-1}^{a(t,v)})$ is

$$\begin{aligned} -z_t^{v,k} = \max \quad & \pi_t^{v,k}(h_t^v - T_t^v x_{t-1}^{a(t,v)}) + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v + \sigma_t^{v,k} e_t^{v,k} + \rho_t^{v,k} d_t^{v,k} \\ \text{s.t. } & (W_t^v)^T \pi_t^{v,k} + (E_t^{v,k})^T \sigma_t^{v,k} + (D_t^{v,k})^T \rho_t^{v,k} = c_t^v \\ & \sum_{i \in \mathcal{I}_{t,k}^v} \sigma_t^{v,a,i} = 1 \quad \forall a \in A_t^v \\ & \lambda_t^{v,k} \geq \mathbf{0}, \mu_t^{v,k} \leq \mathbf{0}, \sigma_t^{v,k} \geq \mathbf{0}, \rho_t^{v,k} \geq \mathbf{0}, \end{aligned} \tag{5.2}$$

where

$$E_t^{v,k} = \bigotimes_{i \in \mathcal{I}_{t,k}^v} \bigotimes_{a \in A_t^v} E_t^{v,a,i}, D_t^{v,k} = \bigotimes_{j \in \mathcal{J}_{t,k}^v} \bigotimes_{s \in S_t^v(j)} D_t^{v,s,j}, e_t^{v,k} = \bigotimes_{i \in \mathcal{I}_{t,k}^v} \bigotimes_{a \in A_t^v} e_t^{v,a,i},$$
$$d_t^{v,k} = \bigotimes_{j \in \mathcal{J}_{t,k}^v} \bigotimes_{s \in S_t^v(j)} d_t^{v,s,i}, \sigma_t^{v,k} = \bigotimes_{i \in \mathcal{I}_{t,k}^v} \bigotimes_{a \in A_t^v} \sigma_t^{v,a,i}, \rho_t^{v,k} = \bigotimes_{j \in \mathcal{J}_{t,k}^v} \bigotimes_{s \in S_t^v(j)} \rho_t^{v,s,j}.$$

$\bigotimes$ denotes row-wise concatenation of row vectors, i.e., the result is a matrix.

The cut generation is different to the two-stage case as the present optimality and feasibility cuts must be considered when optimality and feasibility cuts are generated for all problems in the middle of the tree, i.e., at stages t with $1 < t < T$. The optimality cut coefficients are computed as (Birge & Louveaux, 2011, p. 268)

$$E_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \pi_t^{v,k} T_t^v. \tag{5.3}$$

The right hand side of an optimality cut is computed as (Birge & Louveaux, 2011, p. 268)

$$e_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \left[\pi_t^{v,k} h_t^v + \sigma_t^{v,k} e_t^{v,k} + \rho_t^{v,k} d_t^{v,k} + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v \right]. \tag{5.4}$$

When all nodes of stage t are solved to optimality, optimality cuts can be computed from the precomputed coefficients (5.3) and right-hand sides (5.4) as usual. To describe such a cut for the ancestor node $a(t,v)$ at stage $t-1$, we switch to this node to describe the cut in its own terms, i.e., the ancestor node $a(t,v)$ is now denoted as node (t,v). The final cut for an aggregate $a \in A_t^v$ is then computed as

$$E_t^{v,a,k} x_t^v + \theta_t^{v,a} = \sum_{s \in S_t^{v,a}} E_t^{v,s,k} x_t^v + \theta_t^{v,a} \geq \sum_{s \in S_t^{v,a}} e_t^{v,s,k} = e_t^{v,a,k}, \tag{5.5}$$

where $S_t^{v,a} \subseteq d(t,v)$ is the partition of the child nodes $d(t,v)$ for aggregate a. The final cut coefficients are stored as $E_t^{v,a,k}$ and the right-hand side as $e_t^{v,a,k}$. When all cuts have been computed, the intermediate results $E_t^{v,s,k}, e_t^{v,s,k}, \forall s \in d(t,v)$ are deleted, as they are no longer needed. Note that the intermediate results $E_t^{v,s,k}$ are indexed with a node

index $s \in d(t,v)$ whereas the final coefficients $E_t^{v,a,k}$ are indexed with an aggregate index $a \in \{1,\ldots,A_t^v\}$.

Feasibility cuts from a node (t,v) for its ancestor are generated as

$$D_{t-1}^{a(t,v),v,k} x_{t-1}^{a(t,v)} \geq d_{t-1}^{a(t,v),v,k}, \tag{5.6}$$

where the feasibility cut coefficients are computed as

$$D_{t-1}^{a(t,v),v,k} = \pi_t^{v,k} T_t^v x_{t-1}^{a(t,v)}, \tag{5.7}$$

and the right hand side is computed as (Birge & Louveaux, 2011, p. 267)

$$d_{t-1}^{a(t,v),v,k} = \pi_t^{v,k} h_t^v + \rho_t^{v,k} d_t^{v,k} + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v. \tag{5.8}$$

The pseudocode is shown in Algorithm 2 (see (Birge & Louveaux, 2011, p. 267ff) for proof of correctness).

5.2. Sequencing Protocols

The textbook algorithm we described in Algorithm 2 calls a sequencing protocol that decides the direction. At every stage $1 < t < T$ the choices are forward or backward. In the first implementation of the algorithm (Birge, 1985) the employed protocol was *FastForward (FF)*. This means that the algorithm goes forward in stage t, when the approximation of the recourse function at stage t is not good enough, i.e., when new optimality cuts have been added to stage t after the problems at stage $t+1$ have been solved. The opposing choice is *FastBack (FB)*, where the algorithm goes backward at stage t, when the recourse function approximation at stage $t-1$ is not good enough. These are the extreme choices. FF may take a lot of time to build a good recourse function approximation for an initial solution which was far away from the optimum. The FB algorithm may take a lot of time to build a good recourse function approximation at the earlier stages without taking into account the recourse function approximation at later stages, which may be rather poor due to the missing dual information (see (Morton, 1996) for a further discussion).

A balanced approach is the *FastForwardFastBack (FFFB)* or *Fastpass* protocol, originally proposed by (Wittrock, 1983) (cf. (Gassmann, 1990)). It makes a whole forward sweep, followed by a whole backward sweep. This means it pushes the current primal information as much down the tree as possible, and then pushes the resulting dual information back to the root node. If a problem at a stage is infeasible, all the protocols do a backward sweep to get a new primal solution. This scheme is currently believed to be the most efficient (Birge & Louveaux, 2011, p. 268). With respect to the other protocols, the literature is undecided. Gassmann (1990) concludes that FF is faster than FB, whereas (Birge et al., 1996) and (Morton, 1996) come to the conclusion that FB fares better than FF. These results depend on the investigated problems.

Initialization;
while $UB - LB > \epsilon_{optimality}$ **do**
  for *every node* v *at stage* t **do**
    Solve subproblem $P_t^v(x_{t-1}^{a(t,v),k})$;
    if *Subproblem feasible* **then**
      Let $x_t^{v,k}, \theta_t^{v,1,k}, \ldots, \theta_t^{v,A_t^v,k}$ be the primal solution;
      Let $\pi_t^{v,k}, \sigma_t^{v,k}, \rho_t^{v,k}, \lambda_t^{v,k}, \mu_t^{v,k}$ be the dual solution;
      Generate Optimality-Cut Coefficients;
      $E_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \pi_t^{v,k} T_t^v$;
      $e_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \left[\pi_t^{v,k} h_t^v + \sigma_t^{v,k} e_t^v + \rho_t^{v,k} d_t^v + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v\right]$;
    if *Subproblem infeasible* **then**
      Let $\pi_t^{v,k}, \sigma_t^{v,k}, \rho_t^{v,k}, \lambda_t^{v,k}, \mu_t^{v,k}$ be the dual extreme ray;
      Generate feasibility cut and add it to parent problem;
      $\pi_t^{v,k} T_t^v x_{t-1}^{a(t,v)} \geq \pi_t^{v,k} h_t^v + \rho_t^{v,k} d_t^v + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v$;
  if $t = 1$ **then**
    Set $LB \leftarrow c_t^1 x_t^{1,k}$;
  else if $t = T$ *AND all subproblems are feasible* **then**
    Compute current solution value $z^k = \sum_{t=1}^{T} \sum_{v=1}^{K_t} p_t^v q_t^v x_t^{v,k}$;
    if $UB > z^k$ **then**
      Set $UB \leftarrow z^k$ and store incumbent first stage solution $x_1^{1,k}$;
  Call sequencing protocol to set direction;
  if $t > 1$ *AND direction = backward AND all subproblem are feasible* **then**
    Generate all optimality cuts of the form (5.5) for all nodes at stage $t-1$;
    Add all generated optimality cuts to their respective problem;
  if $t = 1$ **then**
    Set $k \leftarrow k + 1$;
  if $direction = forward$ **then**
    $t = t + 1$;
  else
    $t = t - 1$;
Return incumbent solution;

Algorithm 2: Hybrid-cut nested L-shaped method

Morton (1996) introduces two new protocols; ϵ-variants of FF and FB[1]. These two protocols look at the lower and upper bound for every stage, and move in the opposite direction, if the gap between the bounds is smaller than the ϵ threshold. The concepts *Absolute Error* and *Discrepancy* (Morton, 1996) need to be described first, as the protocols use these concepts.

We define Absolute Error (AE) for a whole stage. This implies that the concept can only be meaningfully defined, if all subproblems for a stage $\hat{t}$ are feasible. Absolute Error is the difference between the expected recourse function approximation at stage $\hat{t}$ compared to the actual expected recourse function value,

$$AE(\hat{t},k) = \sum_{t=\hat{t}+1}^{T} \sum_{v=1}^{K_t} p_t^v c_t^v x_t^{v,k} - \sum_{v=1}^{K_t} \sum_{a=1}^{A_t^v} p_{\hat{t}}^v \theta_{\hat{t}}^{v,a,k}. \tag{5.9}$$

To be able to determine the Absolute Error for a stage $\hat{t}$, the expected recourse function must be evaluated for the current primal solutions. This is done by solving all stages $t > \hat{t}$.

Discrepancy (Disc) is the difference between the expected recourse function approximation at a stage $\hat{t}$ compared to the weighted objective function value and the expected recourse function approximation at stage $\hat{t}+1$,

$$Disc(\hat{t},k) = \sum_{v=1}^{K_{\hat{t}+1}} \left(p_{\hat{t}+1}^v c_{\hat{t}+1}^v x_{\hat{t}+1}^{v,k} + \sum_{a=1}^{A_{\hat{t}+1}^v} p_{\hat{t}+1}^v \theta_{\hat{t}+1}^{v,a,k} \right) - \sum_{v=1}^{K_{\hat{t}}} \sum_{a=1}^{A_{\hat{t}}^v} p_{\hat{t}}^v \theta_{\hat{t}}^{v,a,k}. \tag{5.10}$$

In other words, discrepancy measures the "goodness" of the expected recourse function approximation with respect to the next stage. To compute the discrepancy for stage $\hat{t}$, only the next stage $\hat{t}+1$ must be solved.

Numerical difficulties, which arise in practice, require a slight change with respect to the original description, when implementing these protocols. This holds in particular for FF and FB (cf. (Morton, 1996) with (Birge et al., 1996)). It is advisable to change the strict condition that the algorithm has to wait until no more optimality cuts can be generated before the direction can be changed, towards a condition that uses tolerances and the gaps between lower and upper bounds instead. Typical values for such tolerances lie between 10^{-4} and 10^{-6}. Therefore we describe the FF and FB protocols with tolerances, instead of the original condition that no more cuts could be generated. The absolute gap $|UB - LB|$ is used as a threshold. For comparison, Morton (1996) uses the minimum of the absolute values of global upper and lower bounds as a threshold. This can lead to problems for the FB protocols if the upper bound of a problem becomes zero, and the threshold condition can not be satisfied anymore due to the requirement that the discrepancy has to be lower than zero.

To describe all protocols in a unified way, the different sequencing protocols are called from a general protocol. Algorithm 3 describes the general part that is common for all sequencing protocols, i.e., a direction change at the first and last stage, and a backward

[1]Morton (1996) denotes FF as *shuffle* and FB as *cautious*.

sweep if any subproblem was infeasible. If no specific protocol is called, it is identical to the FFFB protocol.

if $t = 0$ **then**
  $direction = forward$;
  $sweep = false$;
else if $t = T$ **then**
  $direction = backward$;
  $sweep = false$;
if *Subproblem infeasible* **then**
  $direction = backward$;
  $sweep = true$;
if $sweep = true$ **then**
  return;
else if $1 < t < T$ **then**
  Call specific protocol;

Algorithm 3: General sequencing protocol

The FastForward protocol, described in Algorithm 4, is then called from the general protocol, as well as all other protocols for that matter.

if $direction = forward$ **then**
  $sweep = true$;
else if $AE(t,k) < \epsilon_{optimality} \cdot |UB - LB|$ **then**
  $direction = backward$;
else
  $direction = forward$;

Algorithm 4: FastForward sequencing protocol

The opposite protocol, FastBack, is described in Algorithm 5. The FB protocol requires a valid recourse function approximation for every node. This is checked in the first if clause.

The FastForwardFastBack protocol is described in Algorithm 6. The direction is not changed in intermediate stages, except for the case of infeasible subproblems. It does not add anything to the generic protocol given in Algorithm 3. To be consistent in the description of sequencing protocols, we include it for the sake of completeness and to emphasize that FFFB does not take any information about the current bounds into account.

The $\epsilon-$FF protocol decides to go back, if the Absolute Error is less than $\epsilon \cdot |UB - LB|$.

The $\epsilon-$FB protocol goes forward, if the Discrepancy is less than $\epsilon \cdot |UB - LB|$.

If the protocols FF or ϵ-FF are used, additional care has to be taken with respect to iteration counts. The iteration counts used so far, also in the description of the nested L-shaped method in Algorithm 2, are used to index solutions as well as cut components. With FF, $\epsilon-$FF, or any other protocol that can go forward after it went previously backward during the same iteration, this numbering does not suffice, as the same iteration count is

if *no expected recourse function approximation for any stage* $t < T$ **then**
  $direction = forward$;
  $sweep = true$;
  return;
if $direction = backward$ **then**
  $sweep = true$;
else if $Disc(t-1,k) < \epsilon_{optimality} \cdot |UB - LB|$ **then**
  $direction = forward$;
else
  $direction = backward$;

Algorithm 5: FastBack sequencing protocol

return;

Algorithm 6: FastForwardFastBack sequencing protocol

if $direction = forward$ **then**
  $sweep = true$;
else if $AE(t,k) < \epsilon \cdot |UB - LB|$ **then**
  $direction = backward$;
else
  $direction = forward$;

Algorithm 7: ϵ-FastForward sequencing protocol

if *no expected recourse function approximation for any stage* $t < T$ **then**
  $direction = forward$;
  $sweep = true$;
  return;
if $direction = backward$ **then**
  $sweep = true$;
else if $Disc(t-1,k) < \epsilon \cdot |UB - LB|$ **then**
  $direction = forward$;
else
  $direction = backward$;

Algorithm 8: ϵ-FastBack sequencing protocol

used to denote different solutions as well as cut components. In an actual implementation, it is thus advisable to increase the iteration number for the FF and ϵ-FF protocol, when the direction in an intermediate stage was set to *forward*.

Morton (1996) compares the five protocols, albeit for $\min\{|LB|, |UB|\}$ as threshold, where the ϵ values lie in the range of 0.0001 to 0.4096. He comes to the conclusion that FB,ϵ-FB and FFFB reach comparable performance, but they outperform FF and ϵ-FF. The ϵ value is relatively unimportant for the performance of ϵ-FB. For larger ϵ values, ϵ-FF becomes more competitive, but that is due to the fact that for increasing ϵ, the ϵ variants behave more like FFFB. The explanation for the relative performance differences of the protocols lies in the computationally expensive later stages that do not have to be solved so often for the FB,ϵ-FB and FFFB protocols (Morton, 1996).

Altenstedt (2003) proposes another approach. He modifies the FFFB protocol by introducing a *bouncing* stage. When the algorithm reaches the bouncing stage, a backward sweep is done. He calls such an iteration a *minor* iteration. This is in contrast to a full iteration which he calls a *major* iteration. The backward sweeps can be repeated for a fixed number of minor iterations, specified by the parameter *BouncingIterations*. The extension to the FFFB protocol is due to the implementation of the algorithm in which all problems up to the bouncing stage have a separate solver instance. Due to limited main memory, a common solver instance is shared in stages after the bouncing stage. It is faster to resolve problems that have a memory representation than to resolve problems which must be build into a memory representation before they can be solved. The pseudocode is detailed in Algorithm 9.

```
if no expected recourse function approximation for any stage t < T then
    direction = forward;
    sweep = true;
    return;
if t = BouncingStage AND direction = forward then
    if ItCounter < BouncingIterations then
        direction = backward;
        ItCounter = ItCounter +1;
    else
        ItCounter = 0;
        direction = forward;
```

Algorithm 9: Bouncing sequencing protocol

5.3. Parallelization

The two-stage as well as the multi-stage L-shaped method lends itself readily to parallelization. All subproblems that have to be solved at every stage are independent of one another, so a parallel execution can be expected to have good speedup properties. The speed up $S(N)$ is defined as the solution time of the sequential algorithm divided by the wall-clock solution time of the parallel algorithm, for N processors. According to Amdahl's

law (Amdahl, 1967), the achievable speedup is restricted by the amount of sequential code, i.e., $S(N) = \frac{1}{1-P+\frac{P}{N}}$, where P is the amount of code that can be executed in parallel. The most amount of work in the nested L-shaped method lies in the repeated solution of subproblems, which can be parallelized. A single solution process of a subproblem can also be efficiently parallelized, if an interior point method is used. The simplex method is not so easy to parallelize (Shu & Wu, 1993), but attempts have been made (Bixby & Martin, 2000). We are not interested in the parallelization of the underlying LP solution process, but in the parallelization of the decomposition method. However, depending upon the architecture, parallelization can be achieved by different means (see (Culler et al., 1999) for an introduction into parallel computing).

Parallel computing architectures are best described via several dimensions. These are instruction- and datastreams, e.g., Flynn's taxonomy (Flynn, 1972), whether the processes communicate via memory, which can be shared or distributed, or message passing, and how the memory is connected. A symmetric multiprocessing system is a Multiple-Instruction Multiple-Data (MIMD) type architecture with shared memory, which is connected via a system bus. All the cores of one processor are treated as distinct processors. This is the type of architecture that is common for modern PCs and laptops. If several of these systems are connected and can communicate with each other, it is called a *cluster*. The communication typically consists of message passing, where each system has its own memory (distributed memory), but those systems can also be build with distributed shared memory, i.e., with a global address space. A *grid* is a cluster that consists of computers that are not necessarily of the same quality with respect to computing power, main memory and other components. In particular, computers can join and leave the grid in no guaranteed order.

A crucial issue of algorithms, implemented on architectures using distributed memory or message passing, is the cost of communication. Compared to shared memory systems, distributed systems need to communicate to exchange results or to get input data. In our context, the relevant data are primal solutions used to form the right-hand-side of the subproblems and as a result the obtained dual solution. This may differ in implementations, where instead of dual solutions computed cuts can also be send.

The difficulty is to balance the amount of computation done at the distributed nodes with the amount of necessary communication. The two extreme cases are no communication, e.g., one computer solves the whole problem, versus most communication, e.g., every node of the scenario tree is solved by a different computer, so that all the primal and dual solutions must be communicated between the computers. In the first case, parallelization with respect to the number of computers in the cluster is non-existent. In the second case, the algorithm is highly parallelized, but the communication overhead can be so large to dwarf the gains of parallel execution. The main goal is to devise a strategy to effectively use the available resources in parallel while minimizing the communication overhead, such that the wall clock solution time is minimal.

The nested L-shaped method was parallelized by different authors on different architectures. Ariyawansa & Hudson (1991) implemented a two-stage L-shaped method, where only the subproblems are solved in parallel, on a Sequent/Balance, a symmetric multiprocessing system. Dantzig et al. (1991) parallelized Benders decomposition in conjunction with importance sampling on a hypercube architecture, on which the different processors

are connected via a hypercube network. Message passing is used to communicate between the processors. The master processor solves the master problem and coordinates the subprocessors, which solve the subproblems in parallel. Thus the information which has to be send is the primal solution of the master problem and the information which scenario should be solved by which subprocessor. The subproblems send back their objective function value and a cut. The scheduling is done on a first-come first-serve basis, where the next idle processor gets the next scenario subproblem until all subproblems are solved and the master problem can be recomputed with the added cuts. In contrast to the standard definition of speedup, which is defined as the ratio of parallel computing time and sequential computing time, Dantzig et al. (1991) define speedup as the sum of individual processor time divided by the parallel computing time. Efficiency is defined as speedup S divided by the number of processors p, $\frac{S}{p} \times 100\%$. On a system with 64 processors, an efficiency of near to 60 % can be achieved, where the efficiency is higher for a higher sample-size.

Ruszczyński (1993b) introduces parallelization on a symmetric multiprocessing system for the nested regularized decomposition method, but the proposed asynchronous protocol is also valid for the parallelized nested L-shaped method. Each node of the tree is solved by a thread, where the nodes can be scheduled to the worker threads via a dynamic queue or on a predetermined basis. The notion of *buffer* is used, where primal solutions are stored in "boxes" and cuts are stored in "pipes". Whenever any of these receives an update, the corresponding problems can be scheduled for a resolve with the current information. Thus information is propagated not just in one direction, but in two. His results show that the asynchronous protocol accomplishes a higher speedup than the synchronous protocol for the same number of threads. A Java implementation of this method was tested on a beowulf-cluster (Moritsch et al., 2001) and compared with a synchronous implementation using the FFFB protocol, but for very small problem sizes with up to 511 scenarios (see (Moritsch, 2006) for details).

The regularized decomposition approach, although the implementation was not as refined as Ruszczyński's, compares favorably with the L-shaped method in a parallel master/worker message-passing setting for two-stage problems (Vladimirou, 1998). In contrast to the original description, a scenario partitioning approach is used to decrease the amount of communication. The worker processors solve their predetermined set of subproblems, and Vladimirou (1998) finds that it is superior to solve the subproblems sequentially than to construct a larger problem consisting of the independent scenario subproblems and solve this instead. Due to the predetermined set of scenarios, load balancing is more important than in the hypercube approach (Dantzig et al., 1991). Dantzig et al. (1991) identifies the synchronization step at the master problem as a bottleneck that should be reduced by a parallel solution process for the master problem and the usage of asynchronous protocols.

Nielsen & Zenios (1997) employ data parallelism for solving the subproblems with an interior-point method to achieve good performance for the parallelization of Benders decomposition for two-stage problems. They make use of the structural pattern of the W matrix which must be identical for all scenarios in their approach. This can be exploited in a Single-Instruction Multiple-Data (SIMD) type algorithm that can only be run on specialized hardware. Nielsen & Zenios (1997) do not present speedups for their implementation.

Birge et al. (1996) use a message-passing architecture and a master/worker approach for the parallelization of the nested L-shaped method. To minimize communication they

split the scenario tree at a split stage. The master process solves the multi-stage program up to the split stage. Every worker process solves one subtree, for the given primal solution from the master process. The sequencing protocol for the distributed subtrees is FastForwardFastBack. This is contrasted with their "hybrid protocol" that uses the FastForward for the split stage, i.e., subtrees have to be solved to optimality. If an infeasible subtree is discovered, the master process computes a new primal solution. Their results show that the FFFB protocol is more effective than the hybrid protocol, because of load-balancing problems. If communication takes a long time, the hybrid protocol can be more effective, because it needs less communication effort (Birge et al., 1996).

A parallel implementation of the nested L-shaped computer code MSLiP (Gassmann, 1990) was done by Dempster & Thompson (1998) on a distributed memory architecture. The difference to the implementation of Birge et al. (1996) is that every subproblem is solved on its own worker instead of a whole subtree. This mitigates load-balancing problems, but creates more communication, so it can be seen as an extension of similar ideas for two-stage problems (Ariyawansa & Hudson, 1991; Dantzig et al., 1991) to multi-stage problems on a message-passing architecture. It differs from these, because subproblems send back their primal and dual solutions instead of a cut. The primal solution is needed, because the nodes at the next stage need it as an input. The amount of serial computation is thus increased as the master problem has to form the cuts out of the dual solutions. FFFB is the used sequencing protocol.

The trust region method in the context of Benders decomposition is parallelized in a synchronous and asynchronous version (Linderoth & Wright, 2003) in a master/worker approach. The parallel architecture is a computational grid, which is relatively cheap compared to a specialized cluster. This makes the implementation more susceptible to load balancing problems, because the computers are heterogeneous and not available all the time. For the cases where some workers take a long time, the asynchronous approach fares better; otherwise the synchronous variant has a smaller solution time. Linderoth & Wright (2003) compare their trust region implementation to an asynchronous implementation of the L-shaped method, where the master problem does not have to wait until all subproblems reported their results. Instead it can be resolved when a certain number of subproblems reported their results in the form of optimality cuts. The implementation extends a previous approach of Vladimirou (1998) with an asynchronous protocol. In a subsequent study, Buaklee et al. (2002) create an adaptive version of the asynchronous trust region method which chooses parameters differently from the "rules of thumb" used before, to improve its performance. The asynchronous trust region (ATR) implementation (Linderoth & Wright, 2003) is also used to investigate the empirical behavior of SAA with different sampling methods on a computational grid (Linderoth et al., 2006).

Parting from the previously mentioned parallelization techniques for the nested L-shaped method, Latorre et al. (2008) proposes a scheme called "complete scenario decomposition", where the scenario tree is divided by scenarios, i.e., every chunk consists of a path from the root to the corresponding scenario leaf node. If such a decomposition approach is applied, the non-anticipativity constraints that are implicitly fulfilled by the scenario tree structure do not hold and have to be enforced by another approach. Latorre et al. (2008) do not describe how their algorithm ensures that non-anticipativity holds. The algorithm is developed for a computational grid.

For surveys of parallelization approaches applied to stochastic programming solution techniques see (Birge, 1997; Vladimirou & Zenios, 1999). They also handle methods that are not based on Benders decomposition.

5.4. Advanced Start

One weakness of cutting plane techniques is that initial iterations are often ineffective. Advanced start techniques are used to mitigate this effect. Infanger (1992) proposes to compute the expected value problem (EV) solution with Benders decomposition itself and then start Benders decomposition on the original problem instance, but with the cuts generated during the computation of the EV solution. The goal is that the generated cuts will guide the algorithm in the early iterations to avoid ineffective initial iterations. This can be done for problems with a random RHS or technology matrix T and inter-stage independent random variables (Infanger, 1992).

Morton (1996) describes two advanced start variants with "prespecified decisions". A prespecified decision is a set of solution vectors $\{x^i \mid i = 1, \ldots, N_t\}$ for every stage. One simple idea is to use the EV solution as a prespecified solution, where the EV solution can be computed with state-of-the-art LP solvers. The first variant is a "naive implementation", where the subproblems at stage t are solved with the corresponding prespecified solution of stage $t-1$ to generate valid optimality cuts. The more involved variant uses cut sharing between different nodes at a stage to reduce the number of subproblems that have to be solved. A cut generated for one node can be reused for another node via the "dual sharing formula" by adjusting its coefficients (see (Infanger, 1992) and (Morton, 1996) for details). Zverovich et al. (2012) use the EV solution as a prespecified solution in their implementation of Benders decomposition. Other solutions can also be used as a prespecified solution, for example the worst-case, best-case or even the optimal solution.

5.5. Stage Aggregation

If a multi-stage stochastic program is present, but the available solver can only handle two-stage stochastic programs, it is possible to aggregate stages of the problem in such a way that a two-stage problem results, albeit with larger first and/or second-stage problems (see (Gassmann, 1990; Vladimirou, 1998; Dempster & Thompson, 1998)). It is also possible to aggregate a multi-stage problem to a multi-stage problem with fewer stages. This might be computationally worthwhile for problems with a large number of scenarios, but rather small subproblem size. Computational results on a small set of test problems are available and show that a suitable aggregation can reduce the solution time of the algorithm (Dempster & Thompson, 1998). On the other hand, Gassmann (1990) and Vladimirou (1998) find that an aggregation to a two-stage problem is computationally disadvantageous. The problem with stage aggregation is similar to that of cut aggregation: how to choose the "best" aggregation level for a given problem is unclear a priori.

The aggregation idea is further developed by Cerisola & Ramos (2000). They devise different schemes to find an aggregation that generates subtrees with a certain size. Kuhn

(2006) combines discretization with aggregation to yield a problem that gives bounds on the original problem.

6. Modeling Languages

If a stochastic programming problem is modeled, it has to be brought into a form that is computer readable to apply a solver in order to get a solution. For deterministic mathematical programs, the use of algebraic modeling languages is common due to their advantages over static matrix generators (Fourer, 1983). Maturana (1994) discusses more features of algebraic modeling languages in the context of existing modeling languages at the time. Most of the modeling languages he examines are still used today. A recent collection, edited by Kallrath (2012), introduces algebraic modeling languages (AMLs) and algebraic modeling systems (ALSs) and how they are used to model and solve real-world problems, following an earlier collection about AMLs (cf. (Kallrath, 2004)). This shows that modeling languages are widely used in the area of mathematical programming. They should also be considered to model stochastic programs to get the benefits of an AML, e.g., variability, modifiability, and simplicity (Fourer, 1983). It is of course possible to model stochastic programs via the deterministic equivalent formulation, but that is cumbersome and error-prone, especially for the multi-stage case as the non-anticipativity constraints must be entered manually (Gassmann & Ireland, 1995, 1996). The result is then a large-scale LP, where no special stochastic programming solution techniques can be applied as the structure of the stochastic program is not given to the solver.

An in-depth survey that covers existing techniques for modeling, analyzing and solving stochastic programs is provided by Gassmann (1998). The book edited by Wallace & Ziemba (2005) contains several chapters about existing software environments that can be used to model stochastic programs. A more detailed inspection of some environments is done three years later (Kopa, 2008), with the investment problem (Birge & Louveaux, 2011, p. 20-27) as a common example. The situation between 2005 and today changed somewhat, as many commercial vendors now supply some modeling support for stochastic programming. We examine and compare several of the existing environments in Section 6.2. But first we start with some considerations about the features and possible obstacles that are present in designing modeling languages for stochastic programming.

6.1. Theoretical Concepts

There are several challenges in designing modeling environments for stochastic programs. At first, the term stochastic program captures a variety of different problem structures, in contrast to linear programs. In this thesis, we restrict ourselves to recourse problems, but when an AML should be considered for stochastic programming in general, it may be wise to also consider other problem types, see the taxonomy given in Figure 2.2 on page 12.

Second, a recourse problem consists of two things, a deterministic problem and information about stochastic parameters. The latter is often expressed via a scenario tree. A deterministic LP can be represented in memory in a fairly standardized way. Five arrays,

for column lower and upper bounds, objective function coefficients and row lower and upper bounds, together with a sparse matrix representation, suffice. All LP solvers provide facilities to read a model presented in such a way. An in-memory representation of a scenario tree is far from standardized. A tree structure is firstly programming language dependent and secondly not unique, see (Cormen et al., 2001) for several representations. The problem of a non-standardized in-memory representation of stochastic programs is acknowledged in the literature (Fragnière & Gondzio, 2005; Condevaux-Lanloy et al., 2002; Gassmann & Ireland, 1996).

A remedy in the form of the Optimization Services instance Language (OSiL) is presented for general non-linear problems (Fourer et al., 2010), which is recently extended to stochastic programs (Fourer et al., 2009), called OSiL-SE. It is a XML-based description of a stochastic program, from which an in-memory representation can be build using the OSInstance model. At the time of writing, the OSInstance model for the stochastic extensions is not implemented and thus not directly usable.

There are several widely used file formats for LP or MIP problems, e.g., MPS (International Business Machines, 1972), lp (CPLEX, 2013), to allow the interchange of problems. A problem in a standard file format can be read in by most LP solvers. The SMPS format (Birge et al., 1987; Edwards, 1988) is an extension of the MPS format, and is widely accepted as the de facto standard interchange format for stochastic programs. The format itself is extended several times (Gassmann & Schweitzer, 2001; Gassmann & Infanger, 2007), but most software in this area only supports a subset of the specification[1].

SMPS consists of three files. The `core` file is a MPS file of the underlying deterministic problem, ordered by stages. The `time` file specifies the number of stages and divides the column and rows into stages. The scenario tree is stored in the `stoch` file, which allows a variety of different directives. Scenarios can for example be stored directly via a `SCENARIOS` section or stored as independent variables with given outcomes via an `INDEP` section. Some examples that illustrate the versatility of the format are provided by Gassmann & Kristjansson (2007). Contrasted with this de facto text file standard there is no standard for the representation of stochastic programs inside an AML (Fragnière & Gondzio, 2005). Regarding OSil-SE, there are not yet any examples available.

Two important concepts must be considered in AMLs for stochastic programming, namely *stage* and *uncertainty*.

Every variable, parameter and constraint has an associated stage. This can be implemented as an typical index set in most AMLs. However, to export the program written in an AML to the SMPS format, the columns and rows must be sorted by stage. This can not be ensured, as AMLs use internal logic to write out the problem in the MPS format (Condevaux-Lanloy et al., 2002). The multi-case staircase structure depicted in Figure 2.3 is usually lost, when writing a problem with an AML that is stage unaware (Fragnière & Gondzio, 2005). Gassmann & Ireland (1996) propose the introduction of a stage set to allow easier modeling and to perform consistency checks inside the AML.

The second concept is *uncertainty*. Uncertainty is represented in the final deterministic equivalent formulation by matrix coefficients a_{ij} (or right-hand side values, column bounds and objective function coefficients). But this does not mean that the a_{ij} are necessarily

[1] Available online at `http://myweb.dal.ca/gassmann/smps2.htm`

variates of a random variable, at least not directly. As Condevaux-Lanloy et al. (2002) point out, a matrix coefficient a_{ij} is usually the result of some function $f(u_1, \ldots, u_l)$, where u_i are parameters. Some of the parameters may be random variables. Thus one random variable can influence several coefficients, both in the matrix and in other places of the model. When using an AML to generate scenarios, it should know which parameters are random variables, and how they are distributed, as otherwise model generation will result in a scripting process that changes the value of some parameters to generate a new scenario. This process is time consuming, as the whole model must be generated every time when a parameter value changes (see the Simplified Interface for Stochastic Programming (SISP) by (Condevaux-Lanloy et al., 2002)). Gassmann & Ireland (1996) propose the introduction of a random keyword together with a descriptive part that allows arbitrary distributions of random variables.

Additional problems arise depending on the chosen solution technique. Decomposition methods can be based on the explicit or implicit deterministic equivalent formulation, therefore the AML must be able to generate both forms, when both solution techniques should be supported. Exterior sampling techniques can be implemented inside an AML with scripting support. For interior sampling techniques, the distribution must be given in a form that is readable by the solver, either the SMPS format or the OSiL-SE format mentioned above. The AML must then put the algebraic formulation $f(u_1, \ldots, u_l)$ that is needed to compute the coefficients a_{ij} inside the file format, such that the solver can compute the coefficients with the given formula. A support of this procedure is only partially available in the SMPS format for linear transformations of random variables. More complex functions are supported by the OSiL-SE format.

6.2. Practical Examples

The following commercial vendors added stochastic programming support in recent years, e.g., LINDO (LINDO) (see (Atlihan et al., 2010) for examples), Xpress-Mosel (see (Dormer et al., 2005) for examples), MPL (MPL), AIMMS (AIMMS), Frontline (Frontline Solvers), Microsoft Solver Foundation[2] (Microsoft), and GAMS (GAMS). The prediction of Fragnière & Gondzio in 2005 that AMLs would support stochastic programming in the coming years thus proved to be correct.

In the following, we list the efforts that are and were previously made by researchers to add stochastic programming support to AMLs and AMSs. A list of management systems for stochastic programming can also be found in (Kall & Mayer, 2010, p. 376f).

Algebraic Modeling Languages

Several researchers propose extensions to AMPL, namely SAMPL (Valente et al., 2005), StAMPL (Fourer et al., 2009), SML (Colombo et al., 2009) and DET2STO (Thénié et al., 2007) which we will present in turn.

[2] discontinued, see the statement of Nathan Brixius at `http://nathanbrixius.wordpress.com/2012/05/25/no-more-standalone-releases-of-microsoft-solver-foundation/`

The Stochastic Programming Integrated Environment (SPiNE) (Messina, 1997; Valente et al., 2005, 2009) allows to model stochastic programs with their AMPL extension SAMPL, also those with probabilistic constraints. SPiNE is an AMS, as it is integrated with a stochastic programming solver, FortSP (Ellison et al., 2012), and database access to store problem data and results. It also provides tools for solution inspection. SPiNE allows only scenario-based modeling. The new constructs of SAMPL are, among others, a stage and scenario set as well as commands to specify the scenario tree structure. The constructs are entered mostly via the suffix feature already available in AMPL (Fourer & Gay, 2000).

StAMPL (Fourer et al., 2009) is a system that is build on top of AMPL. The stochastic problem is divided by stage, and each stage is modeled separately in AMPL. The StAMPL preprocessor creates AMPL files for every stage and connects these files to get a deterministic core model. Scenario data is initally handled outside of AMPL, such that scenario generation routines are independent of the model. In the processing step, the scenario tree is traversed to create AMPL files. Together with the stage files and the tree node files the final output is generated in SMPS format.

SML (Colombo et al., 2009) is a modeling language that extends AMPL by a pre- and postprocessing phase to implement the `block` keyword. Thus, the problem structure can be conveyed to the solver in an intact manner and is not scrambled by the AML coefficient generation process. It can be used to specify stochastic programming problems with recourse, in a node-wise fashion. The stochastic data is specified explicitly for every tree node, such that a generated scenario tree is necessary to specify a model instance. The idea to declare separate problems for each stage and the model generation process is similar to StAMPL.

DET2STO (Thénié et al., 2007) is a script that takes the deterministic core model and a scenario tree description (explicit or transition-based), both in AMPL syntax, to generate a deterministic equivalent problem in AMPL syntax. The scenario tree description is written programmatically by the user, depending on the used scenario tree generation technique for the stochastic process. The generated deterministic equivalent does not convey special structure and can only be solved with standard LP or MIP solvers. This approach was previously tried by SETSTOCH (Condevaux-Lanloy & Fragnière, 1998) and SISP (Condevaux-Lanloy et al., 2002), except that these tools use GAMS as the modeling language and the scenario tree definition is specified outside the AML.

SISP, in contrast to SETSTOCH, allows to use computed parameter values by the AML in scenarios. The result of SISP is a SMPS file, from which an explicit DEM has to be build before it can be solved by standard LP or MIP solvers. In comparison, SISP specifies more data outside the modeling language than DET2STO, but on the other hand it can create a problem description suitable to specialized solvers.

Entriken (2001) suggests to extend AMPL by the use of a `random` attribute for parameters, but he does not use scenario or stage sets. Thus all constraints containing random parameters have to be declared explicitly for every random value of the parameter. He hopes that more compact formulations will arise that allow to model stochastic programs without this overhead. A partial ordering on variables is introduced with respect to the number of periods, to allow the modeling of multi-stage problems without an explicit stage set.

A recursive formulation of stochastic programs is proposed by Buchanan et al. (2001) in the AML sMAGIC. The random data is specified for each node, and due to the recursive formulation only suitable for symmetric trees. Independent random variables are thus implicitly assumed.

PySP (Watson et al., 2012) is a Python[3]-based modeling environment for stochastic programming problems with recourse. It uses the AML Pyomo (Hart et al., 2011, 2012), which is also Python-based. PySP and Pyomo are both part of the COIN-OR project (Lougee-Heimer, 2003). Pyomo is similar to FlopC++ (Hultberg, 2007) as both AMLs are libraries for general purpose programming languages. Users can take advantage of the strength of programming languages in combination with a modeling language. Watson et al. (2012) argue that modeling in Python comes easy for programming language novices, thus it is not restricted to programmers.

PySP uses a deterministic core model and a scenario tree representation, which is also written in Pyomo, to build the explicit DEM with non-anticipativity constraints. This is similar to SISP or DET2STO. The scenario tree description can be scenario- or node-based, depending on the user needs. PySP can solve the DEM with all solvers that can either read the AMPL solver library NL format (Gay, 2005) or LP format (LP) files. Decomposition-based algorithms can be implemented in Python and work directly with the model, as the already implemented Progressive Hedging algorithm demonstrates (Watson et al., 2012). Due to the lack of SMPS input or output routines, existing problems can not be solved with PySP solvers, and PySP problems can not be solved with already existing stochastic solvers.

Another Python-based AML APLEpy (Karabuk, 2005) is extended by (Karabuk, 2008) to model stochastic programs with recourse. Karabuks main purpose is to reduce the time to build solution algorithms within a modeling environment. APLEpy allows to implement stage- and scenario-based decomposition methods while working with the same model, by abstracting the non-anticipativity constraints and generating the necessary stage-wise or scenario-wise problems on-demand. It uses scenario indexing, similar to SAMPL. The scenario tree description is done outside of the AML and is scenario-based.

Model Management Systems

A model management system for stochastic linear programming, SLP-IOR (see (Kall & Mayer, 2010, p. 377ff) and the references therein), is developed by Kall & Mayer, starting in 1992. Deterministic LP problems formulated in GAMS can be imported. As the user works directly with matrix coefficients, SLP-IOR is not an AML. It supports a wide range of solvers and problem types (Kall & Mayer, 2005). This allows to use a specialized solver for each problem type, resulting in faster solution times compared to more general solvers. SLP-IOR uses the basic SMPS format (Birge et al., 1987), without the extensions developed later, as possible input and output format. Random variables can be specified with several distributions, also multivariate normal and empirical distributions. Support of affine transformations allows to combine random variables.

[3] `http://www.python.org/`

Strums (Fourer & Lopes, 2006) is a decomposition environment that can read in stochastic programs in the SMPS format and can turn them into an implicit or explicit deterministic equivalent, just as required by different solution algorithms, i.e., stage-wise or scenario-wise decomposition, respectively. Stage aggregation and visualization of the scenario tree is also supported. Thus it is a tool that can be used after the AML finished creating the model and before the solver is started, to create the input in the required format.

Summary

Comparing todays situation with that of 1998 (Gassmann, 1998) and 2005 (Fragnière & Gondzio, 2005), quite a few algebraic modeling languages were extended, in particular AMPL, or even newly developed (e.g., PySP), to support stochastic programming. Modeling support for stochastic programming is now also present in several commercial modeling languages, although the implementations differ in key aspects as described above. What is still unresolved is the issue of a standard representation of scenario trees, especially an in-memory representation. This continues to hinder algorithm development. The proposed solution by OSiL-SE is not yet usable. The only de facto standard for interchanging stochastic programs is still the SMPS format (Birge et al., 1987).

7. Required Work

This chapter details the goals that should be reached with this thesis. These goals are based on our review of the state-of-the-art, for both solver development and modeling language development. The state-of-the-art which we presented in the preceding Chapters 4 and 5. We derive the required work for solver development in Section 7.1. We do the same for modeling languages in Section 7.2.

7.1. Solver Development

Solution technique development for stochastic programming is an ongoing research topic. The recent results by Trukhanov et al. (2010) and Zverovich et al. (2012) show that the usage of Benders decomposition based solution algorithms is a sensible approach to solve two-stage problems. It shows also that specialized solvers are necessary, if the deterministic equivalent gets too large to be solved by conventional LP solvers. We surveyed different aspects of Benders decomposition in Chapter 4. These are in particular cut aggregation, advanced start procedures, and techniques to stabilize the master problem.

Our goal is to build upon these techniques and to combine them to further improve the solution process. To evaluate the new techniques thoroughly, a diverse test set is a prerequisite. As of today, several test sets are available in different formats. We aim to create a new test set out of the existing test sets, so that researchers can evaluate new techniques on a wide range of instances.

The specific tasks for extending Benders decomposition are divided into tasks for two-stage and multi-stage problems. The tasks for the two-stage case are the following.

- Consider techniques to remove old redundant cuts from the master problem. Although it is theoretically hard to say which cuts can be removed without being recomputed later on this should not refrain us from implementing cut consolidation techniques, as performance can improve nonetheless. An empirical evaluation can show the benefit or drawback of such an approach.

- Evaluate different advanced start techniques. To our knowledge, it is not tested which advanced start technique works best on a wide range of problems.

- Combine cut aggregation with regularization techniques. Regularized decomposition and level decomposition can be used in conjunction with cut aggregation. This is already done to decrease memory usage (Linderoth et al., 2006; Vladimirou, 1998), but it was not evaluated under the aspect of performance.

- Level decomposition is a regularization technique that uses a projection problem to find the next iterate. This projection problem uses the euclidean distance, but other

distances can also be used. The application of level decomposition to stochastic programming with other distances in the projection problem is not yet researched. Possible distances would be for example the manhattan distance and the infinity distance.

- Level decomposition requires a level parameter λ. It has to be chosen from the interval $(0, 1)$. We did not find computational evidence for a particular choice of λ.
- We want to apply the recently proposed on-demand accuracy approach to the classical L-shaped method in combination with cut aggregation. In addition, the on-demand accuracy approach requires a parameter κ, which has to be set before the algorithm starts. We want to gather computational experience to give guidance towards choosing a good value for κ.
- On-demand accuracy is shown to be computationally efficient on a small set of test problems. We want to study the effect of on-demand accuracy in combination with level decomposition on a more diverse test set. We will also look at the computational efficiency of the parameters κ and λ.

The tasks for the multi-stage case are the following.

- Reconsider sequencing protocols. Our survey in Section 5.2 shows that different authors proposed different sequencing protocols. We want to investigate further the importance of sequencing protocols and how more dynamic rules can be considered for deciding the direction of the algorithm.
- Parallelize the solution process. Benders decomposition based techniques are already parallelized in several ways, see Section 5.3 for a comprehensive survey. With the advent of multiple cores on a single CPU, it is possible to employ parallelization on normal computers, without communication costs and latency issues. Furthermore, the inter-dependencies between solution algorithms and parallelization is not sufficiently researched in our opinion. In addition, the usage of hyper-threading (Marr et al., 2002) can have an influence on the speedup behavior of algorithms.
- Apply regularization techniques to the multi-stage case. The success of regularization techniques for the two-stage case implies that the adoption to the multi-stage case should be considered. We therefore would like to investigate whether this can be done and what changes are necessary to apply regularization techniques in a nested fashion.

7.2. Modeling Languages

In recent years, modeling languages for stochastic programming came a long way. Most of the ideas proposed by Gassmann & Ireland (1996) are implemented in several AMLs and can be used comfortably. The specification of random variables and its combination via algebraic functions allows to build the deterministic equivalent formulation automatically from the data. What is still missing is an agreed upon specification of a standard in-memory

representation of stochastic programs, to be able to call a stochastic programming solver directly. As of now, an intermediate step is required: the creation of SMPS files. One task is therefore to bridge this gap, by calling a solver without the indirection of an SMPS file.

The modeling language FlopC++ that can be used inside a C++ program, has already gathered some attention and was thought of to be combined with Smi to build an open-source modeling language for stochastic programs (Kaut et al., 2008), but this was not yet done. Our goal is therefore to combine FlopC++ and Smi to build a modeling language for stochastic programs inside a general purpose programming language that is able to call specialized stochastic solvers directly from memory, without resorting to SMPS files.

Part III.

Advanced Techniques and Computational Results

8. Accelerating the Nested Benders Decomposition

Based upon the description of the Nested Benders decomposition method given in Section 3.3 and Chapters 4 and 5 we present techniques to accelerate the solution process. Some techniques are only important for the Nested Benders decomposition algorithm, but others are also relevant for the two-stage case.

Cut consolidation is described in Section 8.1. It is a technique to combat cut proliferation in the master problem and is primarily meant for the two-stage case. We describe a new sequencing protocol for nested Benders decomposition in Section 8.2. Our parallelization approach is explained in Section 8.3. It applies to both two- and multi-stage problems. We discuss dynamic cut aggregation in Section 8.4. The on-demand accuracy concept, which allows to use all computed information in a computationally efficient way, is described in Section 8.5. It can be combined with level decomposition, which we present in Section 8.6. We also present two new projection problem variants. Finally, we end this chapter with a discussion of how on-demand accuracy and level decomposition can be extended to the multi-stage case in Section 8.7.

8.1. Cut Consolidation

During a typical run of the algorithm, feasibility and/or optimality cuts are added to the subproblems at all stages $t < T$. If new cuts are added to a problem, it gets bigger. Resolving the problem can then take more time. In addition, memory consumption increases, as the generated cuts have to be stored. Depending on the level of aggregation, at most A_t^i optimality cuts are added to a problem in each iteration. The higher the number of aggregates, the more cuts get added to the problem, thus the problem of cut proliferation is more pronounced. The use of warm start techniques, available for modern simplex solver implementations, mitigates the runtime effects to a certain extend, as the algorithm can start from a dual feasible solution and does not have to start from scratch. We are not concerned with the proliferation of feasibility cuts, because these cuts cut off solutions x_t that are not feasible for the whole problem.

Optimality cuts that were generated in earlier iterations are not necessarily needed anymore for the algorithm to converge. Unfortunately, there is no reliable way to tell which cuts can be safely removed from the master problem (Ruszczyński, 2003). The remaining options are then to either not delete any old cuts, because of the fear of deleting a cut that may still be needed, or to devise heuristics that allow deletions of old cuts but with the drawback that a deleted cut may be recomputed.

To apply a heuristic it is necessary to gather some data on which decisions can be based. We call the set of parameters and their respective values used in a heuristic to remove optimality cuts from a master problem a *cut removal scheme*.

We denote an optimality cut as *redundant* if its corresponding dual variable is zero (see (Trukhanov et al., 2010)). A cut is specified by its aggregate $a \in A_t^v$ and the iteration when it was generated, $i \in \mathcal{I}_{t,k}^v$, where k is the current iteration. It can be seen from the dual problem $D_t^v(\cdot)$ (5.2) for node v at stage t that all the dual variables which correspond to the optimality cuts for a certain aggregate have to sum up to one. This means that at least one cut is always non-redundant for every aggregate.

After a problem was solved to optimality, we can look at the dual variables, especially at those that correspond to the optimality cuts. These are $\sigma_t^{v,a,i}, i \in \mathcal{I}_{t,k}^v, a = 1, \ldots, A_t^v$. We can then see which cut was redundant and which was not. When we do this inspection after every successful solution process, we can count the number of times an optimality cut was redundant or inactive in consecutive iterations and store the information for this cut as $ic_t^{v,a,i}$. When a cut was active, i.e., with a dual value greater than zero, $ic_t^{v,a,i}$ is set to zero.

The first proposed heuristic is thus called `CutRemovalByRedundancy`, see Algorithm 10, and it takes a parameter α that specifies the threshold that $ic_t^{v,a,i}$ has to reach until the corresponding cut is deleted from the problem. As the heuristic is called for a certain node, the stage t and the node number v are known. The effectiveness of the heuristic depends

```
ListOfCuts = ∅;
for i ∈ I^v_{t,k} do
    for a ∈ A^v_t do
        if σ^{v,a,i}_t = 0 then
            ic^{v,a,i}_t = ic^{v,a,i}_t + 1;
            if ic^{v,a,i}_t > α then
                ListOfCuts = ListOfCuts ∪ (a, i);
        else
            ic^{v,a,i}_t = 0;
Remove all cuts in the set ListOfCuts from P^v_t(·);
```

Algorithm 10: CutRemovalByRedundancy heuristic

crucially on a good choice for the parameter α. If it is chosen too low, the removed cuts might be recomputed at a later iteration, because they were still important. If α is chosen too high, the proliferation of cuts may not be sufficiently prevented. The choice of the number of aggregates might also play a role in evaluating the effectiveness of this heuristic.

If cuts are removed from the problem, information about the recourse function which was gained in previous iterations is removed. If this information would have been still useful, it must re recomputed. To mitigate the effects of removing cuts that are still important for the master problem, we propose the concept of cut consolidation (see (Wolf & Koberstein, 2013)). Instead of a simple removal we propose a consolidation of existing cuts to keep

some information. All cuts which were generated at the same iteration $i \in \mathcal{I}^v_{t,k}$ can be combined into a single cut in the following form

$$\sum_{a=1}^{A^v_t} E^{v,a,i}_t x^v_t + \sum_{a=1}^{A^v_t} \theta^{v,a}_t \geq \sum_{a=1}^{A^v_t} e^{v,a,i}_t. \tag{8.1}$$

This cut is identical to a regular single cut with the difference that there is not a single aggregate variable, but A^v_t many, because of the chosen aggregation level. We denote $\sum_{a=1}^{A^v_t} E^{v,a,i}_t$ with $E^{v,-i,k}_t$ and $\sum_{a=1}^{A^v_t} e^{v,a,i}_t$ with $e^{v,-i,k}_t$. To indicate that the cut is a consolidated cut that was generated at iteration i we use the aggregate number $-i$. The pseudocode for the cut consolidation heuristic is presented in Algorithm 11. Another threshold $\beta \in [0,1]$ determines the number of cuts generated at the same iteration $\hat{i}$ that must be marked as removable before all the cuts of that iteration $\hat{i}$ are consolidated into a single cut.

```
NumCuts_i = 0;
RemoveCuts = ∅;
AddCuts = ∅;
for i ∈ I^v_{t,k} do
    for a ∈ A^v_t do
        if σ^{v,a,i}_t = 0 then
            ic^{v,a,i}_t = ic^{v,a,i}_t + 1;
            if ic^{v,a,i}_t > α then
                NumCuts_i = NumCuts_i + 1;
        else
            ic^{v,a,i}_t = 0;
for i ∈ I^v_{t,k} do
    if NumCuts_i ≥ β · A^v_t then
        Generate a cut E^{v,-i,k}_t x + Σ_{a∈A^v_t} θ^{v,a}_t ≥ e^{v,-i,k}_t in the form (8.1) for iteration i;
        AddCuts = AddCuts ∪{E^{v,-i,k}_t x + Σ_{a∈A^v_t} θ^{v,a}_t ≥ e^{v,-i,k}_t};
        RemoveCuts = RemoveCuts ∪ {i};
Remove all cuts for the iterations in RemoveCuts from P^v_t(·);
Add all cuts in the set AddCuts to P^v_t(·);
```

Algorithm 11: CutConsolidation heuristic with thresholds α and β

The scheme can be altered in such a way that only all removable cuts of an iteration $\hat{i}$ that exceed the threshold β are consolidated into a cut, and all cuts that are not yet removable stay in the problem. If in the next iterations other cuts generated in iteration $\hat{i}$ become removable, the previously generated consolidated cut is expanded with the newly removable cuts. The effectiveness of the presented heuristics for particular problems depends upon the values α and β. Of course, such a scheme can not improve the solution time of methods, where the master problem is not subject to cut proliferation.

8.2. Dynamic Sequencing

For multi-stage problems, the direction of the nested L-shaped method has to be chosen at every stage $1 < t < T$. The decision is taken according to a sequencing protocol. The details of the protocols as well as the definitions of the discrepancy concept are explained in Section 5.2.

The protocol FastForwardFastBack is static regarding its decision to change the direction. The protocols FastForward and FastBack incorporate the absolute value and the discrepancy (see (Morton, 1996), respectively, in deciding when to change the direction, so the current state of the algorithm is incorporated into the decision. The threshold $\min\{|LB|, |UB|\}$ is used in the decision in the description by Morton (1996). This is not true for the original description of these protocols, where the requirement for a direction change was that no new optimality cuts can be generated (cf. (Birge et al., 1996)). We suggest to use the absolute gap $|UB - LB|$ as a threshold instead to prevent problems that can happen if the upper bound becomes zero and thus the FB protocol might cycle between the first two stages.

The goal of our new sequencing protocol is to combine the advantages of the FF and FB protocol, namely a good approximation of the recourse function at the last stage with good first stage solutions (see (Wolf & Koberstein, 2013) for more details). The discrepancy measure (5.10) is used to determine if the current approximation at stage t is considered good enough. If this is the case, we proceed to the next stage. If not, we do a backward sweep to update the current approximation. The threshold depends upon the absolute gap and therefore adjusts dynamically during the run of the algorithm.

A stage is declared critical to force the protocol to do a complete forward sweep once this stage is reached. The goal is to reduce the time spent to achieve good solutions for the current last stage approximation, but update the approximation for the last stage instead. The dynamic sequencing protocol is described in Algorithm 12. It is called by the basic protocol (see Algorithm 3).

if $t >$ *CriticalStage* **then**
  $direction = forward$;
  $sweep = true$;
else if *Disc(t-1)* $< |UB - LB|/10$ *AND direction = forward* **then**
  direction = forward;
  else
    direction = backward;

Algorithm 12: Dynamic sequencing protocol

The assignment of the critical stage is pivotal to the success of the strategy. The heuristic approach that we use to assign the critical stage is based upon the first full sweep. It results in a state where every node, except those at the last stage, has a valid recourse function approximation. The time spend in solving each stage is measured. We then assign the critical stage to the first stage that in addition with the time of the previous stages takes over 10 % of the total time for the first full sweep. Of course, the threshold values are subject to experimentation.

The protocol uses a dynamic threshold and the critical stage to trade off the accuracy of recourse function approximations at later stages and good incumbents at earlier stages.

8.3. Parallelization

Modern processors have multiple cores that allow parallel execution at a single computer. To take advantage of that we parallelize the algorithm in a fashion similar to (Ariyawansa & Hudson, 1991) on a symmetric-multiprocessing architecture, see Section 5.3 for details. Communication is not an issue, as the main memory is shared among the different threads and synchronization is reduced to a minimum, e.g., if cuts are added to a subproblem. All nodes (i.e., the corresponding problems) of a stage are solved in parallel. The stages are solved sequentially, so we do not employ non-deterministic techniques (Ruszczyński, 1993a; Moritsch et al., 2001), but make use of sequencing protocols.

Parallelization is achieved via the parallel execution of methods that are called tasks. Solving a node with subsequent cut coefficient generation is encapsulated in the task `HandleSubproblem(v)`. Combining computed cut coefficients to optimality cuts is encapsulated in the task `AggregateCuts(v)`. The tasks are executed by worker threads, which are stored in a thread pool. There are as many worker threads in the thread pool as there are cores on the processors to avoid context switching due to the operating system. A task queue belongs to the thread pool. The tasks in the tasks queue are assigned to idle worker threads on a first-come first-serve basis. The main thread can be blocked until all tasks are successfully completed.

When the number of nodes at a stage t is greater than the number of threads n in the thread pool, the speedup ratio for solving all problems at stage t is between $n/2$ and n, under the assumption that the subproblems are solvable in similar time. The first stage problem can only be solved by one thread, so there is no gain for the solution time on the first stage. However, the used optimization solver can use internal parallelization for its solution process. We do not allow this for nodes at stages other than the first to avoid context switching, and because the dual simplex is not yet parallelized, see the discussion in Section 5.3. The pseudocode for the parallelized algorithm with the use of a thread pool and tasks is given in Algorithm 13. The tasks `HandleSubproblem(v)` and `AggregateCuts(v)` are described in Algorithms 14 and 15, respectively (cf. (Wolf & Koberstein, 2013)).

As explained in Section 5.3 the nested L-shaped method was also parallelized on message passing architectures. It is of course possible to combine the symmetric multiprocessing parallelization that we employ with another parallelization layer on a computational grid. This leads to a two-tier parallelized algorithm, but it was not further investigated in this thesis.

Parallelization can have an effect upon the relative effectiveness of the nested L-shaped method, depending upon the chosen parameters. Some parameter combinations benefit more from parallelization than others. In reverse, not all parameter combinations that are effective for the parallelized version of the algorithm are so in the sequential case. It follows that the results that we obtain from the parallelized algorithm should also be analyzed under the aspect of parallelization. See (Wolf & Koberstein, 2013) for an example regarding the solution time of the algorithm with respect to the number of aggregates.

Initialization;
while $UB - LB > \epsilon_{optimality}$ **do**
  for *every node* v *at stage* t **do**
    Add task `HandleSubproblem(`v`)` to task queue;
  Block until all tasks are finished;
  if $t = 1$ **then**
    Set $LB \leftarrow c_t^1 \overline{x}_t^1$;
  else if $t = T$ *AND all subproblems are feasible* **then**
    Compute current solution value $z^k = \sum_{t=1}^{T} \sum_{v=1}^{K_t} p_t^v q_t^v \overline{x}_t^v$;
    if $UB > z^k$ **then**
      Set $UB \leftarrow z^k$ and store incumbent solution $\overline{x} = \overline{x}_1^1$;
  Call sequencing protocol to set direction;
  if $t > 1$ *AND direction = backward AND all subproblem are feasible* **then**
    for *every node* v *at stage* $t - 1$ **do**
      Add task `AggregateCuts(`v`)` to task queue;
    Block until all tasks are finished;
  if $t = 1$ **then**
    Set $k \leftarrow k + 1$;
  if $direction = forward$ **then**
    $t = t + 1$;
  else
    $t = t - 1$;
Return incumbent solution;

Algorithm 13: Parallel nested L-shaped method

Solve subproblem $P_t^v(\overline{x}_{t-1}^{a(t,v)})$;
if *Subproblem feasible* **then**
  Let $\overline{x}_t^v, \overline{\theta}_t^{v,1}, \ldots, \overline{\theta}_t^{v,A_t^v}$ be the primal solution;
  Let $\pi_t^{v,k}, \sigma_t^{v,k}, \rho_t^{v,k}, \lambda_t^{v,k}, \mu_t^{v,k}$ be the dual solution;
  Generate Optimality-Cut Coefficients;
  $E_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \pi_t^{v,k} T_t^v$;
  $e_{t-1}^{a(t,v),v,k} = \frac{p_t^v}{p_{t-1}^{a(t,v)}} \left[\pi_t^{v,k} h_t^v + \sigma_t^{v,k} e_t^v + \rho_t^{v,k} d_t^v + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v\right]$;
if *Subproblem infeasible* **then**
  Let $\pi_t^{v,k}, \sigma_t^{v,k}, \rho_t^{v,k}, \lambda_t^{v,k}, \mu_t^{v,k}$ be the dual extreme ray;
  Generate feasibility cut and add it to parent problem;
  $\pi_t^{v,k} T_t^v x_{t-1}^{a(t,v)} \geq \pi_t^{v,k} h_t^v + \rho_t^{v,k} d_t^v + \lambda_t^{v,k} l_t^v + \mu_t^{v,k} u_t^v$;

Algorithm 14: `HandleSuproblem(v)`

if $t > 1$ *AND direction = backward AND all subproblem are feasible* **then**
  for *every aggregate* $a = 1, \ldots, A^v_{t-1}$ **do**
    Generate optimality cut $\sum_{s \in S^{v,a}_{t-1}} E^{v,s,k}_{t-1} + \theta^{v,a}_{t-1} \geq \sum_{s \in S^{v,a}_{t-1}} e^{v,s,k}_{t-1}$;
  Add all generated optimality cuts to the problem P^v_{t-1}

Algorithm 15: `AggregateCuts(v)`

8.4. Aggregation

The number of optimality cuts that are generated for a node v at stage t is less or equal to the number of aggregates A^v_t. An optimality cut is usually associated to an aggregate variable that approximates the recourse function of some child nodes, depending on the partitioning of child nodes. The partitioning of the child nodes is done during initialization of the algorithm and not changed afterwards. The important questions with respect to aggregates are

1. What is a good number of aggregates, with respect to solution time?
2. How should the children of the nodes be partitioned to achieve a good solution time?

Answers to both questions are still to be found. Some concrete ideas for partitioning schemes were already given in Section 4.2. We chose the static partitioning scheme presented via equation (4.8).

A note on adaptive aggregation

Trukhanov et al. (2010) analyze a variant of the hybrid-cut method for the two-stage case that changes the number of aggregates and thus the partitions during the course of the algorithm. They claim that a "good" a priori choice of the number of aggregates is not easy and that it would be better for the algorithm to adapt the size during the run. We repeat and extend the analysis done in (Wolf & Koberstein, 2013) with respect to adaptive aggregation. The algorithm, as described by Trukhanov et al. (2010), can not work as it requires that *all* corresponding optimality cuts for an aggregate are redundant. A cut is defined to be redundant if its respective dual variable has a value of zero. To see that not all optimality cuts for an aggregate can be redundant we look at the dual of the master problem (4.7) with A aggregates at some iteration k

$$
\begin{aligned}
-z^k = \max_{\pi,\lambda,\mu,\sigma,\rho} \quad & \pi b + \lambda l + \mu u + \sigma e + \rho d \\
\text{s.t.} \quad & A^T\pi + \lambda l + \mu u + E^T\sigma + D^T\rho = c \\
& \sum_{i \in \mathcal{I}_k} \sigma^{a,i} = 1 \qquad \forall a \in A \\
& \lambda \geq \mathbf{0}, \mu \leq \mathbf{0}, \sigma \geq \mathbf{0}, \rho \geq \mathbf{0},
\end{aligned}
\tag{8.2}
$$

where $E = \bigotimes_{i \in \mathcal{I}_k} \bigotimes_{a \in A} E^{a,i}$, $D = \bigotimes_{j \in \mathcal{J}_k} \bigotimes_{s \in S(j)} D^{s,j}$,$e = \bigotimes_{i \in \mathcal{I}_k} \bigotimes_{a \in A} e^{a,i}$, $d = \bigotimes_{j \in \mathcal{J}_k} \bigotimes_{s \in S(j)} d^{s,i}$, $\sigma = \bigotimes_{i \in \mathcal{I}_k} \bigotimes_{a \in A} \sigma^{a,i}$ and $\rho = \bigotimes_{j \in \mathcal{J}_k} \bigotimes_{s \in S(j)} \rho^{s,j}$. $\bigotimes$ denotes row-wise concatenation of row vectors, i.e., the result is a matrix. From the last constraint

of problem (8.2) it follows that at least one dual variable $\sigma^{a,i}$ for some $i \in \mathcal{I}_k$ must be greater than zero. Therefore, every aggregate $a \in A$ has at least one optimality cut that is not redundant, if optimality cuts were already added to the problem. A scheme that requires all cuts for an aggregate to be redundant can thus not work.

Apart from these considerations Trukhanov et al. (2010) suggest a remedy for the a priori choice of the number of aggregates and thus the partitioning by their adaptive approach. However, the adaptive approach does not solve the problem of setting a "good" number of aggregates, it changes the partitioning of the scenarios instead. The adaptive approach has a parameter, $agg - max$, that specifies how many scenarios can be partitioned into an aggregate. This parameter has to be specified a priori and does not change over the course of the algorithm. If the algorithm runs on a problem with for example 100 scenarios and is allowed to put at most 10 scenarios into an aggregate, the lower bound for the number of aggregates is therefore $\frac{100}{10} = 10$. Therefore the burden of choosing a "good" number of aggregates translates to choosing a good value for $agg - max$ that provides a good lower bound for the number of aggregates. The partitioning itself is different compared to the hybrid-method. It is done dynamically during the course of the algorithm. The influence of this dynamic partitioning seems to be rather small, as Trukhanov et al. (2010) show comparable results for the adaptive approach and static partitioning, for equal parameter settings regarding the number of aggregates. This means that an $agg - max$ value of 100 corresponds to 10 aggregates for a problem with 1000 scenarios and to 20 aggregates for a problem with 2000 scenarios, etc.

8.5. On-Demand Accuracy

The trade-off that is apparent in choosing a good number of aggregates can be shifted towards a smaller number of aggregates with the *on-demand accuracy* approach. It is originally proposed by Oliveira & Sagastizábal (2012) for level bundle methods, which includes level decomposition in several variants, described in Section 4.3.2. Level bundle methods and proximal point methods for both exact and inexact calculations are presented in a unified framework (see (Oliveira & Sagastizábal, 2012) and the references therein). They propose the concept of on-demand accuracy oracles to improve overall solution time. They show advantages of their approach with a small computational study for two-stage stochastic programs.

We describe their technique adapted to the notation used in this thesis, and extend it, in particular, by considering an arbitrary aggregation level instead of the single-cut aggregation and by applying it also to the classical L-shaped method. We keep the notation simple by describing the method for two-stage stochastic programs. We only consider (exact) level decomposition and the classic L-shaped method. It has to be kept in mind that in the original assessment of the strength of the on-demand accuracy oracle approach (Oliveira & Sagastizábal, 2012) the results were compared with the classic L-shaped method, but the subproblems were solved with a primal-dual code that can not be warm started. Thus, all subproblems were solved from scratch every time. We employ the

simplex method to solve the subproblems, thus it is not possible to use inexact variants[1], therefore the $\epsilon_{x_{k+1}}$ parameter can be set to zero and is not considered in the following presentation.

After a subproblem is solved to optimality in iteration k, its dual solution can be used to generate an optimality cut in the multi-cut method. In particular, for every subproblem, cut coefficients $E^{s,k}$ and right-hand sides $e^{s,k}$ are computed according to equation (4.4) that are used to generate A optimality cuts of the form

$$\sum_{s \in S^a} E^{s,k} x + \theta^a \geq \sum_{s \in S^a} e^{s,k} \quad , \forall a \in \{1, \ldots, A\}.$$

For a given aggregation level, the aggregated components $E^{a,k}$ and $e^{a,k}$ are computed out of the the original cut components $E^{s,k}$ and $e^{s,k}$ according to equation (4.6). Once this is done, the original cut components are not used anymore and can be discarded. The information contained in the original components is therefore usually lost and can not be used for cut generation in subsequent iterations. With the introduction of on-demand accuracy this changes, as the original cut components $E^{s,k}$ and $e^{s,k}$ are kept to generate new cuts on-demand.

Note that it is possible to get a valid, although not necessarily tight, approximation for the recourse function for a first stage solution x^k and a subproblem s by computing $e^{s,i} - E^{s,i} x^k$ for an iteration $i \in \mathcal{I}_k$. This translates to

$$p^s \left[\pi^{s,i} \left(h^s - T^s x^k \right) + \lambda^{s,i} l^s + \mu^{s,i} u^s \right],$$

which is a feasible solution for problem (4.3), although it may not be the optimal solution. As the dual feasibility set is not dependent on the current solution x^k, every dual solution $\pi^{s,i}, \forall i \in \mathcal{I}_k$ is a feasible solution, and thus a lower bound for the corresponding primal problem (4.2). To get the largest recourse function approximation for a single subproblem s all stored cut components can be evaluated with the current solution x^k. The goal is therefore to find the index $\bar{i}^s$ that leads to the largest recourse function approximation for scenario s,

$$\bar{i}^s = \arg\max_{i \in \mathcal{I}_k} e^{s,i} - E^{s,i} x^k. \tag{8.3}$$

The complete recourse function approximation given by the stored cut components is

$$\tilde{q}(x^k) = \sum_{s=1}^{S} e^{s,\bar{i}^s} - E^{s,\bar{i}^s} x^k. \tag{8.4}$$

The current solution value of the master problem (4.7) at iteration k is denoted by

$$F^k = c x^k + \sum_{a=1}^{A} \theta^{a,k} = LB.$$

[1] We apply the term exact or inexact only to the solution process of a single subproblem, for a further discussion of inexact solution methods, see Section 4.4

If all subproblems (4.2) are solved to optimality, the recourse function evaluation can be obtained by

$$F^k(x^k) = cx^k + \sum_{s=1}^{S} Q(x^k, s) = cx^k + \sum_{s=1}^{S} p^s q^s y^{s,k}.$$

If the current first-stage solution with the on-demand accuracy recourse function approximation $cx^k + \tilde{q}(x^k)$ is larger than a target value γ, i.e., $\gamma = UB - \kappa\Delta^k$, relation

$$cx^k + \tilde{q}(x^k) \geq \gamma \tag{8.5}$$

holds, and the on-demand accuracy cuts (8.6) defined by the $\bar{i}^s$

$$\sum_{s \in S^a} E^{s,\bar{i}^s} x + \theta^a \geq \sum_{s \in S^a} e^{s,\bar{i}^s}, \quad a = 1, \ldots, A \tag{8.6}$$

are added to the master problem. The usual step that includes evaluating $F^k(x^k)$ by solving all subproblems to generate new optimality cuts is then skipped. Instead the master problem is resolved to get a new solution. This becomes more important the larger the set of scenarios, as the time spent to solve all second stage problems is roughly linear to the number of subproblems. The L-shaped method combined with the on-demand accuracy oracle is described in Algorithm 16.

If the on-demand accuracy (ODA) method is used, $\tilde{q}(x^k)$ has to be computed every time the master problem was solved. In return for that effort the second stage does not need to be solved at every iteration, only if $cx^k + \tilde{q}(x^k) < UB - \kappa\Delta^k$.

If the second stage is solved, such an iteration is called a *substantial* iteration. Otherwise it is called an *insubstantial* iteration. The ODA method allows to use all generated information, but only a small number of aggregates is necessary to incorporate the useful information in the solution process, thereby preventing cut proliferation in the master problem.

If the ODA method is used in conjunction with level decomposition, the solution x^k is the solution from the level projection problem (4.11). Fábián (2013) proposes a non-proximal level method variant which allows the target value

$$\gamma = \kappa \left(cx^k + \sum_{a=1}^{A} \theta^{a,k} \right) + (1 - \kappa)\, UB, \tag{8.7}$$

where the valid range for κ is $0 < \kappa < 1 - \lambda$ and f is the current model function or current approximation of the recourse function, given by the added cuts. We use a specialized version of the level variant (Fábián, 2013), and set the target value accordingly. The complete algorithm for level decomposition with on-demand accuracy is formally described in Algorithm 17 on page 87. On-demand accuracy can be applied to all the variants of level decomposition, described in the next Section 8.6.

The on-demand accuracy principle can be contrasted with the cut deletion strategies described in Section 8.1. In the classical application of Benders decomposition, all generated cuts are added to the master problem, without any knowledge about their "usefulness". A cut consolidation scheme then tries to mitigate the effects of cut proliferation by consoli-

dating the cuts which were not useful. In contrast, in the ODA method cut proliferation is reduced by adding only a few cuts to the master problem in the first place. Cuts that are most likely useful are added on-demand.

The method can be extended by deleting inactive cuts as discussed in Section 8.1. Oliveira & Sagastizábal (2012) discuss techniques to remove generated cuts from the problem, so called bundle management techniques. After a sequence of *critical* iterations, all inactive cuts can be removed from the problem. All iterations are grouped into sets $K^l = \{k(l), \ldots, k(l+1)-1\}, l \geq 1$. K^l contains all the iterations including and following iteration $k(l)$, until the next critical iteration $\bar{k}$ appears. Then l is incremented and $k(l)$ is set to $\bar{k}$. An iteration $\bar{k}$ is called critical, when the gap is closed sufficiently, compared with the gap at the last critical iteration (Oliveira & Sagastizábal, 2012),

$$\Delta^{\bar{k}} < (1-\lambda)\Delta^{k(l)} \Rightarrow k(l+1) = \bar{k}, l = l+1. \tag{8.8}$$

8.6. Level decomposition

In the level decomposition method, see Section 4.3.2, a projection problem is solved to determine the next iterate. This projection problem usually minimizes the squared *euclidean* distance between its solution x and the current iterate x^k, i.e., $\| x - x^k \|_2^2$. It is not necessary to use the euclidean distance. Other distances can also be used, like the l_1 or l_∞-norm, see (Ben-Tal & Nemirovski, 2005; Oliveira & Sagastizábal, 2012).

The linear projection problem for the l_2-norm (4.11) minimizes the squared euclidean distance $\| x - x^k \|_2^2$. The euclidean distance of two vectors $x, x^k \in \mathcal{R}^n$ is defined as

$$\| x - x^k \|_2 = \sqrt{\sum_{i=1}^{n} (x_i - x_i^k)^2}.$$

The squared distance is $\sum_{i=1}^{n}(x_i - x_i^k)^2$. It can be used in the objective function of a quadratic programming problem, once it is written via the binomial theorem as $\sum_{i=1}^{n}(x_i)^2 - 2x_i x_i^k + (x^k)^2$. Another possibility is to introduce new variables w with the constraints $w = x - x^k$ and to minimize $\| x - x^k \|_2^2 = \| w \|_2^2 = \sum_{i=1}^{n} w_i^2$. The objective function contains no linear part in this formulation, but n new variables and constraints must be added to the problem, which we would like to avoid. The projection problem for the first possibility thus reads

$$\begin{aligned}
\min_x \quad & (x)^2 - x^T x^k & & \\
\text{s.t.} \quad & Ax = b & & \\
& E^i x + \theta \geq e^i & & \forall i \in \mathcal{I}_k \\
& D^{s,j} x \geq d^{s,j} & & \forall j \in \mathcal{J}_k, \forall s \in S(j) \\
& c^T x + \theta \leq (1-\lambda)F^k + \lambda F(x^*) & & \\
& l \leq x \leq u. & &
\end{aligned} \tag{8.9}$$

Initialization;
while $UB - LB > \epsilon_{optimality}$ **do**
  Solve Master problem (4.1) and store solution $x^k, \theta^{1,k}, \ldots, \theta^{A,k}$;
  Set $LB \leftarrow cx^k + \sum_{a=1}^{A} \theta^{a,k}$;
  if *Master problem infeasible* **then**
    return Problem infeasible;
  for *every scenario* $s \in S$ **do**
    Find index $\bar{i}^s$ using equation (8.3);
  Compute $\tilde{q}(x^k)$ using equation (8.4);
  if $cx^k + \tilde{q}(x^k) \geq UB - \kappa LB$ **then**
    Compute $E^{a,k} = \sum_{s \in S^a} E^{s,\bar{i}^s}$ and $e^{a,k} = \sum_{s \in S^a} e^{s,\bar{i}^s}$ for every aggregate $a \in A$;
    Generate optimality cuts $E^{a,k}x + \theta^a \geq e^{a,k}, \forall a \in A$ and add them to Master problem ;
  else
    for *every scenario* $s \in S$ **do**
      Solve second-stage problem $Q(x^k, s)$ (4.2) for scenario s;
      if *Subproblem feasible* **then**
        Let $y^{s,k}$ be the primal solution and $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual solution of $Q(x^k, s)$;
        Generate optimality cut coefficients and right-hand side (4.4);
        $E^{s,k} = p^s \pi^{s,k} T^s$;
        $e^{s,k} = p^s \left[\pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s\right]$;
      if *Subproblem infeasible* **then**
        Let $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual extreme ray;
        Generate feasibility cut and add it to Master problem;
        $\pi^{s,k} T^s x \geq \pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s$;
    if *No subproblem was infeasible* **then**
      for *every aggregate* $a \in A$ **do**
        Form optimality cut $\sum_{s \in S^a} E^{s,k} x + \theta^a \geq \sum_{s \in S^a} e^{s,k}$;
        if $\theta^{a,k} < e^a - E^a x^k$ **then**
          Add generated optimality cut to Master problem;
      Compute current solution value $z^k = cx^k + \sum_{s=1}^{S} p^s q^s y^{s,k}$;
      if $UB > z^k$ **then**
        Set $UB \leftarrow z^k$ and store incumbent solution $x^k, y^{1,k}, \ldots, y^{S,k}$;
  Set $k \leftarrow k + 1$;
Return incumbent solution;

Algorithm 16: Hybrid-cut L-shaped method with on-demand accuracy

The l_1-norm distance (also called taxicab or manhattan distance) is defined as

$$\| x - x^k \|_1 = \sum_{i=1}^{n} |x_i - x_i^k|.$$

To use the absolute value in a linear problem it must be modeled explicitly with the introduction of new variables and constraints. The new variables w take the absolute value of $|x - x^k|$. This is ensured via two types of constraints, for all $i = 1, \ldots, n$, in conjunction with the objective function $\min_w w$. The constraints $w_i + x_i \geq x_i^k$ ensure that w_i takes up the slack if $x_i - x_i^k < 0$. This can be seen by rearranging the constraint to $x_i - x_i^k \geq -w_i$. If $x_i - x_i^k \geq 0$, the slack is taken up by w_i via the constraints $x_i - x_i^k \leq w_i$, which is equivalent to $w_i - x_i \geq -x_i^k$.

The projection problem in the case of the l_1-norm reads

$$\begin{aligned}
\min_{w} \quad & w \\
\text{s.t.} \quad & Ax = b \\
& E^i x + \theta \geq e^i && \forall i \in \mathcal{I}_k \\
& D^{s,j} x \geq d^{s,j} && \forall j \in \mathcal{J}_k, \forall s \in S(j) \\
& c^T x + \theta \leq (1-\lambda) F^k + \lambda F(x^*) \\
& w + x \geq x^k \\
& w - x \geq -x^k \\
& l \leq x \leq u \\
& \mathbf{0} \leq w \leq \boldsymbol{\infty}.
\end{aligned} \tag{8.10}$$

The l_∞-norm distance (also called infinity norm or maximum norm distance) is defined as

$$\| x - x^k \|_\infty = \max_{1,\cdots,n} \left(|x_1 - x_1^k|, \ldots, |x_n - x_n^k| \right).$$

The goal is to minimize the distance. Therefore it is possible to derive the LP formulation of $\min_x \| x - x^k \|_\infty$, by using a scalar variable w instead of a vector, with modified constraints from the l_1-norm problem above, namely $w + x_i \geq x_i^k$ and $w - x_i \geq -x_i^k$. Thus w is chosen as the maximum over all component-wise absolute values $|x_i - x_i^k|$, while w is minimized, fulfilling the l_∞-norm.

The projection problem in the case of the l_∞-norm reads

$$\begin{aligned}
\min_{w} \quad & w \\
\text{s.t.} \quad & Ax = b \\
& E^i x + \theta \geq e^i && \forall i \in \mathcal{I}_k \\
& D^{s,j} x \geq d^{s,j} && \forall j \in \mathcal{J}_k, \forall s \in S(j) \\
& c^T x + \theta \leq (1-\lambda) F^k + \lambda F(x^*) \\
& w + x \geq x^k \\
& w - x \geq -x^k \\
& l \leq x \leq u \\
& 0 \leq w \leq \infty.
\end{aligned} \tag{8.11}$$

From the three projection problems (8.9), (8.10), and (8.11), which stand for the euclidean, manhattan and infinity distances, respectively, only the euclidean projection problem requires a quadratic programming solver. The other two are pure linear programming problems. If no quadratic programming solver is available, level decomposition with the l_1 or l_∞ norm can be used. Therefore a comparison of the computational results of the three different projection problems is interesting and done in Section 10.5.3. The complete algorithm for level decomposition combined with on-demand accuracy is given in Algorithm 17. CPS denotes the current projection problem solution value $cx + \sum_{a=1}^{A} \theta^a$.

8.7. Extending techniques to the multi-stage case

It would be ideal, if extensions and modifications of the two-stage L-shaped method that prove to be successful can also be applied to the nested L-shaped method. Cut consolidation, parallelization and cut aggregation can be readily used in the nested L-shaped method. The techniques for stabilizing the master problem can also be extended to the multi-stage case, but not unaltered. The same holds for on-demand accuracy cut generation. We explain why after the nested nature of the algorithm is explored.

The nested L-shaped method is in principle the extension of the two-stage L-shaped method to the multi-stage case by applying the two-stage L-shaped method in a nested fashion (Birge, 1985). This is done by viewing two-stage subtrees rooted at a node (t, v) as a two-stage problem, with the difference that the respective master problem is parameterized with its current parent solution and that the respective subproblems contain optimality and feasibility cuts. Figure 8.1 depicts the nested application of the two-stage L-shaped method. Nodes with solid lines act as master nodes, nodes with dotted lines act as subproblem nodes, and nodes with dashed lines act as a master problem to their subproblems and act as a subproblem to their master problem. The boxes around the nodes depict the different two-stage problems which are nested within each other.

The nested nature leads to two observations, namely the impact on the feasible region of the primal nested master problem and the existence of cuts in the nested subproblem. The master problem at node (t, v) is different from a master problem for a normal two-stage problem. Its feasible region is dependent on its current parent problem solution $x_{t-1}^{a(t,v)}$.

Initialization;
while $UB - LB > \epsilon_{optimality}$ **do**
  Solve Master problem (4.1) and store solution $\bar{x}, \bar{\theta}^1, \ldots, \bar{\theta}^A$;
  Set $LB \leftarrow c\bar{x} + \sum_{a=1}^{A} \bar{\theta}^a$;
  if *Master problem infeasible* **then**
    return Problem infeasible;
  Solve Projection problem (8.9), (8.11), or (8.10);
  Store solution of projection problem $x^k, \theta^{1,k}, \ldots, \theta^{A,k}$;
  Set $CPS \leftarrow cx^k + \sum_{a=1}^{A} \theta^{a,k}$;
  for *every scenario* $s \in S$ **do**
    Find index $\bar{i}^s$ using equation (8.3);
  Compute $\tilde{q}(x^k)$ using equation (8.4);
  if $cx^k + \tilde{q}(x^k) \geq \kappa CPS + (1-\kappa)UB$ **then**
    Compute $E^{a,k} = \sum_{s \in S^a} E^{s,\bar{i}^s}$ and $e^{a,k} = \sum_{s \in S^a} e^{s,\bar{i}^s}$ for every aggregate $a \in A$;
    Generate optimality cuts $E^{a,k}x + \theta^a \geq e^{a,k}, \forall a \in A$ and add them to Master problem;
  else
    for *every scenario* $s \in S$ **do**
      Solve second-stage problem $Q(x^k, s)$ (4.2) for scenario s;
      if *Subproblem feasible* **then**
        Let $y^{s,k}$ be the primal solution and $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual solution;
        Generate optimality cut coefficients and right-hand side (4.4);
        $E^{s,k} = p^s \pi^{s,k} T^s$;
        $e^{s,k} = p^s \left[\pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s\right]$;
      if *Subproblem infeasible* **then**
        Let $\pi^{s,k}, \lambda^{s,k}, \mu^{s,k}$ be the dual extreme ray;
        Generate feasibility cut and add it to Master problem;
        $\pi^{s,k} T^s x \geq \pi^{s,k} h^s + \lambda^{s,k} l^s + \mu^{s,k} u^s$;
    if *No subproblem was infeasible* **then**
      for *every aggregate* $a \in A$ **do**
        Form optimality cut $\sum_{s \in S^a} E^{s,k}x + \theta^a \geq \sum_{s \in S^a} e^{s,k}$;
        if $\theta^{a,k} < e^a - E^a x^k$ **then**
          Add generated optimality cut to Master problem;
      Compute current solution value $z^k = cx + \sum_{s=1}^{S} p^s q^s y^{s,k}$;
      if $UB > z^k$ **then**
        Set $UB \leftarrow z^k$ and store incumbent solution $x^k, y^{1,k}, \ldots, y^{S,k}$;
  Set $k \leftarrow k + 1$;
Return incumbent solution;

Algorithm 17: Hybrid-cut Level decomposition with on-demand accuracy

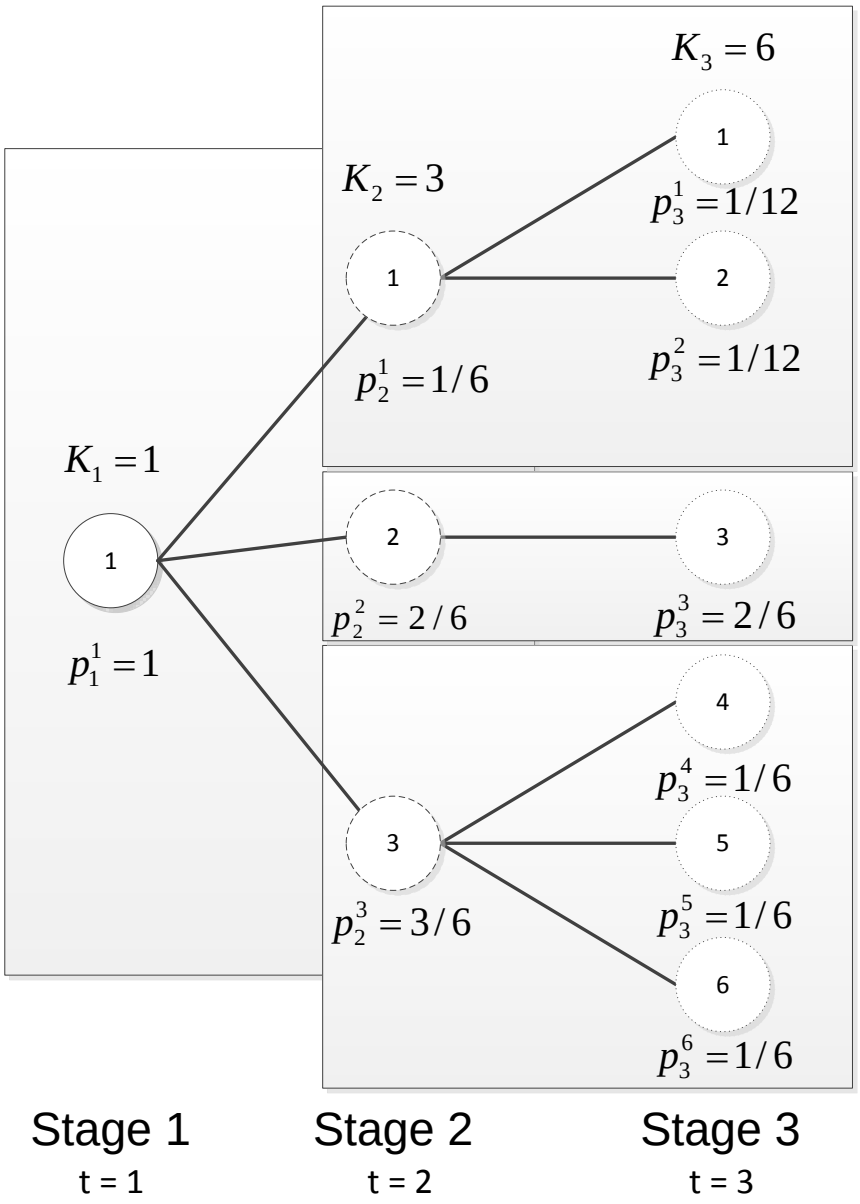

Figure 8.1. Scenario tree with six scenarios and three stages. Grey boxes depict nested two-stage instances.

This can be seen by considering the problem formulation $P_t^v(x_{t-1}^{a(t,v)})$ (5.1). It is restricted by the constraints $W_t^v x_t^v = h_t^v - T_t^v x_{t-1}^{a(t,v)}$. As $x_{t-1}^{a(t,v)}$ changes from iteration to iteration, the feasible region can also change depending on T_t^v and $x_{t-1}^{a(t,v)}$. It does never change for the trivial case of an empty technology matrix, but then P_t^v is independent of its parent problem decision and the solution process becomes trivial.

The subproblems can contain optimality and feasibility cuts. Thus, for a given master solution, the objective function value rises gradually, if the outer linearization of the respective recourse function is refined by new cuts. In turn, the master problem solution can only be thought of as optimal, if its subproblem solutions are optimal with respect to their subproblems, and so on.

These two observations have implications for nested-instance-wise lower and upper bound computations. For a real two-stage problem, the L-shaped method provides converging and valid global lower and upper bounds during the course of the algorithm. If the two-stage L-shaped method is applied in a nested fashion, it is not possible to provide global lower and upper bounds with respect to each nested instance. Only global lower and upper bounds for the whole problem can be computed, see the description in Algorithm 2 where this is done. This is due to the observation that if the solution of the parent of a master problem changes, the current incumbent solution which was used for the upper bound computation for that particular two-stage nested instance is likely to be infeasible, thus the value of the computed bound is meaningless[2].

Additionally, a computed upper bound for a nested two-stage instance, e.g., the left instance in Figure 8.1, is not really an upper bound, as it can increase when the recourse function approximation at the respective subproblems is refined such that the second-stage objective function values have increased overall, for the same incumbent master solution. The nested instance lower bound remains valid in such a case.

To overcome the obstacles of the non global lower and upper bounds, we can use the discrepancy measure (5.10) that we introduced in Section 5.2, but changed to a node-wise definition (as in (Morton, 1996)). Let the discrepancy for a node (t, v) be defined as

$$Disc(t,v) = \sum_{s \in d(t,v)} \frac{p_{t+1}^s}{p_t^v} \cdot \left(c_{t+1}^s x_{t+1}^s + \sum_{a=1}^{A_{t+1}^s} \theta_{t+1}^{s,a} \right) - \sum_{a=1}^{A_t^v} \theta_t^{v,a} \tag{8.12}$$

where $x_t^v, \theta_t^{v,1}, \ldots, \theta_t^{v,A_t^v}$ is the optimal solution of problem $P_t^v(x_t^{a(t,v)})$ and $x_{t+1}^s, \theta_{t+1}^{s,1}, \cdots, \theta_{t+1}^{s,A_{t+1}^s}$ are the optimal solutions of $P_{t+1}^s(x_t^v)$ for all $s \in d(t,v)$, respectively.

Thus it is possible to provide temporary *local* lower and upper bounds for a nested instance rooted at node (t, v) by letting the local lower bound be defined as

$$LB_t^v(x_{t-1}^{a(t,v)}) = c_t^v x_t^v + \sum_{a=1}^{A_t^v} \theta_t^{v,a},$$

[2]We adopted the convention that the objective function value of an infeasible minimization problem is set to ∞, thus the objective function value of an infeasible solution is also ∞ in this context.

and the local upper bound $UB_t^v(x_{t-1}^{a(t,v)})$ as

$$\min\left\{UB_t^v(x_{t-1}^{a(t,v)}), c_t^v x_t^v + \sum_{u\in d(t,v)} \frac{p_{t+1}^u}{p_t^v} \cdot \left(c_{t+1}^u x_{t+1}^u + \sum_{a=1}^{A_{t+1}^u} \theta_{t+1}^{u,a}\right)\right\},$$

where $UB_t^v(x_{t-1}^{a(t,v)})$ is reset to ∞ and $LB_t^v(x_{t-1}^{a(t,v)})$ to the new optimal solution of $P_t^v(x_{t-1}^{a(t,v)})$, if the solution $x_{t-1}^{a(t,v)}$ changes. The gap between these local bounds is nothing else than the discrepancy (8.12), as can be seen by computing $UB_t^v(x_{t-1}^{a(t,v)}) - LB_t^v(x_{t-1}^{a(t,v)})$.

These considerations imply that the adoption of all methods which rely on global lower and upper bounds in the two-stage case, can be adopted to the multi-stage case by considering local lower and upper bounds instead, for all stages $1 < t < T$. It also implies that before a technique can be applied both local bounds must have a finite value, such that the discrepancy can be computed. To use a technique at stage t all problems at stage $t+1$ must be solved with the current stage t solutions.

This is at odds with some of the sequencing protocols, in particular the FB,$\epsilon-FB$,$FFFB$ described in Section 5.2 and the dynamic sequencing protocol described in Section 8.2. These protocols try to give dual information as fast as possible to its parent. When this is achieved, the parent problem gets a new optimal solution, which is likely to be different from the one before, and thus the computed local bounds get invalidated and the regularization techniques can not be applied. The use of regularization techniques in the multi-stage case requires new sequencing protocols and is subject to a trade-off between avoiding zig-zagging behavior and spending more time computing the iterates.

The present sequencing protocols should be extended, so that level or regularized decomposition can be applied at intermediate stages. After an intermediate stage t is solved and the algorithm is on its way back from stage $t+1$ the algorithm can decide to resolve the stages $t+1$ and t with a regularized iterate, or if the current cuts should be applied directly to stage $t-1$. Asynchronous protocols can help in this case, as suggested by Ruszczyński (1993a) or Moritsch (2006). If on-demand accuracy cuts should be used at intermediate stages, the sequencing protocols must also be extended similarly.

9. A Modeling Environment for Stochastic Programs

In this chapter we present our work on integrating FlopC++ with Smi (Wolf et al., 2011) to get a stochastic programming modeling environment for a general purpose programming language following the ideas of Kaut (2008); Kaut et al. (2008). The integration allows to use solely open-source tools to model stochastic programs with recourse and to solve them, either with a specialized solver or a conventional LP or MIP solver. The extensions to Smi allow the creation of SMPS files, such that other stochastic programming solvers can be used, e.g., FortSP.

The C++ library ***F**ormulation of **L**inear **O**ptimization **P**roblems in **C++*** (FlopC++) (Hultberg, 2007) allows to model linear and mixed-integer problems similar to other well-known modeling languages like AMPL or GAMS. The concept behind FlopC++ is to combine a general purpose programming language, in this case C++, with a modeling language for linear and mixed-integer problems. This allows for a tighter integration of the modeling language with a decision support system programmed in the same language.

The stochastic modeling interface (Smi) provides classes and methods to create, store and access multi-stage stochastic programs. The scenario tree is stored as an in-memory equivalent of the storage structure used in the `SCENARIOS` section of a `STOCH` file (Birge et al., 1987). It also serves as a SMPS reader, for SMPS files containing `INDEP` and `SCENARIOS` sections only.

The previous attempts to combine Smi and FlopC++ use the existing projects and build the tree structure directly in C++ code in several variants (Kaut et al., 2008). This approach works for the given example, but it does not contain any of the constructs proposed by Gassmann & Ireland (1996) and it is thus not straightforward to use for a modeler of stochastic programs, as it splits the algebraic notation of the model into several parts which are connected by newly build C++ classes. To really achieve an AML for stochastic programs FlopC++ must be extended with new keywords. The integration of FlopC++ and Smi should be made *inside* the library and therefore hidden from the user.

On the one hand, the modeler should be able to specify random variables with distributions. On the other hand, the support of scenario data given by external tools should also be supported. We identified the following modeling constructs, which are necessary to model stochastic programs with recourse. We coin the extensions StochasticFlopC++ (SFlopC++).

MP_stage A dedicated set which contains all stages of the problems, and is used to index every constraint and variable at a stage other than the first.

MP_scenario_set A set which contains all scenarios. Useful, if scenario-wise data is retrieved by external tools, e.g., simulation results or a database.

RandomVariable Models random variables. Needs distribution information and methods to discretize continuous distributions. If scenario information is given it contains a data entry for every scenario. Distribution information can be given in different ways, e.g., a typical normal distribution with mean μ and standard deviation σ or an empirical distribution with probability-value pairs.

MP_random_data A random parameter. Can combine normal parameters (`MP_data`), random variables and random parameters with the usual FlopC++ algebraic operations, e.g., floor, $+$, $\cdot$.

Smi can be extended with the following items to enable a better integration with the modeling language and possible solvers.

SMPS writer Smi should be able to write out SMPS files. This allows interchanging models and to test different solvers that do not use Smi as input.

BLOCKS support Many existing files use the `BLOCKS` format to describe uncertainty. Supporting this format would allow to read and solve more problems.

Sampling support With `INDEP` or `BLOCKS` sections, it is easily possible to specify stochastic programs with a huge number of scenarios, e.g. storm with 5^{118} ((Linderoth et al., 2006), based on (Mulvey & Ruszczyński, 1995)). Trying to construct such a program in-memory will result in a segmentation fault for the foreseeable future. Storing the distribution information and sampling a given number of scenarios on-demand is possible and necessary to support approximative solution techniques, e.g., SAA.

Wait-and-See Smi should provide results of the wait-and-see model.

We model the well-known dakota problem (see, e.g., (Higle, 2005)) in SFlopC++ to explain the new constructs in Listing 9.1.

```
1  MP_model dakota(new OsiClpSolverInterface());
2  MP_stage T(2);
3
4  enum {desk, table, chair, numProducts};
5  MP_set P(numProducts);
6
7  enum {lumber, finishing, carpentry, numResources};
8  MP_set R(numResources);
9
10 enum {low, normal, high, numScenarios};
11 MP_scenario_set scen(numScenarios);
12
13 MP_data prob(scen);
14 prob(low)    = 0.3;
15 prob(normal) = 0.4;
16 prob(high)   = 0.3;
17 dakota.setProbabilities(prob);
18
19 double scenDemand[1][numProducts][numScenarios] =
20 {// Second Stage, First Stage is always omitted
21   { // L     M    H
```

```
22      {50,  150, 250}, // desk
23      {20,  110, 250}, // table
24      {200, 225, 500}  // chair
25    }
26  };
27  MP_random_data demand(&scenDemand[0][0][0], T, P);
28
29  MP_data resourceCost(R);
30  resourceCost(lumber) = 2;
31  resourceCost(finishing) = 4;
32  resourceCost(carpentry) = 5.2;
33
34  MP_data resourceReq(P, R);
35  resourceReq(desk, lumber) = 8;
36  resourceReq(desk, finishing) = 4;
37  resourceReq(desk, carpentry) = 2;
38  resourceReq(table, lumber) = 6;
39  resourceReq(table, finishing) = 2;
40  resourceReq(table, carpentry) = 1.5;
41  resourceReq(chair, lumber) = 1;
42  resourceReq(chair, finishing) = 1.5;
43  resourceReq(chair, carpentry) = 0.5;
44
45  double prices[3] = {60.0, 40.0, 10.0};
46  MP_data sellingPrice(&prices[0], P);
47
48  MP_variable
49    x(R),    // amount of resources
50    y(T,P); // produced units
51
52  MP_constraint demandConstraint(T,P)
53  MP_constraint productionConstraint(T,R)
54
55  demandConstraint(T+1,P) = y(T,P) <= demand(T,P);
56  productionConstraint(T+1,R) =
57    sum(P, resourceReq(P, R) * y(T,P)) <= x(R);
58
59  dakota.setObjective(
60    sum(P, y(T+1,P) * sellingPrice(P))
61    - sum(R, x(R) * resourceCost(R)) );
62  dakota.attach(dakota.Solver);
```

Listing 9.1 Dakota model in SFlopC++

If we assume that the demand for desk, tables, and chairs is independently distributed, but with the same values, we can model the problem via the scenario approach above, but we would have to build all the combinations manually. Instead it is possible to use random variables with empirical distributions by replacing lines 10-27 in Listing 9.1 with the following code.

```
10  std::vector<double> values_desk = { 50, 150, 250 };
11  std::vector<double> values_table = { 20, 110, 250 };
12  std::vector<double> values_chair = { 200, 225, 500 };
13  std::vector<double> prob = { 0.3, 0.4, 0.3 };
14  RandomVariable* random_vars[1][numProducts] =
```

```
{ // Second Stage
  { // Products
  new EmpiricalRandomVariable(values_desk,prob), // Desks
  new EmpiricalRandomVariable(values_table,prob), // Tables
  new EmpiricalRandomVariable(values_chair,prob) //Chairs
  }
};
MP_random_data demand(&random_vars[0],T, P);
```

Listing 9.2 Extended Dakota model in SFlopC++

This modified problem has three random variables, where each variable has three outcomes. This will result in nine scenarios via the cartesian product of the three random variables. This computation is done automatically, during the coefficient generation phase. The result of the `attach` call is a deterministic model, where the values of the random elements are set to the expected value, and a scenario tree. This information can then be used to solve the deterministic equivalent problem, write the problem into SMPS format, or call a stochastic solver. The objective function is also build automatically. Whenever a stage-indexed variable appears in the objective function, it is readily multiplied with the correct probability.

To compare SFlopC++ with previous integration attempts (Kaut, 2008; Kaut et al., 2008) we state the problem formulation for the financial portfolio problem (Birge & Louveaux, 2011, p. 20-27) in Listing 9.3, with the values used in (Kopa, 2008, p. 10f). The scenario tree is constructed automatically.

```
MP_model investmentModel(new OsiClpSolverInterface());
MP_data initialWealth,goal;
initialWealth() = 55;
goal() = 80;

enum {asset1,asset2,numAssets};
MP_set assets(numAssets);

enum {numStage=4};
MP_stage T(numStage); //Time Stages

enum {numScen=8};
MP_scenario_set scen(numScen);

double scenarios[numStage-1][numAssets][numScen]=
{
  {// stage 2
  {1.25,1.25,1.25,1.25,1.06,1.06,1.06,1.06},//asset 1
  {1.14,1.14,1.14,1.14,1.16,1.16,1.16,1.16} //asset 2
  },
  {// stage 3
  {1.21,1.21,1.07,1.07,1.15,1.15,1.06,1.06},//asset1
  {1.17,1.17,1.12,1.12,1.18,1.18,1.12,1.12} //asset2
  },
  {// stage 4
  {1.26,1.07,1.25,1.06,1.05,1.06,1.05,1.06},//asset1
  {1.13,1.14,1.15,1.12,1.17,1.15,1.14,1.12} //asset2
  }
```

```
29    };
30    MP_random_data returns(&scenarios[0][0][0],T,assets);
31
32    MP_variable x(T,assets);
33    MP_variable wealth(T);
34    MP_variable shortage(T), surplus(T);
35
36    MP_constraint
37      initialWealthConstr,
38      returnConstr(T),
39      allocationConstr(T),
40      goalConstr(T);
41
42    initialWealthConstr() = sum(assets,x(0,assets)) == initialWealth();
43    allocationConstr(T) = sum(assets,x(T,assets)) == wealth(T);
44    returnConstr(T+1) = sum(assets,returns(T,assets)*x(T-1,assets) ) == wealth
          (T);
45    goalConstr(T.last()) = wealth(T) == goal() + surplus(T) - shortage(T);
46
47    MP_expression valueFunction( 1.3*shortage(T.last()) - 1.1*surplus(T.last()
          ));
48    investmentModel.setObjective( valueFunction );
49    investmentModel.attach(investmentModel.Solver);
50    investmentModel.solve();
```

Listing 9.3 Investment model in SFlopC++

After a stochastic model is created with the `attach` call, the `MP_model` contains a scenario tree in the form of an `SmiScnModel` object. The `SmiScnModel` can be retrieved from the `MP_model` instance for further usage, e.g., writing a SMPS file, compute the wait-and-see solution, etc. A call to `solve` creates the deterministic equivalent problem via the `SmiScnModel` instance and solves it. Subsequently, the solution can be accessed via the `MP_variable`s. Correlation of variables and multivariate distributions are not supported at the moment.

The integration of a stochastic solver, which uses Smi to process the scenario tree input, is shown in Listing 9.

```
1     MP_model someModel(new OsiClpSolverInterface());
2
3  // Specify the model here
4
5     someModel.attach(someModel.Solver);
6     OsiStochasticParallelNestedBendersSolverInterface pnb(someModel.Solver,
          someModel.getSmi());
7     pnb.initialSolve();
8  // Do some solution processing
```

Listing 9.4 Call a stochastic solver in SFlopC++

10. Computational Results

The acceleration techniques described in Chapter 8 are evaluated in this chapter. The different techniques were tested on a wide range of test problems, which are described in Section 10.1. The different averages used to evaluate the results, e.g., arithmetic, geometric and shifted geometric mean, as well as the use of performance profiles are explained in Section 10.2. The implementation itself is described in Section 10.3. To give context for the computing times, the computing environment is described in Section 10.4. Section 10.5 contains the computational results for the techniques that can be applied to two-stage problems. These techniques are cut aggregation, cut consolidation, level decomposition, on-demand accuracy and advanced start solutions. As the algorithm is parallelized, Section 10.6 deals with the consequences of parallel execution on the relative order of different methods and gives speedup factors. The next Section 10.7 contains the results for the different sequencing protocols applied to the multi-stage test instances. The application of parallelized level decomposition within the approximative solution method sample average approximation (SAA) is explored in Section 10.8 with respect to computing times. A conclusion of all the results and a final comparison of the fastest algorithms is done in Section 10.9.

10.1. Test Instances

We assembled a wide range of test problems for both the two-stage and the multi-stage case. Due to the large number of instances only the containing collections are given here. For more information regarding the specific problems please consult the references (cf. (Zverovich et al., 2012))[1]. The instances and dimensions of the two-stage test set are presented in Table A.1 in the appendix. The instances and dimensions of the multi-stage test set are presented in Table A.2 in the appendix.

The **POSTS** test set (Holmes, 1995) contains four different problem families of which three problems are multistage problems.

The **slptestset** (Ariyawansa & Felt, 2004) is a collection of mostly two-stage linear stochastic programming problems, compiled by Andrew Felt. It is available online at `http://www4.uwsp.edu/math/afelt/slptestset.html`. It contains nine different problem families with a total of 40 different instances, mostly due to varying scenario size. Three problems are multi-stage problems.

A set of five different linear two-stage stochastic programs with a large number of scenarios is compiled by Linderoth et al. (2006), it can be retrieved at `http://pages.cs.wisc.edu/~swright/stochastic/sampling/`. The problems are solved by the authors

[1] More references and information about particular problems are available online via `http://users.iems.northwestern.edu/~jrbirge/html/dholmes/SPTSlists.html`.

approximately with SAA, and thus we name the test set **sampling**. We sampled from the particular problems 20term, ssn and storm three instances each, with 1000, 2000, and 3000 scenarios. The problems gbd and LandS with 646425 and 1000000 scenarios, respectively, can be solved directly. Thus we have a total of 11 test instances from this test set.

A set of three problems, **rand**, with a total of fifteen instances are randomly generated (Zverovich et al., 2012), using the routine `GENSLP` (Kall & Mayer, 1998). The problems do not possess any real-world structure, but they are useful for evaluating scale-up properties of algorithms.

A two-stage gas portfolio planning problem, **saphir**, which is numerically challenging, is available with five instances (Koberstein et al., 2011).

Consigli & Dempster (1998) present a multi-stage financial portfolio problem, **watson**. It comes in two flavors, independent and dependent variables, with a total of 24 instances.

We also consider ten two-stage problems by Deák (2011) which have normal distributed right-hand side, but only use the problems also considered by Oliveira & Sagastizábal (2012). We named the test set **deak**, after its contributor. It is available online at `http://web.uni-corvinus.hu/~ideak1/kut_en.htm`. We decided to use only the largest three instances of each problem also used by (Oliveira & Sagastizábal, 2012), resulting in 30 instances.

Several mixed-integer stochastic programs are bundled in the **SIPLIB** (Ahmed et al., 2013) test-set, compiled by Shabbir Ahmed. We use the sslp instances without the integer requirements.

Another multi-stage problem with mixed-binary first-stage variables is a supply chain planning problem called **SCDP** (Koberstein et al., 2013).

10.2. Evaluation Techniques

We apply our solution algorithm on a large set of test problems described in the previous section. To evaluate the effects of parameter combinations on these problems we measure the computing time and the iteration counts. The computing time is measured as the wall clock time of the algorithm without input and output routines. To perform more detailed analysis the computing time is also measured for distinct parts of the algorithm, like the time spent in each stage, or for different tasks, e.g., creating a subproblem in memory or solving a subproblem. The iteration counts are also measured, as these are more comparable to other implementations than the sole computing time, which depends on the interaction of hard- and software.

To compare one parameter setting with another the measured computing times and/or iteration counts on the whole test set are compared, as a problem by problem comparison is impractical due to the large number of instances. It would be hard to derive meaningful conclusions from single instance data alone. Thus we use averages of the individual results, namely the arithmetic, geometric and shifted geometric mean.

To prevent problems with the averages due to very small computing times, we set every computing time to at least 0.05 seconds. Instances which exceed the computing time threshold of $3,600$ seconds are counted as if they solved the problem in $3,600$ seconds. This is an advantage for the methods that exceed the time threshold (cf. (Achterberg,

2007)). We denote it in the results if a method fails to solve a problem. Such occurrences are also counted as if they solved the problem in $3,6000$ seconds. It is also be possible to exclude these instances when the averages are computed, but this is unfair to those methods that solve these instances correctly, and especially if this takes a long time. On the other hand, this distorts the overall running time. Therefore we include both averages if some method fails to solve a problem to allow a better comparison.

Let $t_s \geq 0$ denote the time of instance s. Let N denote the number of total instances. Summing up all the individual computing times for all test instances results in the total computing time for the whole test set, $\sum_{s=1}^{N} t_s$. This method neglects the differences that may appear between many instances, as the computing time of large instances dominates the result. The sum divided by the number of instances gives the time which is spend on each instance on average, i.e., the arithmetic mean (AM) $\frac{\sum_{s=1}^{N} t_s}{N}$, but with the same caveat that large instances dominate the result.

To compute an average of the computing time ratios, we use the geometric mean. The geometric mean (GM) is defined as $\left(\prod_{s=1}^{N} t_s\right)^{1/n}$. It is sensitive to small changes in computing times for small instances, see (Achterberg, 2007, p. 321f).

To reduce the influence of the small instances, we use the shifted geometric mean (SGM). The shifted geometric mean is computed as $\left(\prod_{s=1}^{N} t_s + s\right)^{1/n} - s$. The influence of small differences in computing time for instances that can be solved faster than s is less pronounced compared with the geometric mean, in particular for computing times less than one second (cf. (Achterberg, 2007)). The shifted geometric mean is thus a compromise between the arithmetic and geometric mean. We use the shifted geometric mean with $s = 10$ to compare the relative performance of different methods.

Apart from the averages, Dolan & Moré (2002) propose the use of performance profiles for the comparison of different algorithms, a widely adapted concept, see, e.g., (Zverovich et al., 2012), (Wesselmann, 2010) and (Koberstein, 2005). A performance profile is a graphical comparison of solution methods or algorithms that allows to see the influence of test instances on the relative performance of the algorithms. It also handles the case where an algorithm is unable to solve a test instance. To create a performance profile the computing times for all methods $m \in M$ on all problem instances $p \in P$ are recorded as $t_{p,m}$, where M is the set of methods and P is the set of problem instances. To get a comparison between methods we compute ratios for every method and instance as follows

$$r_{p,m} = \frac{t_{p,m}}{\min\{t_{p,m'} : m' \in M\}.} \tag{10.1}$$

If the solution time $t_{p,m}$ exceeds the computing time threshold, it should be set to a high number, e.g., if the threshold is $3,600$ seconds it can be set to $3,600 \cdot 1,000$, to ensure that this instance is counted as unsolvable in the profile. Situations where a method fails to provide a correct solution or fails due to memory limitations are handled in the same way.

The cumulative distribution functions for every solution method is then defined as

$$\rho_m(\tau) = \frac{|\{p \in P | r_{p,m} \leq \tau\}|}{|P|}, \tag{10.2}$$

and it gives the probability that a solution method solves a problem within a ratio of τ of the fastest solution methods. These cumulative distribution functions can be plotted into a diagram for a graphical comparison. Of course, it is also possible to compute these profiles for iteration counts. For further details and properties of performance profiles, see (Dolan & Moré, 2002).

10.3. Implementation Aspects

The algorithmic techniques which we discussed in earlier chapters need an efficient implementation to be usable on real world problems. The theoretical complexity of the simplex method for example has not changed over the years, but the performance of the simplex method has increased by two orders of magnitude (Maros, 2003, p. xviif). This is due to new algorithmic techniques and refinements in the implementation, as well as new hardware (cf. (Maros, 2003; Koberstein, 2005)). Thus this section contains several implementation details which increase performance and are helpful for an efficient implementation.

10.3.1. Implementation

The algorithm is implemented in C++ and compiles with VisualC++ under Windows and with gcc under Linux[2]. It is based upon the COmputational INfrastructure for Operations Research (COIN-OR)[3] framework (Lougee-Heimer, 2003), in particular the stochastic modeling interface (Smi)[4] and the open solver interface (Osi)[5]. These projects build upon CoinUtils[6], which provides classes to store sparse matrix and vector data structures (Maros, 2003, p. 69ff). Osi provides access to different linear programming and mixed-integer solvers via a common interface. This allows to change the underlying LP solver without rewriting the entire program. Smi provides classes to read in SMPS files, store a scenario tree in memory, store the deterministic core problem, and to build an implicit deterministic equivalent presentation solvable by any Osi compatible solver. We extended Smi with a SMPS writer, a reader which can read in scenarios specified via `BLOCKS`, and the capabilities of solving the WS and EEV problems directly.

Apart from the COIN-OR libraries, the code uses the boost libraries[7], as well as google-glog[8] logging library and the googletest[9] testing framework. As the available cores of modern computers should be utilized, the code is thread-based and uses shared main memory to store information; see Section 5.3 regarding parallel computing architectures. One thread with one solver instance is created for every available logical core, and they are managed via a thread pool[10]. This entails that the subproblem structure must be build into memory from the stored data before it can be solved. Those parts of the problem

[2] Visual C++ 2012 and gcc-4.7
[3] `http://www.coin-or.org/`
[4] `https://projects.coin-or.org/Smi`
[5] `https://projects.coin-or.org/Osi`
[6] `https://projects.coin-or.org/CoinUtils`
[7] `http://www.boost.org/`
[8] `https://code.google.com/p/google-glog/`
[9] `https://code.google.com/p/googletest/`
[10] threadpool `http://threadpool.sourceforge.net/`

structure that do not change between different subproblems are retained in memory to reduce problem building time. The master problem is managed by a separate solver instance to avoid unnecessary overhead.

The work, which is done by the threads, is described by *tasks*, which are simply C++ methods. The tasks are assigned to the thread pool by the main process at those points in the algorithm whose parts can be processed in parallel. The thread pool assigns the tasks to the threads on a first-come first-serve basis, until all tasks are processed. The parallel implementation is described in pseudocode in Algorithm 13 together with the main tasks `HandleSuproblem` and `AggregateCuts` in Algorithms 14 and 15, respectively. Additional information that belongs to a specific scenario is stored in the corresponding node in the scenario tree, e.g., warm start information, optimality cuts, etc.

10.3.2. Solving a subproblem

In the original description of the L-shaped method (Van Slyke & Wets, 1969), feasibility of the subproblems has to be checked in a separate step (step 2). To accomplish this, an LP is constructed with additional artificial variables that measure constraint violations. The objective is to minimize the constraint violation. If the objective function value of the optimal solution is greater than zero, a feasibility cut is then generated from the associated simplex multipliers.

In the description of Freund (2004), which we adapted to our notation in Section 3.3, a feasibility cut is generated from a dual extreme ray if a dual subproblem (4.3) is unbounded. This ray is returned by the LP solver if it detects that the dual is unbounded. It is thus not necessary to solve an additional LP problem just to check if a subproblem is feasible.

All variables are non-negative in the original L-shaped description. In practical problems, this might not be the case for all variables (cf. (Dempster & Thompson, 1998)). The necessary changes to allow arbitrary variable bounds are already described in Chapters 4 and 5 in the formulas for the optimality and feasibility cuts, (4.4) and (4.5), respectively. It was not mentioned there how to get the λ and μ values for the lower and upper bounds, respectively, from a solver that usually solves the primal problem with the dual simplex.

The sum $\lambda + \mu$ of the dual slack variables λ and μ corresponds to the reduced cost vector rc (Koberstein, 2005, p. 12), which is equal to $(h - T\bar{x}) - W^T\pi$ for subproblem (4.2). The solver returns the reduced cost vector rc. We must then determine if rc_i corresponds to a value for λ_i or μ_i depending on the actual variable bounds and the current primal solution. The correspondence is the following (cf. (Koberstein, 2005, p. 14)).

$$\lambda_i = rc_i, \quad \text{if } l_i > -\infty \text{ and } u_i = \infty \tag{10.3}$$

$$\mu_i = rc_i, \quad \text{if } l_i = -\infty \text{ and } u_i < \infty \tag{10.4}$$

$$\lambda_i = rc_i, \quad \text{if } l_i > -\infty \text{ and } u_i < \infty \text{ and } x_j = l_j \tag{10.5}$$

$$\mu_i = rc_i, \quad \text{if } l_i > -\infty \text{ and } u_i < \infty \text{ and } x_j = u_j \tag{10.6}$$

If the primal subproblem is infeasible and the dual unbounded, the solver returns an extreme ray $r \in R^m$. If primal variable bounds are present, the corresponding dual slack variables can not be determined via the reduced costs, as they are not available. Instead,

the solver can return a Farkas certificate to prove infeasibility of the primal problem, from which we can deduce the dual slack variable values (Rubin, 2011).

10.3.3. Warm Start

An important part of the algorithm is the use of simplex warm starts, see Section 4.5. The performance of the decomposition algorithm would deteriorate without these (cf. (Morton, 1996),(Dohle, 2010),(Rudolph, 2010)). A warm start is nothing else than the information which variables are basic, and which are nonbasic and at which bound. It is stored after a problem is solved to optimality. If the problem is encountered again, it can be restarted with the warm start basis. If the dual simplex is used, the warm start solution stays dual feasible, even if cuts are added to the problem. Often it takes only a few pivot steps to get from the warm start solution to the new optimal solution. In our algorithm, we use node-wise warm starts. A stage-wise warm start is used when a node-wise warm start is not yet available, e.g., in the first iteration.

10.3.4. Tolerances

Another important part of an algorithm dealing with floating point variables are tolerances, as floating point arithmetic is inexact by design (Goldberg, 1991). A LP solver has an internal threshold from which a variable is considered to be equal to zero. CPLEX considers all values that are smaller than 10^{-6} as zero. Due to rounding errors the output of the LP solver can contain very small values which are smaller than the threshold. The programmer must ensure that these values are set to zero, by checking against a zero tolerance.

When these small values are not zeroed out by applying a zero tolerance, numerical artifacts can occur while processing the output, for example in cut generation. This is the case, because, e.g., $10^6 \cdot 10^{-10} = 10^{-4}$, but $10^6 \cdot 0 = 0$. These errors can accumulate and thus not adopting the check for zero tolerances can lead to unsolvable problems or wrong convergence behavior. It is advisable that the zero tolerance used in the algorithm is the same as the one of the underlying LP solver, as otherwise problems can occur that are hard to understand and debug.

10.4. Computing environment

All the test instances were solved on a separate computer with the only task of solving the problems to reduce measurement errors by computer load due to other processes. The processor is an Intel Core i7-3770 with 3.4 GHz and four physical cores, but it provides eight logical cores via hyper-threading (Marr et al., 2002). The computer has 16 GiB of main memory and the operating system is 64-bit Windows 7 Professional. The code was compiled with VisualC++ 2012 under VisualStudio 2012 in release mode. The external LP, MIP and QP solver is CPLEX 12.4.

10.5. Evaluation of Two-Stage Acceleration Techniques

To evaluate the effect of accelerating techniques on the algorithm we define a base case, from which several parameters can then deviate. The Benders base case (Benders BC) is the single-cut L-shaped method with the dual simplex method as the LP solver. Node-wise warm start is enabled. Parallelization is enabled and all available cores are used by creating a respective number of threads and solver instances. If subproblems become infeasible, only one feasibility cut is generated. The expected value solution is used as an advanced start solution. The time to solve the EV problem is counted towards the solution time. The default settings for all the tolerances is 10^{-6}, $\lambda = 0.5$, and $\kappa = 0.5$. The DEM is solved with the CPLEX barrier method without crossover. It utilizes all available cores.

In Section 10.5.1 we present results achieved with cut aggregation. The effect of cut consolidation, in combination with cut aggregation, is evaluated in Section 10.5.2. Effects of the choice of λ on the performance of level decomposition are studied in Section 10.5.3 for the three different projection problems. The effect of cut aggregation on level decomposition is also evaluated. Computational results of using on-demand accuracy in various combinations are presented in Section 10.5.4. We evaluate the effectiveness of different advanced start solutions in Section 10.5.5. Part of the results in Sections 10.5.1 and 10.5.2 are already published (Wolf & Koberstein, 2013).

10.5.1. Cut Aggregation

The possibility to choose aggregation levels between single and multi cut is recently evaluated (Trukhanov et al., 2010) on instances with 1000, 2000 and 3000 scenarios of the problems 20term, ssn and storm, which are present in the sampling test set. We evaluated the effect of cut aggregation for our whole test set, including the instances used by Trukhanov et al. (2010)[11]. Note that for instances with less scenarios than the aggregation level, the respective computing time of the multi-cut method is taken.

The computing times are presented in Table 10.1. The comparison shows that an aggregation level between 20 and 100 results in an improvement about 40% compared with the single-cut method, as measured by the shifted geometric mean. The multi-cut approach increases the computing time by 60%. Measuring with the arithmetic mean the aggregate levels 50 and 100 take more time than the winner with 20 aggregates. This can be explained by the problem family sslp. For theses problems, cut proliferation becomes already a problem for a low number of aggregates, like 50 and 100, thus the arithmetic mean rises. The shifted geometric means of 20, 50, and 100 are close together, so the effect on the other problems in the test set is not so large.

A performance profile, comparing the effect of the number of aggregates, is shown in Figure 10.1. It shows that cut aggregation does not always lead to faster solution times, as single-cut and multi-cut Benders are the fastest solution methods on at least 13% and 25% of all problems, respectively. On the other hand, a cut aggregation level of 20, 50, and 100 solves more than 90 % of all problems within a τ of 2.

Cut aggregation trades off the number of iterations to solve the problem versus the time to solve the master problem (cf. (Wolf & Koberstein, 2013)). The single-cut method provides

[11] We sampled instances with the same number of scenarios and do not use their specific instances.

Aggregates	AM	GM	SGM	BC
Single	53.27	2.73	13.30	0
5	29.51	2.04	10.05	-24
10	24.86	1.80	9.15	-31
20	23.76	1.62	8.51	-36
50	26.24	1.50	7.93	-40
100	41.53	1.49	8.09	-39
Multi	263.60	3.62	21.33	60

Table 10.1. Computing times for different number of aggregates for the L-shaped method. The times are given as the arithmetic mean (AM), geometric mean (GM), and shifted geometric mean (SGM) with $s = 10$. If an instance has less scenarios than the number of aggregates, the respective multi-cut computing time was used. The last column gives the percentage change of the shifted geometric mean compared with the Benders base case (BC).

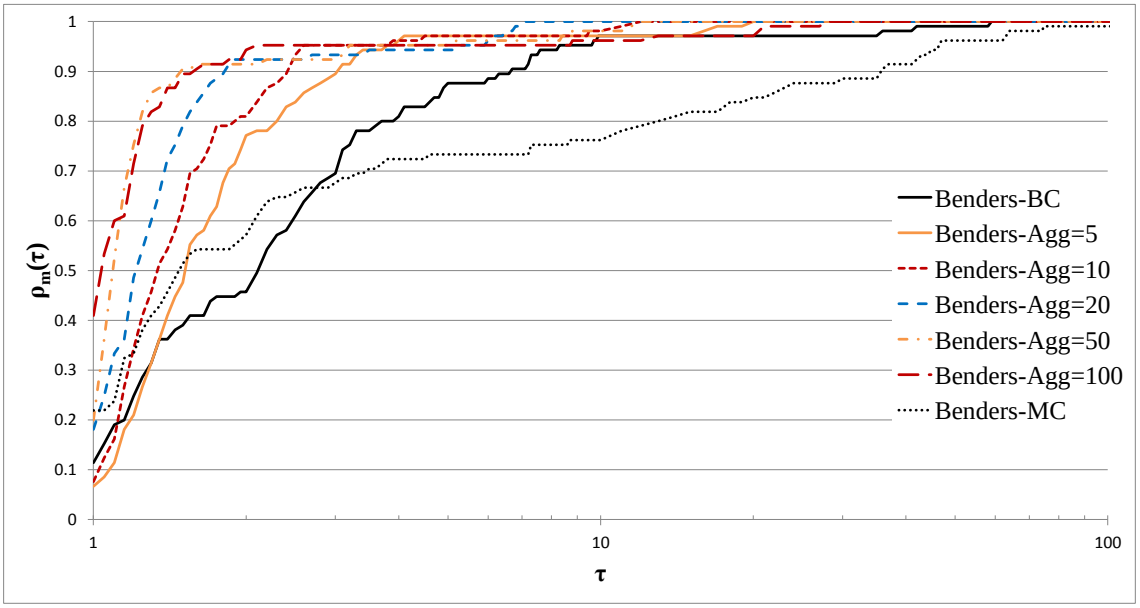

Figure 10.1. Performance profile of the L-shaped method for different cut aggregation levels.

only limited recourse function information, because all the information is aggregated in only one cut, whereas the multi-cut method provides information for every scenario. Thus, the single-cut method needs more iterations to gather a similar approximation quality than the multi-cut method. On the other hand, adding all the information to the master problem in the form of optimality cuts results in a larger master problem with more variables and constraints which is harder to resolve. This effect is called cut proliferation. The trade-off can be seen by the iteration counts and the stage-wise computing times, shown in Table 10.2 and 10.3, respectively. For an increasing number of aggregates the iteration counts are decreasing. The time spend in the master problem increases with the number of aggregates, but the time spend in the second stage decreases. The overall computing time does not vary too much for aggregate numbers between 20 and 100, but the amount of time spend in the first stage increases nonetheless. If the number of aggregates increases further, cut proliferation in the master problem leads to an increase in its solution time, such that the decrease in the iteration numbers can not compensate this. Thus, the first stage computing time dominates, see the last column of Table 10.3.

Aggregates	AM	GM	SGM	BC
Single	274.91	82.95	94.28	0
5	121.97	48.82	56.08	-41
10	88.26	39.22	44.99	-52
20	66.30	31.60	36.32	-61
50	46.20	24.66	28.36	-70
100	38.68	21.17	24.42	-74
Multi	19.84	12.69	14.66	-84

Table 10.2. Iteration counts for different number of aggregates for the L-shaped method. When an instance has less scenarios than the number of aggregates, the respective multi-cut iteration count was used. The last column gives the percentage change of the shifted geometric mean compared with the Benders base case (BC).

We can conclude that the use of cut aggregation is an effective tool for considerably reducing the computing time of the L-shaped method. We thus confirmed the results of Trukhanov et al. (2010) for a much larger set of test instances. Choosing a "perfect" number of aggregates is not easy to do a priori, but at least for our test set, a choice between 10 and 100 reduces the computing time considerably, up to 40%. For problems with a large number of scenarios, e.g., more than 2000, the multi-cut method is almost never a competitive choice, with the exception of ssn.

10.5.2. Cut consolidation

It was recently shown that consolidation of optimality cuts decreases computing time of Benders decomposition (Wolf & Koberstein, 2013). A positive effect of cut consolidation is expected, if cut proliferation shows adverse effects on the runtime, as it is a procedure meant to combat the negative effects of cut proliferation. We investigated the effect of

	Stage 1		Stage 2			
Aggregates	AM	SGM	AM	SGM	$\frac{\text{1. Stage}}{\text{2. Stage}}$	$\frac{\text{1. Stage}}{\text{Computing Time}}$
Single	2.69	1.70	50.27	12.25	13.89	12.79
5	3.37	1.99	25.88	8.56	23.26	19.82
10	4.40	2.12	20.19	7.43	28.48	23.12
20	7.35	2.43	16.15	6.40	37.91	28.50
50	14.14	2.84	11.84	5.29	53.60	35.75
100	30.68	3.51	10.58	4.76	73.71	43.34
Multi	259.07	18.60	4.60	2.29	813.25	87.21

Table 10.3. Stage-wise computing time for different number of aggregates for the L-shaped method. When an instance has less scenarios than the number of aggregates, the respective multi-cut solution times were used. The second last column gives the amount of time the algorithm spends in the first stage, compared with the second stage. The last column gives the amount of time the algorithm spends in the first stage, compared with the overall computing time. Both comparison are done in terms of the shifted geometric mean.

cut consolidation for different levels of the threshold α and different number of aggregates. The computing times are shown in Table 10.4 and the iteration counts in Table 10.5.

A positive effect of cut consolidation can be shown for 100 aggregates and the multi-cut method. Cut consolidation increases the computing time by a small amount for a small number of aggregates. It always increases the number of iterations needed to solve the problems in the test set. Despite this, it reduces the computing time for a low level of cut aggregation. This indicates, especially for the multi-cut method, that cut proliferation is a real problem.

Concluding from the results, the threshold α should be chosen higher for small aggregate numbers, as otherwise cuts get consolidated too early. The original cuts can still provide more useful information in later iterations, but as they are already highly aggregated, they are not active at every iteration and are thus selected for consolidation. α should be chosen smaller for the multi-cut method and higher aggregate numbers, as otherwise the positive effect of cut consolidation, i.e., reducing cut proliferation, is diminished because the master problem has already grown in size before the first cuts get consolidated.

Cut consolidation is more useful, the higher the number of aggregates. This behavior is expected, as the solution of the master problem becomes a bottleneck with an increasing number of aggregates, as was shown in Section 10.5.1, in particular in Table 10.3. Cut consolidation is not useful for situations where the master problem is easy to resolve, due to a low number of added cuts. But for a fairly modest number of cuts, e.g., more than 10, it has no negative effects on the computing time.

10.5.3. Level decomposition

It was recently shown that the regularization technique level decomposition compares favorably with Benders decomposition (Zverovich et al., 2012). In contrast to Benders

Agg.	α	AM	GM	SGM	w/o CC	BC
5	1	35.87	2.15	10.84	8	-19
5	2	32.82	2.09	10.58	5	-20
5	3	33.30	2.07	10.57	5	-21
5	4	33.26	2.07	10.49	4	-21
5	5	30.45	2.07	10.26	2	-23
10	1	25.99	1.81	9.33	2	-30
10	2	25.80	1.83	9.28	1	-30
10	3	25.95	1.81	9.35	2	-30
10	4	25.55	1.79	9.24	1	-31
10	5	25.89	1.82	9.28	1	-30
20	1	24.70	1.65	8.68	2	-35
20	2	24.38	1.65	8.59	1	-35
20	3	23.32	1.61	8.45	-1	-36
20	4	23.60	1.61	8.49	0	-36
20	5	23.93	1.62	8.48	0	-36
50	1	27.39	1.49	8.06	2	-39
50	2	27.31	1.50	8.02	1	-40
50	3	26.48	1.47	7.88	-1	-41
50	4	26.86	1.46	7.88	-1	-41
50	5	26.96	1.47	7.88	-1	-41
100	1	43.55	1.48	8.18	1	-38
100	2	43.25	1.48	8.04	-1	-40
100	3	41.36	1.45	7.93	-2	-40
100	4	41.78	1.45	7.90	-2	-41
100	5	41.43	1.48	7.88	-3	-41
Multi	1	237.05	3.35	19.35	-9	45
Multi	2	238.16	3.32	19.16	-10	44
Multi	3	241.52	3.32	19.38	-9	46
Multi	4	247.02	3.41	19.87	-7	49
Multi	5	251.30	3.48	20.06	-6	51

Table 10.4. Computing times for cut consolidation with different level of cut aggregation. α is the threshold used to determine if a cut can be consolidated. β is fixed to 0.99. The second last column compares cut consolidation with the respective computing time of the L-shaped method without cut consolidation (w/o CC). The last column compares with the Benders base case (BC). Comparisons are done with the shifted geometric mean. Positive values indicate a deterioration, while negative values indicate an improvement.

Agg.	α	AM	GM	SGM	w/o CC
	1	186.69	52.98	61.34	9
	2	166.99	51.18	59.33	6
5	3	167.38	51.44	59.49	6
	4	172.43	51.29	59.24	6
	5	141.36	50.40	58.04	3
	1	106.26	40.94	47.24	5
	2	101.52	40.84	47.07	5
10	3	101.70	40.26	46.38	3
	4	99.91	40.31	46.41	3
	5	101.79	40.24	46.39	3
	1	82.95	33.57	38.67	6
	2	79.38	32.77	37.83	4
20	3	76.44	32.47	37.40	3
	4	75.40	32.64	37.57	3
	5	75.06	32.24	37.14	2
	1	59.25	25.88	29.93	6
	2	56.81	25.57	29.55	4
50	3	54.03	25.14	29.05	2
	4	53.44	25.12	28.97	2
	5	53.51	25.08	28.93	2
	1	51.04	22.29	25.93	6
	2	46.71	21.68	25.20	3
100	3	44.34	21.63	25.08	3
	4	43.65	21.64	25.03	3
	5	42.57	21.45	24.82	2
	1	21.15	13.07	15.21	4
	2	20.40	12.88	14.91	2
Multi	3	20.37	12.86	14.89	2
	4	20.37	12.86	14.91	2
	5	20.10	12.83	14.82	1

Table 10.5. Iteration counts for cut consolidation with different level of cut aggregation. α is the threshold used to determine if a cut can be consolidated. β is fixed to 0.99. The last column compares cut consolidation with the respective iteration counts of the L-shaped method without cut consolidation (w/o CC) using the shifted geometric mean. Positive values indicate a deterioration, while negative values indicate an improvement.

decomposition, level decomposition has a parameter λ that influences the current level set of the algorithm. It is set a priori and kept constant throughout the algorithm. We investigated the effect of the choice of λ on the computing time of the algorithm by comparing the values $0.1, 0.3, 0.5, 0.7$, and 0.9, for the three different projection problems with the l_2, l_1, and l_∞ norm. We call these also the euclidean, manhattan and infinity distance projection problems, respectively. The resulting methods are thus named euclidean level decomposition (LevelE), manhattan level decomposition (LevelM), and infinity level decomposition (LevelI). We also present results on using cut aggregation in combination with level decomposition for the different projection problems.

Projection problem

The parameter λ influences the available solutions of the projection problem and thus the next iterate. The number of iterations required to reach an ϵ-optimal solution is bounded above by $c(\lambda)\left(\frac{D\Lambda}{\epsilon}\right)^2 ln\left(\frac{D\Lambda}{\epsilon}\right)$, where D is the diameter of the feasible polyhedron and Λ is a Lipschitz constant of the objective function. $c(\lambda)$ is a constant that only depends on λ and is minimal for $\lambda = 0.29289\ldots$ (Lemaréchal et al., 1995). We studied the effect of the choice of λ on the computing time and on the number of iterations for the different projection problems. Note that the method is equal to the L-shaped method, if λ is set to zero.

Euclidean-distance projection problem The computing times given in Table 10.6 show that the choice of λ is important for the efficiency of the method. A choice between 0.3 and 0.9 is considerably better than choosing 0.1. The best results are obtained for $\lambda = 0.7$. It shows an improvement of 53% against Benders decomposition. Regardless of the choice of λ, level decomposition is notably faster than Benders decomposition. If we measure the computing times with the arithmetic mean, the improvement upon Benders decomposition is even more pronounced, as level decomposition with $\lambda = 0.7$ improves upon Benders by 73%. These differences show that level decomposition is particularly good on the large instances that take long to solve with the L-shaped method, e.g., those with many scenarios. Despite these differences in quantity, the order stays the same for the arithmetic and shifted geometric mean. The performance profile is shown in Figure 10.2. It can be seen that the

λ	AM	GM	SGM	BC
0.1	20.48	1.93	8.05	-39
0.3	15.93	1.67	6.70	-50
0.5	14.92	1.66	6.42	-52
0.7	14.36	1.68	6.28	-53
0.9	17.82	1.94	6.89	-48

Table 10.6. Computational statistics for euclidean level decomposition with varying λ. The last column compares the performance with the Benders base case (BC), by giving the percentage change of the shifted geometric mean. Negative values indicate an improvement.

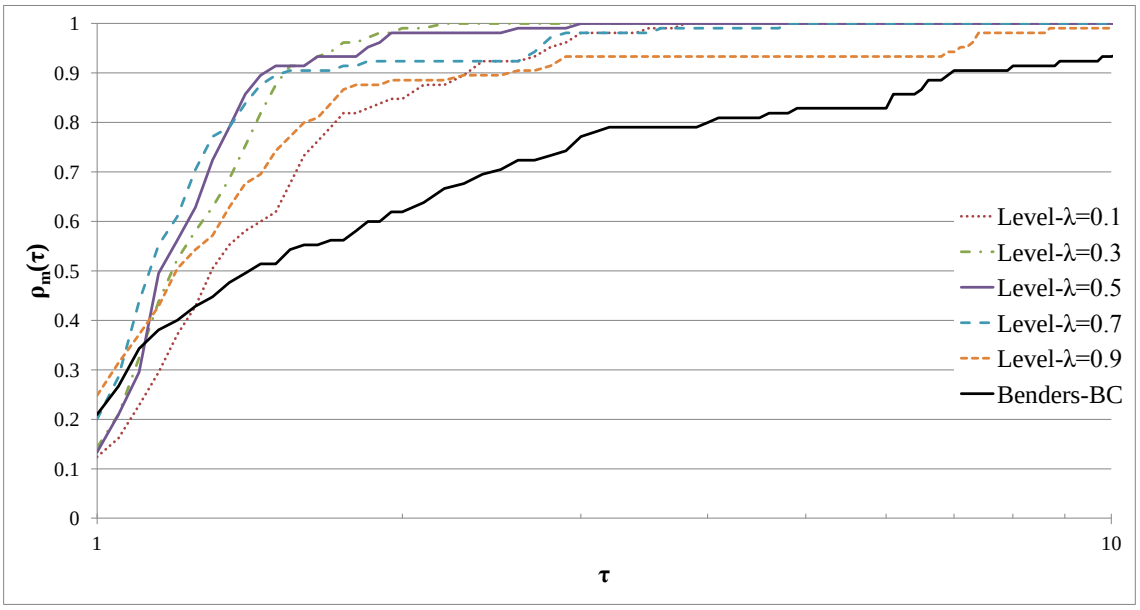

Figure 10.2. Performance profile of the euclidean level decomposition method with varying λ.

values $0.3, 0.5$, and 0.7 are better than 0.1 and 0.9 for most problems. Choosing 0.1 is the fastest algorithm on about 12 % of all investigated instances. For 0.9, after solving approx. 87 % of all problems within a τ of two, it stalls for the remaining problems. Considering the detailed results, the assets and environ problems of the slptestset are the ones where 0.9 is a considerably bad choice. On the other hand, the performance profile indicates also that 0.9 is among the fastest on about 25% of all problems, including the problems storm, sslp, rand1 and rand2.

Manhattan-distance projection problem We analyzed the effect of the choice of λ for level decomposition with the manhattan-distance projection problem similar to the euclidean-distance projection problem. The performance profile is shown in Figure 10.3. It shows that a choice of λ between 0.3 and 0.7 is again better than choosing the more extreme values 0.1 or 0.9. The computing times presented in Table 10.7 support this conclusion. The comparison with Benders decomposition and euclidean level decomposition shows that manhattan level decomposition is much better than Benders, but is around 10% slower than the latter. $\lambda = 0.7$ is the best choice for manhattan level decomposition, with an improvement in total computing time of 69% upon Benders decomposition.

In contrast to euclidean level decomposition, choosing $\lambda = 0.9$ is slower in total computing time than $\lambda = 0.1$, but faster if measured with the shifted geometric mean. This means that some of the large instances were solved faster with $\lambda = 0.1$ than $\lambda = 0.9$. These include the environ and saphir problems as well as the gbd and LandS instances.

Infinity-distance projection problem The performance profile regarding the choice of λ for the infinity-distance level decomposition method is shown in Figure 10.4. The pattern

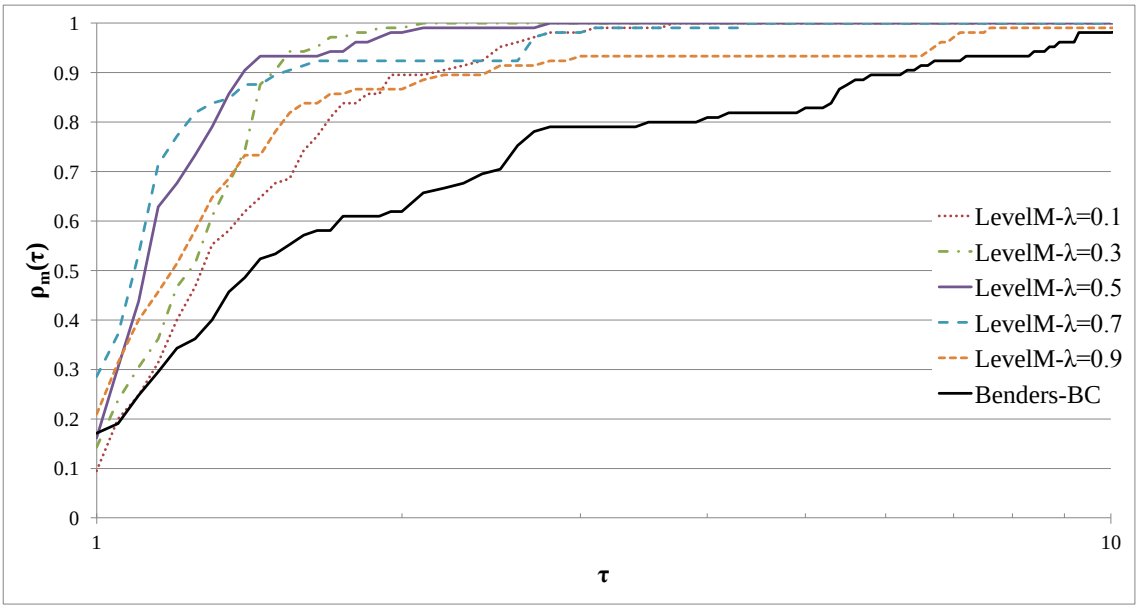

Figure 10.3. Performance profile of the manhattan level decomposition method with varying λ.

λ	AM	GM	SGM	LevelE	BC
0.1	23.13	1.95	8.63	7	-35
0.3	18.35	1.75	7.44	11	-44
0.5	17.19	1.66	7.01	9	-47
0.7	16.42	1.68	6.85	9	-49
0.9	23.27	1.97	7.66	11	-42

Table 10.7. Computational statistics for manhattan level decomposition with varying λ. The second last column compares the perfomance with the euclidean level method (LevelE). The last row compares the performance with the Benders base case (BC). Both performance comparisons give the percentage change of the shifted geometric mean. Negative values indicate an improvement, while positive values indicate a deterioration.

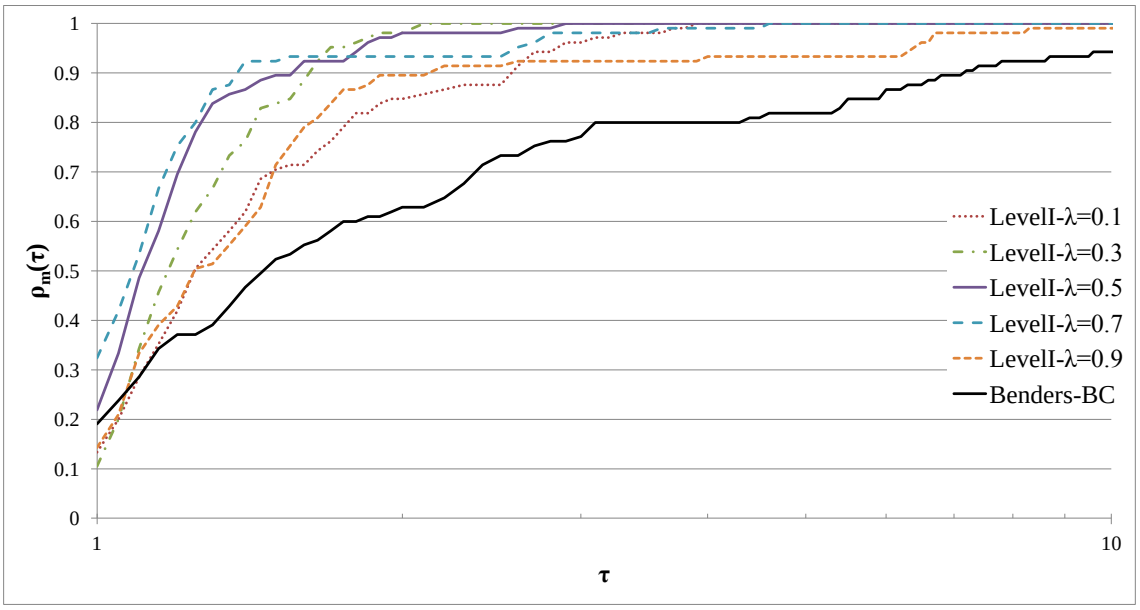

Figure 10.4. Performance profile of the infinity level decomposition method with varying λ.

is similar to the euclidean and manhattan level decomposition method. The computing times presented in Table 10.8 are compared with those for Benders decomposition and both euclidean and manhattan level decomposition. Setting λ between 0.3 and 0.7 results in a good performance with $\lambda = 0.7$ taking the lead again. Infinity level decomposition is a little bit slower than euclidean level decomposition, but a little bit faster than manhattan level decomposition, for all λ. Infinity level decomposition improves upon Benders by 72%, and is only 3% slower than euclidean level decomposition if measured with the arithmetic mean.

λ	AM	GM	SGM	LevelE	LevelM	BC
0.1	22.56	1.91	8.48	5	-2	-36
0.3	17.90	1.68	7.14	7	-4	-46
0.5	15.89	1.60	6.63	3	-5	-50
0.7	14.85	1.60	6.39	2	-7	-52
0.9	21.03	1.98	7.61	10	-1	-43

Table 10.8. Computational statistics for infinity level decomposition with varying λ. The third and second last column compare the perfomance with the euclidean (LevelE) and manhattan level method (LevelM), respectively. The last column compares the performance with the Benders base case (BC). All performance comparisons give the percentage change of the shifted geometric mean. Negative values indicate an improvement, while positive values indicate a deterioration.

					Benders-BC		LevelE	
Dist.	λ	AM	GM	SGM	AM	SGM	AM	SGM
	0.1	92.96	53.06	58.38	-66	-38	0	0
	0.3	73.22	47.57	51.36	-73	-46	0	0
l_2	0.5	74.82	49.10	52.52	-73	-44	0	0
	0.7	77.04	52.01	55.09	-72	-42	0	0
	0.9	90.70	64.18	67.51	-67	-28	0	0
	0.1	107.63	54.74	60.68	-61	-36	16	4
	0.3	89.50	51.03	55.73	-67	-41	22	9
l_1	0.5	85.24	50.50	54.51	-69	-42	14	4
	0.7	90.40	54.17	57.93	-67	-39	17	5
	0.9	107.94	71.10	75.18	-61	-20	19	11
	0.1	105.33	55.60	61.58	-62	-35	13	5
	0.3	77.85	49.57	53.81	-72	-43	6	5
l_∞	0.5	72.80	49.02	52.60	-74	-44	-3	0
	0.7	72.91	51.85	55.09	-73	-42	-5	0
	0.9	103.46	73.07	77.17	-62	-18	14	14

Table 10.9. Iteration counts for level decomposition methods. They are compared with the iteration counts of the Benders base case (Benders-BC) and with those of the euclidean level decomposition method (LevelE), both with the arithmetic and the shifted geometric mean. Comparisons are given as percentage differences, where positive values indicate a deterioration and negative values indicates an improvement.

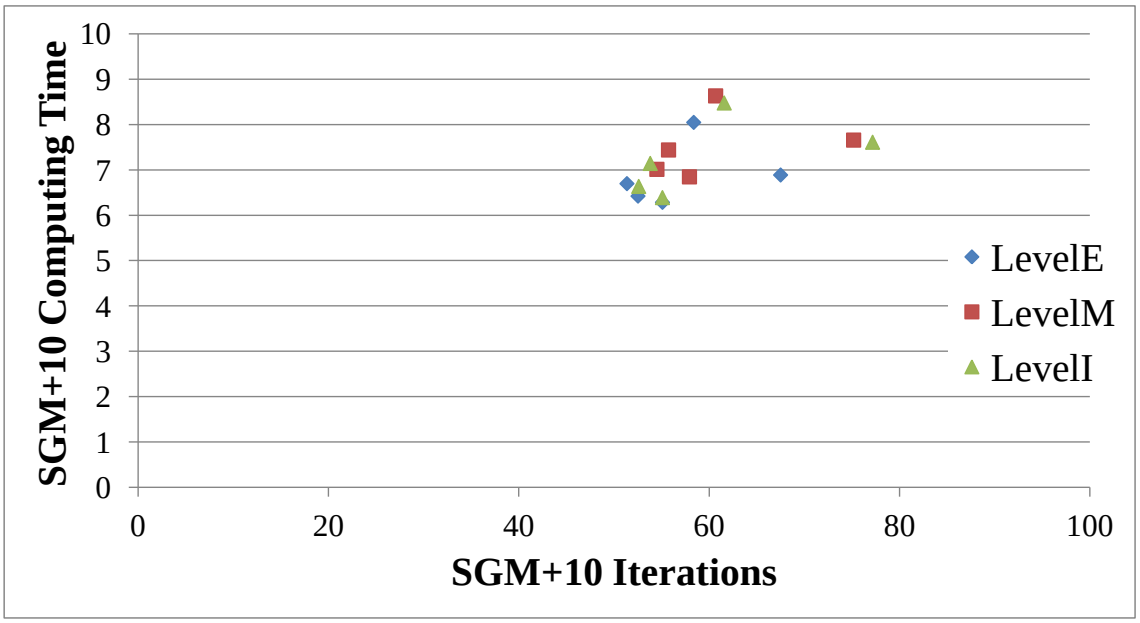

Figure 10.5. Scatter plot of the computing times and the iteration counts, both given via the shifted geometric mean for all level decomposition variants.

Comparing the iteration counts of the three projection problems, given in Table 10.9, the euclidean level decomposition method has the lowest number of iterations, measured with the shifted geometric mean, followed by the infinity and manhattan level decomposition method. The lowest number of overall iterations is achieved by infinity level decomposition with $\lambda = 0.7$, although this is not the fastest method in computational terms. Across the three different projection problems, a pattern emerges: the iteration counts for $\lambda = 0.1$ and $\lambda = 0.9$ are the highest, and the lowest for $\lambda = 0.5$, measured with the arithmetic mean. The shifted geometric mean on the other hand shows that $\lambda = 0.1$ needs considerably less iterations than $\lambda = 0.9$. This difference can be traced back to the environ and assets problem families, where $\lambda = 0.9$ needs much more iterations than $\lambda = 0.1$. All three projection problems decrease the number of iterations considerably compared with Benders decomposition. The improvement is between 61% and 74%.

The results also show that using the infinity distance leads to less iterations than the manhattan distance. In pure iteration counts, infinity level decomposition is comparable to euclidean level decomposition for $\lambda = 0.3, 0.5$, and 0.7, but it takes much more iterations for $\lambda = 0.1$ and $\lambda = 0.9$. The results indicate that the l_2 and l_∞ norm are better suited than the l_1 norm in choosing among the possible candidates. A look at the unit shapes for the three different norms, given in Figure 10.6 for two dimensions, shows that the l_1 norm is the most restricting, meaning that it has the lowest volume for the same distance. The l_∞ norm includes more volume for the same distance than the l_2 norm. It follows that the l_1, l_2, and l_∞ lead to different solutions for a given set of candidate solutions. The rightmost shape in Figure 10.6 shows how much more volume is covered when moving from l_1 to l_2 to l_∞ and which solutions would be selected with a different norm.

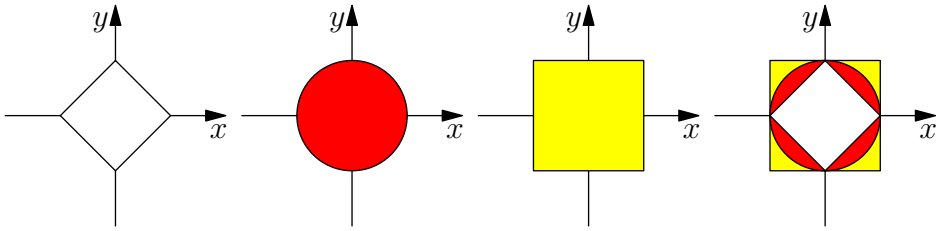

Figure 10.6. Unit shapes for the l_1, l_2, and l_∞ norm in two dimensions.

Trust region also uses the l_∞ norm to restrict the set of possible candidates for the next iterate. As we have not implemented this solution approach, we can not do a direct comparison, but the results of Zverovich et al. (2012) show that euclidean level decomposition compares favorably with their own trust region implementation. As in our implementation l_∞ level decomposition is roughly comparable in performance to l_2 level decomposition, it seems that l_∞ level decomposition should be considered as an alternative solution approach to trust region to solve stochastic programming problems. Zverovich et al. (2012) find also that regularized decomposition is not as computationally efficient as euclidean level decomposition on their test problems. These two findings suggest that solving a separate projection problem instead of a modified master problem results in an efficient regularization approach.

Comparing the computing times of the different projection problems given in Tables 10.6, 10.7, and 10.8 with the respective iteration counts given in Table 10.9 in a scatter plot in Figure 10.5, it can be seen that the results for $\lambda = 0.3, 0.5$, and 0.7 are relatively close together, with 0.1 and 0.9 far apart. Interestingly, the number of iterations is not a good predictor for the overall solution time. This is especially so for $\lambda = 0.9$ due to the observation that iteration count differences are relatively large for many problems with small solution times (e.g., deak, environ, assets), but relatively small on some problems with larger computing times (e.g., rand and sampling).

Cut aggregation

Cut aggregation can also be applied to regularization techniques. Especially the regularized decomposition method (Ruszczyński, 1986) is proposed with no aggregation at all, i.e., multi-cut. As cut aggregation is only recently computationally evaluated on its own (Trukhanov et al., 2010), no computational experience is available regarding the combination of level decomposition and cut aggregation.

We present computational results for level decomposition combined with cut aggregation in Table 10.10 for the three different projection problems. The iteration counts are given in Table 10.11. Our results show that choosing a cut aggregation level of five to ten leads to a further reduction in computing time for all three variants compared with the respective single-cut level decomposition method. It improves upon the single-cut case by 11% to 14%. A further increase up to 50 aggregates is still computationally advantageous for l_1 and l_∞ projection problems. The arithmetic mean measures an improvement of 20% to 26% of level decomposition with five aggregates compared to single-cut level decomposition.

However, the arithmetic mean increases already for an aggregation level around 20. This can be explained mostly by problem sslp. Cut proliferation delays master and projection problem solution times for this problem family, such that the arithmetic mean ratio is much larger than the shifted geometric mean ratio.

The iteration counts presented in Table 10.11 show again that regularization reduces the number of iterations, compared with Benders decomposition. The advantage of level decomposition in terms of iterations is less pronounced, if the number of aggregates increases. This is in line with expectations, as the approximation quality gets better, when more approximation terms are available. Thus the zig-zagging behavior is already reduced by moving from a single linear term approximation to more linear terms approximation, as was already shown in Section 10.5.1. The number of iterations can still be further reduced by using level decomposition. For the fully disaggregated case, i.e., the multi-cut method, level decomposition needs in total more iterations than plain Benders decomposition, regardless of the projection problem. This is an interesting behavior as another regularization technique, regularized decomposition, is effective with the multi-cut method (Ruszczyński, 1993b).

Especially the euclidean level method does not react well to an increasing number of aggregates. This can be traced back to the underlying QP solver, which is used to solve the quadratic programming problems. For the multi-cut method it takes more than twice as long to solve the quadratic projection problems than to solve the linear projection problems in the l_1 and l_∞ case. In addition, two problems pose difficulties for euclidean level decomposition, which explain the larger than expected iteration counts. The ssn problem family gets numerically challenging for some aggregation levels, which is reflected in the iteration counts; they do not fall consistently. For another class of problems, environ from slptestset, a lower bound on θ has to be inserted as the first optimality cuts do not give a finite recourse function approximation value [12]. This lower bound has to be reduced to a smaller value, e.g. -10^6, such that the quadratic programming problems can be solved successfully.

The trade-off of cut aggregation for Benders decomposition, which was analyzed in Section 10.5.1, is much larger than the one for level decomposition. This can be attributed to the fact that a projection problem has to be solved in every iteration. This takes more time when more cuts are added per iteration, although the overall number of iterations is still lower. We can conclude that cut proliferation is even more damaging to level decomposition than to Benders decomposition. Cut aggregation has to be applied judiciously and can slow down the solution process when the number of aggregates is chosen too high.

Overall comparison

The comparison of level decomposition with single-cut Benders and the deterministic equivalent solvers is depicted in Figure 10.7 on page 119. It shows that using level decomposition, regardless of the choice of the distance in the projection problem, is far better than any of the alternatives. In particular, the simplex methods show a worse performance. Part of this can be attributed to the sequential execution of the simplex method, whereas the

[12] Absolute values that are larger than 10^{20} are defined as infinity.

Distance	Agg.	AM	GM	SGM	Level-SC	Benders
	1	14.92	1.66	6.42	0	-52
	5	11.88	1.47	5.74	-11	-43
	10	13.02	1.46	5.78	-10	-37
l_2	20	25.18	1.67	7.33	14	-13
	50	58.17	2.16	9.23	44	17
	100	126.56	2.76	11.09	73	37
	Multi	628.14	17.40	56.57	781	165
	1	17.19	1.66	7.01	0	-47
	5	12.68	1.43	6.04	-14	-40
	10	13.90	1.42	6.19	-12	-32
l_1	20	17.09	1.41	6.36	-9	-25
	50	26.83	1.52	6.85	-2	-13
	100	46.28	1.75	7.77	11	-4
	Multi	407.75	6.94	30.77	339	44
	1	15.89	1.60	6.63	0	-50
	5	12.53	1.38	5.77	-13	-43
	10	13.06	1.34	5.78	-13	-37
l_∞	20	15.71	1.35	5.98	-10	-29
	50	23.66	1.48	6.61	0	-16
	100	40.67	1.68	7.47	13	-8
	Multi	422.13	7.26	32.73	393	53

Table 10.10. Computing times for level decomposition methods with different number of aggregates. The second last column gives the percentage difference of the corresponding level decomposition run with just one aggregate (Level-SC) while the last column gives the percentage difference of the corresponding Benders decomposition run with the same number of aggregates, using the shifted geometric mean. Positive values indicate a deterioration while negative values indicate an improvement.

					Benders	
Proj.	Agg.	AM	GM	SGM	AM	SGM
	1	74.82	49.10	52.52	-73	-44
	5	52.25	36.05	38.64	-57	-33
	10	45.64	32.12	34.50	-49	-26
l_2	20	58.50	31.02	33.77	-13	-11
	50	64.16	28.92	31.67	35	4
	100	72.78	27.36	30.06	80	12
	Multi	25.47	19.71	21.33	31	51
	1	85.24	50.50	54.51	-69	-42
	5	56.16	37.44	40.23	-54	-30
	10	51.14	34.26	36.74	-42	-21
l_1	20	44.34	30.81	33.00	-34	-13
	50	38.07	27.33	29.27	-20	-4
	100	35.80	25.95	27.76	-11	3
	Multi	24.74	19.47	20.95	28	48
	1	72.80	49.02	52.60	-74	-44
	5	51.01	37.07	39.54	-58	-31
	10	45.57	33.26	35.42	-49	-24
l_∞	20	41.28	30.36	32.34	-38	-15
	50	36.93	27.26	29.11	-22	-4
	100	34.40	25.47	27.23	-15	1
	Multi	25.32	20.00	21.51	31	52

Table 10.11. Iteration counts for level decomposition methods with different number of aggregates. The second and last columns compare the iteration counts to Benders decomposition with the same number of aggregates using the arithmetic and the shifted geometric mean, respectively.

other methods are parallelized. Using the barrier method results in better performance than the simplex methods, but plain Benders decomposition solves most problems faster. From the performance profiles and computing times presented in the preceding sections,

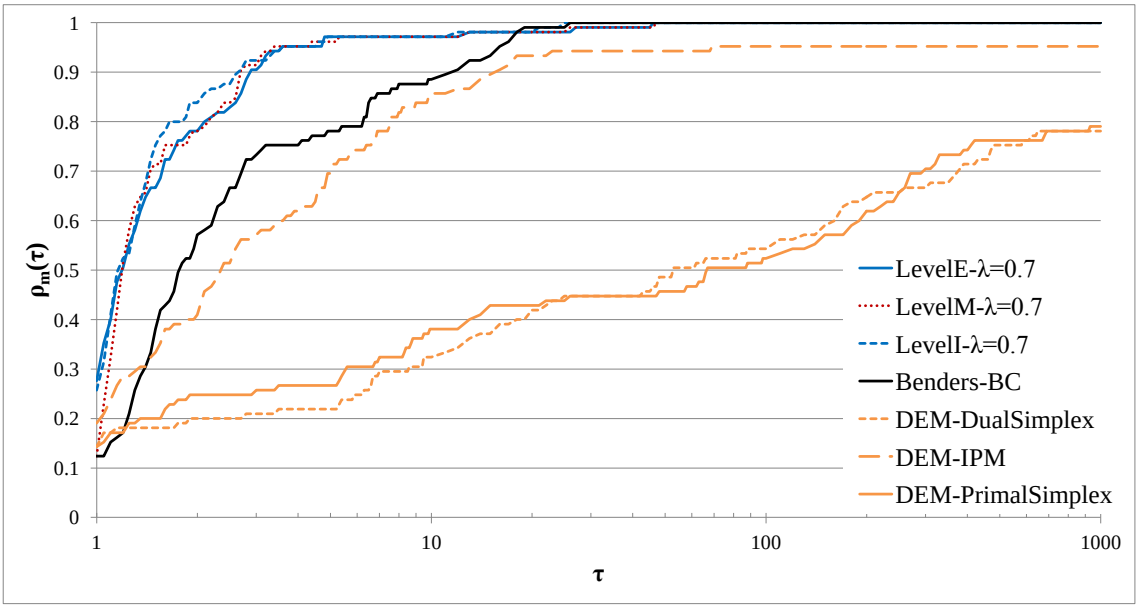

Figure 10.7. Performance profile of the best level decomposition methods, Benders-BC and deterministic equivalent solvers.

we can conclude that $\lambda = 0.7$ results in the lowest computing times for both arithmetic mean and shifted geometric mean, regardless of the distance used in the projection problem. More broadly, setting λ between 0.3 and 0.7 results in good performance. The findings for level decomposition show that it has good scale up properties regardless of the projection problem (Zverovich et al., 2012). Using a modest level of cut aggregation, e.g., five aggregates, the total computing time can be further reduced by up to 20% for euclidean level decomposition.

10.5.4. On-demand accuracy

On-demand accuracy requires a parameter κ that regulates the usage of on-demand accuracy cuts by adjusting the target value. This parameter is set a priori and kept constant throughout the algorithm. The pivotal question is therefore how κ should be chosen. We evaluate the on-demand accuracy approach in combination with the single-cut L-shaped method, with level decomposition variants, and with a varying level of cut aggregation for Benders and level decomposition. We present the results in this order.

Single-cut L-shaped method

The performance profile for the L-shaped method with on-demand accuracy is shown in Figure 10.8 on the next page. It can be seen that κ influences the efficiency of the

algorithm significantly. The trade-off between substantial and insubstantial iterations can

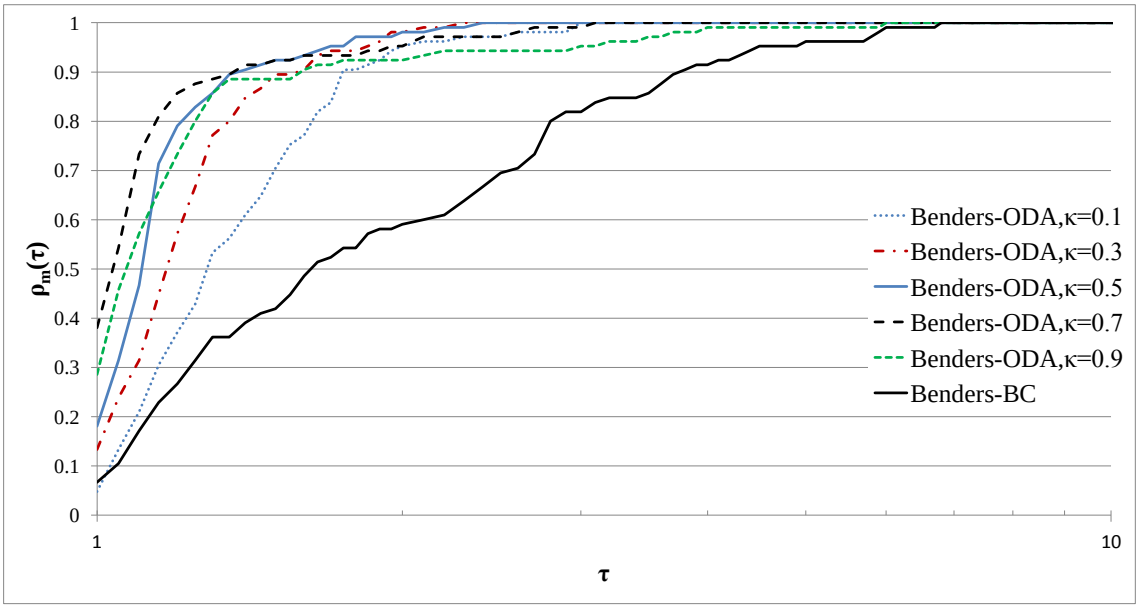

Figure 10.8. Performance profile of the single-cut L-shaped method with on-demand accuracy and varying κ.

κ	AM	GM	SGM	BC
0.1	31.53	1.99	10.12	-24
0.3	26.64	1.79	9.12	-31
0.5	24.08	1.69	8.74	-34
0.7	24.20	1.65	8.77	-34
0.9	30.33	1.76	9.83	-26

Table 10.12. Computational statistics for the L-shaped method with on-demand accuracy. The comparison with the Benders base case (BC) is done with the shifted geometric mean. The values represent percentage changes, where negative values show an improvement.

be evaluated by looking at the computing times shown in Table 10.12 and the iteration counts given in Table 10.13 .

Every insubstantial iteration involves the computation of some on-demand accuracy cuts (for the single-cut method only one) and the subsequent resolve of the master problem with the added cuts, which takes most of the time of an insubstantial iteration (in the single-cut case). In a substantial iteration all subproblems are solved to optimality, and new cuts are computed afterwards. The master problem is then resolved with these new cuts. The relative amount of time the algorithm stays in each stage depends on the number and difficulty of the subproblems, and the time to resolve the master problem.

With rising κ, the amount of insubstantial iterations increases, as the target value $\kappa LB + (1-\kappa)UB$, given by equation (8.7), is reached more often. The increase of insubstantial

	Subst. Iterations		Overall Iterations		$\frac{\text{Subst. It.}}{\text{Overall It.}}$	$\frac{\text{Overall It.}}{\text{BC It.}}$
κ	AM	SGM	AM	SGM	AM	AM
0.1	82.09	41.31	266.50	93.80	0.31	0.97
0.3	63.41	33.75	290.57	98.21	0.22	1.06
0.5	53.25	29.23	328.89	104.56	0.16	1.20
0.7	47.83	25.77	405.98	117.26	0.12	1.48
0.9	40.96	22.83	563.93	139.65	0.07	2.05

Table 10.13. Iteration counts for the L-shaped method with on-demand accuracy. The last column compares the number of overall iterations with the number of iterations of the Benders base case (BC).

iterations is accompanied by a decreasing number of substantial iterations, but the overall number of iterations increases. The balancing needs between insubstantial and substantial iterations, which is controlled via κ, is apparent from the performance profile shown in Figure 10.8 and the computational results given in Table 10.12 and the iteration counts given in Table 10.13.

A too cautious approach, i.e., $\kappa = 0.1$, does not decrease the amount of substantial iterations enough, as the target value is not reached often enough. It still improves upon plain Benders decomposition, as the added on-demand accuracy cuts are more helpful than newly generated Benders cuts, because the overall number of iterations is lower.

A too aggressive approach, i.e., $\kappa = 0.9$, does decrease the number of substantial iterations, but for a price in the form of insubstantial iterations, such that the overall iteration number grows too large, compared with $\kappa = 0.5$ or $\kappa = 0.7$. Still, on-demand accuracy is helpful regardless of the choice of κ, but κ is an important parameter which should be set between 0.3 and 0.7 according to our results.

Note that Benders with on-demand accuracy and $\kappa = 0.5$ runs in about 45% of the time of single-cut Benders, thus on-demand accuracy is particularly helpful for larger problems. This can be deduced from the difference between the arithmetic and the shifted geometric mean.

We conclude that applying on-demand accuracy to the plain L-shaped method results in substantial improvements with respect to the computing time. The downside is an increase in memory usage, which can be combated by deleting older on-demand accuracy information.

Level decomposition

If on-demand accuracy is used in conjunction with level decomposition, the parameter κ must be chosen in addition to the level parameter λ. According to the convergence proof of the algorithm (Fábián, 2013), κ should be chosen smaller than $1 - \lambda$. However, in our computational experiments the algorithm converged also for $\kappa > 1 - \lambda$. Thus we studied the effect of varying κ and λ on the computing time for the different projection problems with the euclidean, manhattan and infinity distance. Due to the high number of

combinations we do not present performance profiles as they would be unreadable. Note that the target value for level decomposition is $\kappa \left(c^T x^k + f(x^k) \right) + (1 - \kappa) UB$.

λ	κ	AM	GM	SGM+10	LevelE	BC
0.5	0.7	10.68	1.41	5.16	-20	-61
0.3	0.9	10.63	1.39	5.20	-22	-61
0.7	0.7	10.84	1.49	5.23	-17	-61
0.5	0.9	10.97	1.43	5.24	-18	-61
0.3	0.7	11.30	1.38	5.25	-22	-61
0.7	0.9	10.86	1.51	5.26	-16	-60
0.3	0.5	10.82	1.41	5.27	-21	-60
0.5	0.5	11.20	1.41	5.28	-18	-60
0.7	0.5	11.19	1.48	5.29	-16	-60
0.5	0.3	11.04	1.46	5.33	-17	-60
0.7	0.3	11.63	1.50	5.44	-13	-59
0.5	0.1	11.33	1.50	5.51	-14	-59
0.3	0.3	11.61	1.45	5.54	-17	-58
0.7	0.1	11.90	1.57	5.62	-11	-58
0.1	0.9	11.82	1.44	5.72	-29	-57
0.3	0.1	12.08	1.50	5.74	-14	-57
0.1	0.7	11.98	1.44	5.81	-28	-56
0.1	0.5	12.46	1.46	5.94	-26	-55
0.1	0.3	12.92	1.50	6.10	-24	-54
0.9	0.5	14.39	1.85	6.21	-10	-53
0.9	0.7	14.22	1.86	6.22	-10	-53
0.9	0.9	14.23	1.90	6.30	-9	-53
0.9	0.3	14.74	1.87	6.32	-8	-52
0.9	0.1	15.19	1.92	6.48	-6	-51
0.1	0.1	14.10	1.60	6.53	-19	-51

Table 10.14. Computational results for euclidean level decomposition with on-demand accuracy, for different λ and κ combinations. The second last column gives the percentage change of the shifted geometric mean for the on-demand accuracy level decomposition approach compared to the level decomposition method with the same λ (LevelE). The last column gives the percentage change compared with the Benders base case (BC). Negative values indicate an improvement.

Euclidean-projection problem The computing times achieved on our test set are given in Table 10.14. First, we can see that using on-demand accuracy in combination with level decomposition is always beneficial. Second, higher κ values lead to smaller computing times than lower κ values, even if the method is not proven to converge for these λ-κ combinations. Across different λ values, a κ value between 0.5 and 0.9 seems to be a good choice. The on-demand accuracy results confirm the results for the level decomposition

method with respect to the choice of λ given in Section 10.5.3 that λ should be set between 0.3 and 0.7. Compared with the L-shaped method with on-demand accuracy κ can be set to higher values and good results can still be obtained. This is due to the regularization of level decomposition that hinders zig-zagging, as can be seen by comparing iteration counts from Table 10.13 and 10.15. Thus the increase of insubstantial iterations is much lower for $\kappa = 0.9$ compared with the L-shaped method with on-demand accuracy.

The ratio of substantial to overall iterations in Table 10.15 show that for an increasing κ parameter the ratio of substantial iterations to overall iterations decreases. Thus more insubstantial iterations are performed with respect to overall iterations. On-demand accuracy cuts are only generated in insubstantial iterations thus κ regulates the usage of on-demand accuracy cuts, similar to the Benders ODA case but less pronounced. This behavior falls in line with expectations as the target value (8.7) is easier to satisfy if κ is larger.

Manhattan-distance projection problem The computing times for manhattan level decomposition with on-demand accuracy are shown in Table 10.16, whereas the iteration counts are given in Table B.2 in the Appendix. The results show a clear picture but different picture than for euclidean level decomposition with on-demand accuracy. κ determines the performance for λ between 0.3 and 0.7. For all λ values, except 0.9, the computing times are strictly increasing with decreasing κ. The comparison with manhattan level decomposition shows a considerable performance improvement for the on-demand accuracy method that is more pronounced the higher κ is chosen. Compared with euclidean level decomposition with on-demand accuracy it is around 6-10% slower for reasonable λ and κ combinations, e.g., λ between 0.3 and 0.7 and κ between 0.3 and 0.9.

Infinity-distance projection problem Computing times for the infinity level decomposition method with on-demand accuracy are shown in Table 10.17, whereas the iteration counts are given in Table B.3 in the Appendix. Differing from the results for the manhattan level decomposition method with on-demand accuracy given above, the choice of λ is more important than the choice of κ. This can be seen from the results, and in particular that combinations with $\lambda = 0.7$ are among the fastest four out of the fastest five. Choosing $\kappa = 0.1$ is no good choice, as it is consistently the slowest combination for every λ. The percentage point increase to infinity level decomposition is highest for $\lambda = 0.1$ and $\lambda = 0.3$, but it is also well above 10% for other λ values. Compared with euclidean level decomposition with on-demand accuracy, level infinity decomposition with on-demand accuracy is slightly ahead for $\lambda = 0.7$ and a little bit slower for other λ choices.

Cut Aggregation

The on-demand accuracy method is not restricted to the fully aggregated or single-cut case even though it was initially introduced in this setting. On-demand accuracy does not make sense for the multi-cut method as no additional information can be gained from the individual scenarios because they are already considered in the current master problem in the form of optimality cuts. Therefore we did not evaluate the on-demand accuracy scheme with the multi-cut method.

		Subst. Iterations		Iterations		$\frac{\text{Subst. It.}}{\text{Overall It.}}$
λ	κ	AM	SGM	AM	SGM	AM
0.1	0.1	45.67	32.15	94.00	59.40	0.49
0.1	0.3	39.56	28.19	93.70	59.74	0.42
0.1	0.5	36.19	25.56	98.77	62.18	0.37
0.1	0.7	34.46	23.92	105.36	66.13	0.33
0.1	0.9	33.02	22.81	111.24	70.89	0.30
0.3	0.1	45.80	33.41	76.67	53.66	0.60
0.3	0.3	41.46	30.15	77.36	54.67	0.54
0.3	0.5	38.49	27.95	78.77	56.26	0.49
0.3	0.7	37.75	26.66	83.01	59.29	0.45
0.3	0.9	36.36	25.67	88.10	62.94	0.41
0.5	0.1	47.74	35.68	75.41	53.75	0.63
0.5	0.3	45.28	33.37	76.98	54.74	0.59
0.5	0.5	43.05	31.04	78.83	55.58	0.55
0.5	0.7	41.52	30.18	81.63	58.47	0.51
0.5	0.9	42.50	29.83	89.25	63.40	0.48
0.7	0.1	52.51	40.87	76.50	56.27	0.69
0.7	0.3	49.28	37.64	76.72	56.03	0.64
0.7	0.5	47.09	36.39	78.53	58.23	0.60
0.7	0.7	46.19	35.31	83.06	61.73	0.56
0.7	0.9	45.73	34.46	88.30	66.08	0.52
0.9	0.1	69.08	55.71	91.49	69.74	0.76
0.9	0.3	66.07	53.41	91.09	69.63	0.73
0.9	0.5	64.22	51.56	93.20	71.13	0.69
0.9	0.7	63.19	50.74	97.84	75.25	0.65
0.9	0.9	63.75	50.90	105.51	82.02	0.60
0.1	-	92.96	58.38	92.96	58.38	1.00
0.3	-	73.22	51.36	73.22	51.36	1.00
0.5	-	74.82	52.52	74.82	52.52	1.00
0.7	-	77.04	55.09	77.04	55.09	1.00
0.9	-	90.70	67.51	90.70	67.51	1.00

Table 10.15. Iteration counts for euclidean level decomposition with and without on-demand accuracy for different λ and κ combinations. The last column gives the amount of substantial iterations with respect to all iterations as measured by the arithmetic mean.

λ	κ	AM	GM	SGM	LevelM	LevelE-ODA	BC
0.5	0.9	11.67	1.38	5.54	-21	6	-58
0.3	0.9	11.83	1.35	5.56	-21	7	-58
0.7	0.9	12.06	1.45	5.60	-18	6	-58
0.5	0.7	11.92	1.39	5.62	-20	9	-58
0.7	0.7	12.06	1.46	5.66	-17	8	-57
0.3	0.7	12.20	1.37	5.68	-19	8	-57
0.7	0.5	12.32	1.48	5.72	-17	8	-57
0.5	0.5	12.35	1.39	5.73	-18	8	-57
0.3	0.5	12.33	1.40	5.82	-17	10	-56
0.5	0.3	12.57	1.44	5.86	-16	10	-56
0.7	0.3	12.74	1.51	5.88	-14	8	-56
0.3	0.3	12.90	1.45	6.02	-14	9	-55
0.5	0.1	13.02	1.49	6.10	-13	11	-54
0.1	0.9	12.96	1.42	6.11	-29	7	-54
0.7	0.1	13.35	1.57	6.13	-10	9	-54
0.1	0.7	13.21	1.42	6.19	-28	6	-53
0.3	0.1	14.08	1.52	6.34	-10	11	-52
0.1	0.5	13.88	1.46	6.38	-26	7	-52
0.1	0.3	15.07	1.51	6.70	-22	10	-50
0.9	0.7	19.13	1.89	6.89	-10	11	-48
0.9	0.5	19.57	1.89	7.07	-8	14	-47
0.9	0.9	19.82	1.95	7.13	-7	13	-46
0.1	0.1	16.50	1.61	7.17	-17	10	-46
0.9	0.3	21.36	1.93	7.26	-5	15	-45
0.9	0.1	27.04	1.99	7.53	-2	16	-43

Table 10.16. Computational results for manhattan level decomposition with on-demand accuracy for different λ and κ combinations. The third last column compares the computing time with the manhattan level decomposition algorithm without on-demand accuracy (LevelM) but with the same λ. The second last column compares the computing times with the euclidean level decomposition with on-demand accuracy (LevelE-ODA) for the same λ and κ combination. The last column compares with the Benders base case (BC). The comparisons are given by the respective percentage change of the shifted geometric mean. Negative values indicate an improvement, while positive values indicate a deterioration.

λ	κ	AM	GM	SGM	LevelI	LevelE-ODA	BC
0.7	0.7	10.09	1.37	5.19	-19	-1	-61
0.7	0.5	10.10	1.38	5.22	-18	-1	-61
0.7	0.9	10.16	1.40	5.25	-18	0	-60
0.3	0.5	10.15	1.31	5.31	-26	1	-60
0.7	0.3	10.36	1.42	5.37	-16	-1	-60
0.3	0.3	10.51	1.33	5.39	-25	-3	-59
0.3	0.9	10.61	1.32	5.42	-24	4	-59
0.7	0.1	10.47	1.46	5.45	-15	-3	-59
0.5	0.7	11.79	1.34	5.47	-18	6	-59
0.3	0.7	11.07	1.32	5.50	-23	5	-59
0.5	0.9	11.88	1.35	5.55	-16	6	-58
0.5	0.3	12.02	1.38	5.58	-16	5	-58
0.5	0.5	11.95	1.37	5.62	-15	6	-58
0.3	0.1	11.47	1.43	5.72	-20	0	-57
0.5	0.1	12.44	1.44	5.80	-13	5	-56
0.1	0.7	11.43	1.36	5.87	-31	1	-56
0.1	0.5	11.56	1.37	5.88	-31	-1	-56
0.1	0.3	12.48	1.41	6.08	-28	0	-54
0.9	0.7	12.60	1.79	6.29	-17	1	-53
0.9	0.9	12.61	1.83	6.34	-17	1	-52
0.1	0.9	14.41	1.42	6.39	-25	12	-52
0.9	0.5	12.87	1.83	6.39	-16	3	-52
0.9	0.3	12.64	1.85	6.56	-14	4	-51
0.1	0.1	13.92	1.54	6.56	-23	0	-51
0.9	0.1	13.71	1.90	6.78	-11	5	-49

Table 10.17. Computational results for infinity level decomposition with on-demand accuracy for different λ and κ combinations. The third last column gives the percentage change compared to the infinity level decomposition method without on-demand accuracy (LevelI) and the same λ and κ values, whereas the second last column gives the percentage change compared to the euclidean level decomposition method with on-demand accuracy (LevelE-ODA) but with the same λ. The last column gives the percentage change compared with the Benders base case (BC). All comparisons are done in terms of the shifted geometric mean. Negative values indicate an improvement, positive values indicate a deterioration.

If a cut aggregation level different from single-cut is chosen the scenarios get partitioned. With on-demand accuracy there are three options of which we evaluated the first two. First, the on-demand accuracy information can be aggregated into a single cut by summing up the optimality cuts similar to how it is done in the cut consolidation technique, in particular equation (8.1). Second, the on-demand accuracy information can be aggregated into the same structure given by the scenario partitions, i.e., for five aggregates the method would generate five on-demand accuracy cuts with the same scenario partitioning. Third, the on-demand accuracy information can be further aggregated, by a partitioning of the existing aggregates. The first and second option are just special cases of the third option, with full aggregation and full disaggregation, respectively.

For the second case, where the on-demand accuracy cuts are partitioned like the optimality cuts, the computing times are shown in Table 10.18 and the iteration counts in Table 10.19. For the L-shaped method with on-demand accuracy aggregation proves to be successful for all the investigated aggregation levels. It reaches an improvement of up to 20% compared with the single-cut L-shaped method with on-demand accuracy. Measured with the arithmetic mean however, the computing times increase above those of the single-cut case for more than 20 aggregates. This behavior can be explained solely by one problem family, sslp, which is prone to cut proliferation and drives up the arithmetic mean. An aggregation level between five and ten is the best choice if measured with the shifted geometric mean. If the computing times are compared with those of the L-shaped method without on-demand accuracy, on-demand accuracy pays off in all cases, with diminishing returns for rising number of aggregates.

For all level decomposition variants an aggregation level of five is a little bit better than single-cut alone. It results in an improvement of 3% to 7% depending on the projection problem type. Choosing a higher number of aggregates leads to worse results than single-cut, if on-demand accuracy is used.

If the computing times are compared against those without on-demand accuracy alone, on-demand accuracy shows an improvement for euclidean and manhattan level decomposition in the range of 1% to 12% for aggregates higher than 10. Applying on-demand accuracy with cut aggregation higher than 10 for infinity level decomposition results in a slightly worse algorithm; a deterioration of around 5% occurs.

We can conclude that on-demand accuracy and modest cut aggregation can be combined successfully. The relative improvement of applying on-demand accuracy shrinks with a rising number of aggregates.

For the case where the on-demand accuracy cuts are partitioned into a single cut (SC-ODA), we give the computing times in Table 10.20. The comparison with the case where on-demand accuracy cuts are partitioned like normal optimality cuts shows that the total aggregation of on-demand accuracy cuts is not a good idea for the L-shaped method with on-demand accuracy. The computing times increase with the number of aggregates by up to 39%.

If we use the arithmetic mean, the data shows a different picture. For 50 and 100 aggregates, SC-ODA is faster than ODA. This is due to the sslp problem family where cut proliferation leads to long master problem solution times, as already mentioned above. If only single on-demand accuracy cuts are added to the master problem, this cut proliferation is less pronounced and thus the overall solution time for these problems is considerably

Proj.	Agg.	AM	GM	SGM	w/o ODA	ODA-SC	BC
-	1	24.08	1.69	8.74	-34	0	-34
	5	17.63	1.41	7.17	-29	-18	-46
	10	18.59	1.33	6.96	-24	-20	-48
	20	24.47	1.29	6.98	-18	-20	-48
	50	43.07	1.32	7.14	-9	-18	-46
	100	88.08	1.40	7.56	-7	-14	-43
l_2	1	11.20	1.41	5.28	-18	0	-60
	5	9.87	1.31	4.89	-15	-7	-63
	10	12.57	1.42	5.48	-5	4	-59
	20	25.29	1.66	7.04	-4	33	-47
	50	77.87	2.23	9.33	1	77	-30
	100	120.30	2.80	10.98	-1	108	-17
l_1	1	12.35	1.39	5.73	-18	0	-57
	5	10.49	1.27	5.22	-14	-9	-61
	10	11.92	1.29	5.46	-12	-5	-59
	20	15.42	1.33	5.79	-9	1	-56
	50	26.57	1.50	6.61	-4	15	-50
	100	48.20	1.74	7.53	-3	32	-43
l_∞	1	11.95	1.37	5.62	-15	0	-58
	5	13.30	1.27	5.44	-6	-3	-59
	10	17.32	1.31	5.74	-1	2	-57
	20	26.66	1.37	6.26	5	11	-53
	50	53.70	1.52	6.90	4	23	-48
	100	105.79	1.76	7.81	5	39	-41

Table 10.18. Computing times of Benders and level decomposition methods with on-demand accuracy for different aggregates, where the on-demand accuracy cuts are partitioned like normal optimality cuts. λ and κ are set to 0.5. The third last column gives the percentage change compared to the method run without on-demand accuracy (w/o ODA), but with the same number of aggregates. The second last column gives the percentage change compared to the original on-demand accuracy method with the fully aggregated model function (ODA-SC), i.e., single-cut method with on-demand accuracy. The last column gives the percentage change compared to the Benders base case (BC). Comparisons are done with the shifted geometric mean. Positive values indicate a deterioration, whereas negative values indicate an improvement.

					w/o ODA		Benders-BC	
Proj.	Agg.	AM	GM	SGM	AM	SGM	AM	SGM
-	1	328.89	92.11	104.56	20	11	20	11
	5	138.66	53.10	60.54	13	5	-50	-36
	10	94.72	41.99	47.65	7	3	-66	-49
	20	68.73	33.97	38.39	3	1	-75	-59
	50	48.44	26.93	30.23	2	0	-82	-68
	100	39.80	23.91	26.53	-1	-1	-86	-72
l_2	1	78.83	51.99	55.58	5	6	-71	-41
	5	52.61	38.00	40.50	1	5	-81	-57
	10	49.30	34.63	37.10	8	8	-82	-61
	20	60.53	32.38	35.18	3	4	-78	-63
	50	72.43	29.54	32.33	13	2	-74	-66
	100	69.35	27.62	30.36	-5	1	-75	-68
l_1	1	89.28	53.76	57.94	5	6	-68	-39
	5	61.06	40.15	43.18	9	7	-78	-54
	10	53.59	36.49	39.04	5	6	-81	-59
	20	46.53	32.46	34.73	5	5	-83	-63
	50	40.22	28.43	30.46	6	4	-85	-68
	100	36.33	26.45	28.28	1	2	-87	-70
l_∞	1	86.48	54.33	58.53	19	11	-69	-38
	5	61.43	40.55	43.44	20	10	-78	-54
	10	54.18	36.51	39.02	19	10	-80	-59
	20	48.76	32.47	34.74	18	7	-82	-63
	50	41.52	28.29	30.33	12	4	-85	-68
	100	37.66	26.28	28.20	9	4	-86	-70

Table 10.19. Iteration counts of Benders and level decomposition methods with on-demand accuracy for different aggregates, where the on-demand accuracy cuts are partitioned like normal optimality cuts. λ and κ are set to 0.5. The fourth and third last columns give the percentage change compared to the method run without on-demand accuracy (w/o ODA), but with the same number of aggregates. The second and last columns give the percentage change compared to the Benders base case (Benders-BC). Comparisons are done with the shifted geometric mean. Positive values indicate a deterioration, whereas negative values indicate an improvement.

less than those achieved with ODA. Without considering sslp, the arithmetic means are in line with the shifted geometric means. Similar reasoning holds for level decomposition with l_2 and l_1. The influence of sslp is stark in the l_∞ case, where the data shows a slight advantage of SC-ODA over ODA for both the arithmetic and the shifted geometric mean. Without taking sslp into account ODA has a slight advantage over SC-ODA for both measures.

Proj.	Agg.	AM	GM	SGM	ODA	BC
-	1	24.08	1.69	8.74	0	-34
	5	22.00	1.68	8.59	20	-35
	10	23.74	1.70	8.81	27	-34
	20	27.72	1.75	9.25	32	-30
	50	39.65	1.82	9.93	39	-25
	100	65.49	1.86	10.49	39	-21
l_2	1	11.20	1.41	5.28	0	-60
	5	10.30	1.39	5.20	6	-61
	10	12.30	1.49	5.57	2	-58
	20	24.99	1.74	7.21	2	-46
	50	65.54	2.25	9.25	-1	-30
	100	117.40	2.79	10.84	-1	-19
l_1	1	12.35	1.39	5.73	0	-57
	5	11.64	1.39	5.72	9	-57
	10	13.07	1.42	5.95	9	-55
	20	16.65	1.46	6.28	8	-53
	50	27.98	1.64	7.19	9	-46
	100	55.55	1.88	8.25	9	-38
l_∞	1	11.95	1.37	5.62	0	-58
	5	11.03	1.31	5.32	-2	-60
	10	12.95	1.34	5.62	-2	-58
	20	16.82	1.38	5.93	-5	-55
	50	27.86	1.52	6.75	-2	-49
	100	51.84	1.75	7.67	-2	-42

Table 10.20. Computing times of Benders and level decomposition methods with on-demand accuracy for different aggregates with on-demand accuracy single-cuts. λ and κ are set to 0.5. The second last column gives the percentage change compared to on-demand accuracy method, where the on-demand accuracy cuts are partitioned like the normal optimality cuts. The last column gives the percentage change compared to the Benders base case (BC). Comparisons are done with the shifted geometric mean. Positive values indicate a deterioration, whereas negative values indicate an improvement.

Conclusion

The on-demand accuracy concept results in signifcant computational savings compared with the respective solution methods without on-demand accuracy. Figure 10.9 gives a graphical comparison of the computing time spend in each stage with and without on-demand accuracy. The first stage computing time increases slightly for the different methods, whereas the second stage computing time decreases considerably if on-demand accuracy is used. This can be attributed to the higher number of overall iterations, but lower number of substantial iterations. The effect is more pronounced for Benders decomposition, but also results in significant computational savings for level decomposition.

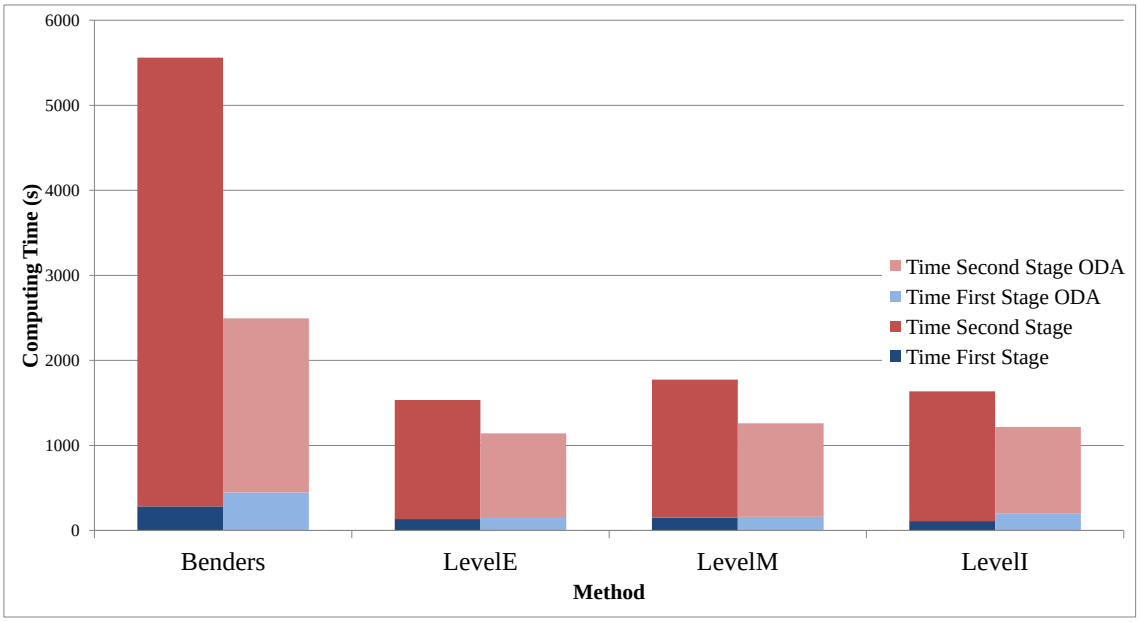

Figure 10.9. The overall computing time that is spent in the first and second stage by every method with and without on-demand accuracy. λ and κ are set to 0.5.

Regarding the choice of κ we can conclude from our experiments that a higher choice of κ mostly leads to better results than a smaller choice. Substantial iterations are traded off against insubstantial iterations. If the latter increase too much, the decrease in substantial iterations can not offset the computational saving, e.g., $\kappa = 0.9$ for Benders decomposition. By applying a regularization technique the negative effects of a high κ can be mitigated as the zig-zagging of the first-stage solution is hindered, and thus longer sequences of insubstantial iterations with very small improvements do not occur.

A modest level of cut aggregation, e.g., five to ten, results in computational savings compared to the single-cut case. An increased level of cut aggregation is only preferable for plain Benders decomposition.

10.5.5. Advanced start solution

Advanced start solution techniques were discussed in Section 5.4. The base case for all tests is the usage of the expected value solution as the starting solution. The computing times for the base case include the solver call to compute the expected value solution, whereas the other solutions are given to the solver at no cost. We tested several advanced starting solutions: no solution (NS), an optimal solution (OS) to the problem, an optimal solution for the worst case scenario (WCS) and an optimal solution for the best case scenario (BCS), where the two last solutions are obtained by solving the Wait-and-See problem. Preliminary test results with the method proposed by Infanger (1992) showed that it is not competitive against the usage of no starting solution by a large margin and is therefore not considered in the following.

L-shaped method The results in Table 10.21 indicate that the effect of an advanced solution is for practical purposes non-existent, at least for two-stage instances. In particular, the usage of the optimal solution as a start solution is of no help, but that is the expected behavior for cutting plane methods.

Advanced Start Solution	AM	SGM
No Solution (NS)	2	2
Optimal Solution (OS)	0	0
Worst Case Solution (WCS)	2	2
Best Case Solution (BCS)	1	-1

Table 10.21. Percentage changes in the computing time compared to the Benders base case with the expected value solution as the advanced solution for the arithmetic and shifted geometric mean. Positive values indicate a deterioration, negative values indicate an improvement.

Level decomposition A starting solution is needed for the level decomposition method. Thus it is interesting to investigate if the choice of the starting solution has any impact on the computing time. Our results indicate that the influence of the starting solution on the runtime of the algorithm is relatively small, as can be seen in Table 10.22 for the different projection problems. Even using an optimal solution does not help very much. Especially for the manhattan distance, the optimal solution increases the computing time by around 6%. The conclusion, which can be drawn from these results, is that it is more important to dampen the zig-zagging effect of Benders than to have a “good” initial solution.

On-demand accuracy The results in Table 10.23 show that the usage of the expected value solution as an advanced starting solution is always the best choice compared with no solution (NS), the optimal solution (OS), the worst-case solution (WCS), and the best-case solution (BCS). The results are more clear than the one for pure Benders and level decomposition, but the influence of the starting solution is still relatively small. The

		OS		WCS		BCS	
Distance	λ	AM	SGM	AM	SGM	AM	SGM
	0.1	-4	-2	1	2	39[a]	7[a]
	0.3	-3	-2	5	3	0	1
l_2	0.5	-1	2	2	-1	-1	-2
	0.7	-2	2	-2	-2	-1	-3
	0.9	13	6	-2	-2	7	0
	0.1	3	5	4	3	1	0
	0.3	6	5	2	0	0	-2
l_1	0.5	2	6	1	1	-3	-1
	0.7	11	8	-2	-2	0	0
	0.9	6	11	14	-1	-4	-1
	0.1	4	5	9	6	6	2
	0.3	-2	3	1	2	4	2
l_∞	0.5	2	1	6	3	6	0
	0.7	8	9	13	6	11	3
	0.9	7	7	-2	1	-3	-2

[a] Numerical difficulties during the solution run of instance saphir_50 lead to the values 39 and 7, respectively. If we exclude this problem, we get the values 1 and 1, respectively.

Table 10.22. Percentage changes in the computing time of using different advanced start solutions for the level decomposition method with different projection problems compared to the respective level decomposition solution run with the expected value solution as the advanced solution for the arithmetic and shifted geometric mean. The advanced start solutions are the optimal solution (OS), the worst-case solution (WCS), and the best-case solution (BCS). Positive values indicate a deterioration, negative values show an improvement.

expected value solution seems to be a better choice than any other tested starting solution. Due to the mixed results for level decomposition, no experiments were done for level decomposition method with on-demand accuracy, let alone different λ and κ combinations.

10.6. Effect of Parallelization

Parallelization of the algorithm is done only on the second and later stages by solving problems at the same stage in parallel. This includes the setup of the problem, the solving of the problem, the warm-start handling, the storing of primal and dual solutions, and the cut coefficient generation. Thus algorithmic variants which spend more time in the first stage can not benefit as much from parallelization as algorithms which spend more time in the second stage.

	NS		OS		WCS		BCS	
κ	AM	SGM	AM	SGM	AM	SGM	AM	SGM
0.1	7	5	2	4	8	6	6	2
0.3	9	8	4	6	7	7	3	2
0.5	4	7	3	5	6	7	4	3
0.7	11	9	6	8	7	8	9	8
0.9	17	8	13	6	13	5	19	9

Table 10.23. Percentage changes in the computing time of compared to the Benders ODA base case with the expected value solution as the advanced solution for the arithmetic and shifted geometric mean. The advanced start solutions are no start solution (NS), the optimal solution (OS), the worst-case solution (WCS), and the best-case solution (BCS). Positive values indicate a deterioration.

We are interested in two effects of parallelization, which we investigate in turn. The first aspect is the effect of parallelization on the computing time. This can be measured with the speedup, which is defined as the time of the sequential algorithm divided by the time of the parallel algorithm. The second aspect is the effect of parallelization on the relative order of the algorithms. In particular, we expect methods like multi-cut Benders, where most of the computing time is spend in the first stage, to not exhibit a good speedup behavior because of Amdahl's law (Amdahl, 1967).

Speedup

The speedup for p threads is defined as T_p/T_1, where T_p denotes the wall-clock computing time of the algorithm run with p threads. Figure 10.10 shows the speedup of the different methods graphically along with the ideal speedup. The speedups are also given in Table B.5 in the appendix.

The computer, where the tests were executed, has four physical cores but eight logical cores due to hyper-threading. The benefit of hyper-threading can be evaluated via the speedups shown in Figure 10.10. The speedup for the different methods is rather linear up to four threads, where every thread can use a physical core exclusively. If the number of threads increases above four the speedup increases less than before for every additional thread. This can be attributed to hyper-threading, as most of the time is spend in CPU intensive tasks, e.g., solving linear programming problems, and thus threads have to wait for a physical core to become available. The speedup of the Benders base case with four threads is already 3.46, but with eight threads it grows only to 4.55. We expect better speedups on systems where more physical cores are available. Note that the speedup of multi-cut Benders is for practical purposes non-existent.

Single-cut Benders reaches the highest speedup of all methods as it spends a particular large amount of its computing time in the second stage. This is depicted in Figure 10.11. It gives the amount of time each method spends in the first stage, for a given number of threads. We did not include multi-cut Benders, to be able to highlight the differences

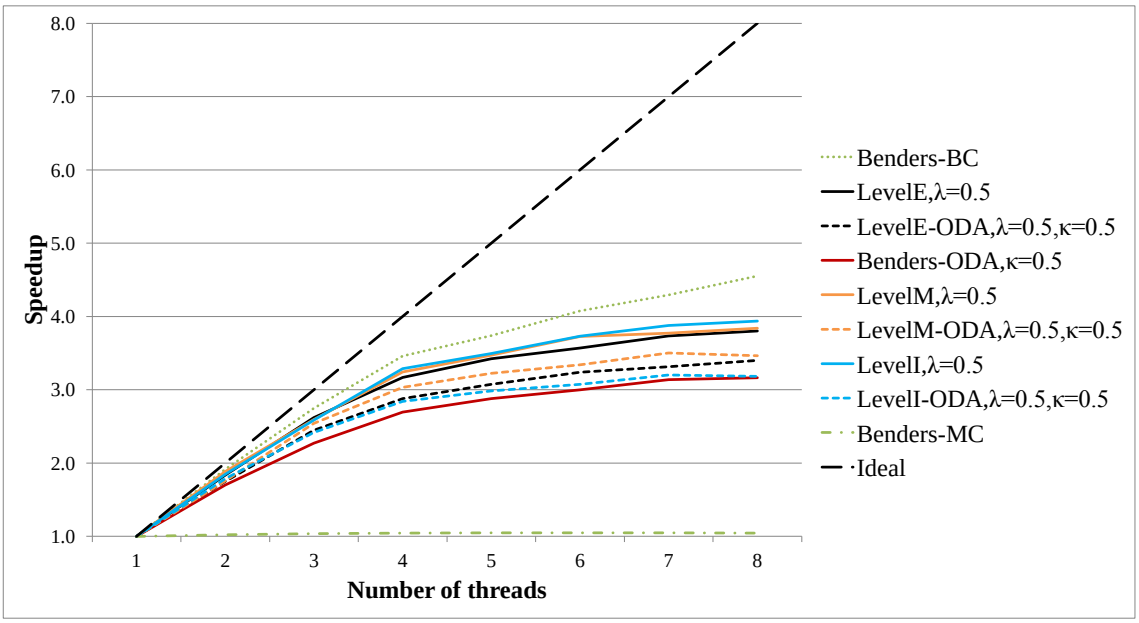

Figure 10.10. Speedup of different algorithms.

among the other methods. We can observe that the amount of time spend in the first stage corresponds to its speedup behavior. The correlation is negative and thus in concordance with Amdahl's law.

Relative order

Performance profiles for the parallel case with eight threads and the sequential case with only one thread are contrasted via the Figures 10.12 and 10.13. In the sequential case, the multi-cut method compares favorably with the single-cut method. This is totally different in the parallel case. Also, the performance differences between the on-demand accuracy variants and their respective methods without on-demand accuracy are more pronounced in the sequential case than in the parallel case. This can be explained via the computational savings due to insubstantial iterations which are higher in the sequential case. The second stage is skipped in every insubstantial iteration. Parallelization speeds up the solving of second stage problems. The time saved by skipping those second stage iterations is thus relatively smaller in the parallel case.

All of our results presented in the previous sections are achieved with parallelization. This performance comparison shows that the positive effects of on-demand accuracy, cut aggregation, and level decomposition would be even more distinct in the sequential case.

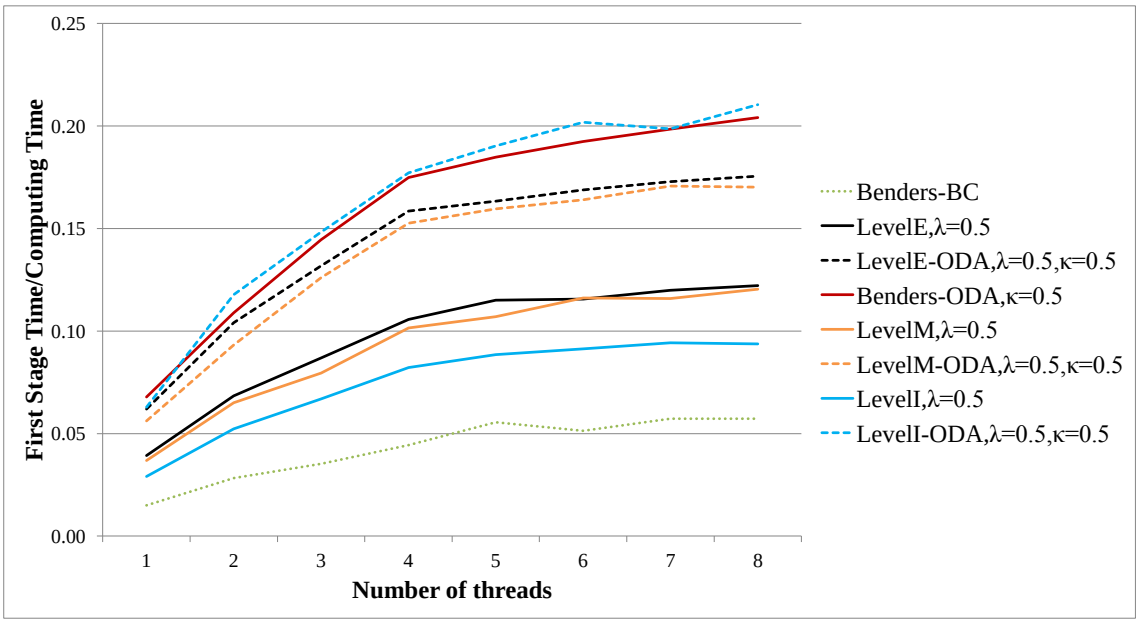

Figure 10.11. Amount of computing time of the first stage compared to overall computing time.

10.7. Evaluation of Multi-Stage Acceleration Techniques

Similar for the two-stage case we define a base case for the multi-stage case. The default sequencing protocol is FastForwardFastBack (FFFB), while the other parameters are the same as for the two-stage case.

10.7.1. Sequencing protocols

We compared the different sequencing protocols FFFB, FastForward (FF), ϵ-FF, FastBack (FB), ϵ-FB, and the dynamic protocol. ϵ is set to $0.1, 0.064$ and 0.01. The FF and FB protocols can also be seen as their ϵ variant with ϵ set to 10^-6 as this is set as the default threshold. The computing times are shown in Table 10.24.

The dynamic protocol proposed in Section 8.2 reaches the lowest computing times. If measured with the arithmetic mean, there are two ϵ-FB variants that are even faster. This can be explained by the pltexp problems. They take long to solve due to the large number of scenarios. They are solved faster with the ϵ-FB approach.

Our results confirm the results of Morton (1996) with respect to the standing of the FF method. It is the slowest protocol of all. The ϵ-FF variants are significantly faster. On our test set it turns out that FFFB is faster in terms of the shifted geometric mean but slower in terms of the arithmetic mean than the FB protocol. This is also due to the plextp problems where a lot of scenario subproblems at the last stage impact the arithmetic solution time.

The performance profile shown in Figure 10.14 gives a broader impression. It shows that the dynamic protocol compares well with all other sequencing protocols over the whole

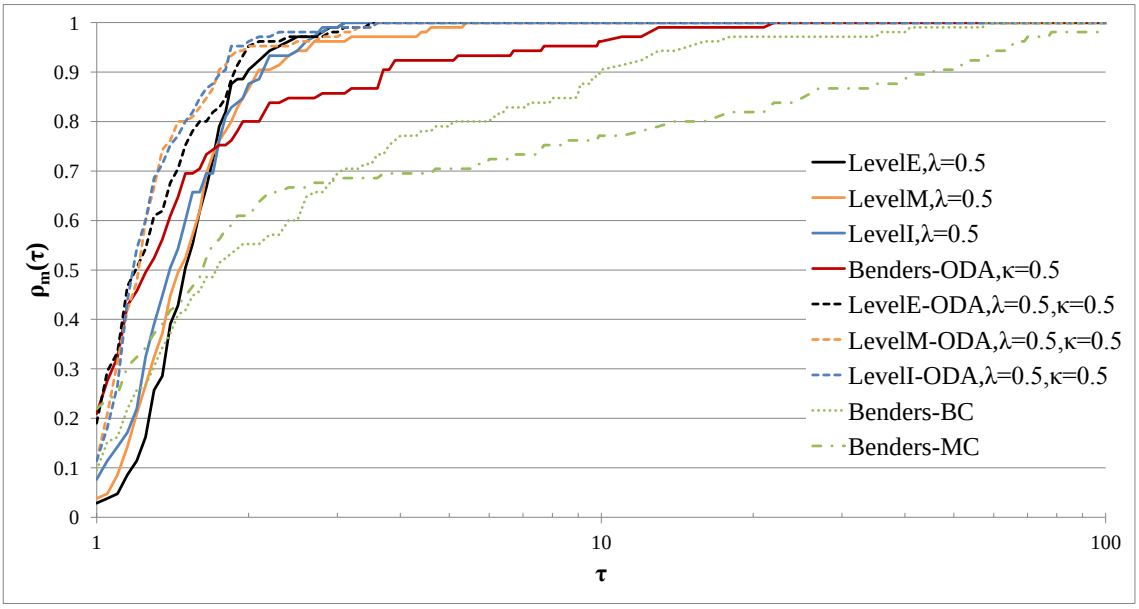

Figure 10.12. Performance profile of several methods with parallel execution.

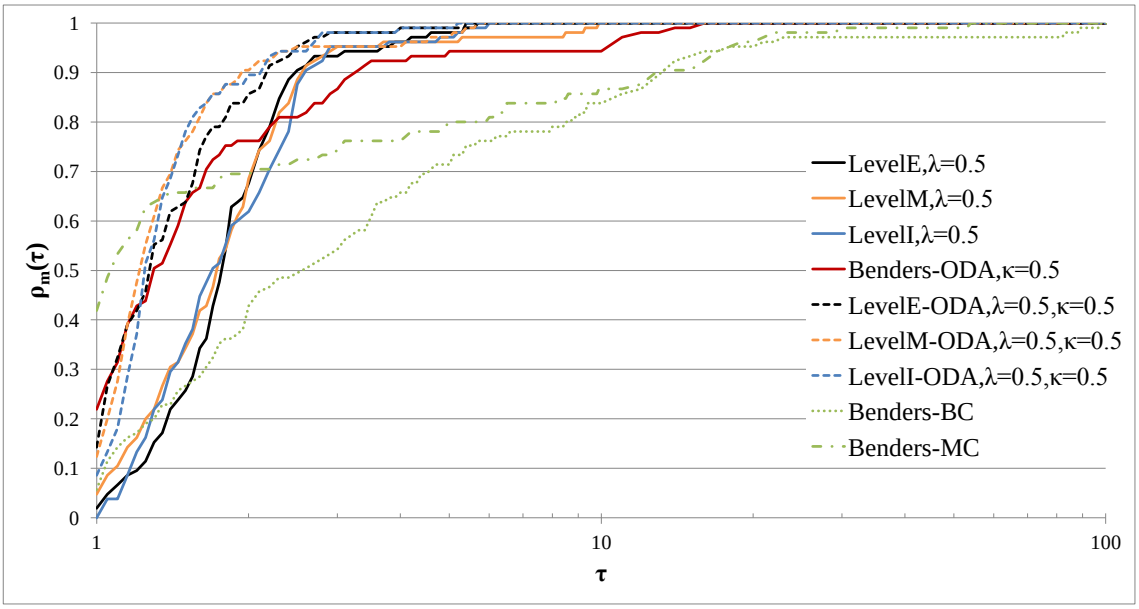

Figure 10.13. Performance profile of several methods with sequential execution.

Sequencing prot.	AM	GM	SGM	FFFB
FFFB	8.53	0.58	2.33	0
FF	241.21	1.91	10.50	351
FB	5.42	0.89	2.80	20
ϵ-FF-0.01	37.54	0.69	3.34	43
ϵ-FF-0.064	19.44	0.64	2.87	23
ϵ-FF-0.1	16.30	0.63	2.81	20
ϵ-FB-0.01	4.73	0.61	1.99	-15
ϵ-FB-0.064	4.91	0.54	1.89	-19
ϵ-FB-0.1	5.17	0.53	1.88	-20
Dynamic	5.05	0.53	1.82	-22

Table 10.24. Computing times of the parallel nested L-shaped method for different sequencing protocols, namely FastForwardFastBack (FFFB), FastForward (FF), FastBack (FB), ϵ variants and the dynamic protocol. The last column compares the shifted geometric mean to the result of the algorithm run with the FFFB protocol.

test set. The FFFB protocol, which the literature assumes to be the fastest protocol, is not as good as the dynamic protocol and two ϵ-FB variants. In contrast to the results of Morton (1996), ϵ-FB and the dynamic protocol are both consistently faster than FFFB both in total computing time and measured with the shifted geometric mean, by up to 45 % and 22 %, respectively.

We decided to count the iterations a little bit different for the FF protocol, as mentioned in Section 5.2. Every time the FF protocol decides to go forward again, after it went backward, is counted as a new iteration. Note that the iterations of the FF protocol are typically computationally expensive iterations, as the later stages with many subproblems are solved. This contrasts with the FB protocol, whose iterations are usually solved fast. The iteration counts are given in Table 10.25.

The FFFB protocol has the lowest number of iterations. This is expected as the primal and dual information traverses the tree as fast as possible in terms of iterations. The dynamic protocol needs less iterations than all of the ϵ-FB variants. This can be attributed to the feature of the dynamic protocol that a full sweep is done once the critical stage is reached, and some otherwise necessary intermediate iterations are saved.

Regarding the ϵ-FB protocols, for a higher ϵ the threshold for the current approximation quality is reached more easily and the algorithm can proceed to the next stage to generate new dual information. With this new dual information, another primal decision can then be found, leading to less cycles in the first few stages. This is reflected in the iteration counts, where a smaller ϵ requires more iterations.

We can conclude that the dynamic protocol is effective in reducing the computing time compared with the default protocol from the literature, FFFB. We showed also that the ϵ-FB variants perform significantly better than FFFB, which is not stated in the literature (Morton, 1996).

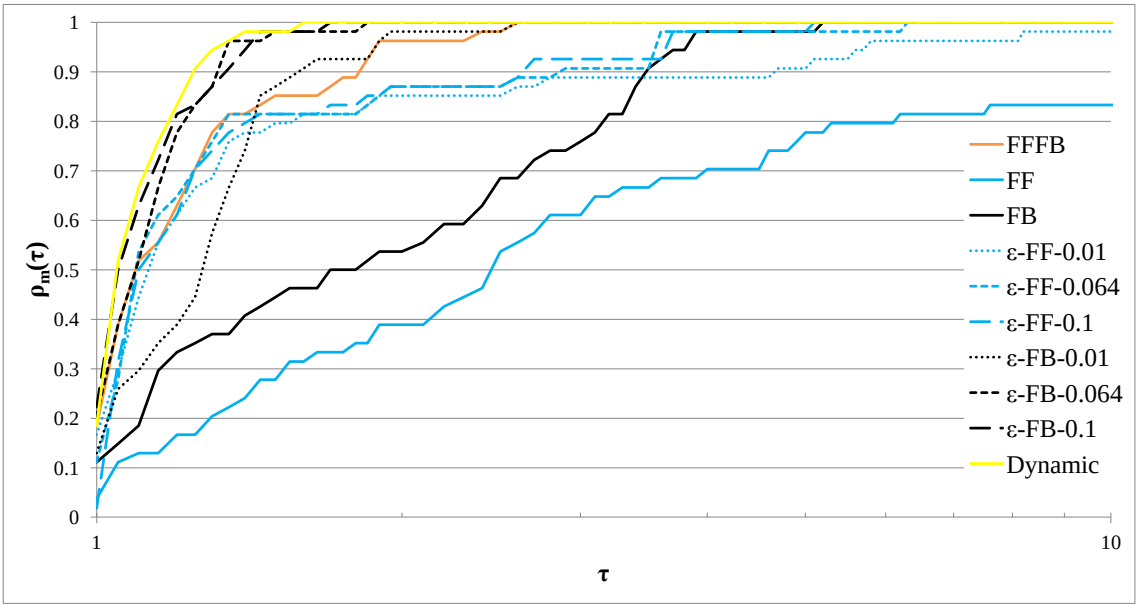

Figure 10.14. Performance profile of different sequencing protocols.

				FFFB	
Sequencing prot.	AM	GM	SGM	AM	SGM
FFFB	16.62	9.44	12.15	0	0
FF	216.67	38.16	52.49	1204	332
FB	286.44	73.93	107.01	1623	781
ϵ-FF-0.01	22.17	11.70	15.66	33	29
ϵ-FF-0.064	18.84	10.52	13.85	13	14
ϵ-FF-0.1	18.77	10.56	13.81	13	14
ϵ-FB-0.01	56.84	28.60	38.47	242	217
ϵ-FB-0.064	35.25	18.52	24.49	112	102
ϵ-FB-0.1	30.49	16.37	21.62	83	78
Dynamic	28.64	15.50	20.59	72	69

Table 10.25. Iteration counts of Benders BC on multi-stage test set with different sequencing protocols, namely FastForwardFastBack (FFFB), FastForward (FF), FastBack (FB), ϵ variants and the dynamic protocol. The last two columns compare the iteration counts with the default sequencing protocol FFFB, using the arithmetic mean and the shifted geometric mean, respectively.

10.8. SAA and Parallel Benders

The sample average approximation method (SAA) is used in a grid computing environment to solve the stochastic programming problems in the **sampling** test set (Linderoth et al., 2006). An asynchronous trust region method (Linderoth & Wright, 2003) is used instead of the L-shaped method to solve the SAA problems, see Section 4.3.3 for further explanations. We use the euclidean level decomposition method with on-demand accuracy with $\lambda = 0.5$ and $\kappa = 0.5$ to get computational results for the same set of problems. This was suggested by Fábián & Szőke (2007).

Our SAA implementation uses the upper bounding method described by Linderoth et al. (2006). We only use latin hypercube sampling instead of also using crude Monte Carlo sampling as this provides better confidence intervals (Linderoth et al., 2006). The number of batches for the lower bounding problems, M, is set to ten and the number of batches for the upper bounding problems, T, is set to 20. After the estimated upper bounds for each of the M solutions are computed by solving T sampled problems with $\bar{N}$ scenarios each, the final upper bound estimate is computed. This is done by computing a new independent upper bound estimate for the lowest upper bound estimate found so far.

We give lower and upper bound confidence intervals in Table 10.26 for all five problems with an increasing number of scenarios and computing times. A comparison of the confidence intervals with those given by Linderoth et al. (2006) shows that we reach similar solution quality, albeit with somewhat wider confidence intervals for the upper bounds. The difference is due to the fact that their results are computed with $T = 50$.

The computing times are presented in Table 10.27. These results show that with the usage of parallelization and regularization techniques, the SAA method reaches good approximate solutions in under an hour, for $N = 20,000$. The results indicate that the sampling time is linear in the number of sampled instances, and the upper bounding time is linear in the number of scenarios to solve. The time spend to solve the lower bounding problems becomes the dominant factor for higher values of N for the problems ssn and 20term. All ten replications together can be solved in about an hour for $N = 50,000$. For comparison, the problems LandS and gbd can be solved directly with the Level-ODA algorithm in 73.98 and 109.86 seconds, respectively.

10.9. Conclusion

We evaluated various acceleration techniques for the nested L-shaped method in this chapter. We provided averages and performance profiles to compare the different methods, due to the large number of instances and parameter combinations. However, detailed results are available for selected methods for the two-stage case in Table B.1 and for the multi-stage in Table B.4 in the appendix. The implementation of the methods allows to compare cut aggregation, cut consolidation, level decomposition, on-demand accuracy and advanced start solutions for two-stage problems. Cut aggregation is an important technique that should be applied to almost all problems, when solved with Benders decomposition. The problem of choosing a cut aggregation level remains, but a number of 20 to 100 aggregates seems promising for many problems. Cut consolidation is an important technique, when

Problem	N	$\bar{N}$	Lower Bound	Upper Bound
LandS	500	20000	225.66 ± 0.09	225.64 ± 0.01
	1000	20000	225.64 ± 0.04	225.63 ± 0.01
	5000	20000	225.65 ± 0.01	225.63 ± 0.00
	10000	20000	225.64 ± 0.01	225.63 ± 0.00
	20000	50000	225.63 ± 0.01	225.63 ± 0.01
	50000	100000	225.63 ± 0.01	225.63 ± 0.00
gbd	500	20000	1655.63 ± 0.00	1655.63 ± 0.00
	1000	20000	1655.63 ± 0.00	1655.63 ± 0.00
	5000	20000	1655.63 ± 0.00	1655.63 ± 0.00
	10000	20000	1655.63 ± 0.00	1655.63 ± 0.00
	20000	50000	1655.63 ± 0.00	1655.63 ± 0.00
	50000	100000	1655.63 ± 0.00	1655.63 ± 0.00
storm	500	20000	15498703.72 ± 254.16	15498742.68 ± 27.61
	1000	20000	15498716.76 ± 243.79	15498747.61 ± 23.36
	5000	20000	15498750.32 ± 91.10	15498725.07 ± 21.52
	10000	20000	15498671.97 ± 90.30	15498737.68 ± 38.33
	20000	50000	15498725.67 ± 37.35	15498740.31 ± 18.67
	50000	100000	15498745.14 ± 28.58	15498734.34 ± 12.54
20term	500	20000	254257.55 ± 115.07	254310.66 ± 8.64
	1000	20000	254338.29 ± 83.68	254309.09 ± 12.41
	5000	20000	254309.32 ± 16.84	254317.79 ± 10.03
	10000	20000	254303.49 ± 17.58	254318.08 ± 8.27
	20000	50000	254311.57 ± 14.72	254308.66 ± 6.33
	50000	100000	254313.26 ± 6.30	254310.57 ± 4.61
ssn	500	20000	9.42 ± 0.47	10.06 ± 0.03
	1000	20000	9.77 ± 0.21	10.00 ± 0.05
	5000	20000	9.90 ± 0.13	9.96 ± 0.04
	10000	20000	9.94 ± 0.08	9.87 ± 0.04
	20000	50000	9.87 ± 0.04	9.89 ± 0.03
	50000	100000	9.91 ± 0.04	9.89 ± 0.02

Table 10.26. Lower and upper bound 95 % confidence intervals for the SAA problems with N sampled scenarios for a single SAA problem and $\bar{N}$ sampled scenarios for a single upper bounding problem. $M = 10$ and $T = 20$, for all solution runs.

Problem	N	$\bar{N}$	LBP	UBP	Sampling	Overall
LandS	500	20000	1.00	47.66	11.59	66.71
	1000	20000	1.25	47.55	11.77	67.03
	5000	20000	3.45	47.86	11.76	69.47
	10000	20000	5.16	47.82	11.89	71.51
	20000	50000	10.16	103.61	29.77	157.76
	50000	100000	23.42	169.27	59.32	278.10
gbd	500	20000	0.98	43.04	17.10	68.03
	1000	20000	1.28	44.37	17.19	69.87
	5000	20000	3.17	45.13	17.28	72.70
	10000	20000	5.37	44.58	17.47	74.68
	20000	50000	11.87	100.58	43.04	171.32
	50000	100000	30.17	161.09	84.94	305.20
storm	500	20000	5.58	534.13	148.40	734.92
	1000	20000	8.70	519.82	148.15	722.77
	5000	20000	31.92	525.97	149.48	754.78
	10000	20000	67.53	544.31	152.96	817.11
	20000	50000	153.29	1323.17	404.48	2042.79
	50000	100000	320.97	2674.22	765.31	4116.86
20term	500	20000	46.46	245.17	65.43	374.39
	1000	20000	80.45	243.44	65.50	407.74
	5000	20000	401.17	241.45	66.02	733.51
	10000	20000	678.85	241.77	66.58	1019.52
	20000	50000	1601.66	572.73	166.24	2432.04
	50000	100000	3402.94	1142.52	336.88	5131.47
ssn	500	20000	26.16	289.91	110.53	435.09
	1000	20000	46.36	288.76	110.70	455.08
	5000	20000	337.10	291.54	111.91	754.32
	10000	20000	692.95	295.55	113.32	1122.59
	20000	50000	1577.89	697.30	287.40	2619.34
	50000	100000	3637.86	1388.48	589.72	5787.00

Table 10.27. Computing times in seconds for lower bounding problems (LBP), upper bounding problems (UBP), sampling and the whole algorithm. $M = 10$ and $T = 20$, for all solution runs.

cut proliferation becomes a problem. This is the case for aggregation levels above 100 and it should be used when such an aggregation level is chosen.

Level decomposition was initially proposed with an euclidean projection problem using the l_2 norm. We showed that other norms can also be used in the projection problem, namely the l_1 and the l_∞ norm. Euclidean level decomposition remains the fastest solution technique, both in iteration counts and in computing time, but infinity level decomposition is nearly as good and manhattan level decomposition is only about 10% slower than euclidean level decomposition. This insight may be important when no quadratic programming solver is available to implement the euclidean level decomposition method.

The choice of λ is an important factor for the computing time of level decomposition, regardless of the choice of the projection problem. Although our default value was set to 0.5, our experiments showed that 0.7 is slightly better. Cut aggregation can also be used together with level decomposition, but only in a very modest form, with 5-10 aggregates. If this is done, the computing times can be further reduced by about 10%.

We evaluated the effectiveness of the on-demand accuracy concept, used together with Benders decomposition and together with level decomposition. It proves to be more effective for Benders decomposition, but still reduces the computing time for level decomposition. The κ parameter which regulates the usage of on-demand accuracy cuts, is set a priori but has significant influence on the overall performance, so it should be set judiciously. For Benders decomposition, a value between 0.5 and 0.7 seems appropriate. For level decomposition, κ and λ must be set, and the effective combinations vary a bit by the projection problem type. In general, combinations of λ between 0.3 and 0.7 and κ between 0.3 and 0.9 show good performance. The usage of cut aggregation in conjunction with on-demand accuracy is dependent on the effectiveness of cut aggregation alone. Level decomposition requires a high aggregation level, whereas Benders decomposition also shows good performance with a lower aggregation level, i.e., more aggregates.

Advanced start solutions are necessary for level decomposition and can also be used within the L-shaped framework. We found no convincing evidence for using a particular start solution, other than the expected value solution. Especially in the on-demand accuracy case, the expected value solution proved to be better than the alternatives.

Parallelization of the algorithm has several impacts. The first impact is that computing times decrease with a relatively good speed-up compared with a sequential implementation. The second impact is that the relative order of the algorithms is changed. This is due to Amdahl's law. Therefore, single-cut Benders and multi-cut Benders switch places. Also, on-demand accuracy is even more effective in the sequential case than in the parallel case.

For multi-stage problems, we proposed a new dynamic sequencing protocol and revisited other sequencing protocols. We found that the FFFB protocol, which was used as the default in the literature, is not the fastest protocol. The dynamic protocol and some ϵ-FB variants showed better performance.

Sample average approximation was evaluated by Linderoth et al. (2006) on a computational grid on some test instances. We found that using the acceleration techniques, i.e., level decomposition with on-demand accuracy, SAA can be used to give meaningful confidence intervals for several problems in under an hour. This would not be possible with plain Benders decomposition, as the lower bounding problem solution time would take too long.

Our results show that developing special solution methods for multi-stage stochastic programming is a worthwhile endeavor. The comparison with deterministic equivalent solvers shows that our implementation is much faster, even for medium-sized problems. For problems that grow too large, mainly because of the number of scenarios, decomposition methods are unavoidable in any case if a solution to these problem should be found in a reasonable time frame or at all.

11. Summary and Conclusion

In this thesis we developed an algebraic modeling language for stochastic programs and computationally efficient solution techniques based on Benders decomposition for multi-stage stochastic programming problems. In Part I we gave an introduction to stochastic programming and basic solution methods. Two-stage and multi-stage stochastic programs were introduced in Chapter 4. We also gave mathematical results, which are required for decomposition algorithms. Different solution methodologies were explained in Chapter 3. Apart from Benders decomposition, which is based on the implicit deterministic equivalent formulation, we described Lagrangean relaxation approaches, which are based on the explicit deterministic equivalent formulation. Both decomposition approaches can be used within approximative solution methods.

A detailed literature review of the state-of-the-art was done in Part II. We looked at acceleration techniques for Benders decomposition for two-stage programs in Chapter 4. Several approaches were later extended in our implementation. Among them are cut aggregation and level decomposition. Chapter 5 introduced techniques that were primarily applied to multi-stage problems. Notable refinements are parallelization approaches and sequencing protocols. Our survey on the development of algebraic modeling languages was given in Chapter 6. We also looked at the challenges that arise in the design of modeling languages. After this extensive literature review we identified the research gaps in Chapter 7 that we aimed to close with this thesis.

Part III described our algorithmic ideas and their implementation in a state-of-the-art solver for stochastic programs. Chapter 8 introduced acceleration techniques to improve Benders decomposition. We described an approach to reduce the negative effects of cut proliferation, which we called cut consolidation. We revisited sequencing protocols for multi-stage stochastic programs and proposed a new dynamic sequencing protocol. The presented parallelization approach allows to benefit from several cores on modern processors. We extended the on-demand accuracy concept (Oliveira & Sagastizábal, 2012) to the classical L-shaped method. The new projection problem norms l_1 and l_∞ were established for level decomposition. Chapter 9 described our extensions to the algebraic modeling language FlopC++ to allow modeling of stochastic programs. It is possible to model stochastic programs with scenario data or with random variables. Chapter 10 described the implementation and evaluation of the algorithmic ideas presented in Chapter 8. We described the diverse set of test instances, which we retrieved from several sources, on which our implementation was evaluated. One of our contributions was the evaluation of several techniques on a large and diverse test set. We found that cut consolidation can be used to combat cut proliferation. Our examination of cut aggregation for the L-shaped method showed that the technique is well suited to reduce computing times. Another contribution is the evaluation of new projection problems for the level decomposition method. This allows to use level decomposition without access to a quadratic programming solver. We

showed that on-demand accuracy can be combined successfully with the classical L-shaped method as well as level decomposition. Our analysis of cut aggregation combined with level decomposition and on-demand accuracy showed that a modest level of cut aggregation can further reduce the computing time. Our evaluation of advanced start solutions showed that using the expected value solution is in most cases a good choice. We found that parallelization changes the relative order of algorithms and that single-cut Benders has the highest speedup of all evaluated algorithms.

A sequencing protocol has to be chosen for the nested L-shaped method. We showed that our proposed dynamic protocol leads to faster solution times than the default protocol FastForwardFastBack.

We demonstrated that sample average approximation can be successfully combined with level decomposition with on-demand accuracy to reach good quality solutions in under an hour.

This thesis showed that research in the field of solution techniques for stochastic programs is an ongoing effort. The development and usage of more efficient data structures can further reduce computing times in the future. Documenting these important details (Maros, 2003) can spark further research that can lead to improved algorithms. One research direction is the combination of our parallelization approach with another parallelization layer on a computational grid. This can be extended by the combination of asymmetric sequencing protocols with on-demand accuracy. Another direction would be to research strategies to combine cut aggregation with on-demand accuracy, where the aggregation levels can adjust dynamically. A stochastic programming test set that combines the existing problems in the literature should be compiled to enable performance comparisons between different solution techniques. The test sets we gathered to evaluate the goals of this thesis can serve as a starting point.

Bibliography

Achterberg, T. (2007). *Constraint Integer Programming*. Ph.D. thesis Technische Universität Berlin.

Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., & Sen, S. (2013). SIPLIB: A Stochastic Integer Programming Test Library. `http://www2.isye.gatech.edu/~sahmed/siplib/`.

Altenstedt, F. (2003). *Aspects on asset liability management via stochastic programming*. Ph.D. thesis Chalmers University of Technology and Göteborg University.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference* (pp. 483–485). ACM volume 126.

Ariyawansa, K. A., & Felt, A. J. (2004). On a new collection of stochastic linear programming test problems. *INFORMS Journal on Computing*, *16*, 291–299.

Ariyawansa, K. A., & Hudson, D. D. (1991). Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the Sequent/Balance. *Concurrency: Practice and Experience*, *3*, 109–128.

Atlihan, M., Cunningham, K., Laude, G., & Schrage, L. (2010). Challenges in Adding a Stochastic Programming/Scenario Planning Capability to a General Purpose Optimization Modeling System. In M. S. Sodhi, & C. S. Tang (Eds.), *A Long View of Research and Practice in Operations Research and Management Science: The Past and the Future* chapter 8. (pp. 117–135). volume 148 of *International Series in Operations Research & Management Science*.

Bauer, H. (1991). *Wahrscheinlichkeitstheorie*. (4th ed.). de Gruyter.

Beale, E. (1955). On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society. Series B (Methodological)*, *17*, 173–184.

Beasley, J. E. (1993). Lagrangean Relaxation. In C. R. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons.

Ben-Tal, A., & Nemirovski, A. (2005). Non-euclidean restricted memory level method for large-scale convex optimization. *Mathematical Programming*, *102*, 407–456.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 238–252.

Bertsekas, D. (1982). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press.

Birge, J., & Louveaux, F. (2011). *Introduction to Stochastic Programming*. (2nd ed.). Springer.

Birge, J. R. (1985). Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Operations Research*, *33*, 989–1007.

Birge, J. R. (1997). Stochastic programming computation and applications. *INFORMS Journal on Computing*, *9*, 111–133.

Birge, J. R., Dempster, M. A. H., Gassmann, H. I., Gunn, E. A., King, A. J., & Wallace, S. W. (1987). A standard input format for multiperiod stochastic linear programs. *COAL newsletter*, *17*, 1–19.

Birge, J. R., Donohue, C. J., Holmes, D. F., & Svintsitski, O. G. (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, *75*, 327–352.

Birge, J. R., & Louveaux, F. V. (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, *34*, 384–392.

Bixby, R. E., & Martin, A. (2000). Parallelizing the Dual Simplex Method. *INFORMS Journal on Computing*, *12*, 45–56.

Brandes, K. T. (2011). *Implementierung und Analyse verschiedener Strategien zur Aggregation und Disaggregation von Multi-Cuts im Benders Dekompositionsverfahren*. Master's thesis Universität Paderborn.

Buaklee, D., Tracy, G. F., Vernon, M. K., & Wright, S. J. (2002). Near-optimal adaptive control of a large grid application. *Proceedings of the 16th international conference on Supercomputing - ICS '02*, (p. 315).

Buchanan, C. S., McKinnon, K. I. M., & Skondras, G. K. (2001). The Recursive Definition of Stochastic Linear Programming Problems within an Algebraic Modeling Language. *Annals of Operations Research*, *104*, 15–32.

Carøe, C. C., & Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, *24*, 37–45.

Cerisola, S., & Ramos, A. (2000). Node Aggregation in Stochastic Nested Benders Decomposition Applied to Hydrothermal Coordination. In *PMAPS2000: 6th International Conference on Probabilistic Methods Applied to Power Systems* 1. Madeira.

Charnes, A., & Cooper, W. (1959). Chance-constrained programming. *Management Science*, *6*, 73–79.

Chvátal, V. (1983). *Linear Programming*. W. H. Freeman and Company.

Colombo, M., Grothey, A., Hogg, J., Woodsend, K., & Gondzio, J. (2009). A structure-conveying modelling language for mathematical and stochastic programming. *Mathematical Programming Computation*, *1*, 223–247.

Condevaux-Lanloy, C., & Fragnière, E. (1998). *SETSTOCH: a tool for multistage stochastic programming with recourse*. Technical Report University of Geneva Geneva.

Condevaux-Lanloy, C., Fragnière, E., & King, A. J. (2002). SISP: Simplified Interface for Stochastic Programming. *Optimization Methods and Software*, *17*, 423–443.

Consigli, G., & Dempster, M. A. H. (1998). Dynamic stochastic programming for asset-liability management. *Annals of Operations Research*, *81*, 131 – 161.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* volume 7. (2nd ed.). The MIT Press.

CPLEX (2013). CPLEX lp files. `http://lpsolve.sourceforge.net/5.5/CPLEX-format.htm`. Last accessed 10/25/13.

Culler, D., Singh, J., & Gupta, A. (1999). *Parallel computer architecture: a hardware/-software approach*. Morgan Kaufmann.

Dantzig, G. B. (1955). Linear programming under uncertainty. *Management Science*, *1*, 197–206.

Dantzig, G. B., & Glynn, P. (1990). Parallel processors for planning under uncertainty. *Annals of Operations Research*, *22*, 1–21.

Dantzig, G. B., Ho, J. K., & Infanger, G. (1991). *Solving Stochastic Linear Programs on a Hypercube Multicomputer*. Technical Report Department of Operations Research, Stanford University Stanford.

Dantzig, G. B., & Infanger, G. (1991). *Large-Scale Stochastic Linear Programs: Importance Sampling and Benders Decomposition*. Technical Report Stanford University Stanford.

Dantzig, G. B., & Wolfe, P. (1961). The decomposition algorithm for linear programs. *Econometrica: Journal of the Econometric Society*, *29*, 767–778.

Deák, I. (2011). Testing successive regression approximations by large-scale two-stage problems. *Annals of Operations Research*, *186*, 83–99.

Dempster, M. A. H., & Thompson, R. T. (1998). Parallelization and Aggregation of Nested Benders Decomposition. *Annals of Operations Research*, *81*, 163–188.

Dempster, M. A. H., & Thompson, R. T. (1999). EVPI-based importance sampling solution procedures for multistage stochastic linear programmeson parallel MIMD architectures. *Annals of Operations Research*, *90*, 161–184.

Dohle, C. (2010). *Eine Implementierung des Benders-Dekompositionsverfahrens für allgemeine zweistufig stochastische Programme mit diskreten Stufe-1-Variablen*. Diplomarbeit Universität Paderborn.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*, 201–213.

Dormer, A., Vazacopoulos, A., Verma, N., & Tipi, H. (2005). Modeling & solving stochastic programming problems in supply chain management using XPRESS-SP. In J. Geunes, & P. M. Pardalos (Eds.), *Supply Chain Optimization* chapter 10. (pp. 307–354). Springer volume 98 of *Applied Optimization*.

Dupačová, J. (1995). Multistage stochastic programs: The state-of-the-art and selected bibliography. *Kybernetika*, *31*, 151–174.

Dupačová, J., Consigli, G., & Wallace, S. W. (2000). Scenarios for multistage stochastic programs. *Annals of Operations Research*, *100*, 25–53.

Dupačová, J., Gröwe-Kuska, N., & Römisch, W. (2003). Scenario Reduction in Stochastic Programming: An Approach Using Probability Metrics. *Mathematical Programming*, *95*, 493–511.

Edwards, J. (1988). A proposed standard input format for computer codes which solve stochastic programs with recourse. In Y. Ermoliev, & R. J.-B. Wets (Eds.), *Numerical techniques for stochastic optimization* (pp. 215–227). Springer volume 10 of *Springer Series in Computational Mathematics*.

Ellison, F., Mitra, G., & Zverovich, V. (2012). FortSP : A Stochastic Programming Solver. http://www.optirisk-systems.com/manuals/FortspManual.pdf.

Entriken, R. (2001). Language constructs for modeling stochastic linear programs. *Annals of Operations Research*, *104*, 49–66.

Ermoliev, Y. (1988). Stochastic quasigradient methods. In Y. Ermoliev, & R. J.-B. Wets (Eds.), *Numerical techniques for stochastic optimization* (pp. 141–185). Springer volume 10 of *Springer Series in Computational Mathematics*.

Escudero, L. F., Garín, M. A., Pérez, G., & Unzueta, A. (2012). Lagrangian Decomposition for large-scale two-stage stochastic mixed 0-1 problems. *Top*, *20*, 347–374.

Fábián, C. I. (2000). Bundle-type methods for inexact data. *Central European Journal of Operations Research*, *8*, 35–55.

Fábián, C. I. (2013). Computational aspects of risk-averse optimization in two-stage stochastic models. *Stochastic Programming E-Print Series*, *2013*.

Fábián, C. I., & Szőke, Z. (2007). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, *4*, 313–353.

Fischetti, M., Salvagnin, D., & Zanette, A. (2008). Minimal infeasible subsystems and Benders cuts. Mathematical Programming, to appear.

Fischetti, M., Salvagnin, D., & Zanette, A. (2010). A note on the selection of Benders' cuts. *Mathematical Programming*, *124*, 175–182.

Fisher, M. L. (1981). The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, *27*, 1–18.

Fisher, M. L. (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces*, *15*, 10–21.

Flynn, M. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, *C-21*, 948–960.

Fourer, R. (1983). Modeling languages versus matrix generators for linear programming. *ACM Transactions on Mathematical Software*, *9*, 143–183.

Fourer, R., Gassmann, H. I., Ma, J., & Martin, R. K. (2009). An XML-based schema for stochastic programs. *Annals of Operations Research*, *166*, 313–337.

Fourer, R., & Gay, D. M. (2000). Conveying Problem Structure from an Algebraic Modeling Language to Optimization Algorithms. In M. Laguna, & J. L. G. Velarde (Eds.), *Computing Tools for Modeling, Optimization and Simulation* (pp. 75–89). Springer volume 12 of *Operations Research/Computer Science Interfaces Series*.

Fourer, R., & Lopes, L. (2006). A management system for decompositions in stochastic programming. *Annals of Operations Research*, *142*, 99–118.

Fourer, R., Ma, J., & Martin, K. (2010). OSiL: An instance language for optimization. *Computational Optimization and Applications*, *45*, 181–203.

Fragnière, E., & Gondzio, J. (2005). Stochastic programming from modeling languages. In S. W. Wallace, & W. T. Ziemba (Eds.), *Applications of Stochastic Programming* chapter 7. (pp. 95–113). Society for Industrial Mathematics.

Freund, R. M. (2004). Benders' Decomposition Methods for Structured Optimization, including Stochastic Optimization.

Frontline Solvers (2013). Robust Optimization, Stochastic Programming, and Simulation Optimization. `http://www.solver.com/robust-decision-making`. Last accessed 10/25/13.

GAMS-EMP (2013). Stochastic Programming (SP) with EMP. `http://gams.com/dd/docs/solvers/empsp.pdf`. Last accessed 10/25/13.

Garstka, S. J., & Rutenberg, D. P. (1973). Computation in discrete stochastic programs with recourse. *Operations Research*, *21*, 112–122.

Gassmann, H. I. (1990). MSLiP: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, *47*, 407–423.

Gassmann, H. I. (1998). Modelling support for stochastic programs. *Annals of Operations Research*, *82*, 107–138.

Gassmann, H. I. (2007). Applied stochastic programming models and computation.

Gassmann, H. I., & Infanger, G. (2007). Modelling history-dependent parameters in the SMPS format for stochastic programming. *IMA Journal of Management Mathematics*, *19*, 87–97.

Gassmann, H. I., & Ireland, A. (1995). Scenario formulation in an algebraic modelling language. *Annals of Operations Research*, *59*, 45–75.

Gassmann, H. I., & Ireland, A. (1996). On the formulation of stochastic linear programs using algebraic modelling languages. *Annals of Operations Research*, *64*, 83–112.

Gassmann, H. I., & Kristjansson, B. (2007). The SMPS format explained. *IMA Journal of Management Mathematics*, (pp. 1–31).

Gassmann, H. I., & Prékopa, A. (2005). On stages and consistency checks in stochastic programming. *Operations Research Letters*, *33*, 171–175.

Gassmann, H. I., & Schweitzer, E. (2001). A comprehensive input format for stochastic linear programs. *Annals of Operations Research*, *104*, 89–125.

Gay, D. M. (2005). *Writing .nl Files*. Technical Report SAND2005-7907P, Sandia National Laboratories.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, *23*, 5–48.

Gondzio, J. (1998). Warm start of the primal-dual method applied in the cutting-plane scheme. *Mathematical Programming*, *83*, 125–143.

Gurobi Optimization, Inc. (2013). Gurobi optimizer reference manual. `http://www.gurobi.com/documentation/5.5/reference-manual/`. Last accessed 10/25/13.

Hart, W. E., Laird, C., Watson, J.-P., & Woodruff, D. L. (2012). *Pyomo - Optimization Modeling in Python*. Springer.

Hart, W. E., Watson, J.-P., & Woodruff, D. L. (2011). Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, *3*, 219–260.

Heitsch, H., & Römisch, W. (2003). Scenario reduction algorithms in stochastic programming. *Computational optimization and applications*, (pp. 187–206).

Heitsch, H., & Römisch, W. (2011). Stability and Scenario Trees for Multistage Stochastic Programs. In G. Infanger (Ed.), *Stochastic Programming - The State of the Art In Honor of George B. Dantzig* (pp. 139–164). Springer.

Held, M., Wolfe, P., & Crowder, H. (1974). Validation of subgradient optimization. *Mathematical programming*, *6*, 62–88.

Higle, J. L. (2005). Stochastic Programming: Optimization When Uncertainty Matters. *TutORials in Operations Research*, (pp. 30–53).

Higle, J. L., & Sen, S. (1991). Stochastic Decomposition: An Algorithm for Two-Stage Linear Programs with Recourse. *Mathematics of Operations Research*, *16*, 650–669.

Higle, J. L., & Sen, S. (1996). *Stochastic decomposition: a statistical method for large scale stochastic linear programming*. Kluwer Academic Publishers.

Holmes, D. (1995). A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS). http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html.

Hultberg, T. H. (2007). FLOPC++ An Algebraic Modeling Language Embedded in C++. In K.-H. Waldmann, & U. M. Stocker (Eds.), *Operations Research Proceedings 2006* (pp. 187–190). Springer Berlin-Heidelberg.

Infanger, G. (1992). *Planning under uncertainty - solving large-scale stochastic linear programs*. Technical Report SOL-92-8, Stanford Univ., CA (United States). Systems Optimization Lab.

International Business Machines (1972). *Mathematical Programming Subsystem - Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description*. Technical Report SH20-0968-1 IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY.

International Business Machines Corporation (2011). IBM ILOG CPLEX V12.4: User's Manual for CPLEX. `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4/topic/ilog.odms.cplex.help/CPLEX/maps/ps_usrmancplex_1.html`. Last accessed 10/25/13.

Kall, P., & Mayer, J. (1998). On testing SLP codes with SLP-IOR. *New Trends in Mathematical Programming: Homage to Steven Vajda*, (pp. 115–135).

Kall, P., & Mayer, J. (2005). Building and Solving Stochastic Linear Programming Models with SLP-IOR. In S. W. Wallace, & W. T. Ziemba (Eds.), *Applications of Stochastic Programming* chapter 6. (pp. 79–93). Society for Industrial Mathematics.

Kall, P., & Mayer, J. (2010). *Stochastic Linear Programming: Models, Theory, and Computation*. (2nd ed.). Springer.

Kall, P., & Wallace, S. W. (1994). *Stochastic Programming*. (2nd ed.). Chichester: John Wiley & Sons.

Kallrath, J. (Ed.) (2004). *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers.

Kallrath, J. (Ed.) (2012). *Algebraic modeling system: Modeling and Solving Real World Optimization Problems*. Springer.

Karabuk, S. (2005). *An open source algebraic modeling and programming software*. Technical Report University of Oklahoma, School of Industrial Engineering Norman.

Karabuk, S. (2008). Extending algebraic modelling languages to support algorithm development for solving stochastic programming models. *IMA Journal of Management Mathematics*, *19*, 325–345.

Kaut, M. (2008). COIN-OR Tools for Stochastic Programming. In M. Kopa (Ed.), *On Selected Software for Stochastic Programming* (pp. 88–116). Prague: Matfyzpress.

Kaut, M., King, A. J., & Hultberg, T. H. (2008). *A C++ Modelling Environment for Stochastic Programming*. Technical Report RC24662 IBM Watson Research Center.

Kaut, M., & Wallace, S. W. (2003). Evaluation of scenario-generation methods for stochastic programming. *Stochastic Programming E-Print Series*, *14*, –.

Kaut, M., & Wallace, S. W. (2007). Evaluation of Scenario-Generation Methods for Stochastic Programming. *Pacific Journal of Optimization*, *3*, 257–271.

Kelley, J. E. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, *8*, 703–712.

King, A. J., & Wallace, S. W. (2012). *Modeling with stochastic programming*. Springer.

Kiwiel, K. C. (1985). *Methods of Descent for Nondifferentiable Optimization*. Springer.

Koberstein, A. (2005). *The Dual Simplex Method , Techniques for a fast and stable implementation*. Ph.D. thesis Universität Paderborn.

Koberstein, A., Lucas, C., Wolf, C., & König, D. (2011). Modeling and optimizing risk in the strategic gas-purchase planning problem of local distribution companies. *The Journal of Energy Markets*, *4*, 47–68.

Koberstein, A., Lukas, E., & Naumann, M. (2013). Integrated Strategic Planning of Global Production Networks and Financial Hedging under Uncertain Demand and Exchange Rates. *BuR - Business Research*, *Forthcoming*.

Kopa, M. (Ed.) (2008). *On Selected Software for Stochastic Programming*. Matfyzpress.

Kuhn, D. (2006). Aggregation and discretization in multistage stochastic programming. *Mathematical Programming*, *113*, 61–94.

Latorre, J. M., Cerisola, S., Ramos, A., & Palacios, R. (2008). Analysis of stochastic problem decomposition algorithms in computational grids. *Annals of Operations Research*, *166*, 355–373.

Lemaréchal, C. (1978). Nonsmooth optimization and descent methods. Research Report 78-4, IIASA, Laxenburg, Austria.

Lemaréchal, C., Nemirovskii, A., & Nesterov, Y. (1995). New variants of bundle methods. *Mathematical Programming*, *69*, 111–147.

Linderoth, J., Shapiro, A., & Wright, S. (2006). The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, *142*, 215–241.

Linderoth, J., & Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, *24*, 207–250.

LINDO Systems (2013). Stochastic Programming Features. `http://www.lindo.com/index.php?option=com_content&view=article&id=130&Itemid=54`. Last accessed 10/25/13.

Lougee-Heimer, R. (2003). The Common Optimization Interface for Operations Research. *IBM Journal of Research and Development*, *47*, 57–66.

Luenberger, D. G., & Ye, Y. (2008). *Linear and nonlinear programming*. (3rd ed.). Springer.

Mak, W., Morton, D. P., & Wood, R. K. (1999). Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, *24*, 47–56.

Maros, I. (2003). *Computational Techniques of the Simplex Method*. Kluwer Academic Publishers.

Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Miller, J. A., & Upton, M. (2002). Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Jounral*, *6*, 1–12.

Maturana, S. V. (1994). Issues in the design of modeling languages for mathematical programming. *European Journal of Operational Research*, *72*, 243–261.

Maximal Software (2013). New Stochastic Extensions for MPL. `http://www.maximalsoftware.com/maximal/news/stochastic.html`. Last accessed 10/25/13.

Messina, E. (1997). Modelling and analysis of multistage stochastic programming problems: A software environment. *European Journal of Operational Research*, *101*, 343–359.

Microsoft (2013). Optimization Modeling Language (OML). `http://msdn.microsoft.com/en-us/library/ff524507(v=vs.93).aspx`. Last accessed 10/25/13.

Mirkov, R., & Pflug, G. C. (2007). Tree Approximations of Dynamic Stochastic Programs. *SIAM Journal on Optimization*, *18*, 1082–1105.

Moritsch, H. (2006). *High Performance Computing in Finance — On the Parallel Implementation of Pricing and Optimization Models*. Ph.D. thesis Technische Universität Wien.

Moritsch, H. W., Pflug, G. C., & Siomak, M. (2001). Asynchronous nested optimization algorithms and their parallel implementation. *Wuhan University Journal of Natural Sciences*, *6*, 560–567.

Morton, D. P. (1996). An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling. *Annals of Operations Research*, *64*, 211–235.

Mulvey, J. M., & Ruszczyński, A. (1992). A diagonal quadratic approximation method for large scale linear programs. *Operations Research Letters*, *12*, 205–215.

Mulvey, J. M., & Ruszczyński, A. (1995). A New Scenario Decomposition Method for Large-Scale Stochastic Optimization. *Operations Research*, *43*, 477–490.

Nemhauser, G. L., & Wolsey, L. A. (1999). *Integer and Combinatorial Optimization*. Wiley-Interscience.

Nering, E., & Tucker, A. (1993). *Linear Programs and Related Problems*. Academic Press, Inc.

Nielsen, S., & Zenios, S. A. (1997). Scalable parallel Benders decomposition for stochastic linear programming. *Parallel Computing*, *23*, 1069–1088.

Oliveira, W., & Sagastizábal, C. (2012). Level bundle methods for oracles with on-demand accuracy. http://www.optimization-online.org/DB_HTML/2012/03/3390.html. Preprint. Instituto Nacional de Matemática Pura e Aplicada.

Paragon Decision Technology (2013). `http://www.aimms.com/operations-research/mathematical-programming/stochastic-programming`. Last accessed 10/25/13.

Parpas, P., & Rustem, B. (2007). Computational Assessment of Nested Benders and Augmented Lagrangian Decomposition for Mean-Variance Multistage Stochastic Problems. *INFORMS Journal on Computing*, *19*, 239–247.

Revuz, D., & Yor, M. (2004). *Continuous martingales and Brownian motion*. Springer.

Rockafellar, R. T. (1976a). Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming. *Mathematics of Operations Research*, *1*, 97–116.

Rockafellar, R. T. (1976b). Monotone Operators and the Proximal Point Algorithm. *SIAM Journal on Control and Optimization*, *14*, 877–898.

Rockafellar, R. T., & Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, *16*, 119–147.

Römisch, W. (2011). Scenario Generation. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, & J. C. Smith (Eds.), *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Ross, S. M. (2004). *Introduction to probability and statistics for engineers and scientists*. Elsevier.

Rubin, P. (2011). Farkas Certificates in CPLEX. `http://orinanobworld.blogspot.de/2011/07/farkas-certificates-in-cplex.html`. Last accessed 10/25/13.

Rudolph, D. (2010). *Eine open-source basierte Implementierung eines Lösers für stochastische zweistufige lineare Programme*. Diplomarbeit Universität Paderborn.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical programming*, *35*, 309–333.

Ruszczyński, A. (1993a). Parallel decomposition of multistage stochastic programming problems. *Mathematical Programming*, *58*, 201–228.

Ruszczyński, A. (1993b). Regularized decomposition of stochastic programs: Algorithmic techniques and numerical results. Working Paper 93-21, IIASA, Laxenburg, Austria.

Ruszczyński, A. (2003). Decomposition Methods. In A. Ruszczyński, & A. Shapiro (Eds.), *Handbooks in Operations Research and Management Science, Volume 10: Stochastic Programming* chapter 3. (pp. 141–211). volume 10.

Ruszczyński, A., & Shapiro, A. (Eds.) (2003). *Handbooks in Operations Research and Management Science, Volume 10: Stochastic Programming*. Elsevier.

Ruszczyński, A., & Świętanowski, A. (1997). Accelerating the regularized decomposition method for two stage stochastic linear problems. *European Journal of Operational Research*, *101*, 328–342.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. Wiley Interscience.

Shapiro, A. (2003). Monte Carlo Sampling Methods. In A. Ruszczyński, & A. Shapiro (Eds.), *Handbooks in Operations Research and Management Science, Volume 10: Stochastic Programming* chapter 6. (pp. 353–425). volume 10.

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: Modeling and Theory*. Society for Industrial Mathematics.

Shu, W., & Wu, M. (1993). Sparse implementation of revised simplex algorithms on parallel computers. In *The Sixth SIAM Conference on Parallel Processing for Scientific Computing* (pp. 501–509).

Thénié, J., Delft, C., & Vial, J. P. (2007). Automatic Formulation of Stochastic Programs Via an Algebraic Modeling Language. *Computational Management Science*, *4*, 17–40.

Trukhanov, S., Ntaimo, L., & Schaefer, A. (2010). Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, *206*, 395–406.

Valente, C., Mitra, G., Sadki, M., & Fourer, R. (2009). Extending algebraic modelling languages for Stochastic Programming. *INFORMS Journal on Computing*, *21*, 107–122.

Valente, P., Mitra, G., & Poojari, C. A. (2005). A Stochastic Programming Integrated Environment. In S. W. Wallace, & W. T. Ziemba (Eds.), *Applications of Stochastic Programming* chapter 8. (pp. 115–136). Society for Industrial Mathematics.

Van Slyke, R., & Wets, R. J.-B. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, *17*, 638–663.

Vanderbeck, F., & Wolsey, L. (2010). Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008* (pp. 431–502). Springer.

Vanderbei, R. (1997). *Linear programming: foundations and extensions*. Kluwer Academic Publishers.

Vespucci, M. T., Maggioni, F., Bertocchi, M. I., & Innorta, M. (2012). A stochastic model for the daily coordination of pumped storage hydro plants and wind power plants. *Annals of Operations Research*, *193*, 91–105.

Vladimirou, H. (1998). Computational assessment of distributed decomposition methods for stochastic linear programs. *European Journal of Operational Research*, *108*, 653–670.

Vladimirou, H., & Zenios, S. A. (1999). Scalable parallel computations for large-scale stochastic programming. *Annals of Operations Research*, *90*, 87–129.

Walkup, D. W., & Wets, R. J.-B. (1967). Stochastic programs with recourse. *SIAM Journal on Applied Mathematics*, *15*, 1299–1314.

Wallace, S. W. (2000). Decision making under uncertainty: Is sensitivity analysis of any use? *Operations Research*, *48*, 20–25.

Wallace, S. W., & Ziemba, W. T. (Eds.) (2005). *Applications of stochastic programming*. Society for Industrial Mathematics.

Watson, J.-P., Woodruff, D. L., & Hart, W. E. (2012). PySP: modeling and solving stochastic programs in Python. *Mathematical Programming Computation*, *4*, 109–149.

Wesselmann, F. (2010). *Generating General-Purpose Cutting Planes for Mixed-Integer Programs*. Ph.D. thesis Universität Paderborn.

Wittrock, R. J. (1983). *Advances in a nested decomposition algorithm for solving staircase linear programs. Technical report SOL 83-2*. Technical Report Stanford Univ., CA (USA). Systems Optimization Lab.

Wolf, C., & Koberstein, A. (2013). Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested L-shaped method. *European Journal of Operational Research*, *230*, 143–156.

Wolf, C., Koberstein, A., & Hultberg, T. H. (2011). Stochastic Extensions to FlopC++. In B. Hu, K. Morasch, S. Pickl, & M. Siegle (Eds.), *Operations Research Proceedings 2010* (pp. 333–338). Springer.

Ye, Y. (1997). *Interior Point Algorithms: Theory and Analysis*. Wiley-Interscience.

Zakeri, G., Philpott, A., & Ryan, D. (2000). Inexact cuts in Benders decomposition. *SIAM Journal on Optimization*, *10*, 643–657.

Zverovich, V., Fábián, C. I., Ellison, E. F. D., & Mitra, G. (2012). A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition. *Mathematical Programming Computation*, *4*, 211–238.

A. Test problems

This chapter contains the problems used in our test set. We give the instance names, number of scenarios, the test set the instance belongs to, and the number of columns and rows of the first stage and second stage. In addition, the number of column and row numbers as well as the number of nonzeroes of the deterministic equivalent problem are given. The two-stage test problems are shown in Table A.1.

For multi-stage problems, we give the number of columns and rows at stage $1 < t \leq T$, if the problem is symmetric and has the same number of columns and rows at every stage as well as the number of stages. The multi-stage test problems are given in Table A.2.

Table A.1. Problem dimensions of two-stage problems in our test set.

			Stage 1		Stage 2		DEQ		
Instance	**Scenarios**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
20x20-1_400	400	deak	20	10	30	20	12020	8010	72483
20x20-1_450	450	deak	20	10	30	20	13520	9010	81533
20x20-1_500	500	deak	20	10	30	20	15020	10010	90583
20x40-1_400	400	deak	20	10	60	40	24020	16010	184083
20x40-1_450	450	deak	20	10	60	40	27020	18010	207083
20x40-1_500	500	deak	20	10	60	40	30020	20010	230083
20x60-1_400	400	deak	20	10	90	60	36020	24010	344083
20x60-1_450	450	deak	20	10	90	60	40520	27010	387083
20x60-1_500	500	deak	20	10	90	60	45020	30010	430083
40x20-1_400	400	deak	40	20	30	20	12040	8020	122725
40x20-1_450	450	deak	40	20	30	20	13540	9020	138025
40x20-1_500	500	deak	40	20	30	20	15040	10020	153325
40x40-1_400	400	deak	40	20	60	40	24040	16020	288325
40x40-1_450	450	deak	40	20	60	40	27040	18020	324325
40x40-1_500	500	deak	40	20	60	40	30040	20020	360325
40x60-1_400	400	deak	40	20	90	60	36040	24020	400325
40x60-1_450	450	deak	40	20	90	60	40540	27020	450325
40x60-1_500	500	deak	40	20	90	60	45040	30020	500325
60x20-1_400	400	deak	60	30	30	20	12060	8030	173127
60x20-1_450	450	deak	60	30	30	20	13560	9030	194677
60x20-1_500	500	deak	60	30	30	20	15060	10030	216227
60x40-1_400	400	deak	60	30	60	40	24060	16030	386727
60x40-1_450	450	deak	60	30	60	40	27060	18030	434977
60x40-1_500	500	deak	60	30	60	40	30060	20030	483227
60x60-1_400	400	deak	60	30	90	60	36060	24030	648727
60x60-1_450	450	deak	60	30	90	60	40560	27030	729727

Table A.1. Problem dimensions (continued)

			Stage 1		Stage 2		DEQ		
Instance	**Scenarios**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
60x60-1_500	500	deak	60	30	90	60	45060	30030	810727
100x20-1_400	400	deak	100	50	30	20	12100	8050	121416
100x20-1_450	450	deak	100	50	30	20	13600	9050	136466
100x20-1_500	500	deak	100	50	30	20	15100	10050	151516
stormG2_8	8	posts	121	185	1259	528	10193	4409	27424
stormG2_27	27	posts	121	185	1259	528	34114	14441	90903
stormG2_125	125	posts	121	185	1259	528	157496	66185	418321
stormG2_1000	1000	posts	121	185	1259	528	1259121	528185	3341696
rand0_2000	2000	rand	100	50	50	25	100100	50050	754501
rand0_4000	4000	rand	100	50	50	25	200100	100050	1508501
rand0_6000	6000	rand	100	50	50	25	300100	150050	2262501
rand0_8000	8000	rand	100	50	50	25	400100	200050	3016501
rand0_10000	10000	rand	100	50	50	25	500100	250050	3770501
rand1_2000	2000	rand	200	100	100	50	200200	100100	3006001
rand1_4000	4000	rand	200	100	100	50	400200	200100	6010001
rand1_6000	6000	rand	200	100	100	50	600200	300100	9014001
rand1_8000	8000	rand	200	100	100	50	800200	400100	12018001
rand1_10000	10000	rand	200	100	100	50	1000200	500100	15022001
rand2_2000	2000	rand	300	150	150	75	300300	150150	6758501
rand2_4000	4000	rand	300	150	150	75	600300	300150	13512501
rand2_6000	6000	rand	300	150	150	75	900300	450150	20266501
rand2_8000	8000	rand	300	150	150	75	1200300	600150	27020501
rand2_10000	10000	rand	300	150	150	75	1500300	750150	33774501
20-1000	1000	sampling	63	3	764	124	764063	124003	4488063
20-2000	2000	sampling	63	3	764	124	1528063	248003	8976063
20-3000	3000	sampling	63	3	764	124	2292063	372003	13464063

Table A.1. Problem dimensions (continued)

			Stage 1		Stage 2		DEQ		
Instance	**Scenarios**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
gbd	646425	sampling	4	2	12	7	12000004	7000002	28000008
LandS	1000000	sampling	17	4	10	5	6464267	3232129	17453492
ssn-1000	1000	sampling	89	1	706	175	706089	175001	2373089
ssn-2000	2000	sampling	89	1	706	175	1412089	350001	4746089
ssn-3000	3000	sampling	89	1	706	175	2118089	525001	7119089
storm-1000	1000	sampling	121	185	1259	528	1259121	528185	3341696
storm-2000	2000	sampling	121	185	1259	528	2518121	1056185	6682696
storm-3000	3000	sampling	121	185	1259	528	3777121	1584185	10023696
saphir_50	50	saphir	53	32	3924	8678	196253	433932	1136753
saphir_100	100	saphir	53	32	3924	8678	392453	867832	2273403
saphir_500	500	saphir	53	32	3924	8678	1962053	4339032	11366603
saphir_1000	1000	saphir	53	32	3924	8678	3924053	8678032	22733103
sslp_10_50_50	50	SIPLIB	10	1	510	60	25510	3001	50460
sslp_10_50_100	100	SIPLIB	10	1	510	60	51010	6001	100910
sslp_10_50_500	500	SIPLIB	10	1	510	60	255010	30001	504510
sslp_10_50_1000	1000	SIPLIB	10	1	510	60	510010	60001	1009010
sslp_10_50_2000	2000	SIPLIB	10	1	510	60	1020010	120001	2018010
sslp_15_45_5	5	SIPLIB	15	1	690	60	3465	301	6835
sslp_15_45_10	10	SIPLIB	15	1	690	60	6915	601	13655
sslp_15_45_15	15	SIPLIB	15	1	690	60	10365	901	20475
airl	25	slptestset	4	2	8	6	204	152	604
airl2	25	slptestset	4	2	8	6	204	152	604
assets-small	100	slptestset	13	5	13	5	1313	505	2621
assets-large	37500	slptestset	13	5	13	5	487513	187505	975021
4node-2	2	slptestset	52	14	186	74	424	162	1191
4node-4	4	slptestset	52	14	186	74	796	310	2127

Table A.1. Problem dimensions (continued)

			Stage 1		Stage 2		DEQ		
Instance	**Scenarios**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
4node-8	8	slptestset	52	14	186	74	1540	606	3999
4node-16	16	slptestset	52	14	186	74	3028	1198	7743
4node-32	32	slptestset	52	14	186	74	6004	2382	15231
4node-64	64	slptestset	52	14	186	74	11956	4750	30207
4node-128	128	slptestset	52	14	186	74	23860	9486	60159
4node-256	256	slptestset	52	14	186	74	47668	18958	120063
4node-512	512	slptestset	52	14	186	74	95284	37902	239871
4node-1024	1024	slptestset	52	14	186	74	190516	75790	479487
4node-2048	2048	slptestset	52	14	186	74	380980	151566	958719
4node-4096	4096	slptestset	52	14	186	74	761908	303118	1917183
4node-8192	8192	slptestset	52	14	186	74	1523764	606222	3834111
4node-16384	16384	slptestset	52	14	186	74	3047476	1212430	7667967
4node-32768	32768	slptestset	52	14	186	74	6094900	2424846	15335679
chem	2	slptestset	39	38	41	46	121	130	289
LandS	3	slptestset	4	2	12	7	40	23	92
env-aggr	5	slptestset	49	48	49	48	294	288	852
env-first	5	slptestset	49	48	49	48	1613521	1580592	4741764
env-loose	5	slptestset	49	48	49	48	294	288	852
env-imp	15	slptestset	49	48	49	48	784	768	2292
env-1200	1200	slptestset	49	48	49	48	58849	57648	172932
env-1875	1875	slptestset	49	48	49	48	91924	90048	270132
env-3780	3780	slptestset	49	48	49	48	185269	181488	544452
env-5292	5292	slptestset	49	48	49	48	259357	254064	762180
env-lrge	8232	slptestset	49	48	49	48	294	288	852
env-xlrge	32928	slptestset	49	48	49	48	403417	395184	1185540
phone	32768	slptestset	8	1	85	23	2785288	753665	9863176

Table A.1. Problem dimensions (continued)

			Stage 1		Stage 2		DEQ		
Instance	**Scenarios**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
stocfor2	64	slptestset	15	15	96	102	6159	6543	26907

Table A.2. Problem dimensions of multi-stage problems in our test-set.

				Stage 1		Stage t		DEQ		
Instance	**Scenarios**	**Stages**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
fxm3_6	64	3	posts	114	92	99	82	9492	6200	54589
fxm3_16	256	3	posts	114	92	99	82	64162	41340	370839
fxm4_6	216	4	posts	114	92	99	82	30732	22400	248989
fxm4_16	4096	4	posts	114	92	99	82	517282	386940	4518039
pltexpA3_6	64	3	posts	188	62	272	104	11612	4430	23611
pltexpA3_16	256	3	posts	188	62	272	104	74172	28350	150801
pltexpA4_6	216	4	posts	188	62	272	104	70364	26894	143059
pltexpA4_16	4096	4	posts	188	62	272	104	1188284	454334	2415889
pltexpA5_6	1296	5	posts	188	62	272	104	422876	161678	859747
pltexpA5_16	65536	5	posts	188	62	272	104	19014076	7270078	38657297
pltexpA6_6	7776	6	posts	188	62	272	104	2537948	970382	5159875
pltexpA6_16	1048576	6	posts	188	62	272	104	304226748	116321982	618519825
pltexpA7_6	46656	7	posts	188	62	272	104	15228380	5822606	30960643
scdp-64000	64000	4	scdp	83	45	61	37	2448923	1910325	10574919
scdp-1024	1024	6	scdp	95	49	85	45	55939	41397	248801
scdp-4096	4096	7	scdp	95	49	85	45	223811	165621	1000929
scdp-16384	16384	8	scdp	95	49	85	45	895299	662517	4009441
scdp-65536	65536	9	scdp	95	49	85	45	3581251	2650101	16043489
sgpf3y3	25	3	slptestset	87	38	51	39	1617	1208	4090
sgpf5y3	25	3	slptestset	139	62	79	63	2509	1952	6570
sgpf3y4	125	4	slptestset	87	38	51	39	7992	6083	20590
sgpf5y4	125	4	slptestset	139	62	79	63	12384	9827	33070
sgpf3y5	625	5	slptestset	87	38	51	39	39867	30458	103090
sgpf5y5	625	5	slptestset	139	62	79	63	61759	49202	165570
sgpf3y6	3125	6	slptestset	87	38	51	39	199242	152333	515590
sgpf5y6	3125	6	slptestset	139	62	79	63	308634	246077	828070

Table A.2. Problem dimensions (continued)

				Stage 1		Stage t		DEQ		
Instance	**Scenarios**	**Stages**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
sgpf3y7	15625	7	slptestset	87	38	51	39	996117	761708	2578090
sgpf5y7	15625	7	slptestset	139	62	79	63	1543009	1230452	4140570
stocfor2_7	2	7	slptestset	15	15	16	17	2031	2157	8847
stocfor3	4	7	slptestset	15	15	16	17	15695	16675	68627
WAT_C_10_16	16	10	watson	15	11	0	0	8401	4573	21368
WAT_C_10_32	32	10	watson	15	11	0	0	15553	8413	39848
WAT_C_10_64	64	10	watson	15	11	0	0	28097	15101	72648
WAT_C_10_128	128	10	watson	15	11	0	0	49153	26237	128648
WAT_C_10_256	256	10	watson	15	11	0	0	82177	43517	218888
WAT_C_10_512	512	10	watson	15	11	0	0	128001	67069	350728
WAT_C_10_768	768	10	watson	15	11	0	0	191994	100598	526078
WAT_C_10_1024	1024	10	watson	15	11	0	0	255987	134127	701428
WAT_C_10_1152	1152	10	watson	15	11	0	0	287949	150869	789028
WAT_C_10_1536	1536	10	watson	15	11	0	0	383927	201155	1052028
WAT_C_10_1920	1920	10	watson	15	11	0	0	479905	251441	1315028
WAT_C_10_2304	2304	10	watson	15	11	0	0	575883	301727	1578028
WAT_C_10_2688	2688	10	watson	15	11	0	0	671861	352013	1841028
WAT_I_10_16	16	10	watson	15	11	0	0	8401	4573	21368
WAT_I_10_32	32	10	watson	15	11	0	0	15553	8413	39848
WAT_I_10_64	64	10	watson	15	11	0	0	28097	15101	72648
WAT_I_10_128	128	10	watson	15	11	0	0	49153	26237	128648
WAT_I_10_256	256	10	watson	15	11	0	0	82177	43517	218888
WAT_I_10_512	512	10	watson	15	11	0	0	128001	67069	350728
WAT_I_10_768	768	10	watson	15	11	0	0	191994	100598	526078
WAT_I_10_1024	1024	10	watson	15	11	0	0	255987	134127	701428
WAT_I_10_1152	1152	10	watson	15	11	0	0	287949	150869	789028

Table A.2. Problem dimensions (continued)

				Stage 1		Stage t		DEQ		
Instance	**Scenarios**	**Stages**	**Testset**	**Cols**	**Rows**	**Cols**	**Rows**	**Cols**	**Rows**	**NZ**
WAT_I_10_1536	1536	10	watson	15	11	0	0	383927	201155	1052028
WAT_I_10_1920	1920	10	watson	15	11	0	0	479905	251441	1315028

B. Test Results

This chapter contains detailed test results of our experiments in the form of tables. We do not give detailed results for every tested parameter combination due to space considerations, but give results for selected parameter combinations instead. Table B.1 gives the results for two-stage problems. Table B.2 contains the iteration counts for manhattan level decomposition with on-demand accuracy, whereas the iteration counts for infinity level decomposition are given in Table B.3. Results for multi-stage problems are given in Table B.4. Finally, detailed speed-up values are shown in Table B.5.

Table B.1. Computing times of several selected algorithms on the two-stage test set. The algorithms are Benders base case (BC), Benders with on-demand accuracy (BC-ODA), Benders multi-cut (MC), euclidean level decomposition (LevelE), euclidean level decomposition with on-demand accuracy (LevelE-ODA), infinity level decomposition (LevelI), infinity level decomposition with on-demand accuracy (LevelI-ODA), manhattan level decomposition (LevelM), manhattan level decomposition with on-demand accuracy (LevelM-ODA), and the deterministic equivalent solved with the barrier method (DEM). $\lambda = 0.7$ for level decomposition without on-demand accuracy and $\lambda = 0.5$ for level decomposition with on-demand accuracy. $\kappa = 0.5$ for all on-demand accuracy methods.

Instance	BC	BC-ODA	MC	LevelE	LevelE	LevelI	LevelI	LevelM	LevelM-ODA	DEM
20x20-1_400	0.13	0.10	0.10	0.15	0.16	0.15	0.13	0.13	0.11	0.36
20x20-1_450	0.13	0.11	0.10	0.19	0.13	0.17	0.12	0.12	0.12	1.02
20x20-1_500	0.14	0.14	0.11	0.20	0.13	0.15	0.13	0.16	0.13	0.35
20x40-1_400	0.13	0.11	0.12	0.15	0.21	0.13	0.16	0.15	0.14	0.59
20x40-1_450	0.15	0.13	0.14	0.18	0.16	0.16	0.18	0.16	0.14	0.66
20x40-1_500	0.15	0.15	0.13	0.23	0.16	0.15	0.16	0.14	0.15	0.64
20x60-1_400	0.96	0.48	0.39	0.49	0.37	0.55	0.41	0.49	0.39	0.48
20x60-1_450	1.04	0.54	0.51	0.55	0.41	0.58	0.45	0.55	0.41	0.65
20x60-1_500	1.15	0.65	0.52	0.52	0.46	0.73	0.46	0.55	0.48	0.82
40x20-1_400	0.19	0.15	0.13	0.19	0.22	0.16	0.16	0.16	0.20	0.42
40x20-1_450	0.18	0.15	0.12	0.21	0.20	0.16	0.16	0.18	0.18	0.40
40x20-1_500	0.23	0.16	0.14	0.20	0.19	0.19	0.17	0.20	0.20	0.38
40x40-1_400	0.23	0.16	0.19	0.25	0.27	0.20	0.18	0.18	0.20	1.35
40x40-1_450	0.24	0.19	0.18	0.26	0.28	0.25	0.19	0.20	0.19	2.96
40x40-1_500	0.28	0.19	0.19	0.22	0.20	0.23	0.23	0.23	0.20	1.48
40x60-1_400	1.43	0.68	0.50	0.62	0.51	0.72	0.52	0.64	0.41	0.98
40x60-1_450	1.46	0.71	0.66	0.76	0.51	0.75	0.58	0.72	0.51	1.11
40x60-1_500	1.88	0.80	0.56	0.91	0.57	0.80	0.55	0.76	0.58	0.99
60x20-1_400	0.50	0.36	0.23	0.44	0.47	0.37	0.37	0.45	0.37	0.37
60x20-1_450	0.57	0.38	0.23	0.45	0.37	0.40	0.38	0.41	0.37	0.58
60x20-1_500	0.64	0.40	0.27	0.48	0.41	0.43	0.41	0.48	0.41	0.56

Table B.1. Computing times (continued)

Instance	BC	BC-ODA	MC	LevelE	LevelE	LevelI	LevelI	LevelM	LevelM-ODA	DEM
60x40-1_400	1.42	0.64	0.46	0.68	0.55	0.61	0.54	0.69	0.52	0.94
60x40-1_450	1.71	0.73	0.50	0.69	0.58	0.76	0.60	0.72	0.58	0.98
60x40-1_500	1.74	0.76	0.56	0.72	0.58	0.72	0.59	0.92	0.53	1.01
60x60-1_400	2.08	0.69	0.52	0.69	0.72	0.88	0.62	0.81	0.55	1.40
60x60-1_450	2.38	0.74	0.64	0.85	0.62	0.89	0.60	0.90	0.62	1.53
60x60-1_500	2.62	0.76	0.73	1.09	0.65	0.96	0.73	1.14	0.68	1.60
100x20-1_400	0.86	0.55	0.30	0.74	0.85	0.56	0.57	0.67	0.77	0.62
100x20-1_450	0.87	0.52	0.32	1.05	1.02	0.70	0.66	0.75	0.79	0.58
100x20-1_500	0.84	0.70	0.34	1.05	1.10	0.76	0.65	0.76	0.72	0.69
stormG2_8	0.11	0.09	0.11	0.14	0.13	0.13	0.12	0.13	0.13	0.14
stormG2_27	0.18	0.14	0.11	0.20	0.17	0.17	0.15	0.20	0.17	0.72
stormG2_125	0.42	0.27	0.21	0.34	0.27	0.35	0.25	0.32	0.28	2.40
stormG2_1000	2.64	1.22	1.04	1.64	1.20	1.58	1.22	1.97	1.27	22.65
rand0_2000	1.90	0.97	0.60	1.17	0.93	1.24	0.86	1.24	1.05	4.00
rand0_4000	3.24	1.83	1.12	1.71	1.38	1.65	1.30	1.71	1.31	11.02
rand0_6000	7.26	3.52	1.94	3.38	2.49	3.56	2.66	3.69	2.73	16.30
rand0_8000	9.77	4.23	2.63	3.18	2.65	4.14	2.94	3.78	2.81	31.14
rand0_10000	22.91	8.61	3.85	5.70	4.50	7.58	5.27	6.58	5.19	45.46
rand1_2000	27.13	12.60	4.77	5.57	4.50	6.02	4.65	6.79	5.44	14.05
rand1_4000	59.97	25.54	9.92	8.74	6.51	11.03	9.46	11.19	7.67	33.38
rand1_6000	72.19	28.96	12.84	11.21	8.08	14.10	9.79	13.85	10.42	54.53
rand1_8000	112.81	41.83	18.78	17.15	11.58	18.91	13.23	21.01	12.35	85.96
rand1_10000	149.92	60.54	22.61	24.13	15.12	27.71	18.88	27.38	17.50	117.21
rand2_2000	168.04	73.65	23.46	13.85	10.32	19.25	14.02	22.09	17.39	41.37
rand2_4000	137.82	40.84	21.68	17.44	13.05	23.14	15.51	20.76	15.74	88.85
rand2_6000	280.81	81.04	39.84	28.87	21.97	39.22	25.33	39.03	30.34	152.73
rand2_8000	345.02	85.11	41.57	31.40	23.57	44.72	29.59	38.89	29.31	213.73

Table B.1. Computing times (continued)

Instance	BC	BC-ODA	MC	LevelE	LevelE	LevelI	LevelI	LevelM	LevelM-ODA	DEM
rand2_10000	607.98	183.96	67.16	47.34	35.88	70.69	46.36	68.05	42.28	269.10
20-1000	116.25	71.57	64.51	18.88	11.77	10.02	10.07	22.51	14.34	6.70
20-2000	204.58	217.00	120.96	34.02	20.19	19.77	28.36	41.07	18.33	13.17
20-3000	291.01	314.25	155.25	46.51	30.47	18.90	28.85	49.67	32.13	3600.00
gbd	120.55	74.94	65.72	143.44	109.86	142.15	106.92	192.26	129.96	245.53
LandS	229.11	83.73	59.41	163.52	73.97	170.65	84.14	186.32	71.12	131.60
ssn-1000	244.08	87.84	13.68	9.49	4.92	11.04	5.75	21.44	12.89	20.45
ssn-2000	437.13	107.34	26.80	26.36	16.18	30.76	16.72	50.08	30.87	58.00
ssn-3000	620.83	129.77	34.59	45.72	26.79	44.04	22.81	69.80	43.80	3600.00
storm-1000	3.20	1.30	1.17	1.86	1.38	1.76	1.34	2.22	1.43	22.92
storm-2000	5.89	2.57	2.24	3.08	2.63	4.03	2.46	3.57	2.28	52.35
storm-3000	8.65	3.49	3.05	4.84	3.57	4.97	3.75	5.90	3.42	83.41
saphir_50	57.25	31.42	12.96	35.80	29.41	36.64	29.64	39.36	36.69	7.54
saphir_100	80.11	50.31	19.80	52.64	45.68	59.64	56.68	60.58	54.24	3600.00
saphir_500	257.13	141.19	61.28	177.68	134.47	174.03	154.21	178.78	151.60	799.34
saphir_1000	393.63	249.46	196.26	256.74	239.32	236.00	190.20	262.20	233.14	3600.00
sslp_10_50_50	15.76	26.31	125.26	15.01	18.53	16.75	29.41	14.93	19.20	5.20
sslp_10_50_100	15.49	22.91	120.57	15.58	17.57	14.57	31.22	14.57	17.35	11.76
sslp_10_50_500	27.01	30.64	142.82	21.57	27.11	30.65	39.44	21.88	25.55	215.04
sslp_10_50_1000	48.03	41.63	144.55	35.05	37.79	49.19	56.01	33.74	34.64	754.38
sslp_10_50_2000	80.52	65.16	220.93	51.43	57.69	81.20	66.13	50.57	57.52	3476.78
sslp_15_45_5	1.90	3.23	1.86	5.00	4.78	2.25	3.17	5.05	3.45	0.13
sslp_15_45_10	5.35	11.59	4.18	6.08	7.29	5.48	7.65	5.87	7.60	0.48
sslp_15_45_15	5.00	11.65	7.50	7.15	7.29	6.41	5.87	6.76	6.68	0.29
airl	0.07	0.06	0.07	0.08	0.09	0.07	0.09	0.07	0.08	0.05
airl2	0.06	0.06	0.06	0.08	0.08	0.07	0.06	0.07	0.07	0.05
assets-small	0.09	0.06	0.07	0.09	0.09	0.09	0.08	0.09	0.08	0.05

Table B.1. Computing times (continued)

Instance	BC	BC-ODA	MC	LevelE	LevelE	LevelI	LevelI	LevelM	LevelM-ODA	DEM
assets-large	1.06	1.10	1.27	5.04	3.43	4.82	3.60	4.63	3.59	3.53
4node-2	0.09	0.09	0.10	0.16	0.14	0.11	0.11	0.12	0.13	0.05
4node-4	0.10	0.08	0.09	0.13	0.16	0.10	0.12	0.12	0.11	0.05
4node-8	0.11	0.11	0.10	0.16	0.13	0.12	0.11	0.13	0.10	0.05
4node-16	0.13	0.09	0.07	0.16	0.18	0.16	0.11	0.13	0.12	0.13
4node-32	0.18	0.12	0.11	0.17	0.18	0.16	0.12	0.15	0.13	0.16
4node-64	0.18	0.16	0.12	0.21	0.24	0.16	0.19	0.19	0.16	0.18
4node-128	0.29	0.18	0.16	0.27	0.30	0.24	0.23	0.21	0.20	0.13
4node-256	0.73	0.30	0.30	0.35	0.39	0.27	0.37	0.30	0.31	0.31
4node-512	1.35	0.55	0.35	0.62	0.55	0.54	0.52	0.48	0.44	1.50
4node-1024	2.18	0.86	0.46	0.94	0.60	0.83	0.69	0.89	0.64	3.03
4node-2048	6.09	2.07	1.50	1.76	1.37	1.40	1.19	1.54	1.26	2.78
4node-4096	15.99	4.76	3.23	3.59	2.82	2.96	2.59	3.33	2.77	6.91
4node-8192	33.73	9.97	6.37	7.64	4.97	5.37	4.43	6.03	6.17	11.21
4node-16384	69.28	18.95	10.13	14.31	12.21	10.79	9.08	12.00	12.13	22.13
4node-32768	145.33	29.78	25.01	27.32	21.81	22.40	17.71	23.24	23.99	52.66
chem	0.06	0.06	0.08	0.09	0.08	0.08	0.08	0.07	0.07	0.05
LandS	0.07	0.06	0.06	0.08	0.08	0.07	0.06	0.06	0.06	0.05
env-aggr	0.07	0.07	0.09	0.11	0.09	0.09	0.09	0.10	0.08	0.05
env-first	0.08	0.08	0.08	0.08	0.07	0.08	0.08	0.08	0.08	0.49
env-loose	0.07	0.07	0.07	0.08	0.08	0.06	0.06	0.06	0.08	0.05
env-imp	0.07	0.07	0.06	0.10	0.10	0.09	0.09	0.11	0.09	0.05
env-1200	0.21	0.18	0.21	0.74	0.53	0.73	0.45	0.63	0.55	2.36
env-1875	0.38	0.40	0.40	1.07	0.71	1.01	0.72	1.01	0.71	2.22
env-3780	0.76	0.73	0.89	2.04	1.41	1.94	1.37	2.05	1.44	5.77
env-5292	1.01	1.04	1.15	2.77	1.84	2.78	1.85	2.70	1.88	9.10
env-lrge	1.54	1.68	1.69	4.25	2.80	3.96	2.71	4.02	2.87	20.09

Table B.1. Computing times (continued)

Instance	BC	BC-ODA	MC	LevelE	LevelE	LevelI	LevelI	LevelM	LevelM-ODA	DEM
env-xlrge	5.91	6.15	6.49	16.75	10.95	16.16	10.82	16.18	11.83	3600.00
phone	1.37	1.34	1.36	1.05	1.12	1.08	1.14	1.05	1.15	18.16
stocfor2	0.10	0.09	0.08	0.11	0.10	0.10	0.10	0.12	0.10	0.06

		Subst. Iterations		Iterations		$\frac{\text{Subst. It.}}{\text{Overall It.}}$
λ	κ	AM	SGM	AM	SGM	AM
0.1	0.1	50.99	33.52	107.16	61.67	0.48
0.1	0.3	44.01	29.29	107.77	63.05	0.41
0.1	0.5	39.45	26.51	112.92	65.73	0.35
0.1	0.7	35.87	24.07	115.53	68.00	0.31
0.1	0.9	34.16	23.06	124.53	74.89	0.27
0.3	0.1	50.61	34.97	86.54	56.18	0.58
0.3	0.3	45.31	31.56	88.20	57.74	0.51
0.3	0.5	43.00	28.64	93.26	60.06	0.46
0.3	0.7	40.59	27.38	97.04	62.94	0.42
0.3	0.9	37.86	25.74	99.84	65.44	0.38
0.5	0.1	53.86	36.83	86.27	56.67	0.62
0.5	0.3	48.79	33.77	86.70	56.69	0.56
0.5	0.5	45.78	31.17	89.28	57.94	0.51
0.5	0.7	43.93	30.09	94.52	61.72	0.46
0.5	0.9	42.05	29.02	98.50	66.11	0.43
0.7	0.1	58.35	41.69	89.35	58.91	0.65
0.7	0.3	53.56	38.58	89.86	59.82	0.60
0.7	0.5	51.42	36.39	93.08	61.84	0.55
0.7	0.7	50.28	35.05	97.48	64.84	0.52
0.7	0.9	48.51	34.06	100.20	68.75	0.48
0.9	0.1	79.65	60.16	108.56	75.71	0.73
0.9	0.3	75.40	57.42	110.10	76.59	0.68
0.9	0.5	72.44	54.67	112.02	77.90	0.65
0.9	0.7	69.10	52.72	114.56	81.91	0.60
0.9	0.9	71.47	53.28	124.80	90.67	0.57
0.1		107.63	60.68	107.63	60.68	1.00
0.3		89.50	55.73	89.50	55.73	1.00
0.5		85.24	54.51	85.24	54.51	1.00
0.7		90.40	57.93	90.40	57.93	1.00
0.9		107.94	75.18	107.94	75.18	1.00

Table B.2. Iteration counts for manhattan level decomposition with and without on-demand accuracy, for different λ and κ combinations. The last column gives the amount of substantial iterations with respect to all iterations as measured by the arithmetic mean.

		Subst. Iterations		Iterations		$\frac{\text{Subst. It.}}{\text{Overall It.}}$
λ	κ	AM	SGM	AM	SGM	AM
0.1	0.1	48.65	34.45	107.70	63.79	0.45
0.1	0.3	40.50	29.00	107.28	63.50	0.38
0.1	0.5	36.70	26.47	112.75	66.79	0.33
0.1	0.7	33.55	24.54	121.59	70.87	0.28
0.1	0.9	32.30	23.37	141.55	77.68	0.23
0.3	0.1	43.52	33.17	80.81	55.96	0.54
0.3	0.3	37.95	29.20	82.56	56.35	0.46
0.3	0.5	35.34	27.03	88.84	59.12	0.40
0.3	0.7	34.61	25.79	99.64	63.15	0.35
0.3	0.9	33.93	25.44	103.95	67.63	0.33
0.5	0.1	43.98	34.37	76.31	54.10	0.58
0.5	0.3	38.82	30.86	78.28	55.20	0.50
0.5	0.5	38.54	29.74	86.48	58.53	0.45
0.5	0.7	35.77	27.94	88.60	61.03	0.40
0.5	0.9	36.33	27.82	96.68	65.59	0.38
0.7	0.1	48.29	39.10	77.30	57.34	0.62
0.7	0.3	44.69	36.32	81.07	58.93	0.55
0.7	0.5	42.38	33.99	83.63	60.44	0.51
0.7	0.7	41.17	32.94	87.40	62.61	0.47
0.7	0.9	41.96	33.26	94.82	69.07	0.44
0.9	0.1	74.23	58.54	102.43	76.63	0.72
0.9	0.3	71.57	56.31	105.28	77.90	0.68
0.9	0.5	66.49	53.13	107.04	80.23	0.62
0.9	0.7	64.81	51.63	113.40	84.28	0.57
0.9	0.9	62.65	49.94	121.51	90.80	0.52
0.1		105.33	61.58	105.33	61.58	1.00
0.3		77.85	53.81	77.85	53.81	1.00
0.5		72.80	52.60	72.80	52.60	1.00
0.7		72.91	55.09	72.91	55.09	1.00
0.9		103.46	77.17	103.46	77.17	1.00

Table B.3. Iteration counts for infinity level decomposition with and without on- demand accuracy, for different λ and κ combinations. The last column gives the amount of substantial iterations with respect to all iterations as measured by the arithmetic mean.

Table B.4. Computional results for the multi-stage test set for different sequencing protocols: FastForwardFastBack (FFFB), FastForward (FF), FastBack (FB), ϵ-variants of the former, and the dynamic protocol.

Instance	FFFB	FF	FB	ϵ-FF-0.01	ϵ-FF-0.064	ϵ-FF-0.1	ϵ-FB-0.01	ϵ-FB-0.064	ϵ-FB-0.1	Dynamic
sgpf3y3	0.06	0.06	0.06	0.06	0.06	0.08	0.06	0.06	0.06	0.06
sgpf3y4	0.08	0.08	0.09	0.08	0.08	0.08	0.08	0.08	0.08	0.08
sgpf3y5	0.11	0.12	0.09	0.10	0.09	0.10	0.09	0.09	0.09	0.09
sgpf3y6	0.27	0.54	0.24	0.29	0.28	0.27	0.24	0.22	0.24	0.22
sgpf3y7	0.95	3.56	0.81	0.97	0.97	0.96	0.73	0.73	0.74	0.74
sgpf5y3	0.06	0.06	0.07	0.06	0.06	0.06	0.06	0.06	0.06	0.06
sgpf5y4	0.08	0.06	0.07	0.07	0.08	0.09	0.08	0.07	0.09	0.08
sgpf5y5	0.09	0.08	0.09	0.09	0.09	0.09	0.08	0.09	0.09	0.09
sgpf5y6	0.19	0.22	0.17	0.21	0.16	0.17	0.19	0.19	0.22	0.20
sgpf5y7	0.58	0.61	0.62	0.62	0.63	0.59	0.59	0.58	0.60	0.63
stocfor2_7	0.12	0.34	0.34	0.14	0.12	0.13	0.18	0.15	0.14	0.13
stocfor3	0.36	1.52	1.11	0.36	0.36	0.36	0.42	0.34	0.36	0.35
fxm3_16	0.34	0.35	0.37	0.30	0.33	0.34	0.36	0.33	0.42	0.32
fxm3_6	0.13	0.15	0.18	0.13	0.13	0.13	0.14	0.16	0.16	0.13
fxm4_16	0.76	0.89	0.58	0.80	0.68	0.71	0.60	0.62	0.64	0.72
fxm4_6	0.17	0.19	0.16	0.17	0.17	0.17	0.16	0.14	0.16	0.16
pltexpA3_16	0.13	0.23	0.11	0.15	0.13	0.13	0.09	0.11	0.10	0.09
pltexpA3_6	0.08	0.11	0.08	0.08	0.08	0.09	0.08	0.08	0.08	0.08
pltexpA4_16	1.08	41.71	0.59	2.78	1.72	1.59	0.64	0.59	0.64	0.66
pltexpA4_6	0.16	1.69	0.15	0.33	0.22	0.19	0.14	0.13	0.12	0.12
pltexpA5_16	16.94	2256.38	9.02	50.20	32.17	32.93	9.09	9.58	10.47	9.94
pltexpA5_6	0.48	36.28	0.55	1.64	1.17	1.19	0.37	0.33	0.34	0.42
pltexpA6_16	343.70	3600.00	144.40	1826.81	895.73	724.72	166.43	182.91	198.80	197.66
pltexpA6_6	2.61	3182.16	3.54	12.32	5.30	4.04	1.51	1.70	1.75	1.73
pltexpA7_6	14.94	3600.00	22.03	51.24	31.04	32.72	8.86	9.57	10.33	9.97
scdp-1024	1.13	1.18	2.48	1.16	1.15	1.15	2.86	2.09	1.88	1.76
scdp-16384	8.23	8.35	7.41	8.29	8.42	8.44	6.08	5.16	4.46	4.41
scdp-4096	2.57	2.60	4.33	2.58	2.59	2.61	3.88	2.01	2.11	2.36
scdp-64000	8.14	8.19	4.56	7.97	8.17	8.10	4.49	4.48	4.50	4.41
scdp-65536	34.45	36.29	20.15	35.00	35.20	35.33	19.21	17.88	17.20	13.72

Table B.4. Computational results for the multi-stage test set (continued)

Instance	FFFB	FF	FB	ϵ-FF-0.01	ϵ-FF-0.064	ϵ-FF-0.1	ϵ-FB-0.01	ϵ-FB-0.064	ϵ-FB-0.1	Dynamic
WAT_C_10_1024	0.65	2.29	2.21	0.64	0.65	0.64	1.03	0.73	0.66	0.70
WAT_C_10_1152	0.71	2.56	1.55	0.72	0.71	0.71	0.83	0.65	0.72	0.68
WAT_C_10_128	0.22	0.43	0.36	0.24	0.23	0.23	0.25	0.23	0.20	0.20
WAT_C_10_1536	0.94	3.06	3.02	0.97	0.96	0.98	1.22	1.02	0.97	0.99
WAT_C_10_16	0.10	0.16	0.14	0.15	0.10	0.11	0.16	0.10	0.11	0.11
WAT_C_10_1920	1.18	3.45	3.65	1.22	1.19	1.18	1.66	1.29	1.15	1.15
WAT_C_10_2304	1.79	15.87	7.25	1.89	1.89	1.87	2.07	1.44	1.42	1.46
WAT_C_10_256	0.29	0.62	0.76	0.32	0.30	0.29	0.37	0.31	0.28	0.29
WAT_C_10_2688	1.86	6.94	5.46	1.84	1.83	1.83	2.17	1.59	1.51	1.58
WAT_C_10_32	0.12	0.21	0.18	0.13	0.12	0.13	0.14	0.14	0.13	0.12
WAT_C_10_512	0.38	0.79	0.69	0.35	0.36	0.34	0.47	0.36	0.34	0.34
WAT_C_10_64	0.18	0.46	0.60	0.21	0.20	0.19	0.34	0.24	0.21	0.19
WAT_C_10_768	0.55	1.52	1.32	0.56	0.54	0.55	0.70	0.52	0.51	0.55
WAT_I_10_1024	0.85	4.95	3.14	0.86	0.87	0.86	1.08	0.83	0.81	0.93
WAT_I_10_1152	2.56	5.44	6.34	2.96	2.96	2.68	3.25	3.41	3.30	2.75
WAT_I_10_128	0.24	0.66	0.80	0.23	0.25	0.24	0.30	0.27	0.23	0.27
WAT_I_10_1536	3.43	157.33	11.33	3.17	3.38	3.71	4.34	4.07	4.16	3.24
WAT_I_10_16	0.11	0.12	0.13	0.09	0.10	0.09	0.15	0.12	0.11	0.11
WAT_I_10_1920	3.69	27.68	14.24	4.01	4.20	3.92	4.81	5.48	3.81	3.82
WAT_I_10_256	0.39	0.94	0.96	0.40	0.40	0.40	0.51	0.43	0.40	0.40
WAT_I_10_32	0.11	0.18	0.20	0.11	0.12	0.12	0.16	0.13	0.13	0.13
WAT_I_10_512	0.42	1.97	1.23	0.46	0.43	0.45	0.57	0.41	0.40	0.40
WAT_I_10_64	0.18	0.45	0.62	0.19	0.20	0.18	0.34	0.23	0.19	0.22
WAT_I_10_768	0.68	3.31	1.87	0.68	0.68	0.69	0.85	0.63	0.66	0.67

	1	2	3	4	5	6	7	8
Benders BC	1.00	1.91	2.75	3.46	3.74	4.08	4.29	4.55
LevelE	1.00	1.84	2.62	3.17	3.42	3.57	3.73	3.80
LevelE-ODA	1.00	1.75	2.45	2.88	3.07	3.24	3.31	3.40
Benders-ODA	1.00	1.70	2.27	2.70	2.88	3.00	3.14	3.16
LevelM	1.00	1.89	2.59	3.24	3.47	3.73	3.77	3.84
LevelM-ODA	1.00	1.76	2.54	3.03	3.22	3.34	3.50	3.47
LevelI	1.00	1.85	2.59	3.29	3.49	3.73	3.88	3.94
LevelI-ODA	1.00	1.79	2.42	2.84	2.98	3.07	3.20	3.18
Benders MC	1.00	1.02	1.04	1.04	1.05	1.05	1.05	1.04

Table B.5. Speedup of the algorithms Benders base case (Benders BC), euclidean level decomposition (LevelE), euclidean level decomposition with on-demand accuracy (LevelE-ODA), single-cut Benders with on-demand accuracy (Benders-ODA), manhattan level decomposition (LevelM), manhattan level decomposition with on-demand accuracy (LevelM-ODA), infinity level decomposition (LevelI), infinity level decomposition with on-demand accuracy (LevelI-ODA), and multi-cut Benders (Benders MC) for different number of threads.

List of Figures

List of Tables

List of Algorithms