VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

STOCHASTIC OPTIMIZATION IN AIMMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAKUB KŮDELA
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

STOCHASTIC OPTIMIZATION IN AIMMS

STOCHASTICKÁ OPTIMALIZACE V PROGRAMU AIMMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAKUB KŮDELA
AUTHOR

VEDOUCÍ PRÁCE RNDr. PAVEL POPELA, Ph.D.
SUPERVISOR

BRNO 2014

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Jakub Kůdela

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Matematické inženýrství (3901T021)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Stochastická optimalizace v programu AIMMS

v anglickém jazyce:

Stochastic optimization in AIMMS

Stručná charakteristika problematiky úkolu:

Optimalizační přístupy hrají důležitou roli v řadě aplikačních oblastí. Navíc je často nutné řešit zahrnutí neurčitosti do modelu. Stochastické programování pak představuje účinný nástroj pro modelování a řešení rozhodovacích problémů v podmínkách neurčitosti. Optimalizační software AIMMS, který není běžně používán na Vysokém učení technickém v Brně nabízí možnou podporu pro řešení stochastických modelů. Student se proto zaměří na problémy podpory implementace vybraných modelů a metod stochastické optimalizace v AIMMS.

Cíle diplomové práce:

Student prostuduje a uvede aktuální teoretické poznatky stochastického programování. Následně vybere vhodné úlohy stochastického programování a zaměří se na formulaci modelů a návrhy metod řešení. Implementace modelů a metod v AIMMS bude podstatnou částí diplomové práce.

ABSTRACT

This master's thesis introduces the basic concepts of mathematical and, most importantly, stochastic programming. Moreover, it gives a description of the usage of the software AIMMS in constructing and solving various optimization problems.
Our main goal is to program several methods for solving these stochastic programming problems in AIMMS and show the usage and usefulness of these methods on chosen problems. One of the problems we chose is an incineration plant model.
All the AIMMS programs, that we describe and use in our text, and their source codes will be enclosed in the appendices.

KEYWORDS

optimization, stochastic programming, AIMMS, scenario-based programs, L-shaped method, incineration

ABSTRAKT

Tato diplomová práce uvádí základní poznatky matematického a především stochastického programování. Navíc se zabývá použitím softwaru AIMMS při vytváření a řešení optimalizačních problémů.
Naším hlavním cílem je naprogramovat v softwaru AIMMS několik metod řešení problémů stochastického programování a ukázat jejich použití a užitečnost na vybraných problémech. Jedním z problémů, který jsme si zvolili, je model spalovny.
Všechny AIMMS programy, které v našem textu použijeme a popíšeme, a jejich zdrojové kódy budou přiloženy v dodatcích.

KLÍČOVÁ SLOVA

optimalizace, stochastické programování, AIMMS, scénářové úlohy, metoda L-shaped, spalovna

KŮDELA, Jakub *Stochastic optimization in AIMMS*: master's thesis. Brno: Brno University of Technology, Faculty of Mechanical Engineering, Institute of Mathematics, 2014. 65 p. Supervised by RNDr. Pavel Popela, PhD.

DECLARATION

I declare that I have written my master's thesis on the theme of "Stochastic optimization in AIMMS" independently, under the guidance of the master's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, as regards the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno . . . .

(author's signature)

ACKNOWLEDGEMENT

I would like to thank my supervisor, RNDr. Pavel Popela, Ph.D., for his valuable advices and helpful comments.
My deepest gratitude goes to my family for their endless support and love.

Brno ....
(author's signature)

CONTENTS

PREFACE

The future is not set. Every day we make decisions whose outcomes are unknown to us and often depend on chance. Deterministic mathematical programming and optimization help us determine what decisions to make, under given circumstances. A downside of these approaches is that they are not well suited for problems that contain uncertainty. This proves to be a major issue since the parameters in optimization models (e.g. prices of certain products, demand, etc.) are in fact random and evolve in time. These are the cases where stochastic optimization takes over.

The aim of this text is to give a description of the basics of stochastic programming and to acquaint the reader with some of the methods used for solving stochastic programmes (for clarification, we will use the word programme in the context of optimization and the word program for software implementations). The implementation of these solution methods will be done in a software called AIMMS, that is designed for optimization modelling. The thesis will be divided into several chapters.

Chapter 1 will be devoted to the introduction of deterministic and stochastic programming. The main objective will be to thoroughly describe the two-stage linear stochastic programme and support this description with a number of solution methods suited for dealing with this particular kind of problem.

Chapter 2 will introduce the software AIMMS. We will cover the creation of optimization programs in AIMMS as well as describe the usage of several important features of AIMMS to show that it is a truly viable software, that is able to successfully deal with stochastic programmes.

Chapter 3 focuses on practical implementation of the knowledge gained from Chapter 2. We will construct a general two-stage linear stochastic program in AIMMS that will allow the user to formulate and solve any problem of this kind with ease. In addition to that we will program the methods for solving two-stage linear stochastic programmes, that we described in Chapter 1, and we will give a comparison of these solution methods. We will also show the usage of this general program on a very well known example.

In Chapter 4 we will encounter a real life stochastic problem. We will use the tools and knowledge we acquired throughout the previous chapters to deal with this problem.

Moreover, we should mention that this thesis contributes to the research activities of Technology Agency of the Czech Republic within the research project No. TE02000236 “Waste-to-Energy (WtE) Competence Centre”.

1 BASIC NOTIONS AND THEORETICAL RESULTS

In this chapter we will describe the basic theoretical ideas of deterministic and stochastic programming, needed for modelling in AIMMS. The deterministic programming will be approached very lightly since its applications in AIMMS were already shown in [8].

1.1 Mathematical Programming

We will start with a brief description of the basic concepts of mathematical programming. For a thorough understanding see [1] or [11].

The goal of mathematical programming is to find an *optimal value* of the *objective function* with respect to given *constraints* (set of inequalities). These constraints form the *feasible set*. The optimal value is either minimum or maximum (depending on a specific problem) of the objective function in the feasible set.

A lot of practical optimization problems, even rather complex ones, are modelled as *linear programmes*. Using the matrix-vector formulation we can write these as follows:

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ \text{s.t. } &\mathbf{A}\mathbf{x} = \mathbf{b} \\ &\mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{1.1}$$

The notation we use follows the conventional notation utilized throughout the field of mathematical programming (see [1], [11] or [12]); we will, thus, omit a thorough description of the equations, which, we fell, are clear from the context. Even though these models have a substantial limitation in the assumption of linearity in the objective function and constraints, they are used in a vast area of applications spanning engineering, transportation, agriculture, etc.

For modelling a closer approximation of the desired real-life problem a more general model must be used

$$\begin{aligned} &\min g_0(\mathbf{x}) \\ \text{s.t. } &g_i(\mathbf{x}) \leq 0, i = 1, \ldots, m \\ &\mathbf{x} \in X \subset \mathbb{R}^n. \end{aligned} \tag{1.2}$$

This form is known as a *mathematical programming* problem. The set $X \subset \mathbb{R}^n$ as well as the real functions $g_i(\mathbf{x}), i = 0, \ldots, m$ are given by the modelling process.

Depending on the properties of the functions g_i and the set X, programme (1.2) is called:

- *linear*, if the set X is convex polyhedral and the functions $g_i(\mathbf{x}), i = 0, \dots, m$ are linear,
- *nonlinear*, if at least one of the functions $g_i(\mathbf{x}), i = 0, \dots, m$ is nonlinear or X is not a convex polyhedral set; among nonlinear programmes, we denote a programme as
  - *convex*, if $X \cap \{\mathbf{x} \mid g_i(\mathbf{x}), i = 1, \dots, m\}$ is a convex set and $g_0(\mathbf{x})$ is a convex function (in particular if the functions $g_i(\mathbf{x}), i = 0, \dots, m$ are convex and X is a convex set), and
  - *nonconvex*, if either $X \cap \{\mathbf{x} \mid g_i(\mathbf{x}), i = 1, \dots, m\}$ is not a convex set or the objective function $g_0(\mathbf{x})$ is not convex.

Another class of problems arises when some of the variables $x_j, j = 1, \dots, n$ can only take integer values. This is called *(mixed) integer programming.*

1.2 Deterministic Programming

Deterministic program is a mathematical programme for which all the parameters and coefficients (in objective function and constraints) are fully known; there is neither uncertainty nor randomness.

A deterministic programme can be expressed in the following form that is further suitable for stochastic programmes:

$$\begin{aligned} &\min g_0(\mathbf{x}, \mathbf{a}) \\ \text{s.t.} \quad &g_i(\mathbf{x}, \mathbf{a}) \leq 0, i = 1, \dots, m \\ &\mathbf{x} \in X \subset \mathbb{R}^n, \end{aligned} \tag{1.3}$$

where $\mathbf{a} \in \mathbb{R}^k$ is a K-dimensional constant vector.

A linear programme (1.1) can be a special case of a deterministic programme if all the coefficients of vectors $\mathbf{b}, \mathbf{c}$ and the matrix $\mathbf{A}$ are fully known.

1.3 Stochastic Programming

As we stated earlier, in the preface, the future is not set. The biggest limitation of deterministic programming is that it requires all the parameters of the model to be fully known. However, the real-world applications can hardly ever fulfil this requirement. To give just some examples we mention: *crop yield* (depending on weather conditions), *demand* on certain product throughout some time period, *prices* of basically anything, changes in legislation (restrictions or liberations of quotas), etc. Using deterministic programming in these situations can return distorted and far-fetched results (see [6]).

Because of all this *uncertainty* a different approach must be adopted. One of the ways to deal with the uncertainty lies in *stochastic programming* where the uncertain parameters are modelled as *random variables* (see [6]).

Let the triplet (Ω, A, P) be a probability space. The mapping $\xi : \Omega \longrightarrow \mathbb{R}$ is called a *random variable* if for all $x \in \mathbb{R}$ holds

$$\{\omega : \xi(\omega) \leq x\} \in A.$$

The general stochastic programme has the following form cf. 1.3:

$$\begin{aligned} &\min g_0(\mathbf{x}, \boldsymbol{\xi}) \\ \text{s.t. } \ &g_i(\mathbf{x}, \boldsymbol{\xi}) \leq 0, i = 1, \dots, m \\ &\mathbf{x} \in X \subset \mathbb{R}^n, \end{aligned} \tag{1.4}$$

where $\boldsymbol{\xi} = (\xi_1, \dots, \xi_K)^T, \boldsymbol{\xi}(\omega) : \Omega \longrightarrow \mathbb{R}^K$ is a finite-dimensional random vector, formed by random variables on the probability space (Ω, A, P).

The feasible set $C(\boldsymbol{\xi})$ of (1.4) can be written in the form:

$$C(\boldsymbol{\xi}) = \{\mathbf{x} \in X \mid g_i(\mathbf{x}, \boldsymbol{\xi}) \leq 0, i = 1, \dots, m\}.$$

Now, a new question arises: How do we solve problems like (1.4)? When a particular realization of random parameters $\boldsymbol{\xi}^p$ is observed and becomes known; one can replace $\boldsymbol{\xi}$ in (1.4) by $\boldsymbol{\xi}^p$, creating a deterministic programme (1.3). However, this does not help in situations when we cannot wait for the particular realization and need to solve the problem now.

This basic partition gives us the two approaches we may take to solve a stochastic programme. The *wait-and-see* approach, being the one that uses the particular realization of $\boldsymbol{\xi}$ and solves a deterministic programme. And the *here-and-now* approach that finds "somehow optimal" solution for all the possible realizations of $\boldsymbol{\xi}$.

1.4 Decisions, Stages and Recourse

Recourse programmes are stochastic programmes in which some decisions or recourse actions can be taken after uncertainty is disclosed (see [6], [12]). In these kind of programmes we can distinguish between two types of decisions:

- A number of decisions have to be taken before the realization of the random vector $\boldsymbol{\xi}$. These are called *first-stage decisions* and the period when these decisions are taken is called the *first stage*.

- A number of decisions have to be taken after the realization of the random vector $\boldsymbol{\xi}$. These are called *second-stage decisions* and the corresponding period is called the *second stage*.

First stage decisions are usually denoted by the vector $\mathbf{x}$, while second-stage decisions are represented by the vector $\mathbf{y}$ or $\mathbf{y}(\omega)$ or even $\mathbf{y}(\omega, \mathbf{x})$, if one wishes to emphasize that second-stage decisions differ as functions of the realization of the random vector and of the first-stage decision. The sequence of events and decisions is thus summarized as

$$\mathbf{x} \longrightarrow \boldsymbol{\xi}(\omega) \longrightarrow \mathbf{y}(\omega, \mathbf{x}).$$

1.5 Two-Stage Programme with Fixed Recourse

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

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} + E_{\boldsymbol{\xi}}[\min \mathbf{q}^T(\omega)\mathbf{y}(\omega)] \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{T}(\omega)\mathbf{x} + \mathbf{W}\mathbf{y}(\omega) = \mathbf{h}(\omega), \text{ a.s.} \\ & \mathbf{x} \geq 0, \mathbf{y}(\omega) \geq 0. \end{aligned} \tag{1.5}$$

As in the previous section, a distinction is made between the first stage and the second stage. The first-stage decisions are represented by the vector $\mathbf{x}$. Corresponding to $\mathbf{x}$ are the first-stage vectors and matrices $\mathbf{c}$, $\mathbf{b}$, and $\mathbf{A}$. In the second stage, a number of random events $\omega \in \Omega$ may be realized. For a given realization ω, the second-stage problem data $\mathbf{q}(\omega)$, $\mathbf{h}(\omega)$ and $\mathbf{T}(\omega)$ become known. Each component of $\mathbf{q}, \mathbf{T}$, and $\mathbf{h}$ is, thus, a possible random variable. $\mathbf{W}$ is called the *recourse matrix*, which is here assumed to be fixed. Piecing together the stochastic components of the second-stage data we obtain a vector $\boldsymbol{\xi}(\omega)$. As indicated before, a single random event ω (or state of the world) influences several random variables, here, all components of $\boldsymbol{\xi}$.

Let, also, $\Xi \subset \mathbb{R}^n$ be the support of $\boldsymbol{\xi}$, i.e. the smallest closed subset in $\mathbb{R}^n$ s.t. $P(\Xi) = 1$. As we just stated, when the random event ω is realized, the second-stage problem data, $\mathbf{q}, \mathbf{T}$ and $\mathbf{h}$ become known. Then the second-stage decision $\mathbf{y}(\omega)$ must be taken. The dependence of $\mathbf{y}$ on ω is of a completely different nature from the dependence of $\mathbf{q}$, or other parameters, on ω. It is not functional but simply indicates that the decisions $\mathbf{y}$ are typically not the same under different realizations of ω. They are chosen so that the constraints of (1.5) hold *almost surely* (denoted a.s.), i.e. for $\forall\omega \in \Omega$ except for sets with zero probability.

The objective function of (1.5) contains a deterministic term $\mathbf{c}^T\mathbf{x}$ and the expectation of the second-stage objective $\mathbf{q}^T(\omega)\mathbf{y}(\omega)$ taken over all realizations of the random event ω. This second-stage term is the more difficult one because, for each ω, the value $\mathbf{y}(\omega)$ is the solution of a linear programme. To stress this fact, one sometimes uses the notion of a deterministic equivalent programme. For a given realization ω, let

$$Q(\mathbf{x}, \boldsymbol{\xi}(\omega)) = \min_y \{\mathbf{q}^T(\omega)\mathbf{y} \mid \mathbf{W}\mathbf{y} = \mathbf{h}(\omega) - \mathbf{T}(\omega)\mathbf{x}, \mathbf{y} \geq 0\} \tag{1.6}$$

be the second-stage value function. Then, we define the expected second-stage value function

$$\mathcal{Q}(\mathbf{x}) = E_{\boldsymbol{\xi}} Q(\mathbf{x}, \boldsymbol{\xi}(\omega)) \tag{1.7}$$

and the *deterministic equivalent programme* (DEP)

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} + \mathcal{Q}(\mathbf{x}) \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq 0. \end{array} \tag{1.8}$$

This representation of a stochastic programme clearly illustrates that the major difference from a deterministic formulation is in the second-stage value function. If that function is given then a stochastic programme is just an ordinary nonlinear programme. Formulation (1.5) is the simplest form of a stochastic two-stage programme. Extensions are easily modelled; for example, if first-stage or second-stage decisions are to be integers. Similarly, nonlinear first-stage and second-stage objectives or constraints can easily be incorporated.

The generalization of the programme (1.8) for a non-linear case may have the following form:

$$\begin{array}{c} \min g_0(\mathbf{x}) + \mathcal{Q}(\mathbf{x}) \\ \text{s.t. } g_i^1(\mathbf{x}) \leq 0, i = 1, \ldots, m_1, \end{array} \tag{1.9}$$

where $\mathcal{Q}(\mathbf{x}) = E_{\boldsymbol{\xi}} Q(\mathbf{x}, \boldsymbol{\xi}(\omega))$ and

$$\begin{array}{c} Q(\mathbf{x}, \boldsymbol{\xi}(\omega) = \min_{\mathbf{y}} q(\mathbf{y}, \boldsymbol{\xi}(\omega) \\ \text{s.t. } t_j(\mathbf{x}, \boldsymbol{\xi}(\omega)) + g_j^2(\mathbf{y}, \boldsymbol{\xi}(\omega)) \leq 0 \text{ a.s. }, j = 1, \ldots, m_2. \end{array} \tag{1.10}$$

By $g_i^1, i = 1, \ldots, m_1$ and $g_j^2, j = 1, \ldots, m_2$ we understand the first and second stage constraints, respectively. A very important aspect of two-stage (and also multi-stage) programmes is the fact that the first-stage decision $\mathbf{x}$ must satisfy so-called *nonanticipativity condition.* The decision $\mathbf{x}$ must be made before the realization of the random vector $\boldsymbol{\xi}$ and, therefore, must be *independent* on it.

1.5.1 Basic Properties

In this section we briefly introduce the basic properties and theory of stochastic programming. All the following results are thoroughly discussed (with examples and proofs) in [6].

Although we set the recourse matrix $\mathbf{W}$ to be fixed, here we study the situation where this matrix can be random. This is because the main issues about definitions of second-stage feasibility sets depend on whether $\mathbf{W}$ is fixed.

For fixed $\mathbf{x}, \boldsymbol{\xi}$, the value $Q(\mathbf{x}, \boldsymbol{\xi})$ of the second-stage programme is given by

$$Q(\mathbf{x}, \boldsymbol{\xi}(\omega)) = \min_{y}\{\mathbf{q}^T(\omega)\mathbf{y} \mid \mathbf{W}(\omega)\mathbf{y} = \mathbf{h}(\omega) - \mathbf{T}(\omega)\mathbf{x}, \mathbf{y} \geq 0\}. \tag{1.11}$$

When the mathematical programme (1.11) is unbounded below or infeasible, the value of the second-stage programme is defined to be $-\infty$ or $+\infty$, respectively.

The expected second-stage value function is, as given in (1.7)

$$\mathcal{Q}(\mathbf{x}) = E_{\boldsymbol{\xi}} Q(\mathbf{x}, \boldsymbol{\xi}(\omega)).$$

Let us first consider the situation when $\boldsymbol{\xi}$ is a finite discrete random variable, namely, $\boldsymbol{\xi} \in \Xi$ with Ξ a finite or countable set. The second-stage value function is then the weighted sum of the $Q(\mathbf{x}, \boldsymbol{\xi})$ values for the various possible realizations of $\boldsymbol{\xi}$. To make the definition complete, we make the additional convention $+\infty + (-\infty) = +\infty$. This corresponds to a conservative attitude, rejecting any first-stage decision that could lead to an undefined recourse action even if there is some realization of the random vector inducing an infinitely low-cost function. Let $K_1 = \{\mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$ be the set determined by the fixed constraints, namely, those that do not depend on the particular realization of the random vector, and let $K_2 = \{\mathbf{x} | \mathcal{Q}(\mathbf{x}) < \infty\}$ be the second-stage feasibility set. We may now redefine the deterministic equivalent programme as follows

$$\begin{aligned} \min z &= \mathbf{c}^T\mathbf{x} + \mathcal{Q}(\mathbf{x}) \\ \text{s.t.} &\quad \mathbf{x} \in K_1 \cap K_2. \end{aligned} \tag{1.12}$$

From a practical point of view, it is not absolutely necessary to have a complete description of the region of finiteness of $\mathcal{Q}(\mathbf{x})$. On the other hand, it is desirable to be able to check if a particular first-stage decision $\mathbf{x}$ leads to a finite second-stage value without having to compute that value. The definition of K_2 is not useful in that respect. Therefore, we consider an alternative definition. Let

$$K_2(\boldsymbol{\xi}) = \{\mathbf{x} | Q(\mathbf{x}, \boldsymbol{\xi}) < +\infty\}$$

be the elementary feasibility sets and

$$\begin{aligned} K_2^P(\boldsymbol{\xi}) &= \{\mathbf{x} | \text{ for all } \boldsymbol{\xi} \in \Xi, \\ &\quad \mathbf{y} \geq 0 \text{ exists s.t. } \mathbf{W}\mathbf{y} = \mathbf{h} - \mathbf{T}\mathbf{x}\} \\ &= \cap_{\boldsymbol{\xi} \in \Xi} K_2(\boldsymbol{\xi}). \end{aligned}$$

The set K_2 is said to define the *possibility interpretation* of second-stage feasibility sets. A decision $\mathbf{x}$ belongs to the set K_2^P if, for all possible values of the random vector $\boldsymbol{\xi}$, a feasible second-stage decision $\mathbf{y}$ can be taken.

Theorem 1.
a. For each $\boldsymbol{\xi}$, the elementary feasibility set is a closed convex polyhedron, hence the set K_2^P is closed and convex.
b. When Ξ is finite, then K_2^P is also polyhedral and coincides with K_2.

Proposition 2. If $\boldsymbol{\xi}$ has finite second moments, then

$$P(\omega | Q(\mathbf{x}, \boldsymbol{\xi}) < \infty) = 1 \text{ implies } \mathcal{Q}(\mathbf{x}) < \infty.$$

Theorem 3. For a stochastic programme with fixed recourse where $\boldsymbol{\xi}$ has finite second moments, the sets K_2^P and K_2 coincide.

Theorem 4. When $\mathbf{W}$ is fixed and $\boldsymbol{\xi}$ has finite second moments:
a. K_2 is closed and convex.
b. If $\mathbf{T}$ is fixed, K_2 is polyhedral.
c. Let Ξ_T be the support of the distribution of $\mathbf{T}$. If $\mathbf{h}(\boldsymbol{\xi})$ and $\mathbf{T}(\boldsymbol{\xi})$ are independent and Ξ_T is polyhedral, then K_2 is polyhedral.

Theorem 5. For a stochastic programme with fixed recourse, $Q(\mathbf{x}, \boldsymbol{\xi})$ is
a. a piecewise linear convex function in $(\mathbf{h}, \mathbf{T})$;
b. a piecewise linear concave function in $\mathbf{q}$;
c. a piecewise linear convex function in $\mathbf{x}$ for all $\mathbf{x}$ in $K = K_1 \cap K_2$.

Theorem 6. For a stochastic programme with fixed recourse where $\boldsymbol{\xi}$ has finite second moments,
a. $\mathcal{Q}(\mathbf{x})$ is a Lipschitzian convex function and is finite on K_2.
b. When $\boldsymbol{\xi}$ is finite, $\mathcal{Q}(\mathbf{x})$ is piecewise linear.
c. If $F(\mathbf{x})$ is an absolutely continuous distribution, $\mathcal{Q}(\mathbf{x})$ is differentiable on K_2.

1.6 Scenario Representations

Let us now look at the expected values that are used in the formulations (1.8) and (1.10). These can be written in the following integral form

$$E_{\boldsymbol{\xi}}(f(\mathbf{x}, \boldsymbol{\xi})) = \int_{\Xi} (f(\mathbf{x}, \boldsymbol{\xi}) \, \mathrm{d}P, \tag{1.13}$$

which brings problems since these integrals are often multidimensional and hard to compute. Because of this fact we will use an approach called *scenario analysis* and create *scenario-based programmes.*

The uncertainty is modelled by *scenarios*, i.e. set of particular realizations $\boldsymbol{\xi}^s$ of the random vector $\boldsymbol{\xi}$. The set of all scenarios is denoted by

$$S = \{s^i, i = 1, \dots, N\},$$

where N is the number of scenarios. We can take all scenarios if the set Ξ is finite and small. However, if the set Ξ is too large, we would have to ask some expert from the desired field to give us a set of most relevant scenarios. We denote p_s the probability of scenario $s \in S : p_s = P(\boldsymbol{\xi} = \boldsymbol{\xi}^s) \geq 0$ and $\sum_{s \in S} p_s = 1$. Therefore, we can rewrite (1.13) as

$$E_{\boldsymbol{\xi}}(f(\mathbf{x}, \boldsymbol{\xi})) = \sum_{s \in S} p_s f(\mathbf{x}, \boldsymbol{\xi}). \tag{1.14}$$

From now on we will use the following notation: $\mathbf{y}_s = \mathbf{y}(\boldsymbol{\xi}^s), \mathbf{q}_s = \mathbf{q}(\boldsymbol{\xi}^s), \mathbf{W}_s = \mathbf{W}(\boldsymbol{\xi}^s), \mathbf{T}_s = \mathbf{T}(\boldsymbol{\xi}^s), \mathbf{h}_s = \mathbf{h}(\boldsymbol{\xi}^s)$.

The scenario-based two-stage stochastic linear programme has now the following form:

$$\begin{array}{rcl} \min z & = & \mathbf{c}^T\mathbf{x} + \sum_{s \in S} p_s Q(\mathbf{x}, \boldsymbol{\xi}^s) \\ \text{s.t.} & & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & & \mathbf{x} \geq 0, \end{array} \tag{1.15}$$

where

$$\begin{array}{rcl} Q(\mathbf{x}, \boldsymbol{\xi}^s) & = & \min_{\mathbf{y}_s} \mathbf{q}_s^T \mathbf{y}_s \\ \text{s.t.} & & \mathbf{T}_s\mathbf{x} + \mathbf{W}_s\mathbf{y}_s = \mathbf{h}_s, \\ & & \mathbf{y}_s \geq 0. \end{array} \tag{1.16}$$

Fusing together (1.15) and (1.16) we get

$$\begin{array}{rcll} \min z & = & \mathbf{c}^T\mathbf{x} + \sum_{s \in S} p_s \mathbf{q}_s^T \mathbf{y}_s & \\ \text{s.t.} & & \mathbf{A}\mathbf{x} = \mathbf{b}, & \\ & & \mathbf{T}_s\mathbf{x} + \mathbf{W}_s\mathbf{y}_s = \mathbf{h}_s, & s = 1, \dots, S, \\ & & \mathbf{x} \geq 0, \mathbf{y}_s \geq 0, & s = 1, \dots, S. \end{array} \tag{1.17}$$

It is easy to see that the size of the programme grows quickly with the number of scenarios. We can rewrite the general two-stage stochastic programme (1.10) in a similar manner.

1.7 Multistage Stochastic Programmes

The previous sections in this chapter were about stochastic programmes with two stages. A lot of real-life decision problems, however, involve a sequence of decisions that react to outcomes that develop over time. These decisions take place in different

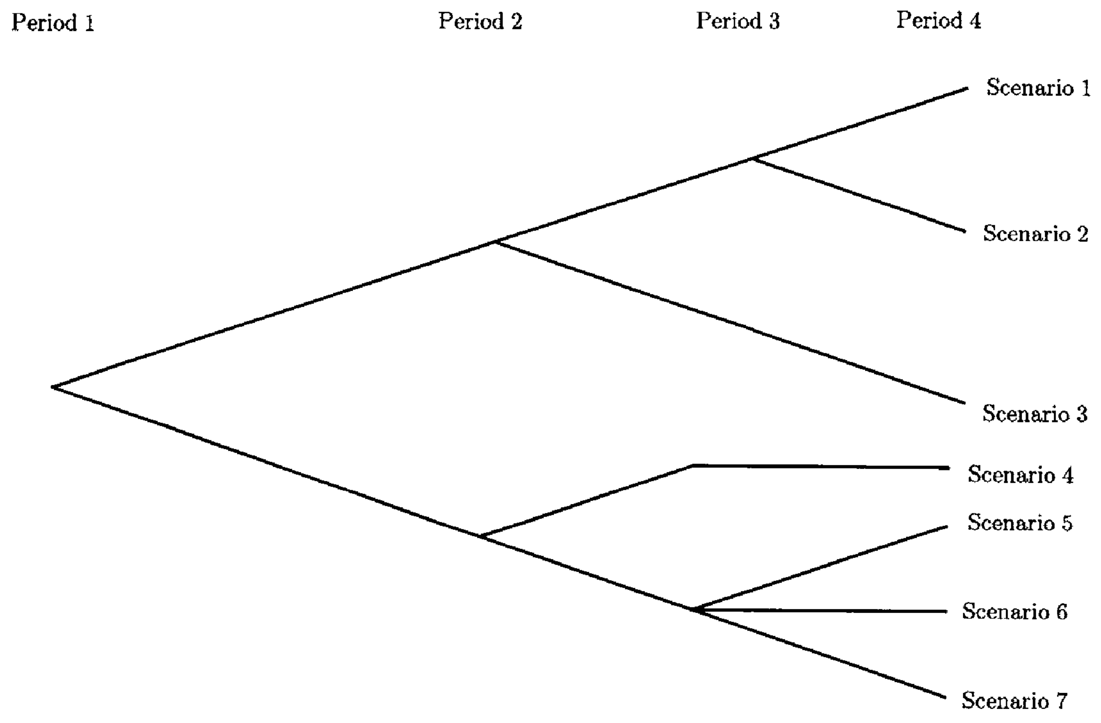


Fig. 1.1: A tree of seven scenarios over four periods.

stages (or periods). We will denote the last stage as H. For further and more detailed information see [6].

The description of scenarios is often made on a tree such as that in Fig. 1.1. Here, there are seven scenarios that are evident in the last stage ($H = 4$). In previous stages ($t \leq 3$), we have a more limited number of possible realizations, which we call the stage t scenarios. Each of these period t scenarios is said to have a single ancestor scenario in stage $(t-1)$ and perhaps several descendant scenarios in stage $(t+1)$. We note that different scenarios at stage t may correspond to the same $\boldsymbol{\xi}$ realizations and are only distinguished by differences in their ancestors.

For a proper description of the multistage stochastic programme, we will use the form that does not use scenarios. The multistage stochastic linear programme with fixed recourse then takes the following form (see [6])

$$\begin{array}{ll} \min & \mathbf{c}^{1^T}\mathbf{x}^1 + E_{\boldsymbol{\xi}^2}[\min \mathbf{c}^{2^T}(\omega)\mathbf{x}^2(\omega)] + \cdots + E_{\boldsymbol{\xi}^H}[\min \mathbf{c}^{H^T}(\omega)\mathbf{x}^H(\omega)] \\ \text{s.t.} & \mathbf{W}^1\mathbf{x}^1 = \mathbf{h}^1, \\ & \mathbf{T}^1(\omega)\mathbf{x}^1 + \mathbf{W}^2\mathbf{x}^2(\omega) = \mathbf{h}^2(\omega), \\ & \vdots \\ & \mathbf{T}^{H-1}(\omega)\mathbf{x}^{H-1} + \mathbf{W}^H\mathbf{x}^H(\omega) = \mathbf{h}^H(\omega), \\ & \mathbf{x}^1 \geq 0, \mathbf{x}^t(\omega) \geq 0, t = 2, \ldots, H. \end{array} \tag{1.18}$$

All the equalities hold a.s., similarly to the model (1.5). The deterministic equivalent

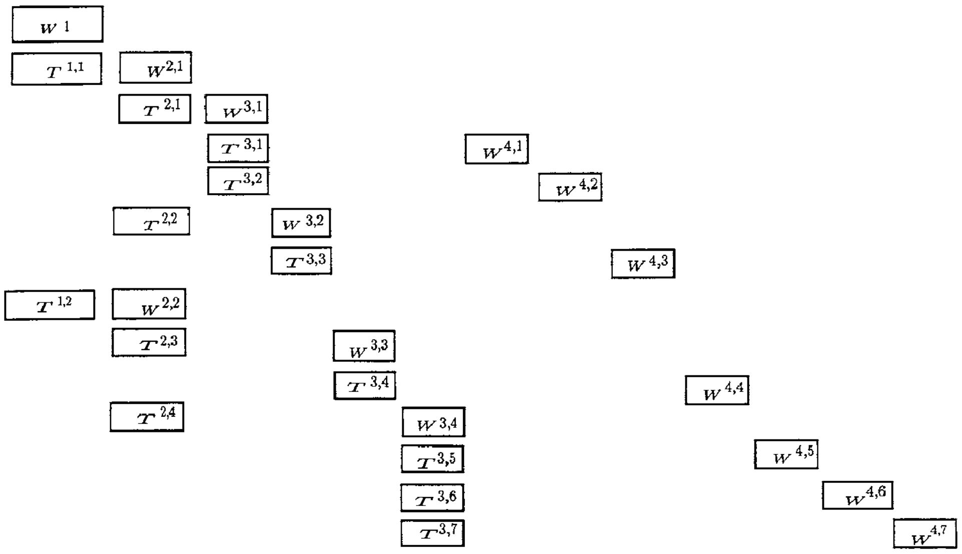


Fig. 1.2: The deterministic equivalent matrix for a problem with seven scenarios in four periods.

programme to (1.18) for the case with a finite number of scenarios is still a linear programme. It has the structural form indicated in Fig. 1.2, where we use an additional superscript to index distinct values of W^t and T^t for different scenarios.

1.8 The L-Shaped Method

In this section we give a brief overview of one of the most commonly used methods for solving large-scale two-stage linear problems. We are going to follow the same notation as presented in [6].

1.8.1 Outer Linearization

Consider the general formulation in (1.5) or (1.8). The basic idea of the L-shaped method is to approximate the nonlinear term in the objective of these problems. A general principle behind this approach is that, because the nonlinear objective term (the recourse function) involves a solution of all second-stage recourse linear programmes, we want to avoid numerous function evaluations for it. Therefore, we use that term to build a master problem in $\mathbf{x}$, but we only evaluate the recourse function exactly as a subproblem.

To make this approach possible, we assume that the random vector $\boldsymbol{\xi}$ has finite support. Let $s = 1, \ldots, N$ index its possible realizations and let p_s be their prob-

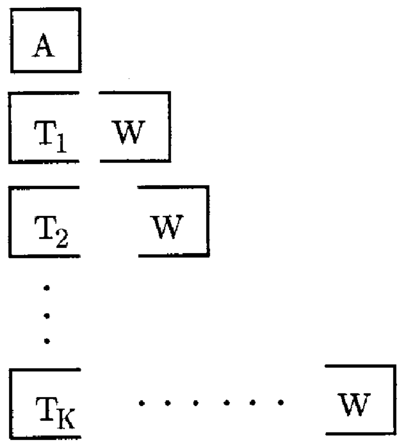


Fig. 1.3: Block structure of the two-stage extensive form.

abilities. Under this assumption, we may now write the deterministic equivalent programme in the extensive form. This form is created by associating one set of the second-stage decisions, say, $\mathbf{y}_s$, to each realization $\boldsymbol{\xi}$, i.e., to each realization of $\mathbf{q}_s, \mathbf{h}_s$, and $\mathbf{T}_s$. It is a large-scale linear problem that we can define as the extensive form (1.17).

The block structure of the extensive form appears in Fig. 1.3. This picture has given rise to the name, *L-shaped method* for the following algorithm:

Step 0: Set $r = q = \nu = 0$.

Step 1: Set $\nu = \nu + 1$. Solve the linear programme

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} + \theta \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{D}_l\mathbf{x} \geq \mathbf{d}_l, \qquad l = 1, \dots, r, \\ & \mathbf{E}_l\mathbf{x} + \theta \geq \mathbf{e}_l, \quad l = 1, \dots, s, \\ & \mathbf{x} \geq 0, \theta \in \mathbb{R}. \end{array} \tag{1.19}$$

Let $(\mathbf{x}^\nu, \theta^\nu)$ be an optimal solution. If $q = 0$, θ^ν is set equal to $-\infty$ and is not considered in the computation of $\mathbf{x}^\nu$.

Step 2: For $s = 1, \dots, N$ solve the linear programme

$$\begin{array}{rcl} \min w' & = & \mathbf{e}^T\mathbf{v}^+ + \mathbf{e}^T\mathbf{v}^- \\ \text{s.t.} & & \mathbf{W}\mathbf{y} + \mathbf{I}\mathbf{v}^+ - \mathbf{I}\mathbf{v}^- = \mathbf{h}_k - \mathbf{T}_k\mathbf{x}^\nu \\ & & \mathbf{y} \geq 0, \mathbf{v}^+ \geq 0, \mathbf{v}^- \geq 0, \end{array} \tag{1.20}$$

where $\mathbf{e}^T = [1, 1, \dots, 1]$, until, for some s, the optimal value $w' > 0$. In this case, let $\boldsymbol{\sigma}^\nu$ be the associated simplex multipliers and define

$$\mathbf{D}_{r+1} = (\boldsymbol{\sigma}^\nu)^T\mathbf{T}_s$$

and

$$d_{r+1} = (\boldsymbol{\sigma}^{\nu})^T \mathbf{h}_s$$

to generate a constraint (called a *feasibility cut*) of type ($\mathbf{D}_l\mathbf{x} \geq \mathbf{d}_l$). Set $r = r+1$, add to the constraint set, and return to Step 1. If for all $s, \mathbf{w}' = 0$, go to Step 3.

Step 3: For $s = 1, \dots, N$ solve the linear programme

$$\begin{array}{rcl} \min w & = & \mathbf{q}_s^T \mathbf{y} \\ \text{s.t.} & & \mathbf{W}\mathbf{y} = \mathbf{h}_k - \mathbf{T}_k \mathbf{x}^{\nu}, \\ & & \mathbf{y} \geq 0. \end{array} \tag{1.21}$$

Let $\boldsymbol{\pi}_s^{\nu}$ be the simplex multipliers associated with the optimal solution of problem s of type (1.21), i.e. optimal solution of the dual problem to (1.16). Define

$$\mathbf{E}_{q+1} = \sum_{s=1}^{N} p_s (\boldsymbol{\pi}_k^{\nu})^T \mathbf{T}_s$$

and

$$e_{q+1} = \sum_{s=1}^{N} p_s (\boldsymbol{\pi}_k^{\nu})^T \mathbf{h}_s.$$

Let $w^{\nu} = e_{q+1} - \mathbf{E}_{q+1}\mathbf{x}^{\nu}$. If $\theta^{\nu} \geq w^{\nu}$, stop; $\mathbf{x}^{\nu}$ is an optimal solution. Otherwise set $q = q+1$, add constraint (called a *optimality cut*) to constraint set ($\mathbf{E}_l\mathbf{x} + \theta \geq \mathbf{e}_l$), and return to Step 1.

The method consists of solving an approximation of (1.8) by using an outer linearization of $\mathcal{Q}$. Two types of constraints are sequentially added: *(i)* feasibility cuts determining $\{x | \mathcal{Q}(x) < \infty\}$ and *(ii)* optimality cuts, which are linear approximations to $\mathcal{Q}$ on its domain of finiteness.

1.8.2 Inner Linearization

The most direct alternative to an outer linearization approach is an inner linearization or column generation approach; this approach is also known as Dantzig-Wolfe decomposition (see [?]). We can derive this approach from the L-shaped method by taking duals.

Consider the following dual linear programme to (1.19).

$$\begin{array}{rcl} \max \zeta & = & \boldsymbol{\rho}^T \mathbf{b} + \sum_{l=1}^{r} \boldsymbol{\sigma}_l d_l + \sum_{l=1}^{q} \boldsymbol{\pi}_l e_l \\ \text{s.t.} & & \boldsymbol{\rho}^T \mathbf{A} + \sum_{l=1}^{r} \boldsymbol{\sigma}_l \mathbf{D}_l + \sum_{l=1}^{q} \boldsymbol{\pi}_l \mathbf{E}_l \leq \mathbf{c}^T \\ & & \sum_{l=1}^{q} \boldsymbol{\pi}_l = 1, \boldsymbol{\sigma}_l \geq 0, l = 1, \dots, r, \boldsymbol{\pi}_l \geq 0, l = 1, \dots, q. \end{array} , \tag{1.22}$$

The linear programme (1.22) includes multipliers $\boldsymbol{\sigma}_l$ on *extreme rays* of the duals of the subproblems. The $\boldsymbol{\pi}_l$ multipliers - the expectations of *extreme points* of the duals of the subproblems. To see this, suppose (1.22) is solved to obtain a multiplier $\mathbf{x}^\nu$. Now consider the following dual to (1.21):

$$\begin{aligned} \max w &= \boldsymbol{\pi}^T(\mathbf{h}_s - \mathbf{T}_s\mathbf{x}^\nu) \\ \text{s.t.} \quad & \boldsymbol{\pi}^T\mathbf{W} \leq \mathbf{q}^T. \end{aligned} \tag{1.23}$$

If (1.23) is unbounded for any s, then we must have some $\boldsymbol{\sigma}^\nu$ s.t. $\boldsymbol{\sigma}^{\nu T}\mathbf{W} \leq 0$ and $\boldsymbol{\sigma}^{\nu T}(\mathbf{h}_s - \mathbf{T}_s\mathbf{x}_s) > 0$ or (1.20) has a feasible dual solution (hence optimal primal solution) with a positive value. So, the second step in the outer linearization is equivalent to checking whether (1.23) is unbounded for any k. In this case we construct $\mathbf{D}_{r+1}$ and d_{r+1} as in the outer linearization and add them to (1.22).

This is the algorithm for the inner linearization:

Step 0: Set $r = q = \nu = 0$.

Step 1: Set $\nu = \nu + 1$ and solve the linear programme (1.22). Let the solution be $(\boldsymbol{\rho}^\nu, \boldsymbol{\sigma}^\nu, \boldsymbol{\pi}^\nu)$ with a dual solution $(\mathbf{x}^\nu, \Theta^\nu)$.

Step 2: For $s = 1, \ldots, N$, solve (1.23). If any infeasibile problem is found, stop and evaluate the formulation. If an unbounded solution with extreme ray $\boldsymbol{\sigma}^\nu$ is found for any s, then form new columns d_{r+1} and $\mathbf{D}_{r+1}$, set $r = r + 1$ and return to Step 1.
If all problems (1.23) are solvable, then form new columns e_{q+1} and $\mathbf{E}_{q+1}$ as in the outer linearization. If $e_{q+1} - \mathbf{E}_{q+1}\mathbf{x}^\nu - \Theta \leq 0$, then stop; $(\boldsymbol{\rho}^\nu, \boldsymbol{\sigma}^\nu, \boldsymbol{\pi}^\nu)$ and $(\mathbf{x}^\nu, \Theta^\nu)$ are the optimal values of the original problem.
If $e_{q+1} - \mathbf{E}_{q+1}\mathbf{x}^\nu - \Theta > 0$, set $q = q + 1$, and return to Step 1.

It is easy to see that the inner linearization method takes the same steps as the outer linearization, except that we solve the duals of the subproblems instead of the primals.

We presented both the inner and outer approximation, because we are going to program the outer linearization algorithm in AIMMS and we felt obliged to acquaint the reader with both of them. Moreover, it is quite clear that the Step 2 in both of those procedures is very well parallelizable (i.e. if we have a multi-core processor, we can command each core to solve one subproblem while at the same time the other core solve the next one and so on, effectively reducing the computing time). We will take advantage of this nice feature once we implement this algorithm into AIMMS.

1.9 Progressive Hedging Algorithm

We will present another solution method for stochastic programming problems, namely the progressive hedging algorithm. Since it is not the purpose of this text, we will not present the proper insight and only give a brief description of the algorithm for two-stage problems. For more information about progressive hedging see [7] and [6]. This algorithm is used for solving problems of the following form:

$$\begin{aligned} &\min E_{\boldsymbol{\xi}}(f(\mathbf{x}, \mathbf{y}(\boldsymbol{\xi}))) \\ \text{s.t. } &g_i(\mathbf{x}, \mathbf{y}(\boldsymbol{\xi})) \leq 0, i = 1, \dots, m. \end{aligned} \tag{1.24}$$

Furthermore, we suppose that we deal with a problem with a finite number of scenarios. The algorithm revolves around the idea, that if $\mathbf{x}$ are the first stage decisions and $\mathbf{y}(\boldsymbol{\xi})$ (or$\mathbf{y}(s)$) are the second stage decisions, at first we suppose different first stage decisions $\mathbf{x}$ for different scenarios s. The condition, that $\mathbf{x}$ must be the same for all scenarios, is enforced by a penalization term.

Step 0: Choose the penalty parameter $\rho > 0$ and the termination parameter $\epsilon > 0$. Set $\mathbf{W}^0(s) = \mathbf{w}_1^0(s) = \mathbf{0}, \hat{\mathbf{X}}^0(s) = (\mathbf{0}, \mathbf{0})$ for all $s \in S$ and $j = 1$.

Step 1: For all $s \in S$ solve the following problem:

$$\begin{aligned} &\min f(\mathbf{x}, \mathbf{y}(s)) + \mathbf{w}_1^{j-1}(s)^T \mathbf{x} + \tfrac{1}{2}\rho(\mathbf{x} - \hat{\mathbf{x}}^{j-1})^2 \\ \text{s.t. } &g_i(\mathbf{x}, \mathbf{y}(s)) \leq 0, i = 1, \dots, m \end{aligned} \tag{1.25}$$

and denote its solution as $\mathbf{X}^j(s) = (\mathbf{x}(s), \mathbf{y}(s))$.

Step 2: For all $s \in S$ calculate an average solution $\hat{\mathbf{X}}^j(s) = (\hat{\mathbf{x}}^j(s), \hat{\mathbf{y}}^j(s))$:

$$\begin{aligned} \hat{\mathbf{x}}^j(s) &= \hat{\mathbf{x}}^j = \textstyle\sum_{s \in S} p_s \mathbf{x}(s), \\ \hat{\mathbf{y}}^j(s) &= \mathbf{y}^j(s). \end{aligned}$$

If the terminal condition

$$\delta = (N \|\hat{\mathbf{x}}^{j-1} - \hat{\mathbf{x}}^j\|^2 + \sum_{s \in S} \|\hat{\mathbf{y}}^{j-1}(s) - \hat{\mathbf{y}}^j(s)\|^2 + \sum_{s \in S} p_s \|\mathbf{x}^j(s) - \hat{\mathbf{x}}^j\|^2)^{\frac{1}{2}} \leq \epsilon$$

holds, then stop. $\hat{\mathbf{X}}^j(s) = (\hat{\mathbf{x}}^j(s), \hat{\mathbf{y}}^j(s))$ is the solution to the original problem with given tolerance. Otherwise set

$$\begin{aligned} &\mathbf{w}_1^j(s) = \mathbf{w}^{j-1}(s) + \rho(\mathbf{x}^j(s) - \hat{\mathbf{x}}^j), \\ &j = j + 1, \end{aligned}$$

and return to Step 1.

1.10 Sample Average Approximation

In this section we introduce a sampling method for solving large scale stochastic programming problems. We concentrate on the "exterior" approach where a random sample is generated outside of an optimization procedure, and then the constructed, so-called sample average approximation (SAA), problem is solved by an appropriate deterministic algorithm. For more detailed description and statistical analysis of this method see [12].

Let us consider a stochastic programming problem in the following form

$$\min_{x\in X}\{f(x) := E_{\boldsymbol{\xi}}[F(\mathbf{x}, \boldsymbol{\xi}(\omega))]\}. \tag{1.26}$$

The expectation in (1.26) is taken with respect to the probability distribution of $\boldsymbol{\xi}$ which assumed to be known. We denote by $\Xi \in \mathbb{R}^d$ the support of the probability distribution of $\boldsymbol{\xi}$, that is, Ξ is the smallest closed set in $\mathbb{R}^d$ such that the probability of the event $\boldsymbol{\xi} \in \mathbb{R}^d \setminus \Xi$ is zero.

Often one can view the optimization problem (1.26) as a two-stage stochastic programming problem with $F(\mathbf{x}, \boldsymbol{\xi}(\omega))$ and $\boldsymbol{\xi}$ being the optimal value and data vector, respectively, of the corresponding second stage programme. For example, in the case of two-stage linear stochastic programming with recourse, $F(\mathbf{x}, \boldsymbol{\xi}(\omega)) := \mathbf{c}^T\mathbf{x} + \mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}(\omega))$, where $\mathcal{Q}(\mathbf{x}, \boldsymbol{\xi}(\omega))$ is the optimal value of the following second stage problem

$$\begin{array}{rl} & \min_{\mathbf{y}} \mathbf{q}^{\mathbf{T}}((\omega))\mathbf{y} \\ s.t. & \mathbf{T}(\omega)\mathbf{x} + \mathbf{W}(\omega)\mathbf{y} = \mathbf{h}(\omega), \text{ a.s.} \\ & \mathbf{y} \geq 0, \end{array} \tag{1.27}$$

with $\boldsymbol{\xi} : (\mathbf{q}, \mathbf{T}, \mathbf{W}, \mathbf{h})$. As such we need to consider situations where $F(\mathbf{x}, \boldsymbol{\xi}(\omega))$ can take values $-\infty$ or $+\infty$. That is, unless stated otherwise, we assume that $F(\mathbf{x}, \boldsymbol{\xi}(\omega))$ is an extended real valued function and the expected value $E_{\boldsymbol{\xi}}[F(\mathbf{x}, \boldsymbol{\xi}(\omega))]$ is *well defined* for every considered $\mathbf{x} \in \mathbb{R}^n$.

If $\boldsymbol{\xi}$ has a finite number of possible realizations (scenarios), say $\Xi = \{\boldsymbol{\xi}^1, \dots, \boldsymbol{\xi}^N\}$ with respective probabilities $p_s, s = 1, \dots, N$, then we can write the expected value function in the form

$$f(\mathbf{x}) = \sum_{s=1}^{N} p_s F(\mathbf{x}, \boldsymbol{\xi}^s). \tag{1.28}$$

Note, however, that even a crude discretization of the probability distribution of $\boldsymbol{\xi}$ leads to an exponential growth of the number of scenarios. For example, if components of the random vector $\boldsymbol{\xi}$ are independent, each having just three possible realizations, then the total number of scenarios $N = 3^d$. No computer in a foreseeable future will be able to handle calculations involving 3^{100} scenarios. Therefore, that way or another, one needs to reduce the number of scenarios to a manageable level. In this section we discuss an approach to solving the expected value problem (1.26), referred to as the *true optimization problem*, by using Monte Carlo sampling technique.

Suppose that we can generate a sample of N replications of the random vector $\boldsymbol{\xi}$. In the Monte Carlo sampling method this is accomplished by generating a random (or rather pseudorandom) sequence $U_1, U_2, \dots,$ of numbers independent of each other and uniformly distributed on the interval $[0, 1]$, and then constructing a sample of $\boldsymbol{\xi}$ by an appropriate transformation. In that way we can consider the sequence $\omega := \{U_1, U_2, \dots\}$ as an element of the probability space equipped with the corresponding (product) probability measure, and the sample $\boldsymbol{\xi}_i = \xi_i(\omega), i = 1, 2, \dots,$ as a function of ω. We can view the generated sample $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots,$ as a sequence of random, vectors, each having the same probability distribution as $\boldsymbol{\xi}$. If the generated random vectors are (stochastically) independent of each other, we say that the sample is independent identically distributed (iid). By $\xi_1, \xi_2, \dots,$ we denote a particular realization of the considered random sample.

With the generated sample $\xi_1, \dots, \xi_N$, we associate the sample average function

$$\hat{f}_N(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^{N} F(\mathbf{x}, \xi_i). \tag{1.29}$$

Since each $\boldsymbol{\xi}_i$ has the same probability distribution as $\boldsymbol{\xi}$, we have that for any $\mathbf{x} \in X, E[F(\mathbf{x}, \xi_i)] = f(\mathbf{x})$ and hence

$$E[\hat{f}_N(\mathbf{x})] = f(\mathbf{x}). \tag{1.30}$$

That is, $\hat{f}_N(\mathbf{x})$ is an *unbiased* estimator of $f(\mathbf{x})$. Moreover, under various conditions the Law of Large Numbers can be applied with the implication that $\hat{f}_N(\mathbf{x})$ converges with probability one to $f(\mathbf{x})$ as $N \longrightarrow \infty$. In that case we say that $\hat{f}_N(\mathbf{x})$ is a *consistent* estimator of $f(\mathbf{x})$. This certainly holds true if the sample is iid.

For the purpose of solving a particular stochastic programming problem, sampling techniques can be applied in different ways. One approach uses sampling in an "interior" fashion. Such algorithms aim at solving the considered problem by resorting to sampling whenever the procedure requires to compute (approximately) the value, and may be derivatives, of the expected value function at a current iteration point. Typically such an algorithm is tailored for a specific class of optimization problems and tries to mimic its deterministic counterpart. Often different samples are used each time the true function or its derivatives are estimated at different iteration points.

We will discuss an alternative approach, referred to as the "exterior" method. First, a sample $\xi_1, \ldots, \xi_N$ is generated, and then the true problem (1.26) is approximated by the optimization problem

$$\min_{\mathbf{x} \in X} \{\hat{f}_N(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} F(\mathbf{x}, \xi_i)\}. \tag{1.31}$$

Note that once the sample is generated, i.e., numerical values of vectors are computed, $\hat{f}_N(\mathbf{x})$ becomes a deterministic function and its value can be calculated at any given point $\mathbf{x} \in X$. From an optimization point of view, problem (1.31) can be considered as a stochastic programming problem with the finite set $\{\xi_1, \ldots, \xi_N\}$ of scenarios each with equal probability $\frac{1}{N}$. Therefore, any numerical algorithm suitable for the considered class of problems can be applied to (1.31). The optimal value $\hat{z}_N$ and an optimal solution $\hat{x}_N$ of the problem (1.31) are considered as statistical estimators of their counterparts of the true problem (1.26).

The above approach is called "exterior" since the sample is generated outside of the considered optimization problem, and then the constructed problem (1.31) is solved by an appropriate deterministic algorithm. It should be noted that this method is not an algorithm, but rather a general approach to solving stochastic programmes. One still needs to employ a particular (hopefully efficient) deterministic algorithm in order to solve the obtained problem (1.31). We refer to (1.31) as the sample average approximation (SAA) problem. The approach is also known as the sample path or the stochastic counterpart method.

We are going to use this approach to generate problems on which we shall test our AIMMS implementation of solution algorithms.

2 AIMMS

In this chapter we will describe the usage of the software we utilized for constructing and solving our optimization problems. The first section will be devoted to the software itself. Other section will deal with its different procedures and features that will be used for constructing our programs.

2.1 About AIMMS

AIMMS is an acronym for "Advanced Interactive Multidimensional Modelling System". It is a software system designed for modelling and solving large-scale optimization problems. It consists of an algebraic modelling language, an integrated development environment, a graphical user interface and a graphical end-user environment. AIMMS is linked to multiple solvers through the AIMMS Open Solver Interface. These solvers are: CPLEX, GUROBI, MOSEK, XA, CP Optimizer, CONOPT, MINOS, SNOPT, LGO, AOA, PATH and CP Optimizer. For more information about AIMMS see [5], [18] or their web page [13].

Formulation of optimization models takes place through declarative language elements such as sets and indices, as well as scalar and multidimensional parameters, variables and constraints, which are common to all algebraic modelling languages, and allow for a thorough description of most problems in mathematical programming.

Procedures and control flow statements are available in AIMMS for

- the exchange of data with external data sources
- data pre- and post-processing tasks around optimization models
- user interface event handling
- the construction of hybrid algorithms for problem types for which no direct efficient solvers are available.

AIMMS supports a wide range of mathematical optimization problem types:

- Linear programming
- Quadratic programming
- Nonlinear programming
- Mixed-integer programming
- Mixed-integer nonlinear programming
- Global optimization
- Complementarity problems (MPECs)
- Stochastic programming
- Robust optimization

- Constraint programming

Uncertainty can be taken into account in deterministic linear and mixed integer optimization models in AIMMS through the specification of additional attributes, such that stochastic or robust optimization techniques can be applied alongside the existing deterministic solution techniques.

Custom hybrid and decomposition algorithms can be constructed using the GMP system library which makes available at the modelling level many of the basic building blocks used internally by the higher level solution methods present in AIMMS, matrix modification methods, as well as specialized steps for customizing solution algorithms for specific problem types.

2.2 Licenses

In order to use the software one must have installed a proper license. There are two types of licenses that AIMMS offers for academic/non-commercial use. For more information about other types of licenses see the company web page [13].

2.2.1 Student License

The AIMMS Student license is limited in the number of identifiers (200) and the size of the optimization models (300x300), allowing students to create and run small AIMMS models on their own computer.

The free student license is equipped with the solvers CPLEX, GUROBI, MOSEK and XA for linear and mixed integer programming, CONOPT and KNITRO for nonlinear programming, AOA for mixed integer nonlinear programming, PATH for mixed complementarity programming, BARON (restricted size 10x10) for global optimization and RO Add-on for Robust Optimization.

This license is not suitable for our purposes but we mention it anyway for the sake of informing the reader about the possibility of obtaining this free license without the need of registration.

2.2.2 Academic License

The Free AIMMS Academic License is an unrestricted license for academic people (students, teachers, professors and researchers).

Because we are going to deal with rather large-scale optimization problems, this is the license we shall use. It contains all the AIMMS-supported solvers and we can introduce as many variables and identifiers as we please.

However, to obtain this license one has to go through a process of registration, which although not being very long may still discourage some people from getting this license.

We strongly recommend to obtain the academic license to everyone who intends to seriously work with AIMMS. You can get the Academic License in here [14].

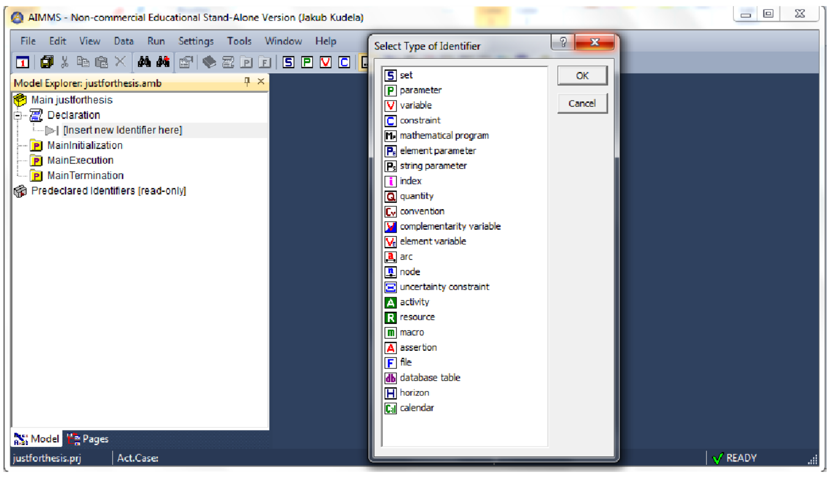
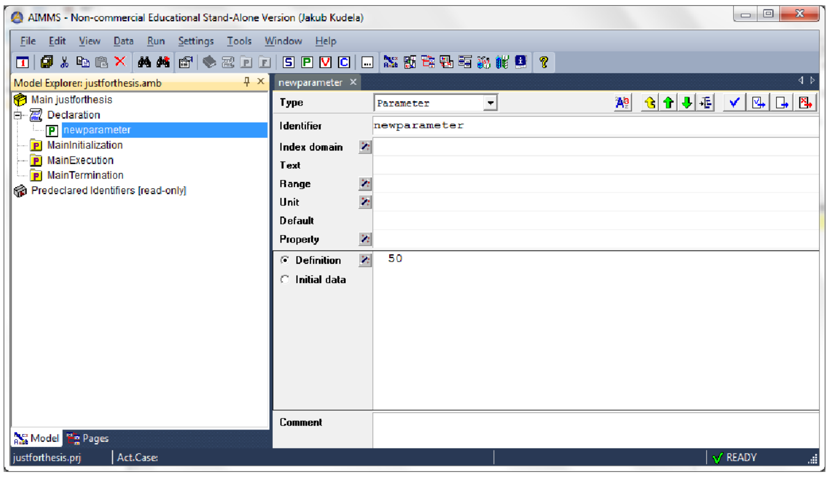


Fig. 2.1: Declaration of identifiers

2.3 Sets, Parameters, Variables and Constraints

In order to successfully create an optimization model, one has to be able to define variables and describe relations between these variables in a form of equalities or inequalities.


AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: justforthesis.amb
Main justforthesis
Declaration
newparameter
MainInitialization
MainExecution
MainTermination
Predeclared Identifiers [read-only]
Model
Pages
justforthesis.prj
Act.Case:
READY

newparameter

Type: Parameter
Identifier: newparameter
Index domain
Text
Range
Unit
Default
Property
Definition: 50
Initial data
Comment

Fig. 2.2: Parameter declaration

In AIMMS the declaration of new variables, sets, parameters, constraints, etc. is done either via the graphical interface, as shown in the figures 2.1 and 2.2, or directly writing the source code (like in GAMS):

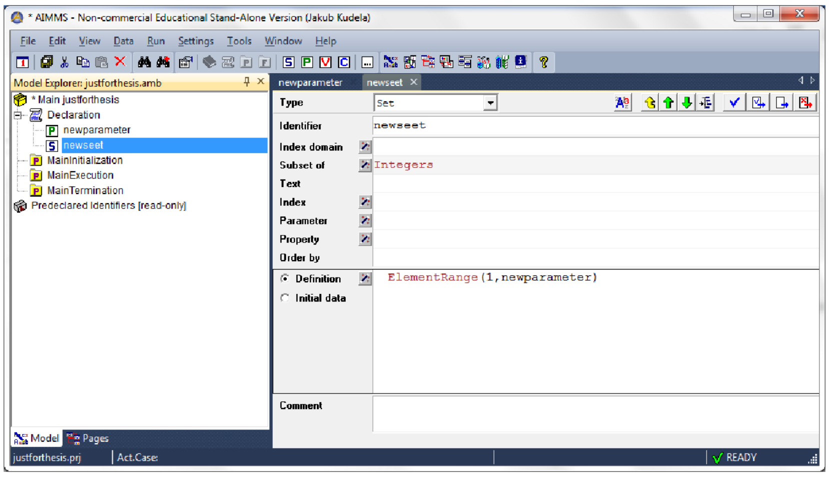
```
DECLARATION SECTION

  PARAMETER:
     identifier :  newparameter
     definition :  50 ;

ENDSECTION  ;
```

These declarations are fairly straight forward and natural, so we will omit a more detailed walk through. However, there are some special features regarding the declaration of sets that will be used throughout our programs, so we decided to highlight them:

- The function *ElementRange* lets us create a set of elements with sequential character, e.g. subsets of integers with fixed distance between each element (2.3).
- Every new set can be handled like a subset of some of the predefined sets. These in addition to others contain Integers, which will get handy when working with flow control statements. But most importantly we can define our set as a subset of *AllVariables* or *AllConstraints*. This allows use to split


* AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: justforthesis.amb
newparameter
newseet
Model
Pages
justforthesis.prj
Act.Case:
READY

* Main justforthesis
- Declaration
  - newparameter
  - newseet
- MainInitialization
- MainExecution
- MainTermination
- Predeclared Identifiers [read-only]

Type	Set
Identifier	newseet
Index domain	
Subset of	Integers
Text	
Index	
Parameter	
Property	
Order by	
[✓ Definition] Initial data	ElementRange(1,newparameter)
Comment	

Fig. 2.3: Set declaration - ElementRange

the AIMMS program into several separate programs, i.e. we can choose the variables and constraints that are going to be used in creating certain mathematical programme (see next section). This will end up being extremely useful in construction of the L-shaped method (section 1.8) and in our effort to compare different techniques for solving stochastic programmes.

For more informations about declaration of different identifiers and set procedures see [4], [2] and [5].

2.4 Mathematical Programmes

Once we have all our variables and constraints fully describing the problem we are about to solve, we can construct a mathematical programme, i.e. we identify the objective function, the direction of optimization and sets of constraints and variables we want to include. Moreover we can inform the solver about the type of our problem as shown in the figure 2.4.

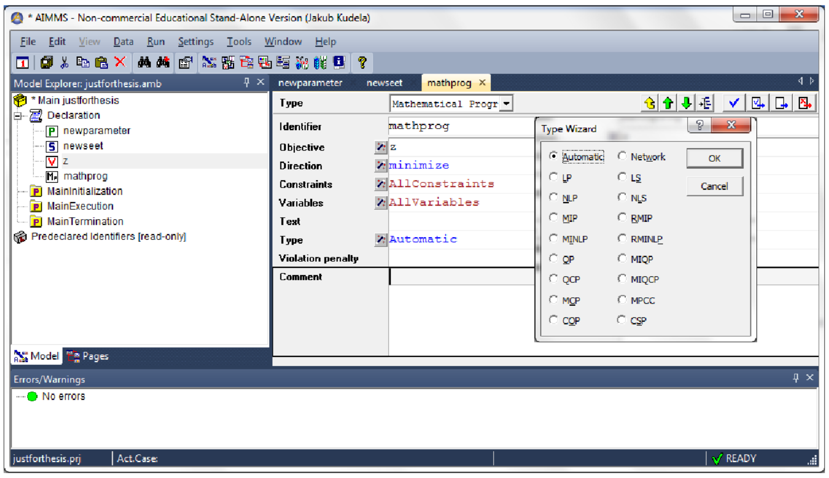
Now, the easiest way of solving this mathematical programme is simply to write a procedure with a *solve* statement and a name of the mathematical programme:

```
PROCEDURE
  identifier :  Procedure_1
  body       :
    solve mathprog;

ENDPROCEDURE  ;
```

The solution of our mathematical programme can be observed in the Math Program Inspector (Tools-Diagnostic Tools-Math Program Inspector).

We will use the *solve* statement quite rarely since it does not offer as much flexibility as working with the GMP library. But we will get to that later on.



Type	Mathematical Progr
Identifier	mathprog
Objective	z
Direction	minimize
Constraints	AllConstraints
Variables	AllVariables
Text	
Type	Automatic
Violation penalty	
Comment	

(• Automatic)

Fig. 2.4: Mathematical programme

2.5 The GMP Library

With every mathematical programme declared as part of our model, the GMP library allows us to associate one or more Generated Math Program instances (GMPs), and with each GMP:

- a conceptual matrix of coefficients that can be manipulated,
- a repository of initial, intermediate or final solutions, and
- a pool of local or remote solver sessions.

There is an extensive amount of procedures in the GMP library. All these procedures are profoundly described in [4] and [2]. These procedures help us manage and adjust our mathematical programmes and solver sessions in such a way, that we are able to program solution algorithms. We are going to pinpoint just those that were crucial in constructing our solution algorithms:

- **GMP::Instance::Generate** generates a mathematical programme instance from a symbolic mathematical programme.
- **GMP::Instance::CreateSolverSession** creates a new solver session for a generated mathematical programme.
- **GMP::Solution::RetrieveFromModel** stores the solution from the model identifiers into the solution repository of a generated mathematical programme.
- **GMP::Solution::SendToSolverSession** initializes a solver session with the values in the solution from the solution repository of a generated mathematical programme.

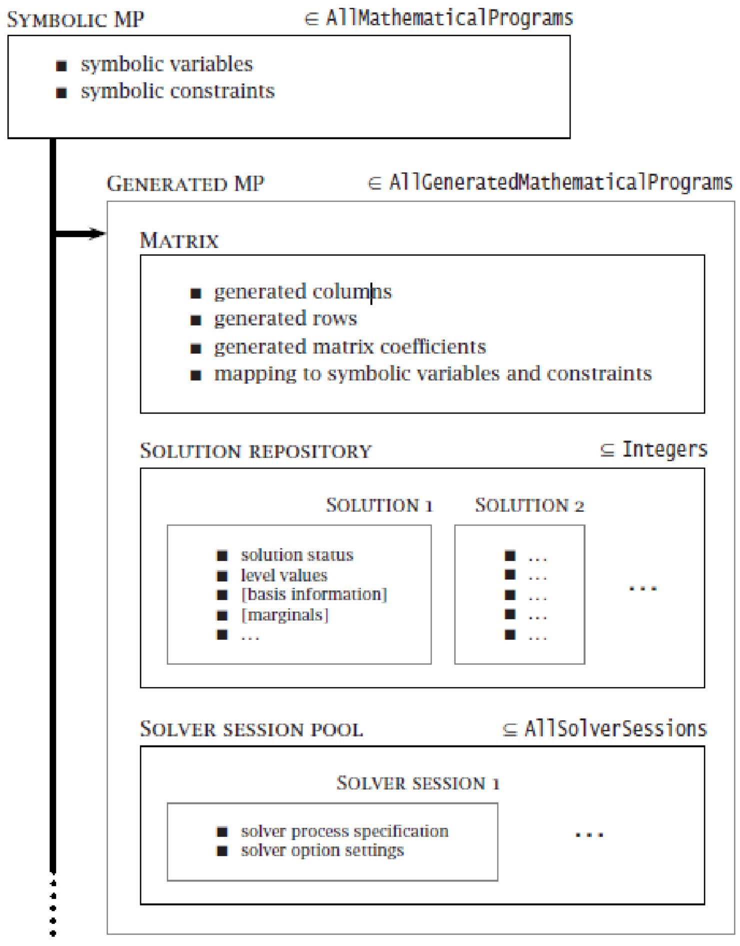


Fig. 2.5: Concepts associated with a GMP

- **GMP::SolverSession::Execute** invokes the solution algorithm to solve the mathematical programme for which it had been generated.
- **GMP::SolverSession::AsynchronousExecute** invokes the solution algorithm to asynchronous solve a generated mathematical programme by using a solver session.
- **GMP::SolverSession::WaitForCompletion** has a set of objects as its input. The set of objects may contain solver sessions that are asynchronous executing and events. This procedure lets AIMMS wait until all the solver sessions have completed their asynchronous execution and all the events get activated.
- **GMP::Solution::RetrieveFromSolverSession** stores the solution from a solver session into the solution repository of a generated mathematical programme.
- **GMP::Solution::SendToModel** initializes the model identifiers with the

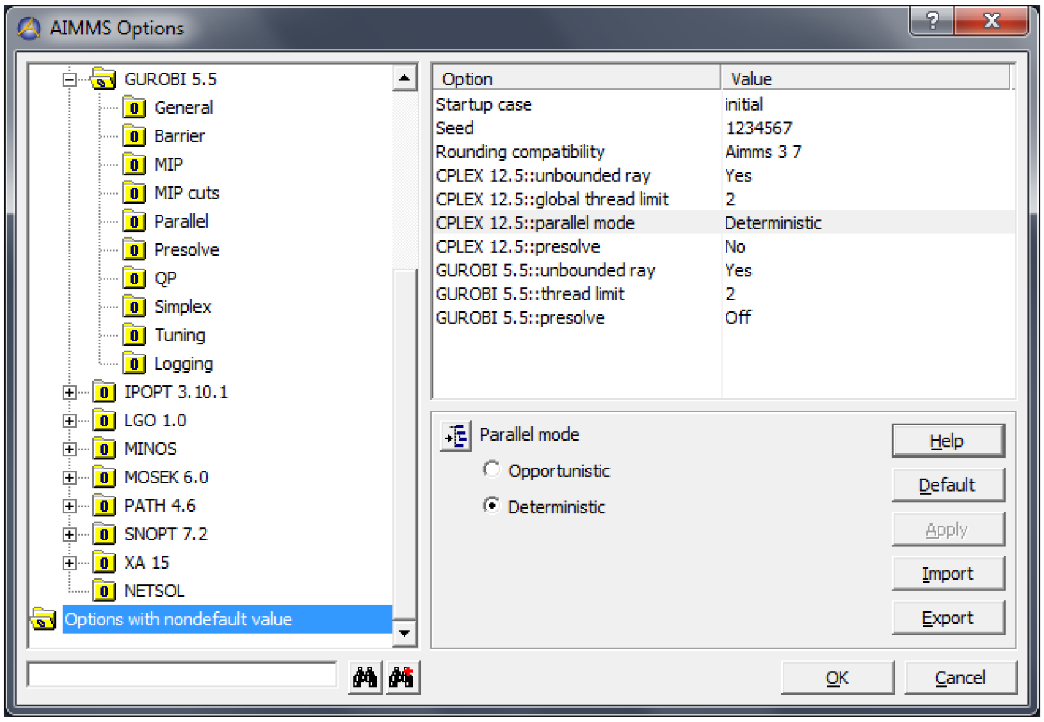
values in the solution from the solution repository of a generated mathematical programme.
- **GMP::SolverSession::GetProgramStatus** returns the program status of the last execution of a solver session.
- **GMP::Instance::DeleteSolverSession** deletes the specified solver session.
- **GMP::Instance::Solve** starts up a solver session to solve a generated mathematical programme. In addition, it copies the initial solution from the model identifiers via solution 1 in the solution repository and stores the final solution via solution 1 back in the model identifiers. This procedure is an equivalent of the *solve* statement, it takes all the necessary steps to solve the mathematical programme (e.g. creating solver sessions), but it does not allow us to choose asynchronous execution.
- **GMP::Row::Add** adds an empty row to the matrix of a generated mathematical programme.
- **GMP::Coefficient::Set** sets the value of a (linear) coefficient in a generated mathematical programme.
- **GMP::Row::SetRightHandSide** changes the right-hand-side of a row in a generated mathematical programme.
- **GMP::Row::SetType** changes the type of a row in the matrix of a generated mathematical programme.

2.6 Particular Solver Settings

In order to successfully implement the L-shaped method we must adjust settings of the solvers we use Fig. 2.6. These adjustments included:
- Disabling presolves and enabling the computation of unbounded rays, which allows us to compute the simplex multipliers in case of an unbounded solution.
- Setting the thread limits to 2, since we worked with a 2 core computer.
- Choosing the deterministic approach in parallel computation. The default value of this setting is opportunistic and leads to different results, but reaches them significantly faster. For more information on this topic see [16].

2.7 Miscellaneous

There are several features of AIMMS that we would like to mention, but we do not feel like writing an entire section about each of these. Therefore, we decided to summarize them in this section. There are, of course, a lot more features and procedures in AIMMS that we do not have the chance to mention. If anyone is



Option	Value
Startup case	initial
Seed	1234567
Rounding compatibility	Aimms 3 7
CPLEX 12.5::unbounded ray	Yes
CPLEX 12.5::global thread limit	2
CPLEX 12.5::parallel mode	Deterministic
CPLEX 12.5::presolve	No
GUROBI 5.5::unbounded ray	Yes
GUROBI 5.5::thread limit	2
GUROBI 5.5::presolve	Off

Parallel mode

Opportunistic [✓ Deterministic]

Fig. 2.6: Solver settings

interested in deeper undestanding of AIMMS we encourage them to delve into the manuals [2], [3], [4] and [5].

Data cases: This feature allows us to solve different problems with the same model. We can save particular values of the model parameters into separate cases and work with these cases without any need to construct a whole new model.

GAMS compatability: If we already have a model description done in GAMS, we can avoid reformulation the whole thing to AIMMS simply by importing the GAMS source code into AIMMS. This is done by selecting the text file with the GAMS source code in **File-Open-Model** and running the compiler.

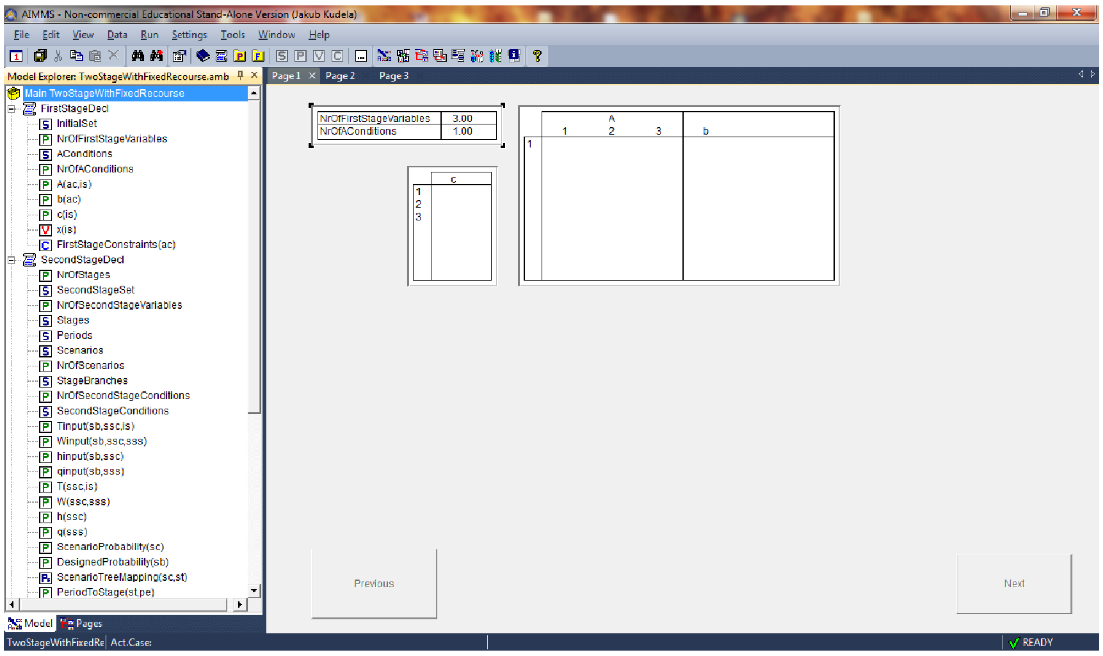
Data import from Excel: Another nice feature is that we can avoid filling the values of parameters in AIMMS altogether by importing them from an Excel file (or other supported database file).

Embedded stochastic optimization procedures: There are procedures already contained in AIMMS that deal directly with stochastic programming. These use the GMP library and as we the Scenario Generation Module to create and solve stochastic programmes. However, as we will discover later on, these procedures do not offer any significant simplification nor do they speed up the solution process.

3 THE GENERAL TWO-STAGE LINEAR PROGRAM

In this chapter, we are going to exploit the possibilities enabled by the AIMMS' programming language and GUI. We will use the properties of the classic formulation of the two-stage stochastic linear programme (1.5) to create a general AIMMS program (from now on we will call it the General Program) for solving these kind of problems.

The purpose of the General Program is to design an easy and end-user friendly way of filling all the necessary parameters of one's model (taking in account the fact that the modelled problem itself must have a two-stage linear structure).



NrOfFirstStageVariables	3.00
NrOfAConditions	1.00

Fig. 3.1: Page 1

The end-user should be able to work with this program without any (or close to none) knowledge of the programming language of AIMMS. Although naturally some understanding of stochastic programming is still required.

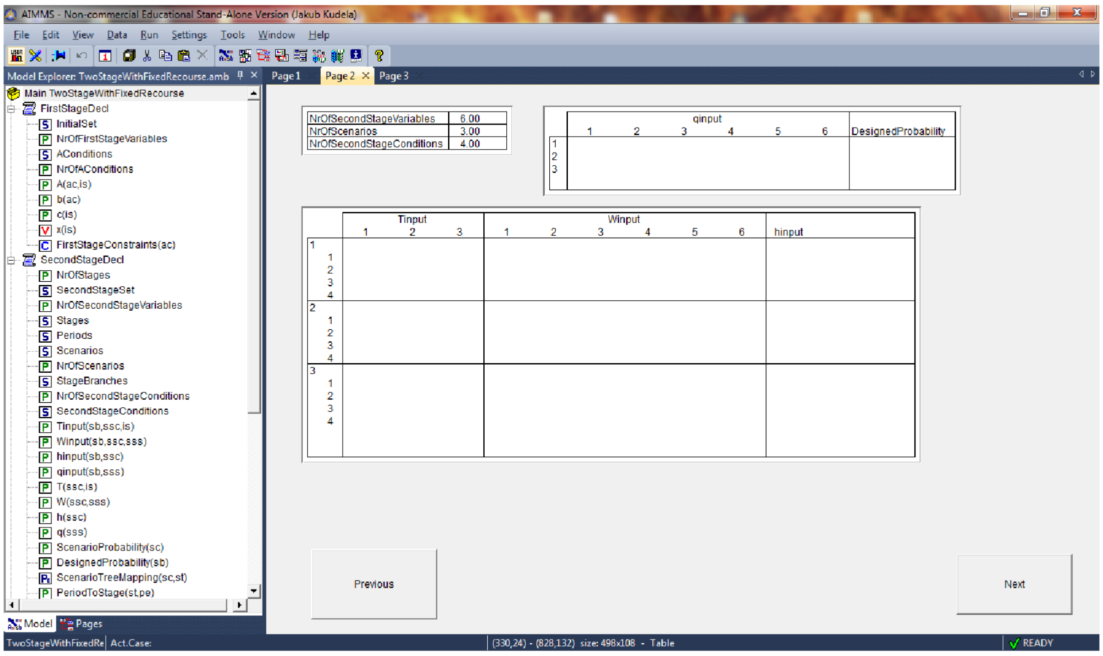
Let us once more look at the equation describing the scenario representation of the two-stage linear stochastic programme with fixed recourse:

$$\begin{aligned} \min z &= \mathbf{c}^T\mathbf{x} + \textstyle\sum_{s\in S} p_s \mathbf{q}_s^T \mathbf{y}_s \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{T}_s\mathbf{x} + \mathbf{W}_s\mathbf{y}_s = \mathbf{h}_s, \quad s = 1,\dots,S, \\ & \mathbf{x} \geq 0, \mathbf{y}_s \geq 0, \quad\quad s = 1,\dots,S. \end{aligned} \tag{3.1}$$

The most important factors in making a general program are the dimensions of the vectors $(\mathbf{c}, \mathbf{x}, \mathbf{q}, \mathbf{y}_s, \mathbf{b}, \mathbf{h}_s)$, the sizes of the matrices $(\mathbf{A}, \mathbf{T}_s, \mathbf{W}_s)$ and number of scenarios (S). These factors and their relations can be described in the following way:

- *Number of scenarios*: S.
- *Number of first stage variables:* $\dim \mathbf{x} = \dim \mathbf{c} =$ number of columns of $\mathbf{A} =$ number of columns of $\mathbf{T}_s, s = 1, \ldots, S$.
- *Number of first stage conditions:* $\dim \mathbf{b} =$ number of rows of $\mathbf{A}$.
- *Number of second stage variables:* $\dim \mathbf{y}_s = \dim \mathbf{q}_s =$ number of columns of $\mathbf{W}_s, s = 1, \ldots, S$.
- *Number of second stage conditions:* $\dim \mathbf{h}_s =$ number of rows of $\mathbf{T}_s =$ number of rows of $\mathbf{W}_s, s = 1, \ldots, S$.

We will neglect the dependence of these numbers on different scenarios. Instead of taking values for all the scenarios, we only consider their maximums. If, for some scenario, is this maximum bigger than the actual value, we just put the extra parameters in the respective vectors and matrices to 0. This will help us to simplify the model a bit; we exchange the need for 2 more numbers for each scenario for a need to fill zeroes in certain places.



NrOfSecondStageVariables	6.00
NrOfScenarios	3.00
NrOfSecondStageConditions	4.00

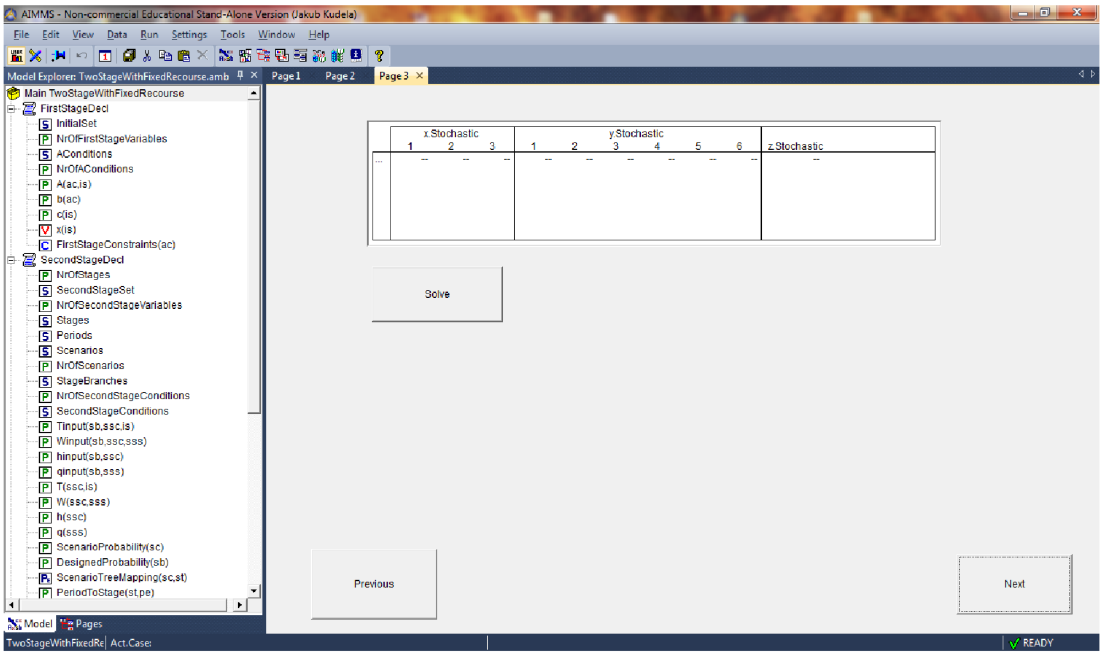
	qinput						
	1	2	3	4	5	6	DesignedProbability
1							
2							
3							

		Tinput			Winput						
		1	2	3	1	2	3	4	5	6	hinput
1	1										
	2										
	3										
	4										
2	1										
	2										
	3										
	4										
3	1										
	2										
	3										
	4										

Fig. 3.2: Page 2

Another slight adjustment of the original programme (3.1) is that the equalities in the conditions will be replaced by inequalities - namely by less that equal. The reason for this is that in order to use equalities we would need to introduce the concept of slack variables (see [11]). This replacement is enabled due to the fact that every equality can be written as 2 inequalities, that have the same coefficients but differ in signs $\leq$ and $\geq$. Moreover every inequality can change its sign from one to another just by multiplying all the coefficients by -1.

These two procedures allow us to design a structure of any two-stage linear stochastic programme given just by the 5 values described above. Since everything is written as general as possible, and we took care of the inconveniences hidden in the equalities and scenarios, we are able to write the AIMMS mathematical programme with ease.



	x.Stochastic			y.Stochastic						
	1	2	3	1	2	3	4	5	6	z.Stochastic
...	--	--	--	--	--	--	--	--	--	--

Fig. 3.3: Page 3

After this it is a simple matter of filling up all the vectors and matrices with desired coefficients. We created a graphical environment, in which the end-user can simply fill and adjust all parameters of the model and potentially solve it just by a click on a button. This environment is shown in Fig. 3.1, 3.2 and 3.3.

Notice the parameter DesignedProbability in Fig. 3.2. Here, the user inserts the probabilities of each scenario. These do not have to add up to 1, since AIMMS will automatically rescale them in a way that they do add up to 1. This allows us to avoid writing numbers like $\frac{1}{7}$ if we have 7 scenarios, all with the same probability, and instead write just 1 everywhere.

The solution process used for solving this problem uses embedded procedures in AIMMS that are designed to deal with stochastic programmes, that are either linear or mix-integer linear programmes. These are closely described in [4].

In the following section, we will show the usage of this General Program on a very well known example.

3.1 The Farmer Example

We will use the notorious farmer example (see [6]) to describe the functionality of the General Program more closely. Moreover, we will expand our General Program by adding procedures that address generating random numbers (depending on given distribution), which is going to help us solve the kind of problems, that instead of giving us set of possible scenarios give us just information about the distribution of some parameters. We shall solve these problems via sample average approximation (1.10).

3.1.1 Problem Formulation

Consider a farmer, who specializes in raising three crops: wheat, corn, and sugar beets. During the winter, he needs to decide how much land, from his 500 acres field, he should devote to each crop, in order to maximize his profit. In other words, how many acres of land should he devote to grain, corn and sugar beets?

The farmer knows that at least 200 tons (T) of wheat and 240 T of corn are needed to feed his cattle. These amounts can be raised on the farm or bought from a wholesaler, whose prices are naturally high compared to the price, at which is the farmer able to produce his own crops. Any production in excess of the feeding requirement would be sold.

Selling prices are \$170 and \$150 per ton of wheat and corn, respectively. The purchase prices are 40% more than this due to the wholesaler's margin and transportation costs.

The third profitable crop, sugar beet, is sold at \$36/T; however, the government imposes a quota on sugar beet production. Any amount in excess of the quota can be sold only at \$10/T. The farmer's quota for next year is 6000 T.

The uncertainty of this problem lies in the weather conditions, that significantly affect the yields of each crop.

Most crops need rain and moisture at the beginning of the planting period, then a lot of sunshine with occasional rain. Sunshine and dry weather is also important during the harvesting period. Due to the above requirements, the yields depend on the weather during the whole planting period.

We will address this problem by modelling uncertainty in the crop's yields via scenario representation (with respect to given scenarios) and sample average approximation (with respect to given distribution function). Moreover, in order to use the general AIMMS program, we need to reformulate the problem from maximization to minimization one. This is done by simply changing the signs of elements in the vectors **c** and **q** in (3.1).

3.1.2 Scenario Representation Approach

Assume, we asked an expert to give us some possible scenarios for the yields, depending on weather. This expert then gave us these three scenarios: mean yields for the ordinary weather (scenario s^1), profitable yields when the weather is favourable (scenario s^2), and lower yields when the weather is unfavourable (scenario s^3). The probabilities of all the scenarios are equal ($p_1 = p_2 = p_3 = \frac{1}{3}$). All data and parameters are given in the following table:

Parameter	Wheat	Corn	Sugar beet
Profitable yield [T/ac]	3	3.6	24
Mean yield [T/ac]	2.5	3	20
Lower yield [T/ac]	2	2.4	16
Planting cost [%/ac]	150	230	260
Selling price [%/T]	170	150	36 under 6000 T 10 over 6000 T
Purchase price [%/T]	238	210	not important
Requirement for feeding [T]	200	240	0

It can be observed, that this model has the two-stage linear structure. This means that there are two decision moments, when the farmer has to decide.

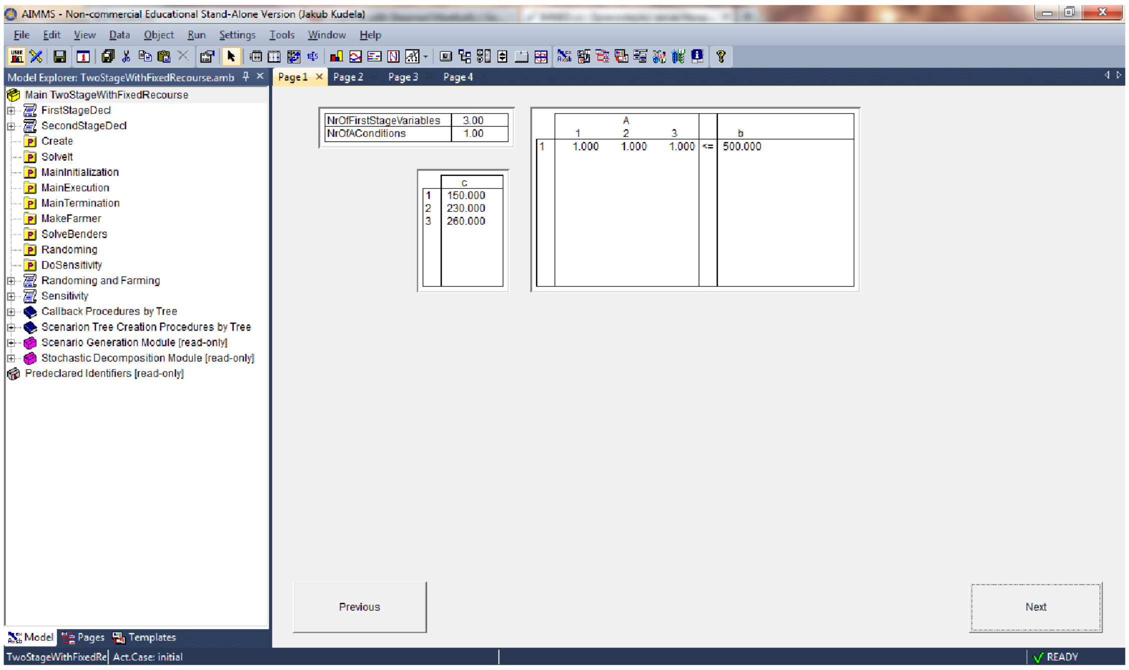
The first one being in winter, when he has to determine how to parcel his land for each crop for the next year. This decision must be taken with no information about future weather (apart from the three possible scenarios). We call this the first stage decision.

The second decision moment comes in place after the realization of the random variable (the weather condition), after the harvest. Now, the farmer has to decide what amount of crops he should sell or buy to fit the feeding requirement and make maximum profit.

We will use the following notation for the model variables:

- x_1: acres devoted to wheat,
- x_2: acres devoted to corn,

- x_3: acres devoted to sugar beet,
- y_1^s: tons of wheat purchased, $s = 1, \ldots, 3$,
- y_2^s: tons of wheat sold, $s = 1, \ldots, 3$,
- y_3^s: tons of corn purchased, $s = 1, \ldots, 3$,
- y_4^s: tons of corn sold, $s = 1, \ldots, 3$,
- y_5^s: tons of sugar beet sold under quota, $s = 1, \ldots, 3$,
- y_6^s: tons of sugar beet sold over quota, $s = 1, \ldots, 3$,
- z: expected profit.



Main TwoStageWithFixedRecourse
- FirstStageDecl
- SecondStageDecl
- Create
- SolveIt
- MainInitialization
- MainExecution
- MainTermination
- MakeFarmer
- SolveBenders
- Randoming
- DoSensitivity
- Randoming and Farming
- Sensitivity
- Callback Procedures by Tree
- Scenarion Tree Creation Procedures by Tree
- Scenario Generation Module [read-only]
- Stochastic Decomposition Module [read-only]

Predeclared Identifiers [read-only]

NrOfFirstStageVariables	3.00
NrOfAConditions	1.00

	c
1	150.000
2	230.000
3	260.000

	A				
	1	2	3		b
1	1.000	1.000	1.000	<=	500.000

Fig. 3.4: First stage part, land requirements

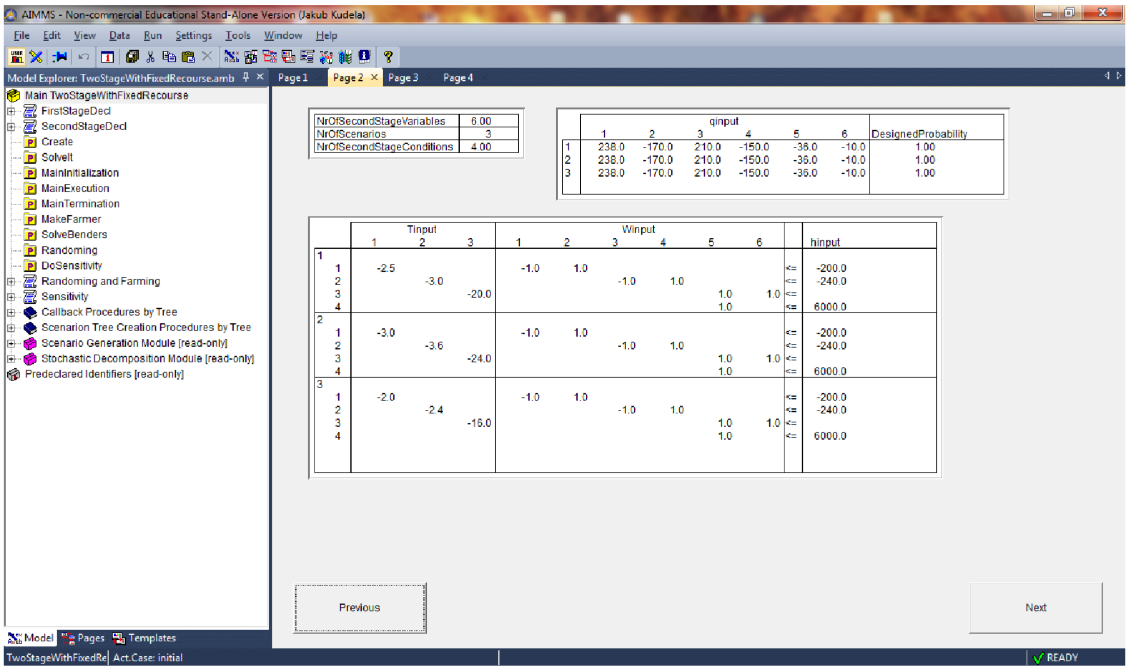
And following notation for the parameters (and their values):

- c_1: planting cost of wheat (= 150),
- c_2: planting cost of corn (= 230),
- c_3: planting cost of sugar beat (= 260),
- q_1: purchasing price of wheat (= 238),
- q_2: selling price of wheat (= -170),
- q_3: purchasing price of corn (= 210),
- q_4: selling price of corn (= -150),
- q_5: selling price of sugar beet sold under quota (= -36),
- q_6: selling price of sugar beet over under quota (= -10),
- t_1^s: yield of wheat, $s = 1, \ldots, 3$ (= {2.5, 3, 2 }),
- t_2^s: yield of corn, $s = 1, \ldots, 3$ (= {3, 3.6, 2.4 }),
- t_3^s: yield of sugar beet, $s = 1, \ldots, 3$ (= {20, 24, 16 }),

- p_s: probability of scenario s $(= \frac{1}{3})$.

Now we can write down the equations describing our model:

$$
\begin{array}{rcll}
\min z & = & \mathbf{c}^T\mathbf{x} + \sum_{s\in S} p_s \mathbf{q}_s^T \mathbf{y}^s & \\
\text{s.t.} & & \sum_{i=1}^{3} x_i \leq 500, & \\
& & t_1^s x_1 + y_1^s - y_2^s \geq 200, & s = 1,\dots,3, \\
& & t_2^s x_2 + y_3^s - y_4^s \geq 240, & s = 1,\dots,3, \\
& & t_3^s x_3 - y_5^s - y_6^s \geq 0, & s = 1,\dots,3, \\
& & y_5^s \leq 6000, & s = 1,\dots,3, \\
& & \mathbf{x} \geq 0, \mathbf{y}_s \geq 0, & s = 1,\dots,3.
\end{array}
\tag{3.2}
$$


AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: TwoStageWithFixedRecourse.amb
Page 1
Page 2
Page 3
Page 4
Main TwoStageWithFixedRecourse
FirstStageDecl
SecondStageDecl
Create
SolveIt
MainInitialization
MainExecution
MainTermination
MakeFarmer
SolveBenders
Randoming
DoSensitivity
Randoming and Farming
Sensitivity
Callback Procedures by Tree
Scenarion Tree Creation Procedures by Tree
Scenario Generation Module [read-only]
Stochastic Decomposition Module [read-only]
Predeclared Identifiers [read-only]
Model
Pages
Templates
Previous
Next
TwoStageWithFixedRe
Act.Case: initial
READY

NrOfSecondStageVariables	6.00
NrOfScenarios	3
NrOfSecondStageConditions	4.00

	qinput						
	1	2	3	4	5	6	DesignedProbability
1	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00
2	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00
3	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00

		Tinput			Winput							
		1	2	3	1	2	3	4	5	6		hinput
1												
	1	-2.5			-1.0	1.0					<=	-200.0
	2		-3.0				-1.0	1.0			<=	-240.0
	3			-20.0					1.0	1.0	<=	
	4								1.0		<=	6000.0
2												
	1	-3.0			-1.0	1.0					<=	-200.0
	2		-3.6				-1.0	1.0			<=	-240.0
	3			-24.0					1.0	1.0	<=	
	4								1.0		<=	6000.0
3												
	1	-2.0			-1.0	1.0					<=	-200.0
	2		-2.4				-1.0	1.0			<=	-240.0
	3			-16.0					1.0	1.0	<=	
	4								1.0		<=	6000.0

Fig. 3.5: Second stage part, equations guarding production and consumption

The first inequality states that we cannot plant crops on more that 500 acres of land. The second and third ones stand as the feeding requirements. The fourth inequality ensures that we do not sell more sugar beets than we produce and the fifth one shows the impact of the quota on sugar beets. The last one is a safe guard against impossible solutions (i.e. we plant a negative amount of acres with wheat,...).

We are almost ready to rewrite the problem into our AIMMS general model. The last step that remains is to change all the inequalities with $\geq$ to $\leq$:

$$\begin{aligned}
-t_1^s x_1 - y_1^s + y_2^s &\leq -200, \quad && s = 1, \dots, 3,\\
-t_2^s x_2 - y_3^s + y_4^s &\leq -240, \quad && s = 1, \dots, 3,\\
-t_3^s x_3 + y_5^s + y_6^s &\leq 0, \quad && s = 1, \dots, 3.
\end{aligned}$$

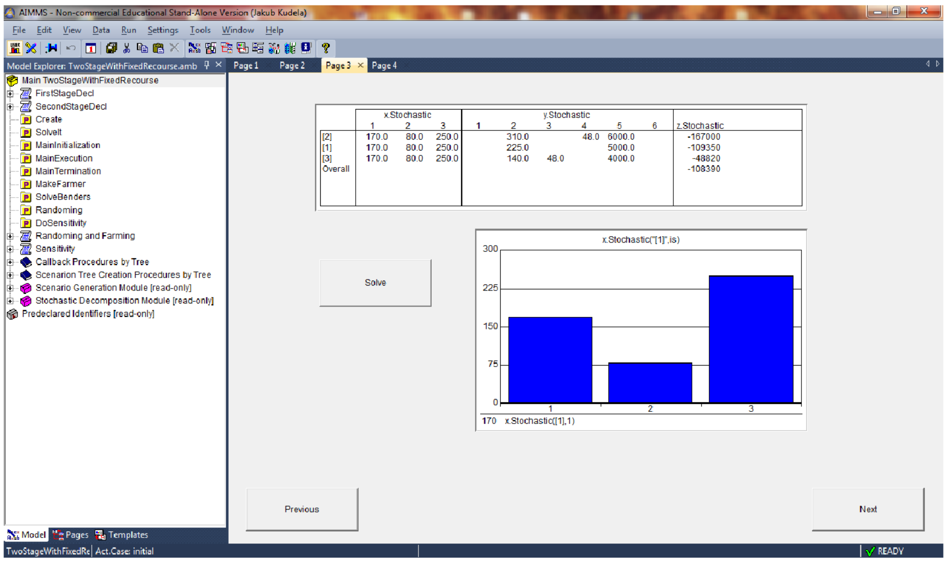
As we stated before in the section about the AIMMS General Program, the most important numbers to create the model are these: number of scenarios, number of first stage variables, number of first stage conditions, number of second stage variables and number of second stage conditions. In our case these take the following values:

- number of scenarios = 3,
- number of first stage variables = 3,
- number of first stage conditions = 1,
- number of second stage variables = 6,
- number of second stage conditions = 4.

We do not need to take into consideration the nonnegativity constraints, since they are already embedded in our AIMMS general model.

Now we can finally proceed to transfer our farmer problem into the AIMMS general model, as shown in Fig. 3.4 and 3.5.

In the Fig. 3.6 we can observe the optimal solution to our farmers problem. This consists of the optimal first stage decision, in the x.Stochastic part of the table (same for all the scenarios), and the optimal strategies in individual scenarios, in the y.Stochastic part. The last part of the table shows the value of the objective function in individual scenarios and the weighted mean of these values, which is the optimal value of the objective function of our farmer's problem. In other words, we advise the farmer to plant 170 acres with wheat, 80 acres with corn and 250 acres with sugar beet. The expected profit is going to be 108390$.



	x.Stochastic			y.Stochastic						z.Stochastic
	1	2	3	1	2	3	4	5	6	
[2]	170.0	80.0	250.0		310.0		48.0	6000.0		-167000
[1]	170.0	80.0	250.0		225.0			5000.0		-109350
[3]	170.0	80.0	250.0		140.0	48.0		4000.0		-48820
Overall										-108390

AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: TwoStageWithFixedRecourse.amb
Page 1
Page 2
Page 3
Page 4
Main TwoStageWithFixedRecourse
FirstStageDecl
SecondStageDecl
Create
SolveIt
MainInitialization
MainExecution
MainTermination
MakeFarmer
SolveBenders
Randoming
DoSensitivity
Randoming and Farming
Sensitivity
Callback Procedures by Tree
Scenarion Tree Creation Procedures by Tree
Scenario Generation Module [read-only]
Stochastic Decomposition Module [read-only]
Predeclared Identifiers [read-only]
Solve
x.Stochastic("[1]",is)
300
225
150
75
0
1
2
3
170 x.Stochastic([1],1)
Previous
Next
Model
Pages
Templates
TwoStageWithFixedRe
Act.Case: initial
READY

Fig. 3.6: Results and optimal strategies

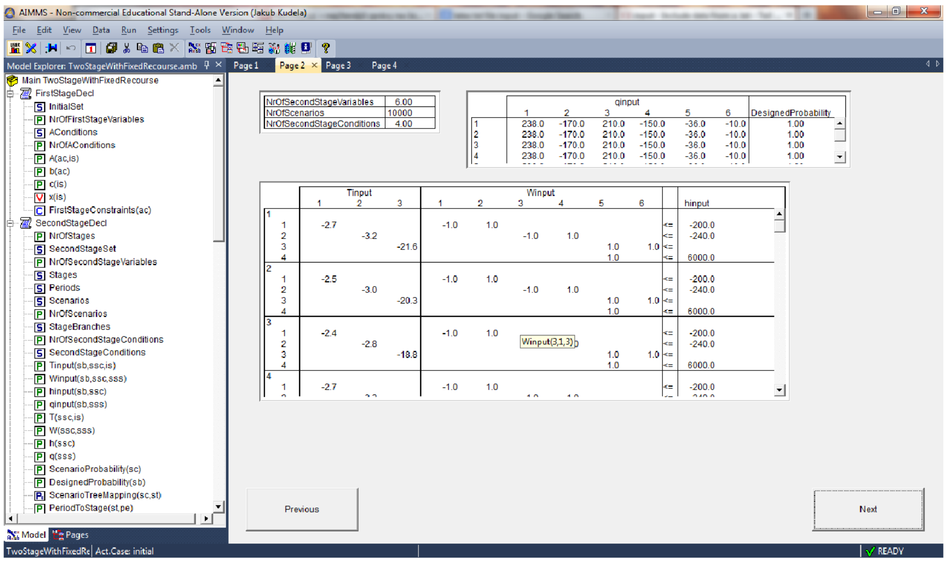
3.1.3 Sample Average Approximation Approach

Imagine, we asked a different expert, than the one from the previous section, and he told us, that the yield is a continuous random variable. To illustrate this, let us assume that the yield can be appropriately described by a uniform random variable, inside some range $[l, u]$. All other parameters of the model remain the same, so we will not mention how to deal with them again.

For the sake of comparison, we may take l to be 80% of the mean yield and u to be 120% of the mean yield (as given in the previous section), so that the expectations for the yields will be the same as in the previous section. This random yield applies to all the crops at once, i.e. in our sample average approximation will be the mean yield for every crop multiplied by a number from $[0.8, 1.2]$ in each generated scenario.

Our goal, now, is to generate a certain number (N) of scenarios that will obey this distribution requirement and incorporate these scenarios into our General Model. Luckily, AIMMS supports a wide variety of functions that generate random numbers, based on given distribution (see [4]). For our purpose serves the *Uniform*(min,max) function.

We use this function to generate a $N \times 1$ vector of random numbers from *Uniform*(0.8,1.2) and, for each scenario, multiply the appropriate values of $\mathbf{T}_s$ (in our case the whole matrix $\mathbf{T}_s$) with a corresponding value from our random vector. Since the rest of the second stage parameters ($\mathbf{W}_s, \mathbf{h}_s$ and $\mathbf{q}_s$) remain for each scenario unchanged, we can proceed in creating a procedure, that will fill up the vectors and matrices of our General Model with appropriate coefficients. This procedure is a bit



```
Main TwoStageWithFixedRecourse
  FirstStageDecl
    InitialSet
    NrOfFirstStageVariables
    AConditions
    NrOfAConditions
    A(ac,is)
    b(ac)
    c(is)
    x(is)
    FirstStageConstraints(ac)
  SecondStageDecl
    NrOfStages
    SecondStageSet
    NrOfSecondStageVariables
    Stages
    Periods
    Scenarios
    NrOfScenarios
    StageBranches
    NrOfSecondStageConditions
    SecondStageConditions
    Tinput(sb,ssc,is)
    Winput(sb,ssc,sss)
    hinput(sb,ssc)
    qinput(sb,sss)
    T(ssc,is)
    W(ssc,sss)
    h(ssc)
    q(sss)
    ScenarioProbability(sc)
    DesignedProbability(sb)
    ScenarioTreeMapping(sc,st)
    PeriodToStage(st,pe)
```

NrOfSecondStageVariables	6.00
NrOfScenarios	10000
NrOfSecondStageConditions	4.00

	qinput						
	1	2	3	4	5	6	DesignedProbability
1	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00
2	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00
3	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00
4	238.0	-170.0	210.0	-150.0	-36.0	-10.0	1.00

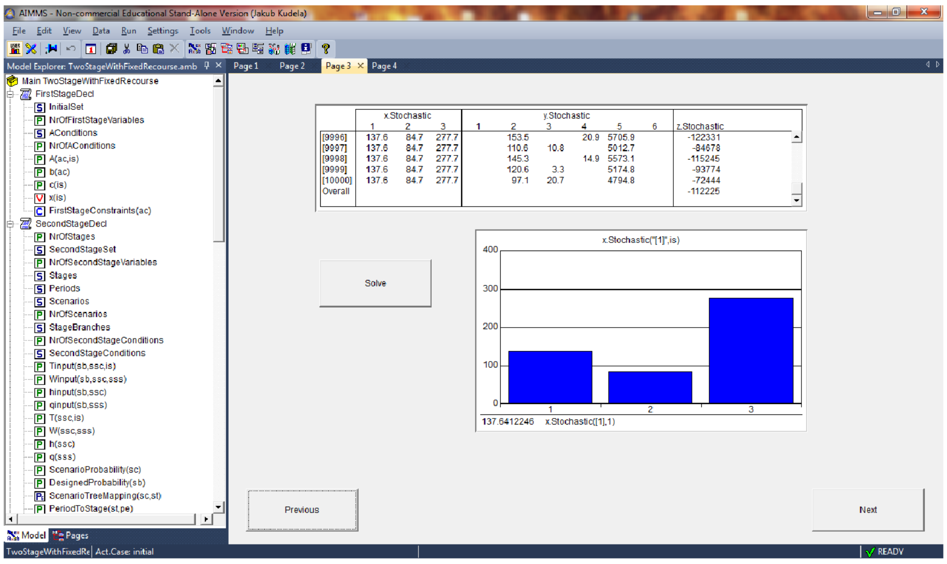
		Tinput			Winput							
		1	2	3	1	2	3	4	5	6		hinput
1	1	-2.7			-1.0	1.0					<=	-200.0
	2		-3.2				-1.0	1.0			<=	-240.0
	3			-21.6					1.0	1.0	<=	
	4								1.0		<=	6000.0
2	1	-2.5			-1.0	1.0					<=	-200.0
	2		-3.0				-1.0	1.0			<=	-240.0
	3			-20.3					1.0	1.0	<=	
	4								1.0		<=	6000.0
3	1	-2.4			-1.0	1.0					<=	-200.0
	2		-2.8				Winput(3,1,3)				<=	-240.0
	3			-18.8					1.0	1.0	<=	
	4								1.0		<=	6000.0
4	1	-2.7			-1.0	1.0					<=	-200.0

AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: TwoStageWithFixedRecourse.amb
Page 1
Page 2
Page 3
Page 4
Previous
Next
Model
Pages
TwoStageWithFixedRe | Act.Case: initial
READY

Fig. 3.7: Scenarios generated by distribution

longer so we will not present it here, however, it can be found in the source code appendix.

In our case we generated $N = 10000$ scenarios (shown in Fig. 3.7) and solved the problem. The results are shown in Fig. 3.8. We can, again, observe the optimal first stage decisions in x.Stochastic: devoting 137.6 acres to wheat, 84.7 acres to corn and 277.7 acres to sugar beet; the recourse actions in y.Stochastic and the expected profit of 112225$.



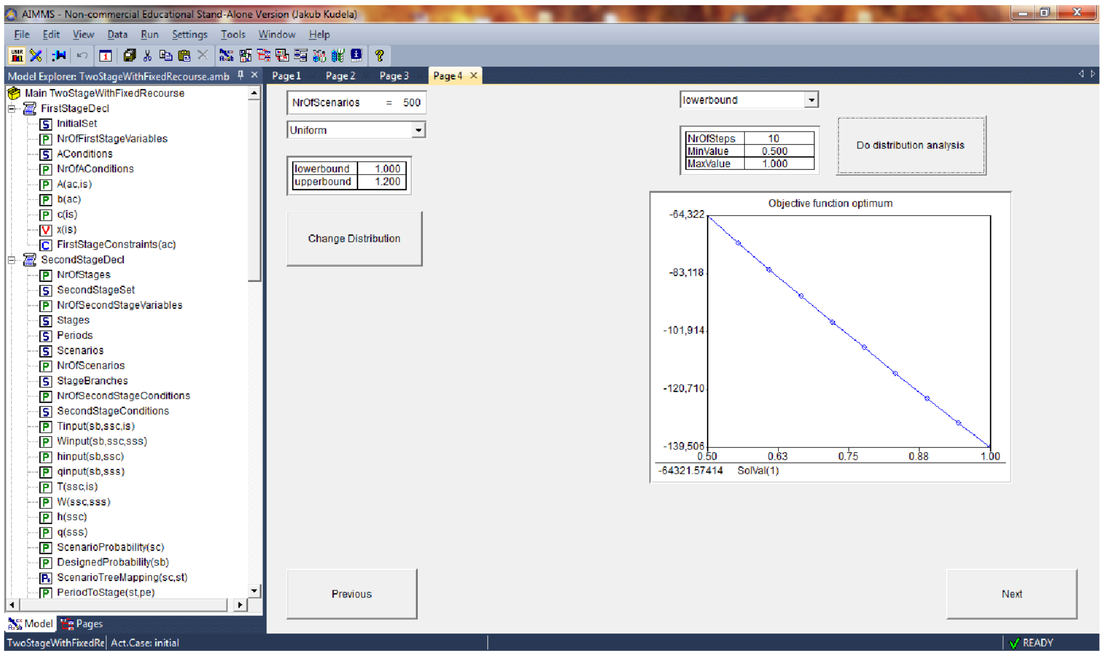
	x.Stochastic			y.Stochastic						
	1	2	3	1	2	3	4	5	6	z.Stochastic
[9996]	137.6	84.7	277.7		153.5		20.9	5705.9		-122331
[9997]	137.6	84.7	277.7		110.6	10.8		5012.7		-84678
[9998]	137.6	84.7	277.7		145.3		14.9	5573.1		-115245
[9999]	137.6	84.7	277.7		120.6	3.3		5174.8		-93774
[10000]	137.6	84.7	277.7		97.1	20.7		4794.8		-72444
Overall										-112225

AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: TwoStageWithFixedRecourse.amb
Page 1
Page 2
Page 3
Page 4
Solve
x.Stochastic("[1]",is)
400
300
200
100
0
1
2
3
137.6412246
x.Stochastic([1],1)
Previous
Next
Model
Pages
TwoStageWithFixedRe | Act.Case: initial
READY

Fig. 3.8: Results of the sample average approximation

3.1.4 Extension: Distribution Analysis

As we created the model using sample average approximation approach, some very natural questions arrived: What if the bounds of the distribution were different? What if the distribution itself is different from the one we used? How does a slight change of the distribution affect the optimal solution?



NrOfScenarios = 500

Uniform

lowerbound	1.000
upperbound	1.200

lowerbound

NrOfSteps	10
MinValue	0.500
MaxValue	1.000

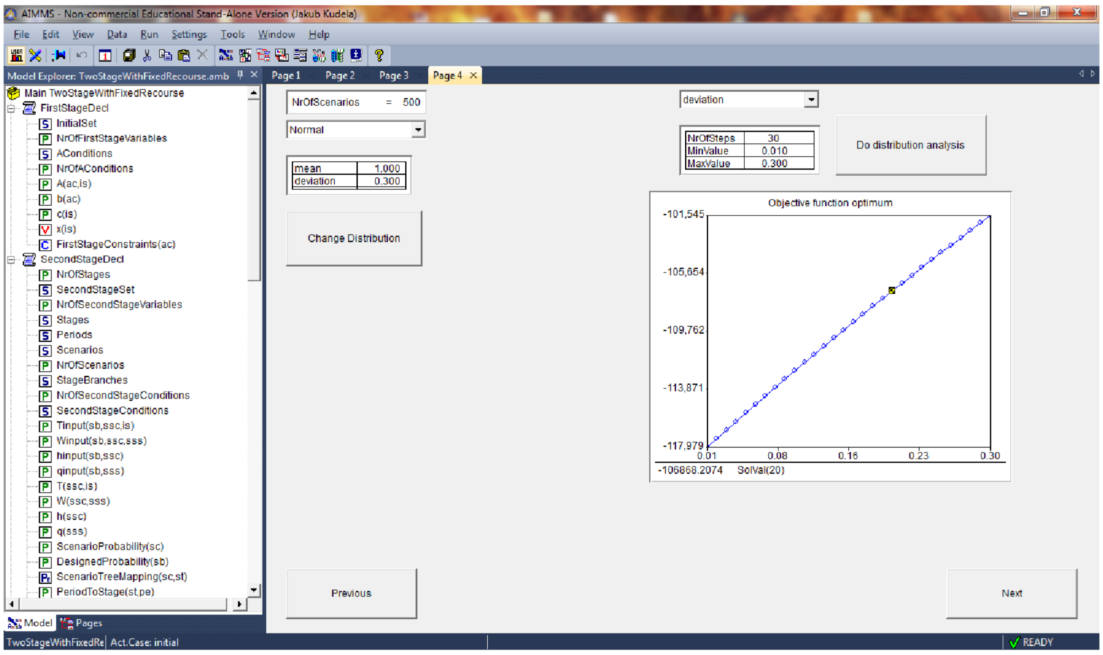
-64321.57414 SolVal(1)

Fig. 3.9: Change in uniform distribution and distribution analysis

All these questions are addressed in the last page of our program (Fig. 3.9 and Fig. 3.10). Here, the end-user is enabled to change the parameters of the uniform distribution (lower and upper bound), choose the number of scenarios and create his own sample average approximation, and see how it changes the results.

Or he can choose normal distribution instead, decide on its parameters (mean and deviation), and see how this change in distribution alters the results. (Just to clarify: the procedure guarding these manipulations creates a vector of random numbers from desired distribution, multiplies the mean yield with numbers from this vector and creates scenarios.)

On the right-hand side of the page, there is the section dealing with the last question about slight changes in parameters of the distribution. The end-user can observe the effects of increasing a distribution parameter on the optimal expected value. The tables and graphs on this page are self-explanatory.



NrOfScenarios = 500

Normal

mean	1.000
deviation	0.300

deviation

NrOfSteps	30
MinValue	0.010
MaxValue	0.300

-106868.2074 SolVal(20)

Fig. 3.10: Change in normal distribution and distribution analysis

3.2 Extension: Solution Methods

As we stated earlier, the solution method used for solving our General Program is the one that is embedded in AIMMS. In addition to this one we decided to program another solution methods. These methods consist of the L-shaped method (the Benders decomposition) - its single and dual-core version, the progressive hedging algorithm (both described in the chapter 1) and the last algorithm, that simply solves the problem in the form 1.17; this last algorithm is basically the simplex method (or its variation used by the solver) and since it does not utilize the particular structure of the two-stage linear programme, we shall denote it as the Naive algorithm. The source codes of these solution method are quite extensive and can be found in the appendix; we decided to briefly describe the crucial parts of the source code for single core Benders decomposition (as described in section 1.8.2), to give the reader at least some insight into its AIMMS implementation.

First of, we generate and solve the master problem:

```
MasterGMP:=GMP::Instance::Generate(master);
...
GMP::Instance::Solve(MasterGMP);
```

Then, we start solving the subproblems, generated by scenarios:

```
SubGMP:=GMP::Instance::Generate(subprog);
```

```
GMP::Instance::Solve(SubGMP);
solstatus:=GMP::Solution::GetProgramStatus(SubGMP, 1);
```

If we detect an unbounded solution, we break the cycle, that solves the subproblems, and compute appropriate coefficients:

```
if solstatus = 'Unbounded' then
zsubsol(sb):=zsub;
usol(sb,ssc):=u(ssc);
FeasF(is):=sum(ssc,u(ssc)*Tsub(ssc,is));
FeasSmallF:=sum(ssc,u(ssc)*hsub(ssc));
break;
endif;
```

If we detect an infeasible solution, we break the procedure altogether and stop:

```
if solstatus = 'Infeasible' then
Errorstatus:="Infeasibility";
break;
endif;
```

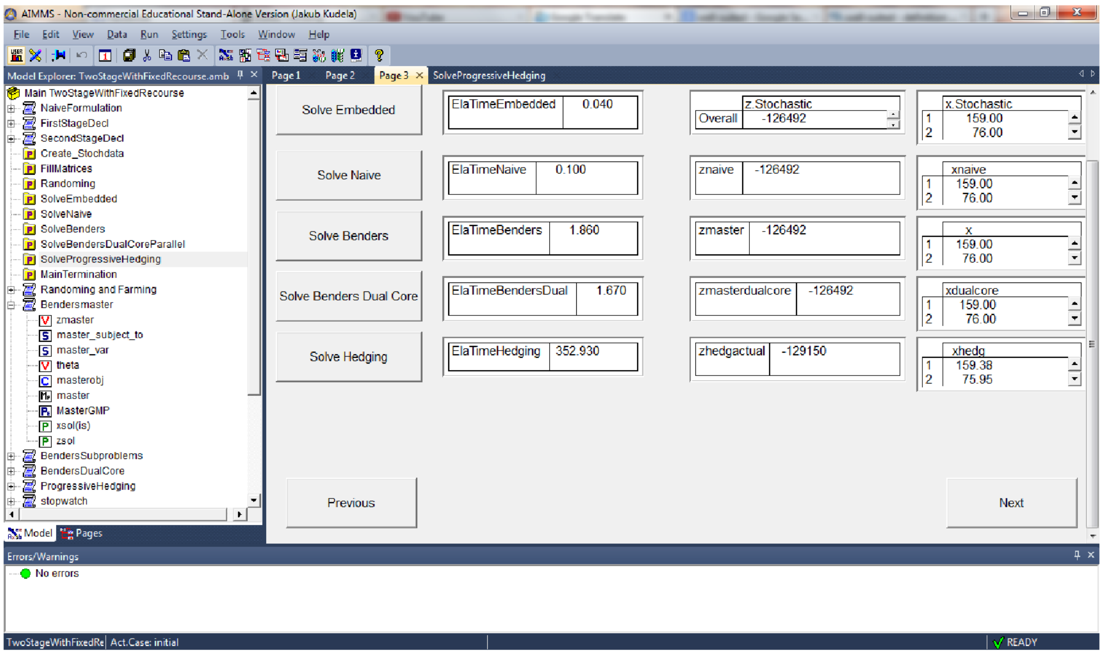
After the solving of subproblems ends (either by a detection of unbounded/infeasible solution or by completing the solution process of all the subproblems), we check for the solution status of the last subproblem.

If the solution status was unbounded, we update the matrices of the master problem (and, thus, generate a feasibility cut) and repeat:

```
if solstatus = 'Unbounded' then
GMP::Row::Add(MasterGMP,addconst(lastIterSet));
for (is) do
GMP::Coefficient::Set(MasterGMP,addconst(lastIterSet),x(is),FeasF(is));
endfor;
GMP::Coefficient::Set(MasterGMP,addconst(lastIterSet),theta,0);
GMP::Row::SetRightHandSide(MasterGMP,addconst(lastIterSet),FeasSmallF);
endif;
```

If the last solution status was optimal, then all of the subproblems have optimal solution. If the following condition is satisfied, we arrived at the optimal solution of the whole problem and stop:

```
if solstatus = 'Optimal' then
OptD(is):=sum(sb, (DesignedProbability(sb)/ScenarioProbSum)*
sum(ssc,-usol(sb,ssc)*Tinput(sb,ssc,is)));
OptSmallD:=sum(sb, (DesignedProbability(sb)/ScenarioProbSum)*
sum(ssc,-usol(sb,ssc)*hinput(sb,ssc)));
if theta+epsilon>=OptSmallD - sum(is,xsol(is)*OptD(is)) then
solutionstatus:="solution found";
errorval:=OptSmallD - sum(is,xsol(is)*OptD(is)) - theta;
break;
```



No errors

ElaTimeEmbedded	0.040
ElaTimeNaive	0.100
ElaTimeBenders	1.860
ElaTimeBendersDual	1.670
ElaTimeHedging	352.930

	z.Stochastic
Overall	-126492

znaive	-126492
zmaster	-126492
zmasterdualcore	-126492
zhedgactual	-129150

	x.Stochastic	xnaive	x	xdualcore	xhedg
1	159.00	159.00	159.00	159.00	159.38
2	76.00	76.00	76.00	76.00	75.95

Fig. 3.11: Page devoted to different solution methods.

And if not, we generate an optimality cut and repeat:

```
else errorval:=OptSmallD - sum(is,xsol(is)*OptD(is)) - theta;
GMP::Row::Add(MasterGMP,addconst(lastIterSet));
for (is) do
GMP::Coefficient::Set(MasterGMP,addconst(lastIterSet),x(is),OptD(is));
endfor;
GMP::Coefficient::Set(MasterGMP,addconst(lastIterSet),theta,1);
GMP::Row::SetRightHandSide(MasterGMP,addconst(lastIterSet),OptSmallD);
GMP::Row::SetType(MasterGMP,addconst(lastIterSet),'>=');
endif;
```

The progressive hedging algorithm is mainly designed for solving convex problems and is not well-suited for simple linear problems. It does not utilize the advantage of linearity in the case of scenario-based linear programme, as the solved scenario-related programme contains an additional quadratic term that ruins the linearity. In spite of this fact, we chose to incorporate this solution method because our General Program can be very easily modified (by adding constraints and/or adding terms in the objective function) to be a convex programme.

To compare these solution methods we programmed a stopwatch procedure (see [15]) that measures the time it took the given method to reach the optimal solution. We also designed one page in the GUI to summarize the results obtained by different solution methods (Fig. 3.11). The source code of this program can be found in Appendix 2.

3.2.1 Results comparison

The aim of this section is to compare the used solution methods. For this purpose we generated in our General Program several linear programmes of a different size and observed how fast did the methods reach the optimum. We will omit the progressive hedging algorithm from the full comparison, since it is not designed for solving linear programmes (it does solve them, but the amount of time it takes is rather large).

First of all, we let all the algorithms solve the original farmer's problem, as presented in section 3.1.2. The results (computational times) are presented in the following table:

algorithm	computational time
Embedded	0.02
Naive	0.02
Benders: single core	0.48
Benders: dual core	0.46
Progressive hedging	5.28

From these results it is fairly obvious, that the progressive hedging algorithm is not very well suited for these kinds of problems.

For further comparison, we decided to test the speed of our algorithms on randomly generated problems (we have used the straightforward and internal support of AIMMS for random number generation). Furthermore, we decided to find out how does the size of the second stage part of the two-stage linear programme affect the computational times of the presented algorithm. However, it is not our goal to give a thorough statistical insight into this problematic; we just want to gain primal knowledge of this phenomena and leave a profound analysis for our future studies. We constructed the problems as follows:

- number of first stage variables = 10,
- number of first stage conditions = 5,
- **A** = Round(Uniform(0.3,1)),
- **b** = Uniform(1000,2500),
- **c** = Uniform(-4500,-3700),
- **T** = Uniform(50,100),
- **W** = Uniform(7,20),
- **h** = Uniform(700,1500),
- **q** = Uniform(-300,-180).

The other three parameters, namely, a number of second stage conditions, a number of second stage variables and a number of scenarios, will vary for each programme. We chose 2 values for the number of scenarios: 200 and 500. The number of second stage conditions and the number of second stage variables will be both from $\{50, 60, 70, 80, 90, 100\}$, for the problems with 200 scenarios, and from $\{20, 30, 40, 50\}$, for the problems with 500 scenarios.

All the inequalities are chosen to be of a $\leq$ type and all the parameters are chosen in such a way, that there always exists an optimal solution. This is easy to see, since a zero solution (i.e. a solution where all the variables are zero vectors) is a feasible solution.

To summarize, the comparison process looks like this:

- We choose a number of scenarios, a number of second stage conditions and a number of second stage variables.
- We generate the vectors and matrices.
- We run all our algorithms on this generated problem and find the amount of time it took the specific algorithm to find optimum.

The results of this comparison are presented in the following tables and figures. First for the problems with 200 scenarios:

200 scenarios - Naive algorithm						
# of second stage variables	# of second stage conditions					
	50	60	70	80	90	100
50	26.46	39.12	79.38	151.11	41.88	103.48
60	43.3	64.4	96.54	73.81	89.55	45.15
70	39.01	44.85	98.62	63.9	163.69	219.39
80	97.38	67.24	143.79	141.52	98.27	108.3
90	49.09	84.02	98.39	208.52	175.79	136.89
100	53.45	49.51	102.42	184.66	66.41	308.59

200 scenarios - Embedded algorithm						
# of second stage variables	# of second stage conditions					
	50	60	70	80	90	100
50	25.92	39.21	80.57	147.57	41.54	98.38
60	42.55	64.97	95.92	74.83	89.1	45.64
70	39.34	41.78	100.1	65.19	186.54	212.85
80	91.9	68.45	140.99	144	99.58	108.31
90	46.85	84.92	98.72	211.85	176.56	124.32
100	69.83	51.27	96.92	184.58	67.15	327.13

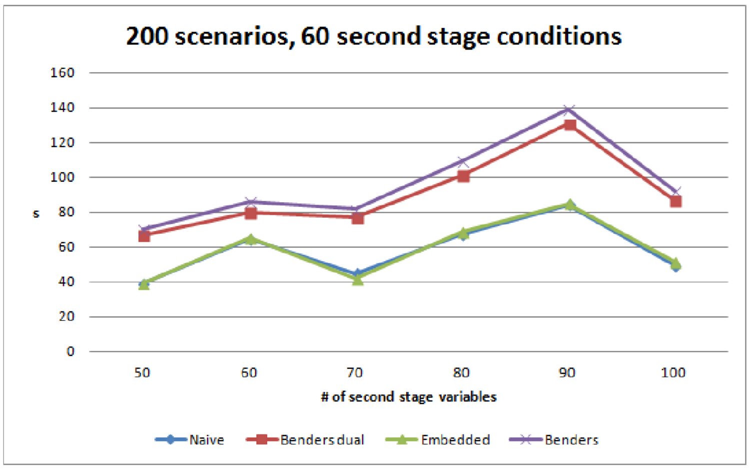


Fig. 3.12: Graph: 200 scenarios, 60 second stage conditions

200 scenarios - Benders decomposition: single core						
# of second stage variables	# of second stage conditions					
	50	60	70	80	90	100
50	90.38	70.15	79.64	84.9	44.76	110.3
60	112.15	86.12	77.44	87.47	77.54	68.67
70	172.16	81.66	102.64	82.57	72.47	131.57
80	118.43	109.45	63.19	87.39	55.82	101.5
90	79.48	138.98	93.92	121.91	88.87	100.69
100	50.58	92.28	65.06	75.21	96.18	117.73

200 scenarios - Benders decomposition: dual core						
# of second stage variables	# of second stage conditions					
	50	60	70	80	90	100
50	52.03	66.69	74.43	80.47	42.73	94.91
60	57.31	80.07	71.2	82.13	71.15	64.52
70	83.38	77.01	96.34	79.19	69.59	125.81
80	79.65	101.39	59.05	82.1	54.21	97.14
90	75.03	130.44	89.38	114.25	85.09	94.63
100	43.95	86.72	61.79	74.75	95.36	112.45

And here for the problems with 500 scenarios:

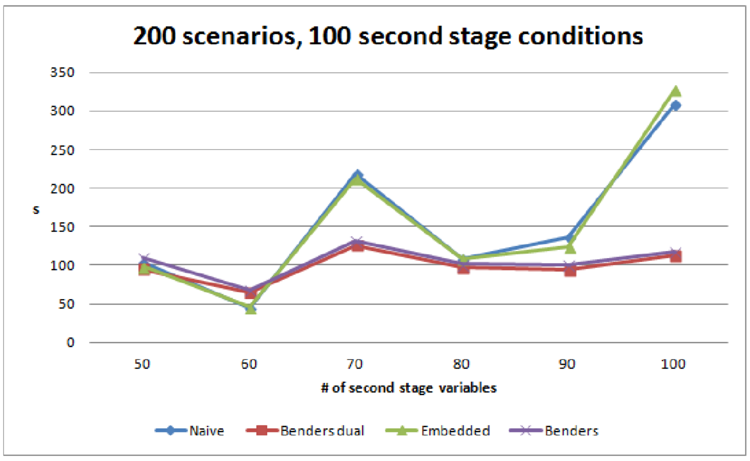


Fig. 3.13: Graph: 200 scenarios, 100 second stage conditions

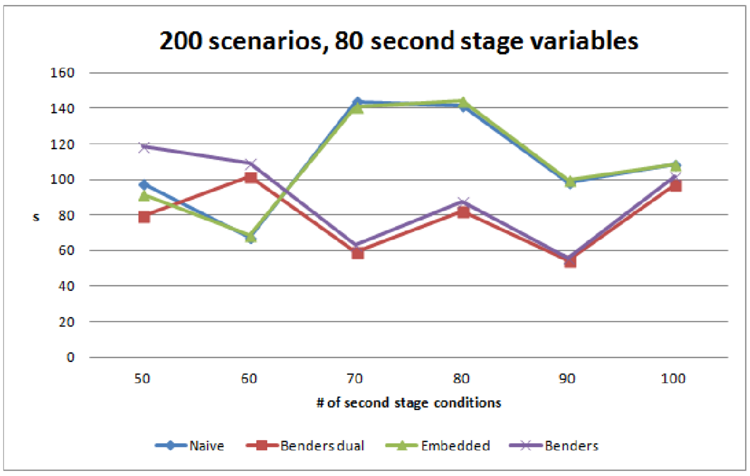


Fig. 3.14: Graph: 200 scenarios, 80 second stage variables

500 scenarios - Naive algorithm				
# of second	# of second stage conditions			
stage variables	20	30	40	50
20	94.56	101.96	88.26	230.43
30	74	113.93	123.94	361.59
40	57.72	100.36	81.06	208.51
50	27.94	103.25	126.2	119.87

500 scenarios - Embedded algorithm				
# of second	# of second stage conditions			
stage variables	20	30	40	50
20	93.81	100.48	82.85	234.09
30	62.8	114.6	122.75	358.48
40	66.74	103.54	82.88	207.11
50	28.42	102.42	129.17	126.98

500 scenarios - Benders decomposition: single core				
# of second	# of second stage conditions			
stage variables	20	30	40	50
20	94.81	94.03	182.88	132.19
30	106.53	114.35	161.52	125.38
40	86.81	176.75	147.26	100.54
50	92.98	49.93	154	118.57

500 scenarios - Benders decomposition: dual core				
# of second	# of second stage conditions			
stage variables	20	30	40	50
20	92.62	89.85	174.19	124.08
30	112.03	107.77	152.6	125.28
40	80.06	162.41	154.02	99.84
50	88.24	48.39	149.14	115.96

From the results above, we deduce that (at least for our generated problems) the Naive algorithm and the AIMMS Embedded algorithm are basically the same algorithm. We can, also, notice that the parallelization done in the dual core version of the Benders decomposition offers a slight, but noticeable, improvement in the computing time. On the other hand, the results of the comparison between the Naive algorithm and the Benders decomposition (either dual or single core version) are not very clear. Each of these algorithms performed better than the other one

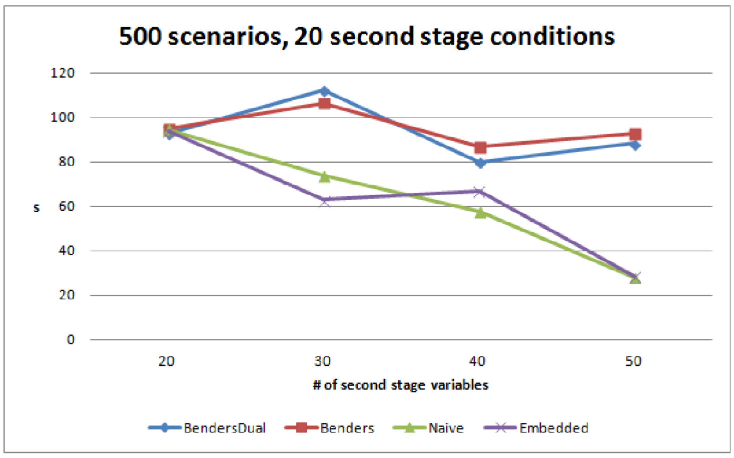


Fig. 3.15: Graph: 500 scenarios, 20 second stage conditions

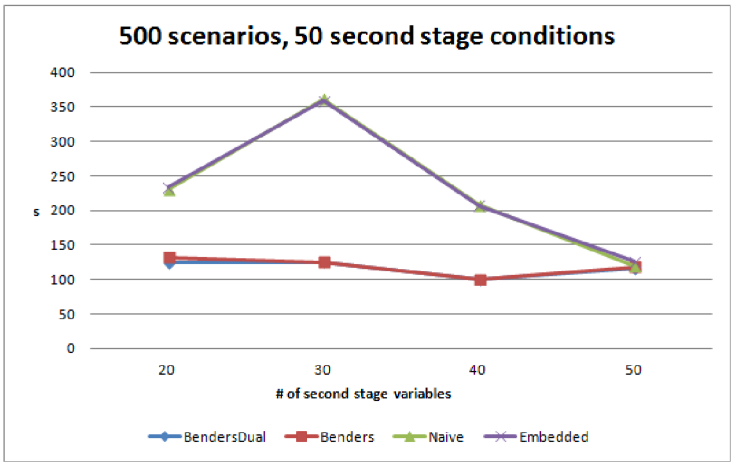


Fig. 3.16: Graph: 500 scenarios, 50 second stage conditions

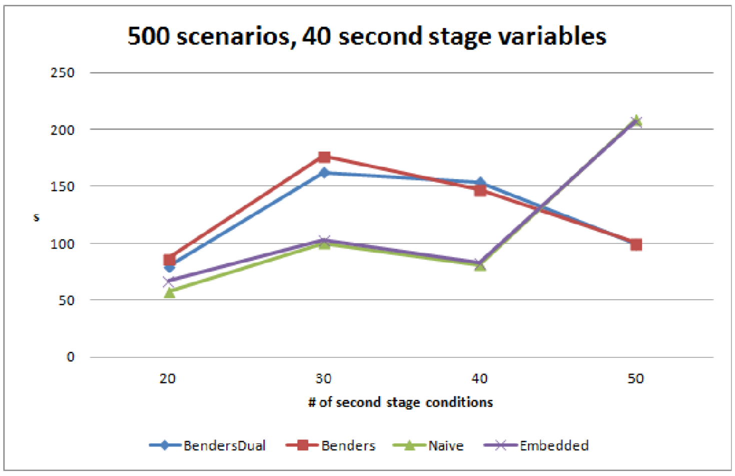


Fig. 3.17: Graph: 500 scenarios, 40 second stage variables

at some problems. However, we only generated one instance for each configuration so any conclusions we make may end up being premature. Generally speaking, as the problem grows in size, the Benders decomposition should be faster than our Naive algorithm. Another important factor is the particular structure of the generated problems, that we chose to test our algorithms on. This dependence remains unknown for us and hopefully will be a part of our future studies.

4 REAL-LIFE EXAMPLE: AN INCINERATION PLANT

The task is to model the waste-to-energy process of an incineration plant, optimize this process and determine the parameters of its most important components, namely a steam turbine or a boiler, as an important part of the incinerator. In fact, also the capacity of incinerator can be optimised in the similar way. The focus is on the simplified model that may easily utilize AIMMS user interface and may help to give a rough estimate for the incinerator optimal design. It also allows visualization of the function $\mathcal{Q}(\mathbf{x})$ in 1.7.

4.1 The Incineration Process

Incineration is a waste treatment process that consists of the combustion of organic substances, contained in waste materials.

The heat, produced by a boiler, is used to generate steam which is then used to drive a steam turbine and, thus, produce electricity. Other option for the usage of this steam is district heating (industrial or municipal). More can be found in ([19], [9]).

Our incineration plant deals with burning a municipal solid waste (MSW), which is a waste type consisting of everyday items that are discarded by the public. The composition of this waste varies throughout the year and its lower heating value (LHV), i.e. the amount of heat we are able to get from a certain quantity of the material by burning it, changes randomly. We want to optimize the waste-to-energy process for 24 years.

The technological process of the incineration plant (Fig. 4.1) can be described as follows:

1. MSW, that was transported to the plant, is stored and regularly mixed. This ensures that the structure of the waste is roughly the same throughout a day.
2. The waste is moved by a feeding unit into a firing grate and is burned. The heat generated by the combustion heats up water inside a boiler and turns this water into a steam.
3. The steam is, then, run through a steam turbine and generates electricity.
4. The steam that has not been used for generating electricity either goes to other technological processes or heats up water, that will end up in district heating.
5. The slag that remains after burning is decomposed into ashes and metals and can be used in other processes.

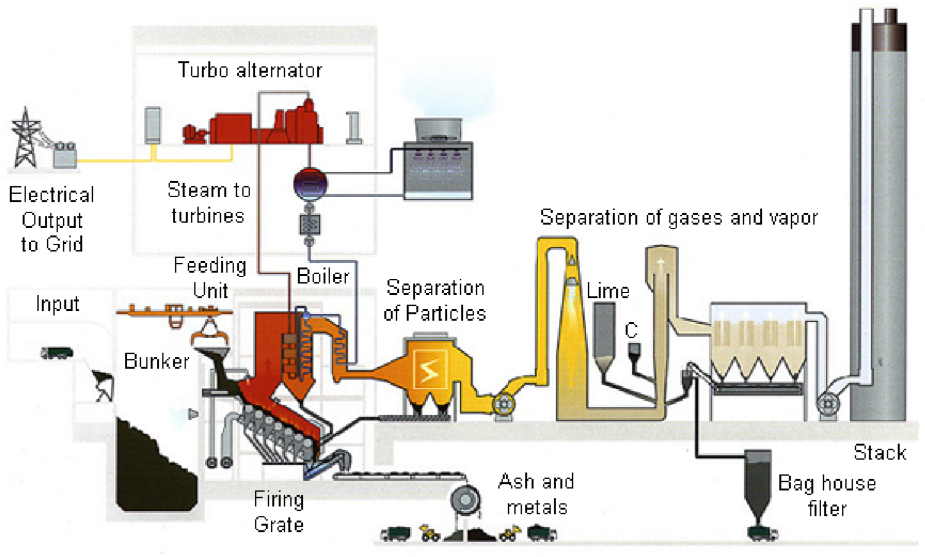


Fig. 4.1: The technological process of the incineration plant. [17]

6. Separation of gases and vapour takes place; this treatment reduces the pollutant emissions to the atmosphere.

4.2 Simplified Problem Formulation

We will present here a profound insight into a model that is the simplest possible one, however, it is still approximately suitable for the introduced incineration problem. After that, we will show that AIMMS is useful even without its optimization tools.

The goal is to determine the size of a boiler. The objective function comprises of the investment cost of the boiler (linearly depending on the boiler's size) and the recourse, which in this case is the amount of money that we will get from using that boiler. Since we deal with stochastic programming, some of the problem parameters will depend on a realization of the random vector ($\boldsymbol{\xi}$).

First of, we define our variables and parameters:

- variable x: the size of the boiler,
- variable y: the amount of energy (generated by the boiler) that we sell,
- parameter $t(\boldsymbol{\xi})$: energy transformation coefficient,
- parameter $h(\boldsymbol{\xi})$: energy demand,
- parameters l, u: lower and upper bounds on the size of the boiler,
- parameter c: cost for a unit size of a boiler,
- parameter $q(\boldsymbol{\xi})$: cost of energy,

We will proceed to define our stochastic programme as follows:

$$\begin{array}{rcl} \min z &=& c \cdot x + \mathcal{Q}(x) \\ \text{s.t.} && x \in [l, u]. \\ \mathcal{Q}(x) &=& E_{\boldsymbol{\xi}} Q(x, \boldsymbol{\xi}). \\ Q(x, \boldsymbol{\xi}) &=& \min_y q(\boldsymbol{\xi}) \cdot y \\ \text{s.t.} && y \leq t(\boldsymbol{\xi}) \cdot x, \\ && y \leq h(\boldsymbol{\xi}), \\ && y \geq 0. \end{array} \tag{4.1}$$

The randomness will be treated via scenarios (as shown in chapter 1). Given a fixed number of scenarios N (i.e. $\boldsymbol{\xi}_s$: the realizations of the random vector $\boldsymbol{\xi}$), our problem becomes:

$$\begin{array}{rcl} \min z &=& c \cdot x + \sum_{s=1}^{N} p_s Q(x, \boldsymbol{\xi}_s) \\ \text{s.t.} && x \in [l, u]. \\ Q(x, \boldsymbol{\xi}_s) &=& \min_{s \in S} q_s \cdot y_s \\ \text{s.t.} && y \leq t_s \cdot x, \\ && y \leq h_s, \\ && y \geq 0. \end{array} \tag{4.2}$$

We have to emphasize the fact that 4.2 model is the simple version of real-world models published in [10]. Its motivation was discussed with colleagues from ÚPEI (Institute of Process and Environmental Engineering): Michal Touš, Radek Šomplák, Martin Pavlas and my supervisor, as the suitable tool for initial estimates of the capacity of a boiler or an incinerator, before the advanced models are available. The advantage of the model is that only aggregated data is needed and the user can easily check whether the optimal capacity tends to boundary capacities/sizes or interior point solution can be expected.

We shall, now, derive the solution for a particular realization of the random vector $\boldsymbol{\xi}$, i.e. we will treat the random parameters as deterministic ones. (This corresponds to solving the problem for only one scenario).

To obtain it we will firstly deal with the second stage problem:

$$\begin{array}{rcl} Q(x, \boldsymbol{\xi}_s) &=& \min q_s \cdot y_s \\ \text{s.t.} && y \leq t_s \cdot x, \\ && y \leq h_s, \\ && y \geq 0. \end{array}$$

We can easily derive a relation between y and x: $y \in [0, \min(t_s \cdot x, h_s)]$. We will define a point $\hat{x}$ as $\hat{x} = h_s/t_s$. Now, given that $q < 0, t_s > 0, h_s > 0$ (which are

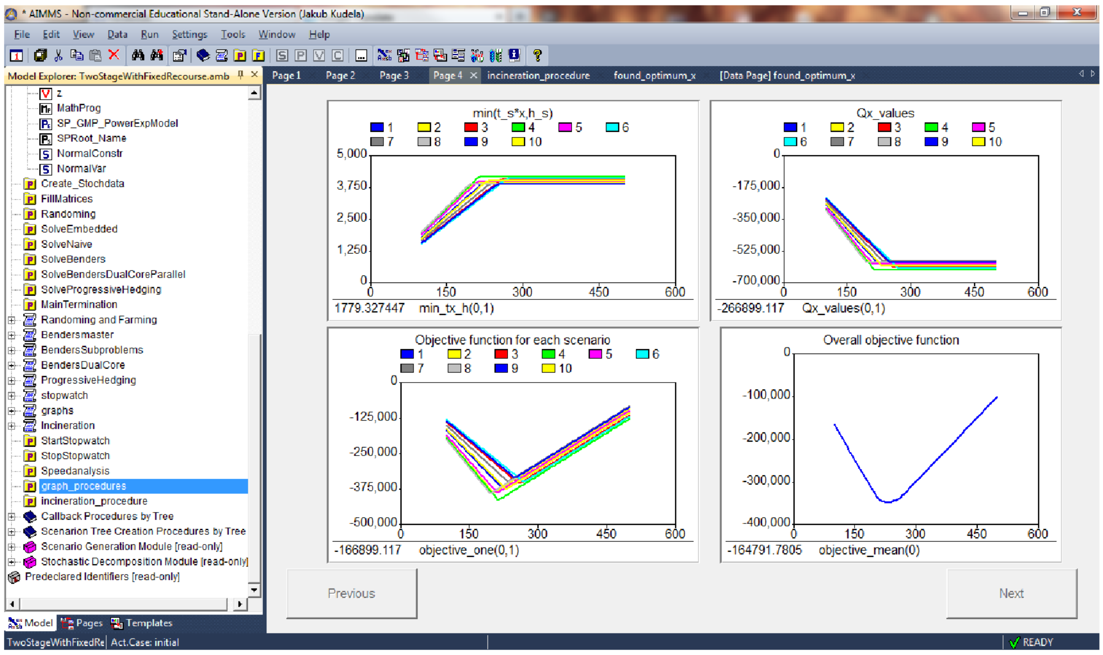

* AIMMS - Non-commercial Educational Stand-Alone Version (Jakub Kudela)
File Edit View Data Run Settings Tools Window Help
Model Explorer: TwoStageWithFixedRecourse.amb
Page 1 Page 2 Page 3 Page 4 incineration_procedure found_optimum_x [Data Page] found_optimum_x
z
MathProg
SP_GMP_PowerExpModel
SPRoot_Name
NormalConstr
NormalVar
Create_Stochdata
FillMatrices
Randoming
SolveEmbedded
SolveNaive
SolveBenders
SolveBendersDualCoreParallel
SolveProgressiveHedging
MainTermination
Randoming and Farming
Bendersmaster
BendersSubproblems
BendersDualCore
ProgressiveHedging
stopwatch
graphs
Incineration
StartStopwatch
StopStopwatch
Speedanalysis
graph_procedures
incineration_procedure
Callback Procedures by Tree
Scenarion Tree Creation Procedures by Tree
Scenario Generation Module [read-only]
Stochastic Decomposition Module [read-only]
Predeclared Identifiers [read-only]
Model Pages Templates
TwoStageWithFixedRe Act.Case: initial
min(t_s*x,h_s)
1 2 3 4 5 6 7 8 9 10
5,000 3,750 2,500 1,250 0
0 150 300 450 600
1779.327447 min_tx_h(0,1)
Qx_values
0 -175,000 -350,000 -525,000 -700,000
-266899.117 Qx_values(0,1)
Objective function for each scenario
0 -125,000 -250,000 -375,000 -500,000
-166899.117 objective_one(0,1)
Overall objective function
0 -100,000 -200,000 -300,000 -400,000
-164791.7805 objective_mean(0)
Previous
Next
READY

Fig. 4.2: Multiple scenarios case.

natural condition on the parameters), we obtain $Q(x, \boldsymbol{\xi}_s) = q \cdot \min(t_s \cdot x, h_s)$. There are three possible outcomes, depending on the position of $\hat{x}$:

- $Q(x, \boldsymbol{\xi}_s) = q \cdot t_s \cdot x$ for $\hat{x} \geq u$,
- $Q(x, \boldsymbol{\xi}_s) = q \cdot h_s$ for $\hat{x} \leq l$,
- $Q(x, \boldsymbol{\xi}_s) = q \cdot t_s \cdot x$ for $x \in [l, \hat{x}]$ and $Q(x, \boldsymbol{\xi}_s) = q \cdot h_s$ for $x \in [\hat{x}, u]$

Now, we focus on the first stage part:

$$\begin{aligned} \min z &= c \cdot x + Q(x, \boldsymbol{\xi}_s) \\ \text{s.t.} & \quad x \in [l, u]. \end{aligned}$$

Once we take into account the results of the second stage, the overall results easily emerge. We will denote the overall optimal solution as x^* and obtain it as follows:

- if $|t_s \cdot q| > c$ and $\hat{x} \in [u, l]$ then $x^* = \hat{x}$,
- if $|t_s \cdot q| > c$ and $\hat{x} \leq l$ then $x^* = l$,
- if $|t_s \cdot q| > c$ and $\hat{x} \geq u$ then $x^* = u$,
- if $|t_s \cdot q| \leq c$ then $x^* = l$.

This whole procedure was basically an effort to find a combination of values of the problem parameters, that would lead to an optimal solution, which is not a boundary value of the interval $[l, u]$. Thus, our effort was successful since we found such a combination.

In the case of multiple scenarios we were not able to arrive at any similar general results. We used AIMMS GUI to illustrate the process of obtaining the overall objective function in the case of multiple scenarios (Fig. 4.2).

Moreover, we programmed the same procedure in MATLAB; we did this because we feel that the MATLAB environment is more suited for this kind of computations and, for anyone not completely familiar with AIMMS, even more user-friendly. Apart from easy adjustment of the model parameters, it gives a graphical result as shown in Fig. 4.3. The MATLAB program is enclosed in the appendix.

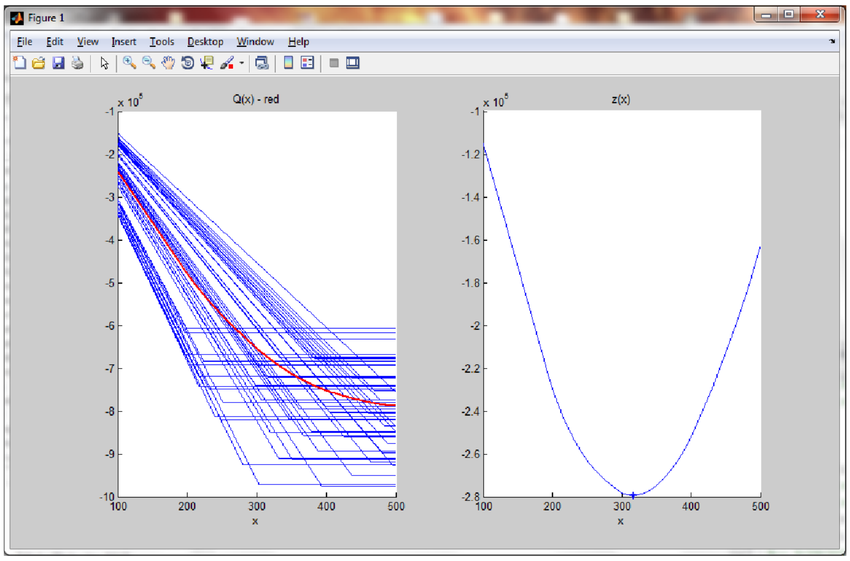


Fig. 4.3: Matlab implementation

4.3 Nonlinearity and Real Data

We asked our colleagues from Institute of Process and Environmental Engineering (BUT) for some real data, regarding our incineration model. The data, we were given, can be summarized in the following expressions:

- The task is to find the optimal capacity of a new boiler for an incineration plant, whose lifespan is 24 years. The range of this capacity is between 50000 and 400000 tons per year. The objective function is comprised of costs (for construction, maintenance, etc.) and earned money (from selling heat, ...).
- The function describing the cost of both constructing and running this boiler was determined as:

$$\text{cost} = 2688000x^{0.7},$$

  where x denotes the capacity of the boiler in tons.

- Our incineration plant produces both heat and electricity. The relationship between the capacity and the amount of produced heat and electricity is governed by the following inequality:

$$0.001x \geq 0.2876 + 0.0126y_1 + 0.0828y_2,$$

where y_1 and y_2 are the amount of heat and electricity, respectively; their unit is $\frac{\text{GWh}}{24\text{ years}}$.
- The incineration plant makes money by selling this heat and electricity and, also, by charging the processing of municipal solid waste. The prices are going to be treated as random parameters with the following properties:
  - the price of 1 GWh of heat: $q_1 \sim N(-1044000, 72000)$,
  - the price of 1 GWh of electricity: $q_2 \sim N(-1650000, 100000)$,
  - the price of 1 ton of processed MSW: $q_3 \sim N(-1500, 100)$,

  where $N(\mu, \sigma)$ stands for normal distribution.
- The amount of heat, we can produce, is restricted by demand (in $\frac{\text{GWh}}{24\text{ years}}$): $h_1 \sim N(6000, 600)$.
- The amount of generated electricity is also restricted by (in $\frac{\text{GWh}}{24\text{ years}}$): $h_2 \sim N(1000, 100)$.
- The amount of processed solid waste over the lifespan of the incineration plant will be equal to 24 times the capacity of the boiler (i.e. we use the boiler to its full potential). Since the price of this procedure is random, we will use an additional variable y_3.

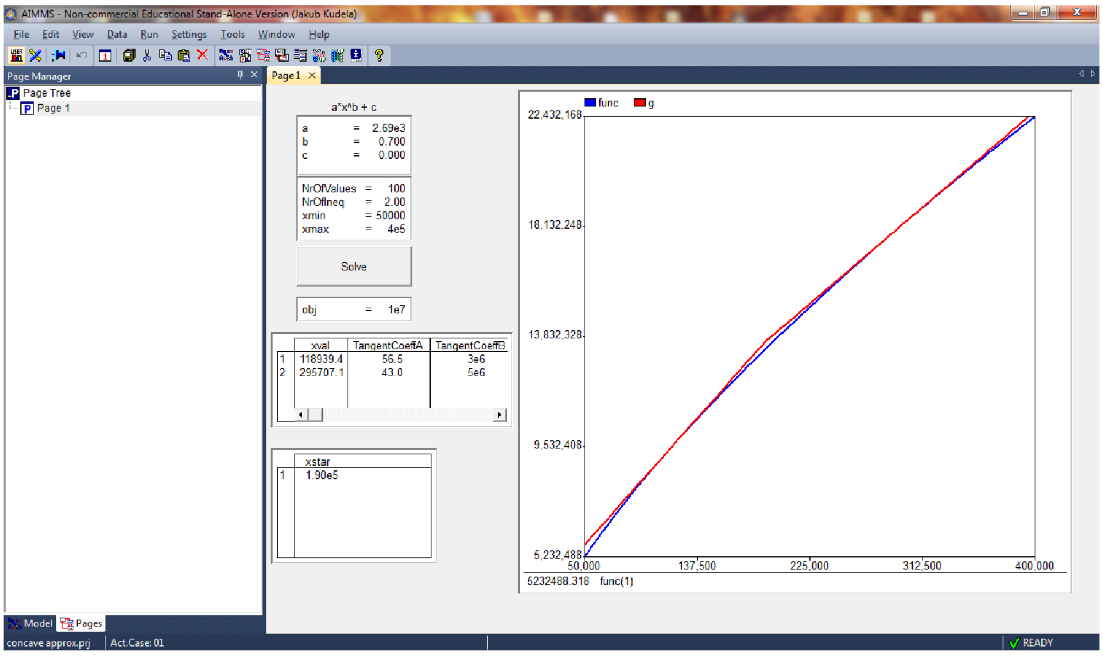
First of, we need to deal with the nonlinearity in the function describing costs. We approach this by constructing an outer approximation of this function. This approximation is done via construction of tangents of the original function. Since the cost function is concave, its tangents have their function value always $\geq$ than the concave function itself.

We programmed an AIMMS implementation of this procedure; since it is not the purpose of this text, we will omit any description and just present the results for our problem in Fig. 4.4.

We decided to use for our approximation 2 tangent lines. The program gave us the following results:

- $\tau_1(x) = 56500x + 2879244000$, for $x \in [50000, 190140]$,
- $\tau_2(x) = 43000x + 5446954000$, for $x \in [190140, 400000]$.

This allows us to split out problem in two linear problems. The first one with $x \in [50000, 190140]$ the cost function described as $\tau_1(x)$ and the second one with $x \in [190140, 400000]$ the cost function described as $\tau_2(x)$. These problems can be



a*x^b + c

```
a             =   2.69e3
b             =    0.700
c             =    0.000

NrOfValues  =     100
NrOfIneq    =    2.00
xmin        =   50000
xmax        =     4e5

obj         =     1e7
```

	xval	TangentCoeffA	TangentCoeffB
1	118939.4	56.5	3e6
2	295707.1	43.0	5e6

	xstar
1	1.90e5

Fig. 4.4: Approximation of a concave function (blue) by 2 tangents (red).

summarized for a fixed number of scenarios N, as follows:

$$
\begin{array}{rcll}
\min z & = & \tau_k(x) + \sum_{s=1}^{N} p_s \mathbf{q}_s^T \mathbf{y}^s & \\
\text{s.t.} & & 0.001x - 0.0126y_1^s - 0.0828y_2^s \geq 0.2876, & s = 1,\dots,N, \\
 & & y_1^s \leq h_1^s, & s = 1,\dots,N, \\
 & & y_2^s \leq h_2^s, & s = 1,\dots,N, \\
 & & 24x - y_3^s = 0, & s = 1,\dots,N, \\
 & & x \in [l_k, u_k], \mathbf{y}_s \geq 0, & s = 1,\dots,N, k = 1,2,
\end{array}
$$

where $[l_1, u_1] = [50000, 190140]$ and $[l_2, u_2] = [190140, 400000]$. The optimal solution of the whole problem will be determined as the best one of the two partial solutions. We used our General Program to model and solve this problem; we chose to generate 100 scenarios for the random parameters. Without further ado, we present the results:

- For $x \in [50000, 190140]$ the optimal capacity is $\hat{x} = 93125$ with objective $z = -7.85e9$.
- For $x \in [190140, 400000]$ the optimal capacity is $\hat{x} = 247386$ with objective $z = -8.4e9$.

Just for clarification, the negative value of the objective z indicates that the incineration plant is profitable. From this we gain the optimal capacity for the whole problem $x^* = 247386$ tons for year.

CONCLUSION

In this master's thesis we dealt with optimization and stochastic programming. We described the basic tools of mathematical programming in Chapter 1; in the same chapter, we presented the main ideas of stochastic programming along with the solution methods used for solving these stochastic programming problems. This chapter laid the theoretical foundations for implementations and algorithms we constructed in Chapter 3.

Chapter 2 served as the introduction to the optimization modelling software AIMMS. We mentioned the crucial ideas and functions in AIMMS, that allowed us to use the theoretical results from Chapter 1 and create stochastic programmes in AIMMS.

The most important part of our work is contained in Chapter 3. Here, we combined the theoretical results from stochastic programming with the functionality of AIMMS. We constructed in AIMMS a general two-stage linear stochastic program and an end-user interface, for this program. This program can be used for solving any kind of two-stage linear stochastic problem. Moreover, we implemented and compared several solution methods described in Chapter 1, namely, the sample average approximation, the benders decomposition and the progressive hedging algorithm. This comparison was carried out on generated problems of different size and on the Farmer example.

In Chapter 4 we have shown that AIMMS is useful even without its optimization tools and functions; we used its graphical interface to obtain the solution of a simplified incineration problem.

We hope, that this text will be helpful to anyone, who is interested in stochastic programming and, mainly, in the usage of AIMMS in dealing with stochastic programming problems. We enclosed in the appendices all the constructed AIMMS programs we described throughout the thesis.

BIBLIOGRAPHY

[1] BAZARAA, M. S. - SHERALI, H. D. - SHEETY, C. M.:*Nonlinear Programming: Theory and Algorithms.* John Wiley Sons, New York, second edition, 1993. ISBN 0-471-55793-5.

[2] BISSCHOP, J.: *Aimms Function Reference.* Haarlem: Paragon Decision Technology, 2012. [Cited 7.4.2014]
`http://www.aimms.com/downloads/other-references-and-guides/function-reference`

[3] BISSCHOP, J.: *Aimms Optimization Modeling.* Haarlem: Paragon Decision Technology, 2007. ISBN 1847539122.

[4] BISSCHOP, J. - ROELOFS, M.: *Aimms Language Reference.* Haarlem: Paragon Decision Technology, 2007. ISBN 1847539114.

[5] BISSCHOP, J. - ROELOFS, M.: *Aimms - User's Guide.* Haarlem: Paragon Decision Technology, 2007. ISBN 1847537820.

[6] BIRGE, J. R. - LOUVEAUX, F.: *Introduction to Stochastic Programming.* New York: Springer Series in Operations Research, Springer Verlag, 1997. ISBN 0-387-98217-5.

[7] KLIMEŠ, L.: Stochastic programming algorithms. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010.

[8] KŮDELA, J.: *Optimalizační úlohy v programu AIMMS.* Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2012.

[9] ŠOMPLÁK, R.: *Využití metod stochastického programování pro hodnocení investic v energetických zdrojích.* Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2011.

[10] ŠOMPLÁK, R. - UCEKAJ, V. - POPELA, P. - PAVLAS, M.: *Waste-to-Energy Facility Planning Supported by Stochasting Programming - Part I Introduction.* Chemical Engineering Transactions, 2012, vol. 29, no. 1, p. 649-654. ISSN: 1974-9791.

[11] RARDIN, R. L.: *Optimization in operations research.* New Jersey: Prentice Hall, 1998. ISBN 0023984155.

[12] RUSZCZYNSKI, A. - SHAPIRO, A.: *Handbooks in Operations Research and Management Science, vol. 10: Stochastic Programming.* Amsterdam: Elsevier, 2003. ISBN 978-0-444-50854-6

[13] AIMMS. Cited [7.4.2014].
`http://business.aimms.com/`

[14] AIMMS: Academic License. Cited [27.5.2014].
`http://www.aimms.com/academic/free-academic-license`

[15] AIMMS blog: Creating StopWatch in AIMMS to time execution. Cited [2.5.2014].
`http://blog.aimms.com/2011/12/`
`creating-stopwatch-in-aimms-to-time-execution/`

[16] IBM:CPLEX Cited [16.4.2014].
`http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/index.jsp?topic=`
`%2Filog.odms.ide.help%2FContent%2FOptimization%2FDocumentation%`
`2FOPL_Studio%2F_pubskel%2Fglobals%2Feclipse_and_xplatform%2Fps_`
`opl990.html`

[17] WtERT: *Incineration*. Cited [2.1.2014].
`http://www.wtert.eu/default.asp?ShowDok=13>`

[18] Wikipedia: *AIMMS*. Cited [7.4.2014].
`http://en.wikipedia.org/wiki/AIMMS`

[19] Wikipedia: *Incineration*. Cited [7.4.2014].
`http://en.wikipedia.org/wiki/Incineration`