FortSP: A Stochastic Programming Solver
Version 1.2

January 31, 2014

http://www.optirisk-systems.com

http://www.carisma.brunel.ac.uk

Version: 1.2

Prepared by
Victor Zverovich, Cristiano Arbex Valle, Francis Ellison and Gautam Mitra
OptiRisk Systems

OptiRisk Systems
One Oxford Road
Uxbridge, Middlesex, UB9 4DA
United Kingdom
www.optirisk-systems.com
+44 (0) 1895 256484

Acknowledgements

We would like to acknowledge the following persons for their contribution to the software:

- Originally implemented by Dr Chandra Poojari and Dr Frank Ellison
- Subsequent extensive reengineering and updates made by Dr Victor Zverovich

We also acknowledge Dr Csaba I. Fábián for his contribution in respect of Level Decomposition / Regularisation methods, and Dr Suvrajeet Sen. Finally, we thank Cristiano Arbex Valle who is now the custodian of Fortsp.

Preface

FortSP is a large scale stochastic programming (SP) solver, which processes linear and mixed integer scenario-based SP problems with recourse. It also supports scenario-based problems with chance constraints and integrated chance constraints. Several different SP algorithms are available for the solution, stochastic measures such as expected value of perfect information (EVPI) and value of the stochastic solution (VSS) may be calculated, and it can use CPLEX, FortMP or Gurobi as its embedded, underlying solver engine.

Contents

1 Introduction to FortSP

1.1 The Problem

FortSP is a solver for stochastic linear, and stochastic mixed integer programs. In such a problem the decision variables are governed by linear constraints, with a linear expression for the objective. Decision variables may be continuous, binary or general integer. Certain data values will be unknown precisely, and only represented by a discrete range with a probability given for each set of uncertain values. Knowledge of the random values will become known in a stage-by-stage progression, and in recourse problems there will be decision variables reserved for each stage which adapt the solution to unfolding events. User may add a rider to some constraints that they need only hold with a certain probability in the case of chance constraints or with a limit on expected of shortfall or surplus in the case of integrated chance constraints.

Recourse Problems

A recourse problem is one in which only some decision variables must be fixed immediately. Other variables are fixed in stages - those of one period taking into account the scenario values that have become known in current and in previous stages, but with future stages still unknown. These postponed decisions are known as recourse variables.

Single Stage

A single-stage problem is the one which has no recourse variables, i.e. all decision variables must be defined without knowing the realisations of random values.

Chance and Integrated Chance Constraints

These appear as normal constraints, but whether satisfied or not is subject to the uncertainty. Chance constraints need only hold with a certain probability in the final solution. Integrated chance constraints limit the expected violation of the underlying inequalities. The expected violation can be either expected shorfall or surplus depending on the type of the underlying constraint. FortSP allows these types of constraints in single and two-stage problems only.

1.2 Command-line Interface

The `fortsp` executable provides a command-line interface (CLI) to all the features of FortSP. It can be used to solve SP problems, compute stochastic measures and generate solution reports.

Usage:

```
fortsp [options] <smps-basename>
fortsp [options] <core-file> <stoch-file> <time-file>
```

The first form can be used if the SMPS filenames have standard extensions namely `cor` for the core file, `sto` for the stoch file and `tim` for the time file and they are otherwise equal. The common part of the filenames without extensions is referred to as `<smps-basename>`. Alternatively it is possible to specify all three filenames as arguments to the `fortsp` executable.

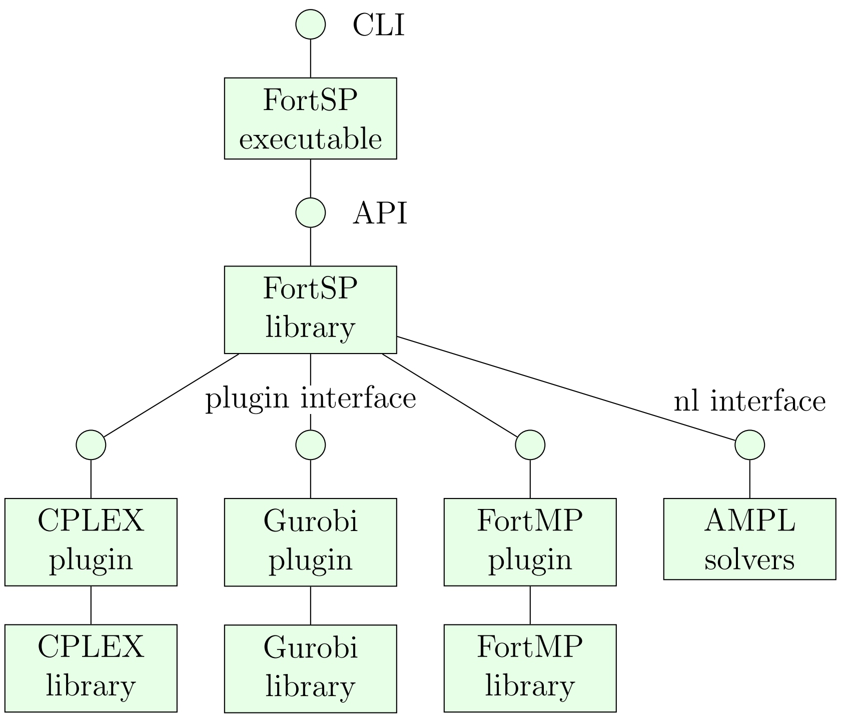


Figure 1: FortSP system architecture

Examples:

```
fortsp sgpf5y3
fortsp fxm.cor fxm2-6.sto fxm2.tim
```

Options start with `--` and may have an argument separated from the name with `=`, for example, `--solver=cplex`. Both option names and arguments are case sensitive except for the arguments that represent paths on a case-insensitive filesystem. For options that take values `0` or `1` there is a shorthand notation `--<option>` in CLI which is equivalent to `--<option>=1`, for example, `--compute-evpi`. Options are described in the subsequent sections and the complete list can be found in Appendix A. If no options specified fortsp uses the default algorithm which is problem-dependent to solve the given problem.

1.3 System Architecture

Figure 1 gives an architecture of the FortSP solver system.

Figure 2 is a simplified summary of the SP algorithm structure. The actual path taken by execution depends on a variety of factors, for example:

- The model types chosen for solution (one or more of HN, EV, WS)
- The model types needed for stochastic measures (EVPI and VSS)
- The algorithm to solve the HN model

However there are various limitations to bear in mind

- The special cutting plane algorithm applies only to single-stage problems with ICC. Other problems with chance and integrated chance constraints must be solved using deterministic equivalent approach.

1.4 Data Provision

FortSP accepts input in the SMPS format directly and in the SAMPL format through a separate SAMPL translator. It is also possible to use FortSP as a library with an application programming interface (API) in C.

SMPS Format

FortSP uses a subset of SMPS (Birge et al., 1987), which is an early format for stochastic programming problems. In this format three separate data files are provided:

- Core file: a generic presentation of the variables and constraints in MPS (MPSX) layout. Data values need not feature any particular scenario, but every random data element must be represented.
- Time file: specifying the subdivisions of the core file that belong to each stage.
- Stoch file: specifying every scenario, the values of random data, and probabilities. Only discrete forms of random distributions are considered.

SAMPL Format

AMPL is an algebraic modelling language for mathematical programming problems. SAMPL (Valente et al., 2009) is an extension of AMPL for stochastic programming. It allows representation of SP problems using syntax similar to the algebraic notation which is more economical and intelligible compared to SMPS. Unlike SMPS the whole problem can be specified in one file or divided into several files according to user requirements. Through the interactive environment SAMPL makes it easy to create or import models and data, specify options, solve the problems and examine solutions.

FortSP Library Interface

FortSP provides a powerful C API which gives programmatic access to most of the functionality of the solver.

1.5 Solution Methods

A variety of stochastic algorithms are available and these can obtain solutions in one of the following forms:

- 'Here and Now' (HN) solution
- 'Wait and See' (WS) solution
- 'Expected Value' (EV) solution

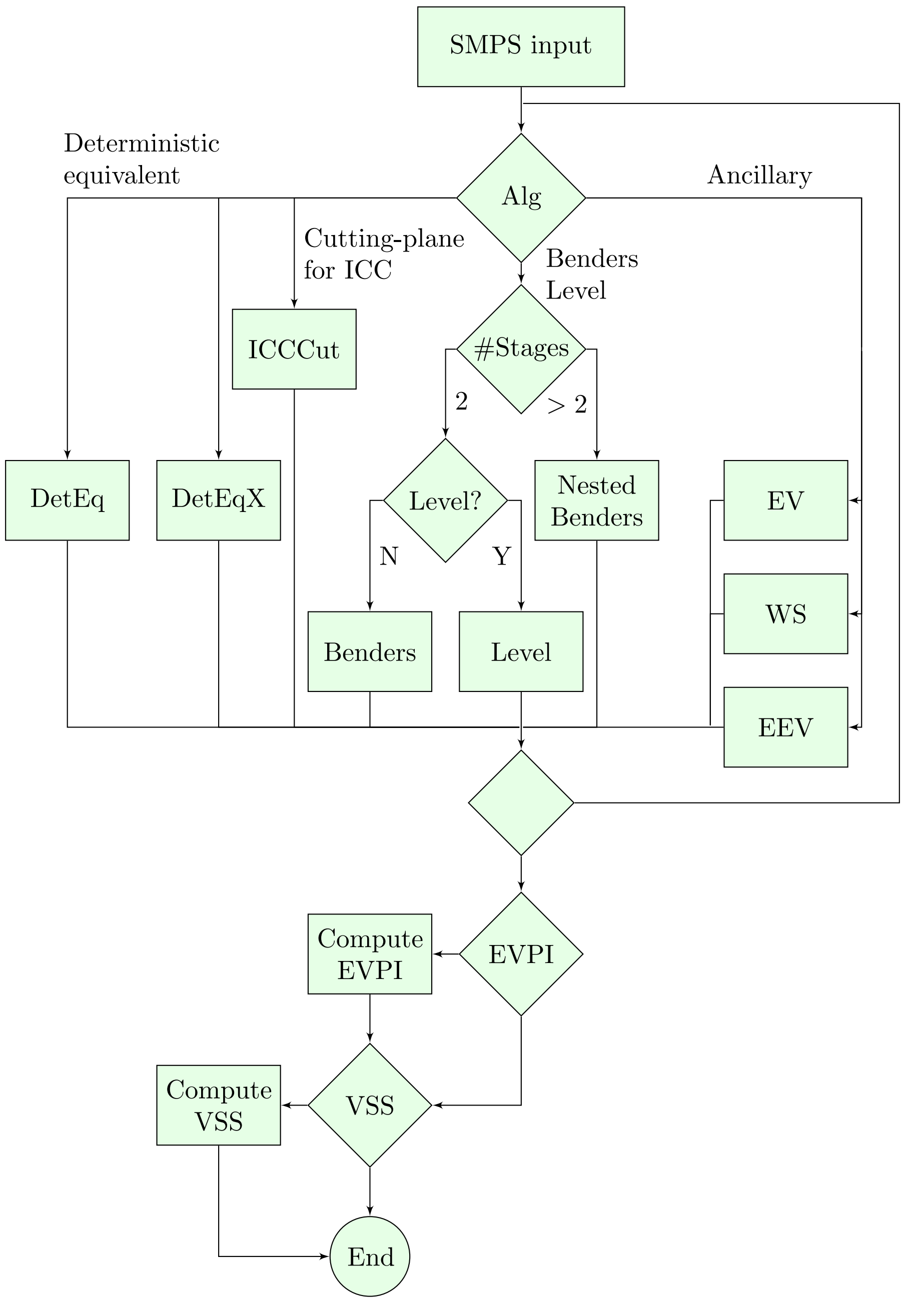


Figure 2: Simplified FortSP flowchart

HN solution provides the most exact answer to the original SP problem (also the most difficult).

The FortSP solver supports the following classes of problems and algorithms:

- Two-stage linear SP recourse problems (Here and Now)
  - Benders decomposition (L-shaped method)
  - Variant of the level decomposition of Fábián and Szőke (2007)
  - Trust region method of Linderoth and Wright (2003)
  - Regularised decomposition
  - Deterministic equivalent reformulation
- Two-stage mixed integer SP problems
  - Integer L-shaped method
  - Deterministic equivalent reformulation
- Multistage mixed integer SP problems
  - Deterministic equivalent reformulation
- Multistage linear SP problems
  - Nested Benders decomposition
  - Deterministic equivalent reformulation
- Chance constraints
  - Deterministic equivalent reformulation
- Integrated chance constraints
  - Cutting-plane algorithm of Klein Haneveld and Vlerk (2002)
  - Deterministic equivalent reformulation

The following stochastic measures may be derived from the solutions of HN, WS and EV problems:

- The expected value of perfect information (EVPI)
- The value of the stochastic solution (VSS).

1.6 External Solvers

FortSP supports several external solvers that are invoked to optimise deterministic equivalent problems or subproblems in decomposition methods. Those listed below are connected through the efficient plug-in interface:

- FortMP (Ellison et al., 2008)
- CPLEX
- Gurobi

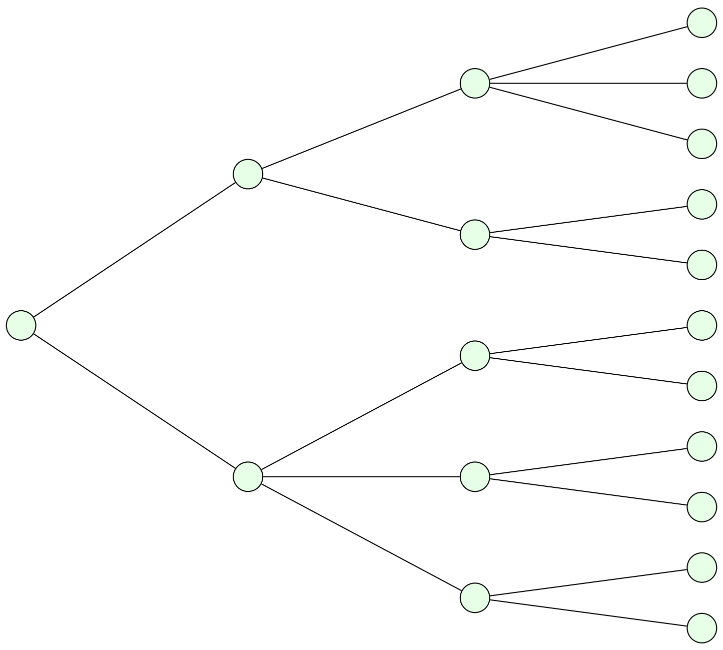


Figure 3: Scenario tree example

2 Mathematical Description of the Problem

Refer to *Introduction to Stochastic Programming* (Birge and Louveaux, 1997) for a detailed problem statement and mathematical background.

2.1 Scenario Tree

Stochastic programming can handle large numbers of decision variables and capture their complex interrelationships stated as constraints in algebraic form. The essence of SP is the confluence of optimum decision-taking model with the modelling of the random parameter behaviour. In order to model the behaviour of random parameter values we consider a limited, discrete sample of the events that may occur between any two stages of the decision-taking process, and this gives rise to a tree-like branching illustrated in figure 3.

Figure 3 illustrates the scenario tree of a 4-stage decision problem. Each node represents an optimisation problem for the decisions to be taken there, and the bundle of arcs leading from the node represents the sampled behaviour in that situation. By scenario is meant the unfolding of all events (arcs) connecting the current (first) decision (root node) to some decision in the final stage (leaf node). In general, decision tree analysis can handle only small data sets, so for realistic problem sizes there is a need for multi-stage SP.

2.2 Two-stage Recourse Models

A two-stage planning horizon is one where immediate (Here and Now) decisions (x_1) have to be taken before all the problem elements have become known. Once this happens there are further, second-stage decisions (x_2) to be taken according to the newly discovered events. After splitting the problem into known and unknown (uncertain) elements we have a first-stage problem as follows:

$$\begin{array}{lrclcl} \text{Minimize} & & & c_1^T x_1 + \theta(x_1) & & \\ \text{Subject to} & g_1 & \leq & A_{11}x_1 & \leq & h_1 \\ & l_1 & \leq & x_1 & \leq & u_1 \end{array} \tag{1}$$

where the function θ is the expectation of second-stage utility, given the decisions x_1 in the first stage.

If we select a particular scenario, this utility function will be expressed as follows:

$$\begin{array}{lrclcl} \text{Minimize} & & & c_2^T x_2 & & \\ \text{Subject to} & g_2 & \leq & A_{21}x_1 + A_{22}x_2 & \leq & h_2 \\ & l_2 & \leq & x_2 & \leq & u_2 \end{array} \tag{2}$$

without any further θ-function. Now since x_1 is known, the second-stage problem for all scenarios can be solved, from which we can derive the expectation of utility (probability-weighted average of the minima), and this defines the θ-function in the first-stage objective.

To express this more exactly we assume the (hypothetical) existence of separate second-stage decision variables x_{2s} for each scenario $s = 1, 2, \ldots, S$. Couple these with corresponding values for the uncertain data, and the second-stage model for each scenario becomes:

$$\begin{array}{lrclcl} \text{Minimize} & & & c_{2s}^T x_{2s} & & \\ \text{Subject to} & g_{2s} & \leq & A_{21s}x_1 + A_{22s}x_{2s} & \leq & h_{2s} \\ & l_{2s} & \leq & x_{2s} & \leq & u_{2s} \end{array} \tag{3}$$

So for the expectation we combine the probability-weighted minima of all the second-stage models, and the entire problem becomes:

$$\begin{array}{lrclcl} \text{Minimize} & & & c_1^T x_1 + \sum_{s=1}^{S} p_s(c_{2s}^T x_{2s}) & & \\ \text{Subject to} & g_1 & \leq & A_{11}x_1 & \leq & h_1 \\ & g_{2s} & \leq & A_{21s}x_1 + A_{22s}x_{2s} & \leq & h_{2s}, \forall s = 1, \ldots, S \\ & l_1 & \leq & x_1 & \leq & u_1 \\ & l_{2s} & \leq & x_{2s} & \leq & u_{2s}, \forall s = 1, \ldots, S \end{array} \tag{4}$$

where p_s is the probability of scenario s. This formulation of the problem is known as the Deterministic Equivalent (DEQ).

It was observed in the 50's that the form (4) is precisely the form solvable with the dual of Danzig-Wolfe decomposition, also known as Benders' decomposition or the L-shaped method. In this method a solution x_1 to the model (1) allows dual-solutions of model (3) to be calculated and applied to form an aggregated 'cut', which is a constraint added to model (1) - thus giving a new solution x_1, and so an iterative process is developed. Theory shows that the iterations converge to precisely the solution of the deterministic equivalent model (4).

It would seem simpler just to solve the DEQ model (4) were it not for the greatly increased size of the problem. However, the DEQ is useful if the number of scenarios is fairly small.

There is a second form of the DEQ that is obtained by postulating separate stage 1 decision variables for each scenario, and equating them by adding explicit constraints

$$x_{1s_1} = x_{1s_2}$$

for all pairs of scenarios s_1 and s_2 (enough pairs to make all scenario-values the same). This is known as DEQ with Explicit Non-Anticipativity (NA). The original form (4) is known as DEQ with Implicit NA. For these large problems Interior Point Method (IPM) is usually chosen in the solution. However, Implicit NA formulation may give difficulty with IPM owing to column density, and this can be overcome in some cases by using Explicit NA.

2.3 Multi-stage Recourse Models

In a multi-stage (multi-period) planning horizon with more than two stages decisions must be taken at each stage with knowledge only of the uncertainty in that stage and in previous stages. We can easily extend the modelling shown in (1) and (3) by considering a θ-function in all stages except the last. Thus stage 1 is:

$$\begin{array}{lllll} \text{Minimize} & & c_1^T x_1 + \theta_1(x_1) & & \\ \text{Subject to} & g_1 \leq & A_{11}x_1 & \leq & h_1 \\ & l_1 \leq & x_1 & \leq & u_1 \end{array} \tag{5}$$

The stages are now given by subscript t where $t = 1, \ldots, T$, and so for an intermediate stage $t < T$ we can say:

$$\begin{array}{lllll} \text{Minimize} & & c_t^T x_t + \theta_t(x1, x2, \ldots, x_t) & & \\ \text{Subject to} & g_t \leq & A_{t1}x_1 + A_{t2}x_2 + \ldots + A_{tt}x_t & \leq & h_t \\ & l_t \leq & x_t & \leq & u_t \end{array} \tag{6}$$

with variables $x_1, x_2, \ldots, x_{t-1}$ known already. For the last stage we have:

$$\begin{array}{lllll} \text{Minimize} & & c_T^T x_T & & \\ \text{Subject to} & g_T \leq & A_{T1}x_1 + A_{T2}x_2 + \ldots + A_{TT}x_T & \leq & h_T \\ & l_T \leq & x_T & \leq & u_T \end{array} \tag{7}$$

with variables $x_1, x2, \ldots, x_{T-1}$ known already. Note that each θ-function depends on the decisions in that stage and in previous stages, up until the last stage, which has no θ-function. The constraints of the t-th stage involve $x_t, x_{t-1}, \ldots, x_1$ (see [1]).

The solution of such a model requires 'nesting'. A specific model corresponds to a specific node of the scenario tree (exampled in figure 3). Hence to solve the sub-model for a given node we need the values of decisions along the path leading up to that node, and all the solutions to the sub-tree of nodes leading from it. Given a proposed solution for everything up to stage $T-1$, we can adjust the solution of stage $T-1$ by applying 2-stage Benders to each bundle of paths leading from stage $T-1$. The same process applies to stage $T-2$ by nesting the last-stage 2-stage Benders inside to form a 3-stage Benders solver. And so on for

[1] In a 'Markovian' situation the t-th stage involves only x_t and x_{t-1}. FortSP handles non-Markovian as well as Markovian situations

the whole tree. Actually the multi-stage Benders algorithm is much faster, using the 'Fast Forward, Fast Back' algorithm described in (Birge and Louveaux, 1997) chapter 7.

Solving multi-stage problems with Deterministic Equivalent also involves nesting, and here the nesting is in the formulation. Consider the (hypothetical) existence of scenario decision variables for every sub-path on the scenario tree connecting one node to its parent, and remember that the probability of that path is the sum of the probabilities of all paths leading through it. Assemble the constraints for all the scenarios and combine in the objective the sub-path-probability-weighted sum of all scenario objectives (too complicated to express here in mathematical terminology). This gives the implicit NA version of the DEQ. For the explicit NA version we consider separate variables for every scenario in every stage, and add all the constraints needed to equate all the variables for different scenarios that lie along the same sub-path everywhere in the tree.

2.4 Chance and Integrated Chance Constraints

In addition to multistage recourse problems described above, FortSP supports single and two-stage problems with individual chance constraints and integrated chance constraints (ICC). By a singe-stage SP problem we mean the one in which all decisions take place in the first stage and then the random parameters realise. The difference from a two-stage problem is that the latter has also a recourse action.

A probabilistic or chance constraint is a constraint that must hold with some minimum probability level. In the framework of model (2) an individual chance constraint corresponding to i-th row ($1 \leq i \leq m_2$) can be formulated as:

$$\mathrm{P}\{g_2^i \leq A_{21}^i \boldsymbol{x_1} + A_{22}^i \boldsymbol{x_2} \leq h_2^i\} \geq \alpha, \tag{8}$$

where $0 < \alpha < 1$ is a reliability level, g_2^i and h_2^i, denote i-th elements of vectors $\boldsymbol{g_2}$ and $\boldsymbol{h_2}$; A_{21}^i and A_{22}^i denote i-th rows of matrices A_{21} and A_{22}.

The chance constraint constraint (8) has the following deterministic equivalent form:

$$\begin{array}{lllll}
g_{2s}^i & \leq & A_{21s}^i \boldsymbol{x_1} + A_{22s}^i \boldsymbol{x_{2s}} + M v_s & & (s = 1, \ldots, S) \\
 & & A_{21s}^i \boldsymbol{x_1} + A_{22s}^i \boldsymbol{x_{2s}} - M w_s & \leq \quad h_{2s}^i & (s = 1, \ldots, S) \\
v_s & \leq & z_s & & (s = 1, \ldots, S) \\
w_s & \leq & z_s & & (s = 1, \ldots, S) \\
\multicolumn{3}{l}{\displaystyle\sum_{s=1}^{S} p_s z_s \leq 1 - \alpha,} & &
\end{array}$$

where M is a suitably chosen large constant, v_s, w_s and z_s are additional binary variables.

Similarly, below is the formulation of an individual ICC if g_2^i is infinite for all realisations of random parameters:

$$\mathrm{E}[(h_2^i - A_{21}^i \boldsymbol{x_1} - A_{22}^i \boldsymbol{x_2})^-] \leq \beta, \tag{9}$$

where $\beta \geq 0$ and $(a)^- := \max\{-a, 0\}$ is a negative part of $a \in \mathbb{R}$.

The ICC (9) has the following deterministic equivalent form:

$$\begin{aligned}
&A^i_{21s}\boldsymbol{x_1} + A^i_{22s}\boldsymbol{x_{2s}} - w_s \leq h^i_{2s} \quad (s = 1, \dots, S) \\
&\sum_{s=1}^{S} p_s w_s \leq \beta,
\end{aligned}$$

where w_s are additional variables. Note that in the case of integrated chance constraints introduced variables are continuous which makes deterministic equivalents of ICCs computationally more tractable than those of chance constraints.

If h^i_2 is infinite for all realisations of random parameters the constraint will look like:

$$\mathrm{E}[(A^i_{21}\boldsymbol{x_1} + A^i_{22}\boldsymbol{x_2} - g^i_2)^-] \leq \beta,$$

The case of both g^i_2 and h^i_2 finite results in a special case of a joint ICC and is not currently supported.

3 Data Provision in SMPS

3.1 SMPS Input Format

Whereas MPSX defines the format for LP data input, Stochastic MPS (SMPS) defines the input format for stochastic programming problems. FortSP implements the most important provisions of SMPS which is described by Birge et al. (1987). Since the solver is initially designed for use within the SPInE system its stochastic input fulfils the requirements of models generated by SPInE, and other provisions of SMPS are not supported in full. SPInE generates solver stochastic data in the form of discrete scenarios, and FortSP supports this form and also discrete blocks form and discrete independent form.

Three input files are required in order to specify a stochastic problem:

- Core file which is the fundamental problem template in the form of an LP problem using the MPSX format
- Time file specifying which rows and columns of the core-file belong to which time stage
- Stoch file specifying the alternative values taken by each random parameter value in the core file

The user may specify precise names for the input files or may give the basename - or 'Generic' name - of these files so that extensions are appended automatically. Denoting a base name by `<problem>` the resulting filenames are:

`<problem>.cor` for core file
`<problem>.tim` for time file
`<problem>.sto` for stoch file

3.2 Core File and Random Parameter Values

The core file expresses a linear programming problem or linear mixed integer problem in the format known as MPSX, familiar to the users of LP solvers and described in the manuals of many of such system, for example, FortMP (Ellison et al., 2008). In this format the file is divided into `ROWS`, `COLUMNS`, `RHS`, `RANGES` and `BOUNDS` sections, and data records have a fixed format as follows:

Field 1.	Positions 2-3 (code)
Field 2.	Positions 5-12 (1st name field)
Field 3.	Positions 15-26 (2nd name field)
Field 4.	Positions 30-37 (1st numeric field)
Field 5.	Positions 40-47 (3rd name field)
Field 6.	Positions 50-61 (2nd numeric field)

This layout is also used for data in the time and stoch files described below.

In the stochastic model a number of scalars will have uncertain values - they are denoted here as *random parameter values*. They may be anywhere in the core file other than in the `ROWS` section. Each random parameter value must have a representative, finite value assigned to it in the core problem, and this value must be recognisable by the input. It may not be zero in the COLUMNS or the RANGES section, or left as infinite in the BOUNDS section. However it does not have to be a value corresponding to any particular scenario.

If the time file is in implicit format then both constraints and variables must be grouped according to the stage at which they apply. These separate groups are to be in the order of time stage in the core file (constraints in the ROWS section and variables in the COLUMNS section). As a result of this ordering the constraint matrix should have a lower block triangular form, with blocks for each stage.

MIP in the form of binary or integer descriptions may be applied to any decision variable (SOS and semi-continuous are not supported). However if it applies to variables other than the first here-and-now stage then the HN model must be solved using deterministic equivalent method - Benders' decomposition would not be suitable.

3.3 Time File

The time file in implicit format specifies the first member of each stage grouping in the constraints and variables of the core problem (hence the need to group these items by stage). The first line is as follows:

Positions 1-4	Keyword TIME
Field 3 (15-22)	Problem name

This is followed by the period header line as follows:

Positions 1-7	Keyword PERIODS
Field 3 (15-22)	Keyword IMPLICIT (optional)

After this one line is included for each stage as follows:

Field 2 (5-12)	Starting variable name
Field 3 (15-22)	Starting constraint name
Field 5 (40-47)	Stage name

Time stages are indexed $1, 2, \ldots$ with stage number 1 being the first, here-and-now stage.

Finally the data ends with the following line:

Positions 1-6	Keyword ENDATA

The following is an example:

```
TIME          EXAMPLE
PERIODS       IMPLICIT
    C1        R1                       STAGE1
    C6        R3                       STAGE2
    C8        R19                      STAGE3
ENDATA
```

Note that in place of R1 it is possible to use the objective row name. The objective row is moved by the input into the first row position, wherever it is found in the data.

3.4 Stoch File

All random data and the discrete distributions are specified in the stoch file. The first line is as follows:

Positions 1-5	Keyword `STOCH`
Field 3 (15-22)	Problem name

This is followed by a header line that specifies the form of data input as follows:

Positions 1 onwards	Keyword defining the data form as one of `INDEP` `BLOCKS` `SCENARIOS` `CHANCE` `ICC`
Field 3 (15-22)	Keyword `DISCRETE`

After this header line there are data-lines as described below and the file is terminated as before:

Positions 1-6	Keyword `ENDATA`

Sample values for random parameter values are presented in the stoch file in the same form as they appear in the core file, that is:

	Random parameter value in section:			
Field	`COLUMNS`	`RHS`	`RANGES`	`BOUNDS`
1: 2-3				Bound type
2: 5-12	Column name	Vector name	Vector name	Vector name
3: 15-22	Row name	Row name	Row name	Column name
4: 25-36	Sample value	Sample value	Sample value	Sample value
5: 40-47	2nd row name	2nd row name	2nd row name	
6: 50-61	Sample value	Sample value	Sample value	

Fields 5 and 6 have a different use for `INDEP`-form data (see below), and otherwise can only be used for random parameter values with the same description in fields 1 and 2.

Stochastic Data Form `INDEP`

`INDEP` is used when each separate random parameter value has an independent distribution. Scenarios are built by selecting one possibility for each random parameter value, the set of all scenarios is then formed by taking all combinations of possible selections.

Each `INDEP` data line can describe only one random parameter value as follows:

Fields 1-4	As described above
Field 5 (40-47)	Stage name
Field 6 (50-61)	Probability value of the sample (must sum to 1 for each random parameter value)

Sequence should be according to time stage, and with separate samples of the same random parameter value collected into consecutive lines. The following is an example:

```
STOCH          EXAMPLE
INDEP          DISCRETE
    C6         OBJ        2.5            STAGE2    0.5
    C6         OBJ        3.0            STAGE2    0.5
    C6         R3         5.0            STAGE2    0.33
    C6         R3         5.5            STAGE2    0.67
    C8         R19        1.0            STAGE3    0.25
    C8         R19        2.0            STAGE3    0.25
    C8         R19        3.0            STAGE3    0.5
ENDATA
```

The above example has 12 ($2 \times 2 \times 3$) scenarios, illustrated in the following table:

	Base		Value of			
Scen	Scen	Stage	`(C6,OBJ)`	`(C6,R3)`	`(C8,R19)`	Probability
1	core	2	2.5	5.0	1.0	0.04125
2	1	3	-	-	2.0	0.04125
3	1	3	-	-	3.0	0.08250
4	1	2	-	5.5	1.0	0.08375
5	4	3	-	-	2.0	0.08375
6	4	3	-	-	3.0	0.16750
7	1	2	3.0	5.0	1.0	0.04125
8	7	3	-	-	2.0	0.04125
9	7	3	-	-	3.0	0.08250
10	7	2	-	5.5	1.0	0.08375
11	10	3	-	-	2.0	0.08375
12	10	3	-	-	3.0	0.16750

Here the Base Scen column is the scenario containing the default values for any unstated random parameter values. The first scenario is always based on the core problem and specifies values for all random parameter values. The Stage column states the stage at which a difference appears from the base.

Stochastic Data Form `BLOCKS`

The nature of this form is very similar to `INDEP`, except that individual independent random parameter values give place to independent blocks (or sets) of random parameter values. The stage number and probability distribution become properties of the block rather than the individual random parameter value. Each new block and each new set of block values is introduced with a header line as follows:

Field 1 (2-3)	Keyword `BL`
Field 2 (5-12)	Block name
Field 3 (15-22)	Stage name
Field 4 (25-36)	Probability value of the sample (must sum to 1 over the samples of each block)

Blocks with the same name should be grouped together.

Values for the members of each block are entered in a way similar to `INDEP` data, except that fields 5 and 6, not being required for stage and probability, may contain a second data entry for fields 3 and 4 as tabled above in the general stoch file description.

The following is an example:

```
STOCH          EXAMPLE
BLOCKS         DISCRETE
 BL BLOCK1     STAGE2     0.5
    C6         OBJ        2.5           R3        5.0
 BL BLOCK1     STAGE2     0.5
    C6         OBJ        3.0           R3        5.5
 BL BLOCK2     STAGE3     0.25
    C8         R19        1.0
    RHS        R19        100.0
 BL BLOCK2     STAGE3     0.25
    C8         R19        2.0
    RHS        R19        200.0
 BL BLOCK2     STAGE3     0.5
    C8         R19        3.0
ENDATA
```

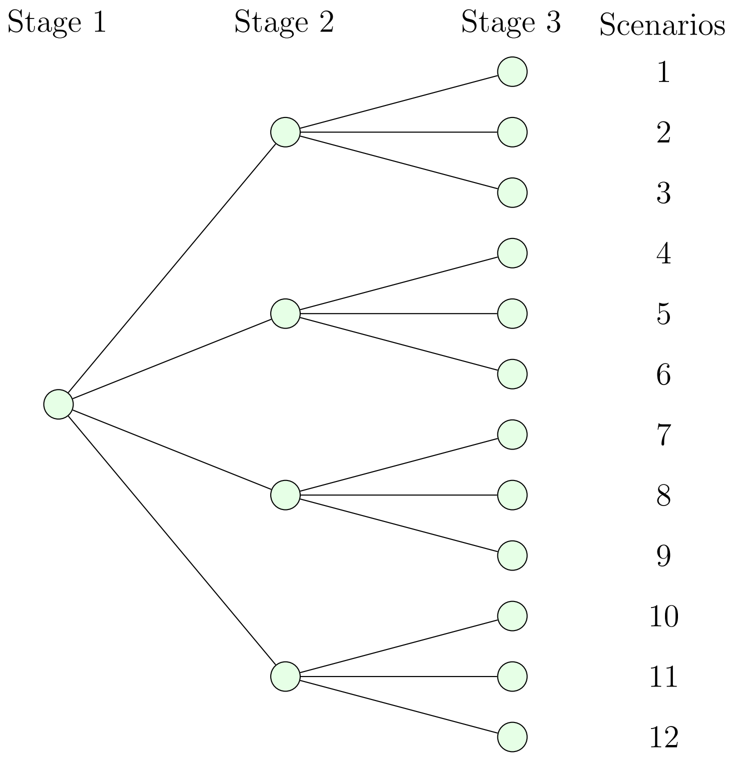
This example gives rise to scenarios in the same manner as before, illustrated as follows:

Scen	Base Scen	Stage	Value of (C6,OBJ) (C6,R3)	Value of (C8,R19) (RHS,R19)	Probability
1	Core	2	2.5 5.0	1.0 100.0	0.125
2	1	3	-	2.0 200.0	0.125
3	2	3	-	3.0 -	0.250
4	1	2	3.0 5.5	1.0 100.0	0.125
5	4	3	-	2.0 200.0	0.125
6	5	3	-	3.0 -	0.250

Note that block samples do not have to restate values in the block if they duplicate the previous sample. Hence the base for samples other than the first of a block is the previous sample. So in the above example scenarios 3 and 6 assign value 200.0 to `(RHS,R19)`.

Stochastic Data Form SCENARIOS

Scenarios have been introduced in the examples above. It may be observed that the branching of scenarios from each other (i.e. the base scenario connection) forms an event tree in which decisions may be taken at the nodes. The tree for the `INDEP` example above looks as follows:



In this diagram each scenario is represented by a full path through the nodes from left to right.

Scenario-form data is prepared from such a tree, which should be known directly or implicitly. For each scenario it is only necessary to enter the information that differs from its base scenario - that is the earlier scenario from which it branches. Where several branches issue from one node (to the right) the later scenarios may be considered as branching from any earlier scenario in that bundle. Thus for example:

Scenario 10 could branch from 1, 4 or 7

Scenario 6 could branch from 4 or 5

Scenario 1 must always branch from the core problem (and provide values for all the random parameter values).

Each scenario is preceded in the stoch file by a scenario header line as follows:

Field 1 (2-3)	Keyword `SC`
Field 2 (5-12)	Scenario name
Field 3 (15-22)	Should contain `ROOT` for scenario 1. For other scenarios enter the name of the base scenario
Field 4 (25-36)	Probability value of the scenario (must sum to 1 over all scenarios)
Field 5 (40-47)	Stage index with optional prefix

Data lines for scenarios follow the layout tabled in the general stoch file description.

Here is how the `BLOCKS` example can be presented in `SCENARIOS` form:

```
STOCH         EXAMPLE
SCENARIOS     DISCRETE
 SC SCEN1     ROOT       0.125         STAGE1
    C6        OBJ        2.5           R3         5.0
    C8        R19        1.0
    RHS       R19        100.0
 SC SCEN2     SCEN1      0.125         STAGE3
    C8        R19        2.0
    RHS       R19        200.0
 SC SCEN3     SCEN2      0.250         STAGE3
    C8        R19        3.0
 SC SCEN4     SCEN1      0.125         STAGE2
    C6        OBJ        3.0           R3         5.5
    C8        R19        1.0
    RHS       R19        100.0
 SC SCEN5     SCEN4      0.125         STAGE3
    C8        R19        2.0
    RHS       R19        200.0
 SC SCEN6     SCEN5      0.250         STAGE3
    C8        R19        3.0
ENDATA
```

3.5 CHANCE and ICC Sections

Data for chance constraints and ICC are presented on the STOCH file in additional sections preceding the random data.

`CHANCE` Section

Chance constraints can be represented in the stoch file using the `CHANCE` section where the reliability parameters α are supplied (see section 2.4). The constraints themselves are defined in the core file and the distributions of their stochastic elements are defined in extra sections of the stoch file.

After a section header consisting of a single keyword `CHANCE` in position 1 each line describes a single chance constraint and has the following structure:

Field 1 (2-3)	`L` or `G` denoting constraint sense as in the `ROWS` section
Field 2 (5-12)	Name of a group of constraint
Field 3 (15-22)	Row name
Field 4 (25-36)	Reliability parameter α, see section 2.4

Example:

```
CHANCE
 G  CC1       R1          0.95
 L  CC1       R2          0.10
```

The `CHANCE` section allows one or more groups of chance constraints to be defined. In the above example, the name of the group is `CC1`. FortSP uses the first group and ignores all others.

`ICC` Section

The `ICC` section is very similar to the `CHANCE` section. It starts with the keyword `ICC` followed by the lines in the form described below:

Field 1 (2-3)	`L` or `G` denoting constraint sense as in the `ROWS` section
Field 2 (5-12)	Name of a group of constraint
Field 3 (15-22)	Row name
Field 4 (25-36)	Parameter β for ICC, see section 2.4

Example:

```
ICC
 L  ICC1      R8          10.0
```

The `ICC` section allows one or more groups of ICCs to be defined. In the above example, the name of the group is `ICC1`. FortSP uses the first group and ignores all others.

3.6 Objective Sense

SMPS doesn't provide a way to specify the objective sense. This can be done with the following option instead.

CLI Name	`--smps-obj-sense`
SAMPL Name	`smps_obj_sense`
Description	The sense of optimisation for SMPS problems
Value	`minimize` or `maximize`
Default	`minimize`

4 Data Provision in SAMPL

The input format used in the AMPLDev modelling system is Stochastic AMPL, or SAMPL, which is described in details in *SAMPL/SPInE User Manual* (Valente et al., 2008). SAMPL is an extension of the AMPL modelling language for stochastic programming. It has many advantages over the legacy SMPS format and therefore it is recommended to use SAMPL for creating new models. The SAMPL translator fully supports FortSP and can export problem instances to SMPS for compatibility with other solvers.

4.1 SAMPL Input Format

Current version of the SAMPL translator accepts only two-stage SP problems expressed in a subset of the language. The syntax can be inferred from the example in Section 4.7. Details of modelling with SAMPL can be found elsewhere (Valente et al., 2008), this section only describes the scripting features that can be used to control FortSP and present the results.

4.2 The `solve` Statement

The `solve` statement instantiates the current problem and solves it.

Syntax

solve-stmt:
    solve ;

4.3 The `print` Statement

The `print` statement evaluates each expression in the list and prints the result to the standard output.

Syntax

print-stmt:
    print *[indexing* **:** *]* *expr-list* **;**

expr-list:
    expr
    expr-list **,** *expr*

Example: `print {p in Products}:  sell[p];`

4.4 The `write` Statement

The `write` statement writes the current problem in the SMPS form.

Syntax

write-stmt:
    write s*filename* **;**

Example: `write sout;`

4.5 The `model` and `data` Statements

The `model` and `data` statements have two forms. The one without arguments switches the current mode. For example the statement `data;` enters the data mode. The second form which takes a filename argument translates the specified file.

Syntax

model-stmt:
    model *[filename]* ;

data-stmt:
    data *[filename]* ;

4.6 The `option` Statement

The `option` statement sets and/or prints the option values.

Syntax

option-stmt:
    option fortsp_options *option-list* ;

option-list:
    option
    option-list , *option*

option:
    name [expr]

Here *name* is an option name and *expr* is an optional expression of compatible type. If the expression is not specified the option value is printed. Otherwise the expression is evaluated and the result is assigned to the option. Example: `option solver cplex, lp_alg dual;`

4.7 Example

As an example let's consider the farmer's problem from *Introduction to Stochastic Programming* (Birge and Louveaux, 1997).

A European farmer has 500 acres of land where he plans to grow wheat, corn, and sugar beets. He wants to decide how much land to devote to each crop in order to maximize profit and produce enough grain to feed his cattle. The farmer knows that at least 200 tons (T) of wheat and 240 T of corn are needed for cattle feed. All that remains after satisfying the feeding requirements is sold. Selling and purchase prices as well as planting costs are given in the following table.

	Wheat	Corn	Sugar Beets
Planting cost ($/acre)	150	230	260
Selling price ($/T)	170	150	36 under 6000 T 10 above 6000 T
Purchase price ($/T)	238	210	-
Min. requirement (T)	200	240	-

Note that sugar beet has two selling prices because the European Commission imposes a quota on its production. Any amount above the quota is sold at a lower price.

The uncertainty in the problem comes from the weather conditions that affect yields. In this problem three possible scenarios are considered. The yields in tons per acre are given below for each crop and scenario.

	Wheat	Corn	Sugar Beets
Above	2.0	2.4	16.0
Average	2.5	3.0	200.0
Below	3.0	6.0	24.0

Here is a SAMPL formulation of the model:

```
set Crops;

scenarioset Scenarios;
probability P{Scenarios};
tree Tree := twostage;

param TotalArea;               # acre
random param Yield{Crops, Scenarios}; # T/acre
param PlantingCost{Crops};     # $/acre
param SellingPrice{Crops};     # $/T
param ExcessSellingPrice;      # $/T
param PurchasePrice{Crops};    # $/T
param MinRequirement{Crops};   # T
param BeetsQuota;              # T

# Area in acres devoted to crop c
var area{c in Crops} >= 0;

# Tons of crop c sold (at favourable price in case of beets)
# under scenario s
var sell{c in Crops, s in Scenarios} >= 0, suffix stage 2;

# Tons of sugar beets sold in excess of the quota under
# scenario s
```

```
var sellExcess{s in Scenarios} >= 0, suffix stage 2;

# Tons of crop c bought under scenario s
var buy{c in Crops, s in Scenarios} >= 0, suffix stage 2;

maximize profit: sum{s in Scenarios} P[s] * (
    ExcessSellingPrice * sellExcess[s] +
    sum{c in Crops} (SellingPrice[c] * sell[c, s] -
                     PurchasePrice[c] * buy[c, s]) -
    sum{c in Crops} PlantingCost[c] * area[c]);

s.t. totalArea: sum {c in Crops} area[c] <= TotalArea;

s.t. requirement{c in Crops, s in Scenarios}:
    Yield[c, s] * area[c] - sell[c, s] + buy[c, s]
        >= MinRequirement[c];

s.t. quota{s in Scenarios}: sell['beets', s] <= BeetsQuota;

s.t. beetsBalance{s in Scenarios}:
    sell['beets', s] + sellExcess[s]
        <= Yield['beets', s] * area['beets'];
```

The data for the farmer's problem are as follows:

```
data;

set Crops := wheat corn beets;
set Scenarios := below average above;

param TotalArea := 500;

param P :=
    below   0.333333
    average 0.333333
    above   0.333333;

param Yield:
           below average above :=
    wheat    2.0     2.5   3.0
    corn     2.4     3.0   3.6
    beets   16.0    20.0  24.0;

param PlantingCost :=
    wheat 150
    corn  230
    beets 260;
```

```
param SellingPrice :=
    wheat 170
    corn  150
    beets  36;

param ExcessSellingPrice := 10;

param PurchasePrice :=
    wheat 238
    corn  210
    beets 100; # Set to a high value to simplify the objective

param MinRequirement :=
    wheat 200
    corn  240
    beets   0;

param BeetsQuota := 6000;
```

Finally the script file needs to be provided which loads model and data files, solves the problem and retrieves the optimal value and solution. Due to the flexibility of the AMPL language SAMPL is based on, it is possible to combine model, data and script in one file which can be convenient in some cases. However, in general it is not recommended since it makes more difficult to use the same model with different data sets.

Let's assume that the model and data are stored in the files `farmer.mod` and `farmer.dat`. The following script loads these files, solves the problem and prints the results:

```
# Read the model and data.
model farmer.mod;
data farmer.dat;

# Set options.
option solver fortsp;

# Instantiate and solve the problem.
solve;

# Print the results.
print 'Optimal value =', profit;
print;
print 'First-stage solution:';
print {c in Crops}: 'area[', c, '] =', area[c], '\
';
print 'totalArea =', totalArea.body;
```

Running sampl with the command `sampl <script filename>` will produce the following output:

```
...
optimal solution; objective 108390.00000000003
Optimal value = 108390.00000000003

First-stage solution:
area[ wheat ] = 170.00000000000003

area[ corn ] = 79.99999999999997

area[ beets ] = 250.00000000000003

totalArea = 500
```

5 Stochastic Programming Solution Methods (Continuous)

5.1 Deterministic Equivalent

The simplest solution approach is to formulate a deterministic equivalent of the SP problem and use a linear programming (LP) solver to optimise it. FortSP fully supports automatic formulation of deterministic equivalent problems either with implicit or with explicit non-anticipativity. This method is feasible and sometimes advantageous especially if the number of scenarios is relatively small. A more detailed explanation of the Deterministic Equivalent can be found on appendix B, section B.1.

FortSP can also formulate deterministic equivalents of two-stage problems with individual chance constraints and integrated chance constraints.

5.2 Cutting Plane (Benders)

The following decomposition algorithms are available in FortSP for solving the Here-and-Now (HN) problem:

- Benders' decomposition - L-shaped method
- Level decomposition variant
- Nested Benders' decomposition

The first two methods are applicable for two-stage problems and the last allows solving multi-stage problems. These algorithms take advantage of a specific structure of stochastic programming problems and make it possible to solve problems with large number of scenarios. The level decomposition applies a regularisation that is particularly effective for larger numbers of scenarios. For the interested reader, section B.2 on appendix B presents a more detailed mathematical background on decomposition methods.

In addition to finding here-and-now values for decision variables in the first stage the system may extend this to recourse values for the various scenarios in future stages.

For integrated chance constraints an efficient cutting-plane algorithm (Klein Haneveld and Vlerk, 2002) is provided.

In the stochastic Bender's decomposition and its variants, a variable θ is used as an appropriate approximation of the second stage value function. In certain cases the addition of optimality cuts (refer to Birge and Louveaux (1997)) creates an unbounded situation as θ is a free variable. As an ad-hoc fix for this a large negative lower bound is applied to θ, which is retained until no longer needed. If not large enough then the algorithm may halt prematurely with a cycling status. It may then be possible to obtain the correct solution by specifying a lower bound for θ with the `--theta-lower` option.

5.3 Generic L-Shaped Method

Consider the two-stage stochastic programming problem

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \mathrm{E}[Q(\boldsymbol{x},\omega)] \\ \text{subject to} & A\boldsymbol{x} \geq \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+, \end{array} \tag{10}$$

where $Q(\boldsymbol{x},\omega)$ is the value function of the recourse problem

$$\begin{array}{ll} \text{minimize} & \boldsymbol{q}(\omega)^T\boldsymbol{y} \\ \text{subject to} & T(\omega)\boldsymbol{x} + W(\omega)\boldsymbol{y} \geq \boldsymbol{h}(\omega), \\ & \boldsymbol{y} \in \mathbb{R}^{n_2}_+. \end{array} \tag{11}$$

Assume that the vector of random coefficients has finite discrete distribution with S realisations (scenarios) $\omega_1, \omega_2, \dots \omega_S$ and probability $P(\omega_i) = p_i, i = 1, 2, \dots, S$.

Let $Q(x)$ denote the objective in (10):

$$Q(x) = \boldsymbol{c}^T\boldsymbol{x} + \mathrm{E}[Q(\boldsymbol{x},\omega)]$$

This section describes a generic L-shaped algorithm that is used to implement various specific L-shaped based methods such as regularised decomposition. The extensibility is achieved by allowing to redefine certain procedures. The following algorithms were implemented based on this generic framework:

- The L-shaped method
- The multicut L-shaped algorithm
- L-shaped algorithm with regularisation by the level method
- Trust region method based on l_∞ norm
- Regularised decomposition

5.3.1 Scenario clustering

Consider the multicut version of the L-shaped method where scenarios are divided into C clusters of sizes $S_1, S_2, \dots, S_C$. The master problem at iteration k is

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{C} \theta_j \\ \text{subject to} & A\boldsymbol{x} \geq \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & F^k_j\boldsymbol{x} + \theta_j\boldsymbol{e} \geq \boldsymbol{f}^k_j, \quad j = 1, 2, \dots, C, \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+, \boldsymbol{\theta} \in \mathbb{R}^C, \end{array} \tag{12}$$

where $D^k\boldsymbol{x} \geq \boldsymbol{d}^k$ are feasibility cuts, $F^k_j\boldsymbol{x}+\theta_j\boldsymbol{e} \geq \boldsymbol{f}^k_j$ are optimality cuts and $\boldsymbol{e} = (1, 1, \dots, 1)$.

Algorithm 2 divides scenarios into clusters of approximately the same size. It takes an input parameter $r \in [0, 1]$ which denotes the cluster size relative to the number of scenarios.

Algorithm 1 Generic L-shaped method

```
choose iteration limit $k_{max} \in \mathbb{Z}_+$
choose relative stopping tolerance $\epsilon \in \mathbb{R}_+$
solve the expected value problem to get a solution $\boldsymbol{x}^0$ (initial iterate)
$k \leftarrow 0$, $Q^* \leftarrow \infty$
*initialise*()
**while** time limit is not reached **and** $k < k_{max}$ **do**
    solve the recourse problems (11) with $\boldsymbol{x} = \boldsymbol{x}^k$ and compute $Q(\boldsymbol{x}^k)$
    **if** all recourse problems are feasible **then**
        add $C$ optimality cuts
        **if** $Q(\boldsymbol{x}^k) < Q^*$ **then**
            $Q^* \leftarrow Q(\boldsymbol{x}^k)$
            $\boldsymbol{x}^* \leftarrow \boldsymbol{x}^k$
        **end if**
    **else**
        add a feasibility cut
    **end if**
    *get-next-iterate*()
    $k \leftarrow k + 1$
**end while**
```

Here *initialise* and *get-next-iterate* are procedures to be defined by specific methods.

Algorithm 2 Scenario clustering

```
$s \leftarrow 0$, $i \leftarrow 1$
**while** $s < S$ **do**
    $S_i \leftarrow \left\lceil i \max\left(\frac{S}{\lceil 1/r - 0.5 \rceil}, 1\right) - s - 0.5 \right\rceil$
    $s \leftarrow s + S_i$
    $i \leftarrow i + 1$
**end while**
```

5.4 The L-shaped method

When the relative cluster size $r = 1$, which is the default, there is only one cluster of size S resulting in the original L-shaped method of Van Slyke and Wets (1969). $r = 0$ results in the multicut version (Birge and Louveaux, 1988) with each cluster consisting of a single scenario. Intermediate values are also possible, for example if $S = 7$ and $r = \frac{1}{3}$, then scenarios will be divided into 3 clusters of sizes 2, 3 and 2.

To get the classic L-shaped method the *initialise* and *get-next-iterate* procedures are defined as shown in Algorithms 3 and 4 respectively. The second check of optimality condition in Algorithm 4 ensures that the recouse problems are not solved unnecessarily when the optimality gap became acceptable due to increase in the lower bound.

Algorithm 3 *initialise* (original L-shaped)

```
$Q^0 \leftarrow -\infty$
```

Algorithm 4 *get-next-iterate* (original L-shaped)

```
if $(Q^* - Q^k)/(|Q^*| + 10^{-10}) \le \epsilon$ then
    stop
end if
solve the master problem (12) to get an optimal solution $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ and the optimal objective value $Q^{k+1}$; $\boldsymbol{x}^{k+1}$ is the next iterate
if $(Q^* - Q^{k+1})/(|Q^*| + 10^{-10}) \le \epsilon$ then
    stop
end if
```

5.5 Regularisations

In the following subsections we briefly discuss variations of the L-Shaped method.

5.5.1 L-shaped method with regularisation by the level method

Regularisation by the level method (Lemaréchal et al., 1995; Fábián and Szőke, 2007) is achieved by defining the *initialise* and *get-next-iterate* procedures according to Algorithms 5 and 6. For more details see appendix section B.5

Algorithm 5 *initialise* (level regularisation)

```
choose $\lambda \in (0, 1)$
$Q^0 \leftarrow -\infty$
```

5.5.2 Trust region method based on the infinity norm

Algorithms 7 and 8 show definitions of the procedures for the l_∞ trust region method (Linderoth and Wright, 2003). See section B.6 for more details.

5.5.3 Regularised decomposition

Algorithms 9 and 10 show definitions of the procedures for the regularised decomposition method with dynamic adaptation of σ as described by Ruszczyński and Świętanowski (1997). This method is explained is section B.4

Algorithm 6 *get-next-iterate* (level regularisation)

```
if $(Q^* - Q^k)/(|Q^*| + 10^{-10}) \leq \epsilon$ then
    stop
end if
solve the master problem (12) to get an optimal solution $(\boldsymbol{x}', \boldsymbol{\theta}')$ and the optimal objective value $Q^{k+1}$.
if $(Q^* - Q^{k+1})/(|Q^*| + 10^{-10}) \leq \epsilon$ then
    stop
end if
solve the projection problem:
```

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{x} - \boldsymbol{x}'\|^2 \\ \text{subject to} & \boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{C} \theta_j \leq (1-\lambda)Q^{k+1} + \lambda Q^* \\ & A\boldsymbol{x} \geq \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & F_j^k\boldsymbol{x} + \theta_j\boldsymbol{e} \geq \boldsymbol{f}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \end{array} \tag{13}$$

```
let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the projection problem; then $\boldsymbol{x}^{k+1}$ is the next iterate
```

Algorithm 7 *initialise* (trust region method)

```
choose $\xi \in (0, 1/2)$ and maximum trust region radius $\Delta_{hi} \in [1, \infty)$
choose initial radius $\Delta \in [1, \Delta_{hi}]$
counter $\leftarrow 0$
$\hat{Q} \leftarrow \infty$
```

Algorithm 8 *get-next-iterate* (trust region method)

```
if $\hat{Q} < \infty$ then
  if $\hat{Q} - Q(\boldsymbol{x}^k) \geq \xi(\hat{Q} - Q^k)$ then
    if $\hat{Q} - Q(\boldsymbol{x}^k) \geq 0.5(\hat{Q} - Q^k)$ and $\|\boldsymbol{x^k} - \hat{\boldsymbol{x}}\|_\infty = \Delta$ then
      increase the radius:
      $\Delta \leftarrow \min(2\Delta, \Delta_{hi})$
    end if
    set a reference point:
    $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
    $\hat{Q} \leftarrow Q(\hat{\boldsymbol{x}})$
    counter $\leftarrow 0$
  else
    $\rho \leftarrow -\min(1, \Delta)(\hat{Q} - Q(\boldsymbol{x}^k))/(\hat{Q} - Q^k)$
    if $\rho > 0$ then
      counter $\leftarrow$ counter $+ 1$
    end if
    if $\rho > 3$ or (counter $\geq 3$ and $\rho \in (1, 3]$) then
      decrease the radius:
      $\Delta \leftarrow \Delta / \min(\rho, 4)$
      counter $\leftarrow 0$
    end if
  end if
else
  set a new reference point:
  $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
  $\hat{Q} \leftarrow Q(\hat{\boldsymbol{x}})$
end if
solve the master problem with additional trust region bounds:
```

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \displaystyle\sum_{j=1}^{C} \theta_j \\ \text{subject to} & A\boldsymbol{x} \geq \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & F_j^k\boldsymbol{x} + \theta_j\boldsymbol{e} \geq \boldsymbol{f}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \\ & \hat{\boldsymbol{x}} - \Delta \leq \boldsymbol{x} \leq \hat{\boldsymbol{x}} + \Delta. \end{array} \tag{14}$$

```
let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the problem (14) and $Q^{k+1}$ be its optimal value
if $|\hat{Q} - Q^{k+1}|/(|\hat{Q}| + 10^{-10}) \leq \epsilon$ then
  stop
end if
```

Algorithm 9 *initialise* (regularised decomposition)

```
$\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^0$
$Q^0 \leftarrow \infty$
choose $\sigma$ and $\gamma$
```

Algorithm 10 *get-next-iterate* (regularised decomposition)

```
if $k = 0$ or $|Q(x^k) - Q^k|/(|Q(x^k)| + 10^{-10})$ then
  set a new reference point:
  $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}^k$
  $\hat{Q} \leftarrow Q(\hat{\boldsymbol{x}})$
end if
if $Q(\boldsymbol{x}^k) < \infty$ then
  if $Q(\boldsymbol{x}^k) > \gamma\hat{Q} + (1-\gamma)Q^k$ then
    $\sigma \leftarrow \sigma/2$
  else if $Q(\boldsymbol{x}^k) < (1-\gamma)\hat{Q} + \gamma Q^k$ then
    $\sigma \leftarrow 2\sigma$
  end if
end if
solve the master problem with an additional quadratic term in the objective:
```

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{C}\theta_j + \frac{1}{2\sigma}\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 \\ \text{subject to} & A\boldsymbol{x} \geq \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & F_j^k\boldsymbol{x} + \theta_j\boldsymbol{e} \geq \boldsymbol{f}_j^k, \quad j = 1, 2, \ldots, C, \\ & \boldsymbol{x} \in \mathbb{R}_+^{n_1}, \boldsymbol{\theta} \in \mathbb{R}^C, \end{array} \tag{15}$$

```
let $(\boldsymbol{x}^{k+1}, \boldsymbol{\theta}^{k+1})$ be an optimal solution of the problem (15) and $Q^{k+1} = \boldsymbol{c}^T\boldsymbol{x}^{k+1} + \sum_{j=1}^{C}\theta_j^{k+1}$
if $|\hat{Q} - Q^{k+1}|/(|\hat{Q}| + 10^{-10}) \leq \epsilon$ then
  stop
end if
delete constraints that have corresponding dual variables zero in the solution of (15),
keeping the last $C$ added constraints intact
```

5.6 Remarks

Preliminary experiments showed that keeping all the cuts in regularised decomposition, while resulting in smaller number of iterations, increases overall solution time due to master problem becoming much more difficult to solve. At the same time the trust region method based on l_∞ norm is exposed to this issue to a less extent and therefore no cut deletion was done.

The default relative stopping tolerance $\epsilon = 10^{-5}$ is used for the L-shaped method with and without regularisation by the level method. The stopping criteria in trust region algorithm and regularised decomposition are different because these methods do not provide global lower bound. Therefore ϵ is set to a lower value of 10^{-6} for them.

Default values for other parameters are listed below.

- L-shaped algorithm with regularisation by the level method:
  $\lambda = 0.5$,
- Trust region method based on l_∞ norm:
  $\Delta = 1, \Delta_{hi} = 10^3, \xi = 10^{-4}$,
- Regularised decomposition:
  $\sigma = 1, \gamma = 0.9$.

6 Stochastic Integer Programming Solution Methods

6.1 Deterministic Equivalent

The simplest solution approach is to formulate a deterministic equivalent of the SIP problem and use a Integer Programming (IP) solver to optimise it via a generic Branch-and-Bound algorithm. FortSP fully supports automatic formulation of deterministic equivalent problems either with implicit or with explicit non-anticipativity. This method is feasible and sometimes advantageous especially if the number of scenarios is relatively small.

FortSP can also formulate deterministic equivalents of two-stage problems with individual chance constraints and integrated chance constraints.

6.2 Integer L-Shaped Method

The Integer L-shaped method is an algorithm for solving two-stage SIP problems with complete recourse, which is similar to the continuous L-Shaped method for stochastic linear programming. The method operates on the current problem based on the first-stage subproblem with added feasibility and optimality cuts. The current problem at iteration k can be defined as follows:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} + \theta \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \\ & D^k\boldsymbol{x} \geq \boldsymbol{d}^k, \\ & E^k\boldsymbol{x} + \theta \geq \boldsymbol{e}^k \\ & \boldsymbol{x} \in \mathbb{R}^{n_1}_+, \boldsymbol{\theta} \in \mathbb{R}, \end{array} \tag{16}$$

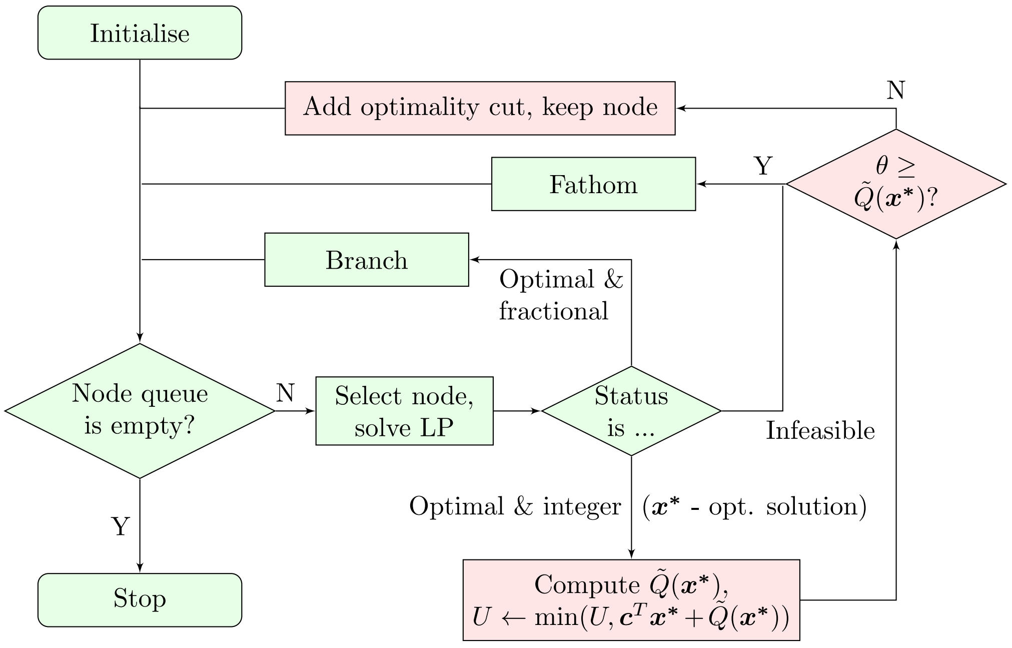
where $D^k\boldsymbol{x} \geq \boldsymbol{d}^k$ are the feasibility cuts and $E^k\boldsymbol{x} + \theta \geq \boldsymbol{e}^k$ are the optimality cuts. The main difference of this algorithm from its continuous counterpart is branching on the first-stage binary variables which results in a branch and cut procedure. As in the continuous L-shaped method, the second-stage subproblems are solved only during the evaluation of the expected recourse function when computing the optimality cuts. In this way both decomposition of stages and decomposition of scenarios are achieved.

The integer L-shaped method is applicable to a wide range of SIP problems with binary first stage, continuous second stage and complete fixed recourse. A restricted class of problems with discrete second-stage variables is supported. The flowchart in Figure 4 outlines the Integer L-Shaped method.

6.3 VNDS Heuristic

Variable Neighbourhood Decomposition Search (VNDS) is a two-level variant of the variable neighbourhood search heuristic, based upon the decomposition of the problem. Introducing neighbourhood structures into the solution space of a SIP makes the application of VNDS possible as a solution method for the first stage problem. The heuristic result can be used to speed up the overall solution process for a given two-stage SIP problem.

By fixing a portion of variables in the first-stage problem of a given two-stage SIP problem, an easier problem is obtained, which can usually be solved faster than the original problem. Ideally, if all the variables from the first-stage problem are fixed, then only solving the



Initialise
Add optimality cut, keep node
N
Y
$\theta \geq \tilde{Q}(\boldsymbol{x}^*)$?
Fathom
Branch
Optimal & fractional
Node queue is empty?
N
Select node, solve LP
Status is ...
Infeasible
Y
Optimal & integer
($\boldsymbol{x}^*$ - opt. solution)
Stop
Compute $\tilde{Q}(\boldsymbol{x}^*)$, $U \leftarrow \min(U, \boldsymbol{c}^T\boldsymbol{x}^* + \tilde{Q}(\boldsymbol{x}^*))$

Figure 4: Flowchart of the integer L-shaped method within a branch and cut framework

second-stage problems of a specific block-structure remains. By systematically changing the number of variables to be fixed, a VNDS scheme for two-stage SIP problem is obtained. The general pseudo-code of the VNDS heuristic for the two-stage SIP problem is given in Algorithm 11. Input parameters for the algorithm are an input two-stage SIP problem P and initial integer feasible solution $\boldsymbol{x}^*$.

Algorithm 11 VNDS for two-stage SIP

```
1: Given: SIP problem instance $P$, integer feasible solution $\boldsymbol{x}^*$,
   stopping tolerance $\varepsilon$
2: $L \leftarrow -\infty, \quad U \leftarrow f(\boldsymbol{x}^*)$
3: while time limit is not reached do
4:    Solve the LP relaxation of $P$ to obtain the solution $\bar{\boldsymbol{x}}$
5:    $L \leftarrow f(\bar{\boldsymbol{x}})$
6:    if $\bar{\boldsymbol{x}} \in \{0,1\}^{r_1} \times \mathbb{R}_+^{n_1-r_1}$ then
7:       $\boldsymbol{x}^* \leftarrow \bar{\boldsymbol{x}}, \quad U \leftarrow L$
8:       stop
9:    end if
10:   $\Delta_j = |x_j^* - \bar{x}_j|, \quad j = 1, \dots, r_1$
11:   Index $x_j$ so that $\Delta_j \leq \Delta_{j+1}, \quad j = 1, \dots, r_1 - 1$
12:   $k \leftarrow r_1$
13:   while time limit is not reached and $k \geq 0$ do
14:      $J_k \leftarrow \{1, \dots, k\}$
15:      Add constraint $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) = 0$ to $P$
16:      Solve the problem $P$ to obtain the solution $\boldsymbol{x}'$
17:      if optimal solution found or problem is infeasible then
18:         Replace the last added constraint with $\Delta(J_k, \boldsymbol{x}^*, \boldsymbol{x}) \geq 1$
19:      else
20:         Delete the last added constraint
21:      end if
22:      if $f(\boldsymbol{x}') < f(\boldsymbol{x}^*)$ then
23:         $\boldsymbol{x}^* \leftarrow \boldsymbol{x}', \quad U \leftarrow f(\boldsymbol{x}')$
24:         if $|U - L| \leq \varepsilon |U|$ then
25:            stop
26:         end if
27:      end if
28:      $k \leftarrow k - 1$
29:   end while
30: end while
```

7 Computing Stochastic Measures

7.1 Ancillary Algorithms - EV and WS

The system may also evaluate the following special problems:

- Expected value (EV) problem, which assumes that all data will take their expected values.
- Wait-and-see (WS) problem, which is obtained by solving a separate sub-problem for each scenario assuming that all random data is already known. The final WS solution is the probability-weighted average of these solutions for each scenario.

Each special problem can be evaluated in addition to the main HN problem. When the calculation of stochastic measures are required, the EV and/or the WS problems will be solved as needed, whether or not a corresponding option has been set. See section 7.2 below.

7.2 Stochastic Measures - EVPI and VSS

The expected value of perfect information (EVPI) is computed as the difference between the optimal values of the wait-and-see (WS) and the here-and-now (HN) problems. Therefore the EVPI option implies the solution of both the HN and WS problems.

In order to calculate the value of the stochastic solution (VSS), we need to know the expectation of the expected value solution (EEV). EEV is calculated by solving the EV problem, fixing the obtained solution in the WS sub-problems, and computing the probability weighted objective value. Hence the VSS option implies the solution of all three classes of problems, HN, EV and WS, whether or not it is specified in the corresponding option.

7.3 Algorithm Controls and Options

Options for SP algorithms are as follows:

CLI Name	`--solve-hn`
SAMPL Name	`solve_hn`
Description	Flag specifying whether to solve the here-and-now problem
Value	0 or 1
Default	1
CLI Name	`--solve-ev`
SAMPL Name	`solve_ev`
Description	Flag specifying whether to solve the expected value problem
Value	0 or 1
Default	0
CLI Name	`--solve-ws`
SAMPL Name	`solve_ws`
Description	Flag specifying whether to solve the wait-and-see problem
Value	0 or 1
Default	0

CLI Name	`--compute-evpi`
SAMPL Name	`compute_evpi`
Description	Flag specifying whether to compute the expected value of perfect information (EVPI) The expected value of perfect information requires the solution of both HN and WS models. Setting this option to 1 implies both `--solver-hn` and `--solve-ws` to be set. EVPI is the absolute difference between the HN and WS solution objectives.
Value	`0` or `1`
Default	`0`

CLI Name	`--compute-vss`
SAMPL Name	`compute_vss`
Description	Flag specifying whether to compute the value of the stochastic solution (VSS)[2]
Value	`0` or `1`
Default	`0`

CLI Name	`--vss-fstage`
SAMPL Name	`vss_fstage`
Description	Flag specifying whether to fix only the first stage when computing the value of the stochastic solution[2]
Value	`0` or `1`
Default	`0`

CLI Name	`--sp-alg`
SAMPL Name	`sp_alg`
Description	Stochastic programming algorithm to be used
Value	The possible values for this option are listed in the table below.

Name	Description
`auto`	The algorithm is chosen automatically (default)
`deteq`	The deterministic equivalent problem with implicit non-anticipativity is constructed and solved
`deteqx`	The deterministic equivalent problem with explicit non-anticipativity constraints is constructed and solved
`benders`	Benders' decomposition
`level`	Variant of level decomposition
`trustregion`	The trust region method by Linderoth and Wright (2003)
`rd`	Regularised decomposition

The default algorithm - `auto` - chosen by the system - is Benders' decomposition for two-stage recourse problems and deterministic equivalent with implicit nonanticipativity for other problems. An exception to this is single-stage ICC problems, which by default are solved with the special cutting plane algorithm. All other CC and ICC problems are solved only with deterministic equivalent.

Default	`auto`

CLI Name	`--time-limit`
SAMPL Name	`time_limit`
Description	Time limit in seconds
Value	Nonnegative number
Default	`3600`

Options for Benders' decomposition:

CLI Name	`--ben-pp-expval`
SAMPL Name	`ben_pp_expval`
Description	Flag specifying whether to obtain the initial first stage solution by solving the EV problem
Value	`0` or `1`
Default	`1`

[2]VSS - Value of Stochastic Solution - requires the solution of both HN and EV models. Setting this option to 1 forces both `--solver-hn` and `--solve-ws` to be set. In order to calculate VSS we need to know the EEV - expected value of the expected value solution. EEV is calculated by solving the EV model, fixing the result so obtained in all the WS models (all stages but the last), which are then solved to give a probability-weighted average value for the objective - which is the VSS. Option `--vss-fstage` can be used to restrict the fix that is performed to first stage variables only (although in theory this is not correct, the theoretical result is often meaningless as a complete fix may be infeasible).

CLI Name	`--ben-fffb`
SAMPL Name	`ben_fffb`
Description	Flag specifying whether to use fast forward, fast back method for multi-stage
Value	`0` or `1`
Default	`0`

CLI Name	`--ben-theta-lower`
SAMPL Name	`ben_theta_lower`
Description	Lower bound for θ used when necessary to avoid unbounded situations. In certain cases the addition of optimality cuts creates an unbounded situation as θ is a free variable. As an ad-hoc fix for this, a large negative lower bound is applied to θ, which is retained until no longer needed. If not large enough then the Benders algorithm may halt prematurely with a cycling status. It may then be possible to obtain a correct solution by specifying a lower value for this option, for example -1000000.
Value	Number
Default	`-100000`

CLI Name	`--ben-cut-factor`
SAMPL Name	`ben_cut_factor`
Description	Maximum cuts per child scenario
Value	Integer
Default	`20`

CLI Name	`--ben-max-iter`
SAMPL Name	`ben_max_iter`
Description	Iteration limit for Benders' decomposition
Value	Nonnegative integer
Default	`10000`

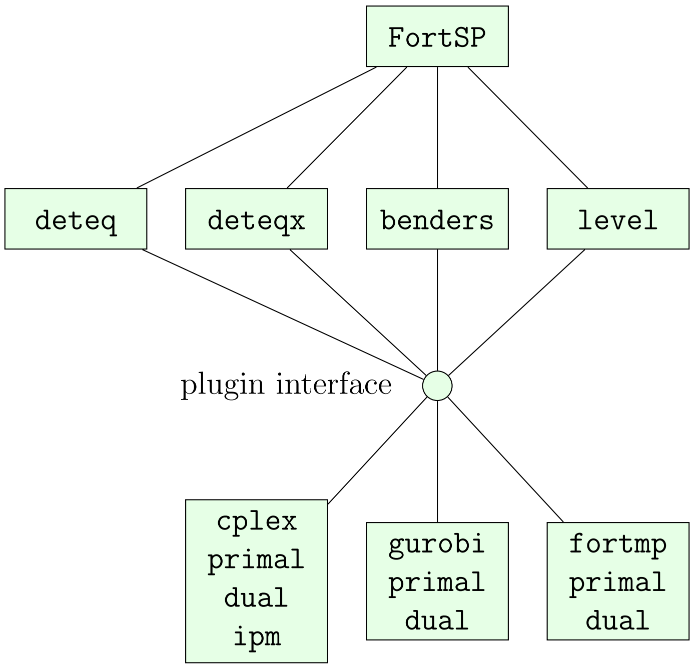


Figure 5: FortSP algorithms and solvers

8 External Solvers

8.1 Solvers Available

FortSP has a powerful plug-in system that allows to connect it to third-party LP, QP and MIP solvers. Plugin-in is a dynamic library (DLL on Windows, shared object on Unix) that provides an implementation of the plug-in interface. Currently FortSP is connected to the following solvers: CPLEX, FortMP and Gurobi. The external solver can be selected using the `--solver` option which takes on the solver name (lowercase) as a value, for example, `--solver=gurobi`.

Figure 5 illustrates which combinations of algorithms and plug-ins are supported in FortSP. Compatible modules are connected by arcs so, for instance, it is possible to solve deterministic equivalent problems with any solver and LP algorithm.

8.2 Solver Options and Controls

Options for LP or QP solver execution are as follows:

CLI Name	`--solver`
SAMPL Name	`solver`
Description	External solver name
Value	String
Default	`fortmp (windows) or cplex (linux/mac)`

CLI Name	`--lp-alg`
SAMPL Name	`lp_alg`
Description	This option specifies which LP algorithm should be used to solve a deterministic equivalent problem and all linear programming subproblems that are constructed in the course of solving the SP problem.
Value	The possible values for this option are listed in the table below.

Name	Description
`auto`	The algorithm is chosen automatically (default)
`primal`	Primal simplex method
`dual`	Dual simplex method
`ipm`	Interior point method

Default	`auto`

CLI Name	`--basis-restart`
SAMPL Name	`basis_restart`
Description	Flag specifying whether to use warm start
Value	`0` or `1`
Default	`1`

CLI Name	`--use-fortmp-specs`
SAMPL Name	`use_fortmp_specs`
Description	Flag specifying whether to use extra SPECS-command file (only with the FortMP solver.) A SPECS command file with the name `fortmp.spc` may be used to refine the options when FortMP is the solver in use. See the FortMP manual (Ellison et al., 2008). Commands are to be provided in sections corresponding to the type of sub-problem that is being solved, according to the following table:

Section ID	Description
ALL	Section that applies to every call to the solver. Must appear first in the SPECS file.
DeqImna	Section to handle Deterministic Equivalent - Implicit NA
DeqExna	Section to handle Deterministic Equivalent - Explicit NA
ExpVal	Section to handle Expected Value solutions
Wsprob	Section to handle Wait and See scenario sub-problems
BendRoot	Section to handle Benders root-node sub-problem solutions (multi-stage)
BendNode	Section to handle Benders node sub-problem solutions other that root or leaf (multi-stage)
BendLeaf	Section to handle Benders leaf sub-problem solutions with no warm restart (multi-stage)
BenRLeaf	Section to handle Benders leaf sub-problem solutions with warm restart (multi-stage)
Ben2Mast	Section to handle Benders master-problem solutions (two-stage)
Ben2Sprb	Section to handle Benders sub-problem solutions (two-stage)
LevelQP	Section to handle Benders Level-method QP solutions (two-stage)

The section ID is named in a `BEGIN` line - e.g. `BEGIN (DeqImna)` - which is followed by the SPECS commands for that section. Each section is terminated with a line `END`.

Value	0 or 1
Default	0

9 Solution File and Logging

Solution file gives the model-type solutions that are requested with status and values for both primal and dual solutions. This is limited by default to values for the first stage only, extendable to further stages by option.

Diagnostics of any unusual events or errors occurring is reported to the console. The log file gives solver messages that are normally not of interest to the user and therefore logging is disabled by default.

9.1 Output Controls and Options

Options for solution output and logging are as follows:

CLI Name	`--sol-file`
SAMPL Name	`sol_file`
Description	Actual name of the solution file. If the filename is empty solution output is disabled.
Value	String
Default	

CLI Name	`--sol-include-second-stage`
SAMPL Name	`sol_include_second_stage`
Description	Include information for the 2nd stage solution if it is available. The information will be appended at the end of the solution file. Currently the option `--sol-format` must be set to `'ampl'` for the information to be written in the output file.
Value	`0` or `1`
Default	`0`

CLI Name	`--log-file`
SAMPL Name	`log_file`
Description	Actual name of the log file. If the filename is empty logging is disabled.
Value	String
Default	

CLI Name	`--ben-log-print`
SAMPL Name	`ben_log_print`
Description	Code for items to be logged (Benders multi-stage only) Additional logged output can be generated with the option `--ben-log-print`. This should be used with caution as the log-file can easily be swamped. Certain values of use are: • 3 - for solution status of each node (plus the default) • 19 - for details of the node tree (plus the above) • 95 - for description of every cut applied (plus the above) with option 3 the output volume may be reduced by specifying `--ben-log-freq` - that is the interval to leave between node solution-status logs.
Value	Integer
Default	`1`

CLI Name	`--ben-log-freq`
SAMPL Name	`ben_log_freq`
Description	Logged every this number of passes (Benders multi-stage only)
Value	Integer
Default	`1`

9.2 Solution Format for Second Stage

By default, information is included for the first stage solution. If the user explicitly selects the corresponding option, information for the second stage will be appended (if available) at the end of the solution file. Second stage solution will be in an AMPL solution format for suffixes as follows:

```
suffix <kind> <n> <namelen> <tablen> <tablines>
<sufname>
<index 1> <value 1>
<index 2> <value 2>
...
<index n> <value n>
```

Field *kind* is the type of data being represented (0 - primal variable suffix, 1 - dual variable suffix), *n* is the number of values (for example, number of second stage variables), *namelen* is a length of the suffix name (the *sufname* string) + 1, *tablen* and *tablines* are irrelevant and will be equal to zero. These values are maintained for AMPL compatibility purposes. Field *index i* is a zero-based index of an item (primal or dual variable), *value i* is the value associated with an item at *index i*.

Both scenario indices and types of values are enconded in suffix names (represented by *sufname*). For example, primal variables suffix names are enconded in the form *v<scenario i>*, where v stands for value *scenario i* is a zero-based scenario index. Similarly for constraints (dual variables) suffix names can be of the form d<*scenario i*> with d standing for dual.

References

Ariyawansa, K. A. and Felt, A. J. (2004). On a new collection of stochastic linear programming test problems. *INFORMS Journal on Computing*, 16(3):291–299.

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252. Re-publised in *Computational Management Science 2* (2005), 3–19.

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

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

Birge, J. R. and Louveaux, F. V. (1997). *Introduction to Stochastic Programming*. Springer-Verlag, New York.

Colombo, M. and Gondzio, J. (2008). Further development of multiple centrality correctors for interior point methods. *Computational Optimization and Applications*, 41:277–305.

Dantzig, G. B. and Madansky, A. (1961). On the solution of two-stage linear programs under uncertainty. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 165–176. University of California Press, Berkeley.

Dantzig, G. B. and Wolfe, P. (1960). The decomposition principle for linear programs. *Operations Research*, 8:101–111.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

Ellison, E. F. D., Hajian, M., Jones, H., Levkovitz, R., Maros, I., Mitra, G., and Sayers, D. (2008). *FortMP Manual*. Brunel University: London, Numerical Algorithms Group: Oxford. http://www.optirisk-systems.com/manuals/FortmpManual.pdf.

Fábián, C. I. (2000). Bundle-type methods for inexact data. *Central European Journal of Operations Research*, 8:35–55. Special issue, T. Csendes and T. Rapcsák, eds.

Fábián, C. I. and Szőke, Z. (2007). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, 4:313–353.

Gassmann, H. (1990). MSLiP: a computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, 47:407–423.

Gondzio, J. (1995). Hopdm (version 2.12) a fast lp solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225.

Holmes, D. (1995). A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS). http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html.

Kall, P. and Mayer, J. (1998). On testing SLP codes with SLP-IOR. In *New Trends in Mathematical Programming: Homage to Steven Vajda*, pages 115–135. Kluwer Academic Publishers.

Kiwiel, K. C. (1985). *Methods of descent for nondifferentiable optimization.* Springer-Verlag, Berlin, New York.

Klein Haneveld, W. K. and Vlerk, M. H. (2002). Integrated chance constraints: reduced forms and an algorithm. Technical report, University of Groningen, Research Institute SOM (Systems, Organisations and Management).

König, D., Suhl, L., and Koberstein, A. (2007). Optimierung des Gasbezugs im liberalisierten Gasmarkt unter Berücksichtigung von Röhren- und Untertagespeichern. In *Sammelband zur VDI Tagung "Optimierung in der Energiewirtschaft" in Leverkusen.*

Lemaréchal, C. (1978). Nonsmooth optimization and descent methods. *Research Report* 78-4, IIASA, Laxenburg, Austria.

Lemaréchal, C., Nemirovskii, A., and Nesterov, Y. (1995). New variants of bundle methods. *Mathematical Programming*, 69:111–147.

Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250.

Nemirovski, A. (2005). Lectures in modern convex optimization. ISYE, Georgia Institute of Technology.

Oliveira, W., Sagastizábal, C., and Scheimberg, S. (2011). Inexact bundle methods for two-stage stochastic programming. *SIAM Journal on Optimization*, 21:517–544.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877–898.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333.

Ruszczyński, A. (2003). Decomposition methods. In Ruszczyński, A. and Shapiro, A., editors, *Stochastic Programming, Handbooks in Operations Research and Management Science*, volume 10, pages 141–211. Elsevier, Amsterdam.

Ruszczyński, A. (2006). *Nonlinear Optimization.* Princeton University Press.

Ruszczyński, A. and Świętanowski, A. (1997). Accelerating the regularized decomposition method for two-stage stochastic linear problems. *European Journal of Operational Research*, 101:328–342.

Valente, C., Mitra, G., Sadki, M., and Fourer, R. (2009). Extending algebraic modelling languages for stochastic programming. *Informs Journal on Computing*, 21(1):107–122.

Valente, P., Mitra, G., Poojari, C., Ellison, E. F., Di Domenica, N., Mendi, M., and Valente, C. (2008). *SAMPL/SPInE User Manual.* OptiRisk Systems. http://www.optirisk-systems.com/manuals/SpineAmplManual.pdf.

Van Slyke, R. and Wets, R. J. B. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17:638–663.

Zverovich, V., Fábián, C. I., Ellison, F., and Mitra, G. (2012). A computational study of a solver system for processing two-stage stochastic lps with enhanced benders decomposition. *Mathematical Programming Computation*, 4(3):211–238.

APPENDICES

A Option Summary

The following is a complete list of options.

SMPS Input Options

CLI Name	`--smps-obj-sense`
SAMPL Name	`smps_obj_sense`
Description	The sense of optimisation for SMPS problems
Value	`minimize` or `maximize`
Default	`minimize`

Algorithm Options

CLI Name	`--solve-hn`
SAMPL Name	`solve_hn`
Description	Flag specifying whether to solve the here-and-now problem
Value	`0` or `1`
Default	`1`
CLI Name	`--solve-ev`
SAMPL Name	`solve_ev`
Description	Flag specifying whether to solve the expected value problem
Value	`0` or `1`
Default	`0`
CLI Name	`--solve-ws`
SAMPL Name	`solve_ws`
Description	Flag specifying whether to solve the wait-and-see problem
Value	`0` or `1`
Default	`0`
CLI Name	`--compute-evpi`
SAMPL Name	`compute_evpi`
Description	Flag specifying whether to compute the expected value of perfect information (EVPI) The expected value of perfect information requires the solution of both HN and WS models. Setting this option to 1 implies both `--solver-hn` and `--solve-ws` to be set. EVPI is the absolute difference between the HN and WS solution objectives.
Value	`0` or `1`
Default	`0`

CLI Name	`--compute-vss`
SAMPL Name	`compute_vss`
Description	Flag specifying whether to compute the value of the stochastic solution (VSS)[3]
Value	0 or 1
Default	0

CLI Name	`--vss-fstage`
SAMPL Name	`vss_fstage`
Description	Flag specifying whether to fix only the first stage when computing the value of the stochastic solution[3]
Value	0 or 1
Default	0

CLI Name	`--sp-alg`
SAMPL Name	`sp_alg`
Description	Stochastic programming algorithm to be used
Value	The possible values for this option are listed in the table below.

Name	Description
`auto`	The algorithm is chosen automatically (default)
`deteq`	The deterministic equivalent problem with implicit non-anticipativity is constructed and solved
`deteqx`	The deterministic equivalent problem with explicit non-anticipativity constraints is constructed and solved
`benders`	Benders' decomposition
`level`	Variant of level decomposition
`trustregion`	The trust region method by Linderoth and Wright (2003)
`rd`	Regularised decomposition

The default algorithm - `auto` - chosen by the system - is Benders' decomposition for two-stage recourse problems and deterministic equivalent with implicit nonanticipativity for other problems. An exception to this is single-stage ICC problems, which by default are solved with the special cutting plane algorithm. All other CC and ICC problems are solved only with deterministic equivalent.

Default	`auto`

CLI Name	`--time-limit`
SAMPL Name	`time_limit`
Description	Time limit in seconds
Value	Nonnegative number
Default	3600

[3]VSS - Value of Stochastic Solution - requires the solution of both HN and EV models. Setting this option to 1 forces both `--solver-hn` and `--solve-ws` to be set. In order to calculate VSS we need to

Options for Benders' Decomposition

CLI Name	`--ben-pp-expval`
SAMPL Name	`ben_pp_expval`
Description	Flag specifying whether to obtain the initial first stage solution by solving the EV problem
Value	`0` or `1`
Default	`1`

CLI Name	`--ben-fffb`
SAMPL Name	`ben_fffb`
Description	Flag specifying whether to use fast forward, fast back method for multi-stage
Value	`0` or `1`
Default	`0`

CLI Name	`--ben-theta-lower`
SAMPL Name	`ben_theta_lower`
Description	Lower bound for θ used when necessary to avoid unbounded situations. In certain cases the addition of optimality cuts creates an unbounded situation as θ is a free variable. As an ad-hoc fix for this, a large negative lower bound is applied to θ, which is retained until no longer needed. If not large enough then the Benders algorithm may halt prematurely with a cycling status. It may then be possible to obtain a correct solution by specifying a lower value for this option, for example -1000000.
Value	Number
Default	`-100000`

CLI Name	`--ben-cut-factor`
SAMPL Name	`ben_cut_factor`
Description	Maximum cuts per child scenario
Value	Integer
Default	`20`

CLI Name	`--ben-max-iter`
SAMPL Name	`ben_max_iter`
Description	Iteration limit for Benders' decomposition
Value	Nonnegative integer
Default	`10000`

know the EEV - expected value of the expected value solution. EEV is calculated by solving the EV model, fixing the result so obtained in all the WS models (all stages but the last), which are then solved to give a probability-weighted average value for the objective - which is the VSS. Option `--vss-fstage` can be used to restrict the fix that is performed to first stage variables only (although in theory this is not correct, the theoretical result is often meaningless as a complete fix may be infeasible).

Solver Options

CLI Name	`--solver`
SAMPL Name	`solver`
Description	External solver name
Value	String
Default	`fortmp (windows) or cplex (linux/mac)`

CLI Name	`--lp-alg`
SAMPL Name	`lp_alg`
Description	This option specifies which LP algorithm should be used to solve a deterministic equivalent problem and all linear programming subproblems that are constructed in the course of solving the SP problem.
Value	The possible values for this option are listed in the table below.

Name	Description
`auto`	The algorithm is chosen automatically (default)
`primal`	Primal simplex method
`dual`	Dual simplex method
`ipm`	Interior point method

Default	`auto`

CLI Name	`--basis-restart`
SAMPL Name	`basis_restart`
Description	Flag specifying whether to use warm start
Value	`0` or `1`
Default	`1`

CLI Name	`--use-fortmp-specs`
SAMPL Name	`use_fortmp_specs`
Description	Flag specifying whether to use extra SPECS-command file (only with the FortMP solver.) A SPECS command file with the name `fortmp.spc` may be used to refine the options when FortMP is the solver in use. See the FortMP manual (Ellison et al., 2008). Commands are to be provided in sections corresponding to the type of sub-problem that is being solved, according to the following table:

Section ID	Description
ALL	Section that applies to every call to the solver. Must appear first in the SPECS file.
DeqImna	Section to handle Deterministic Equivalent - Implicit NA
DeqExna	Section to handle Deterministic Equivalent - Explicit NA
ExpVal	Section to handle Expected Value solutions
Wsprob	Section to handle Wait and See scenario sub-problems
BendRoot	Section to handle Benders root-node sub-problem solutions (multi-stage)
BendNode	Section to handle Benders node sub-problem solutions other that root or leaf (multi-stage)
BendLeaf	Section to handle Benders leaf sub-problem solutions with no warm restart (multi-stage)
BenRLeaf	Section to handle Benders leaf sub-problem solutions with warm restart (multi-stage)
Ben2Mast	Section to handle Benders master-problem solutions (two-stage)
Ben2Sprb	Section to handle Benders sub-problem solutions (two-stage)
LevelQP	Section to handle Benders Level-method QP solutions (two-stage)

The section ID is named in a `BEGIN` line - e.g. `BEGIN (DeqImna)` - which is followed by the SPECS commands for that section. Each section is terminated with a line `END`.

Value	0 or 1
Default	0

Output Options

CLI Name	`--sol-file`
SAMPL Name	`sol_file`
Description	Actual name of the solution file. If the filename is empty solution output is disabled.
Value	String
Default	

CLI Name	`--sol-include-second-stage`
SAMPL Name	`sol_include_second_stage`
Description	Include information for the 2nd stage solution if it is available. The information will be appended at the end of the solution file. Currently the option `--sol-format` must be set to ‘`ampl`’ for the information to be written in the output file.
Value	`0` or `1`
Default	`0`

CLI Name	`--log-file`
SAMPL Name	`log_file`
Description	Actual name of the log file. If the filename is empty logging is disabled.
Value	String
Default	

CLI Name	`--ben-log-print`
SAMPL Name	`ben_log_print`
Description	Code for items to be logged (Benders multi-stage only) Additional logged output can be generated with the option `--ben-log-print`. This should be used with caution as the log-file can easily be swamped. Certain values of use are: • 3 - for solution status of each node (plus the default) • 19 - for details of the node tree (plus the above) • 95 - for description of every cut applied (plus the above) with option 3 the output volume may be reduced by specifying `--ben-log-freq` - that is the interval to leave between node solution-status logs.
Value	Integer
Default	`1`

CLI Name	`--ben-log-freq`
SAMPL Name	`ben_log_freq`
Description	Logged every this number of passes (Benders multi-stage only)
Value	Integer
Default	`1`

B Solution Methods for Two-Stage Stochastic Programming Problems

In this appendix we only consider linear SP models and assume that the random parameters have a discrete finite distribution. This class is based on two key concepts: (i) a finite set of discrete scenarios (of model parameters) and (ii) a partition of variables to first stage ("here and now") decision variables and a second stage observation of the parameter realisations and corrective actions and the corresponding recourse (decision) variables.

B.1 Deterministic Equivalent

The first stage decisions are represented by the vector $\boldsymbol{x}$. Assume there are S possible outcomes (*scenarios*) of the random event, the ith outcome occurring with probability p_i. Suppose the first stage decision has been made with the result $\boldsymbol{x}$, and the ith scenario is realised. The second stage decision $\boldsymbol{y}$ is computed by solving the following *second-stage problem* or *recourse problem*

$$R_i(x): \quad \begin{array}{rl} \min & \boldsymbol{q_i^T y} \\ \text{subject to} & T_i\boldsymbol{x} + W_i\boldsymbol{y} = \boldsymbol{h_i}, \\ & \boldsymbol{y} \geq 0, \end{array} \tag{17}$$

where $\boldsymbol{q_i}$, $\boldsymbol{h_i}$ are given vectors and T_i , W_i are given matrices. Let K_i denote the set of those x vectors for which the recourse problem $R_i(x)$ has a feasible solution. This is a convex polyhedron. For $x \in K_i$, let $q_i(x)$ denote the optimal objective value of the recourse problem. We assume that $q_i(x) > -\infty$ (or equivalently, we assume that the dual of the recourse problem $R_i(x)$ has a feasible solution, solvability of the dual problem does not depend on x). The function $q_i : K_i \to \mathbb{R}$ is a *polyhedral* (i.e., piecewise linear) convex function.

The customary formulation of the *first-stage problem* is stated as:

$$\begin{array}{rl} \min & \boldsymbol{c^T x} + \sum_{i=1}^{S} p_i q_i(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in X, \\ & \boldsymbol{x} \in K_i, \ (i = 1, \ldots, S), \end{array} \tag{18}$$

where $X := \boldsymbol{x}|A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0$ is a non-empty bounded polyhedron describing the constraints, $\boldsymbol{c}$ and $\boldsymbol{b}$ are given vectors and A is a given matrix, with compatible sizes. The expectation part of the objective, $F(x) := \sum_{i=1}^{S} p_i q_i(x)$, is called the *expected recourse function.* This is a polyhedral convex function with the domain $K := K_1 \cap \ldots \cap K_S$.

This two-stage stochastic programming problem ((18) and (17)) can be formulated as a single linear programming problem called the *deterministic equivalent problem*:

$$
\begin{array}{rllll}
\min & \boldsymbol{c^T x} + p_1 \boldsymbol{q_1^T y_1} + & \dots & + p_S \boldsymbol{q_S^T y_S} & \\
\text{subject to} & A\boldsymbol{x} & & & = \boldsymbol{b}, \\
& T_1\boldsymbol{x} + W_1\boldsymbol{y_1} & & & = \boldsymbol{h_1} \\
& \vdots & \ddots & & \vdots \\
& T_S\boldsymbol{x} + & & W_S\boldsymbol{y_S} & = \boldsymbol{h_S} \\
& \boldsymbol{x} \geq 0, \boldsymbol{y_1} \geq 0, \dots, \boldsymbol{y_S} \geq 0 & & &
\end{array}
\tag{19}
$$

B.2 Decomposition Methods

The deterministic equivalent problem (19) is a linear programming problem of a specific structure: for each scenario, a subproblem is included that describes the second-stage decision associated with the corresponding scenario realisation. The subproblems are linked by the first-stage decision variables. Dantzig and Madansky (1961) observed that the dual of the deterministic equivalent problem fits the prototype for the Dantzig-Wolfe decomposition (Dantzig and Wolfe (1960)).

Van Slyke and Wets (1969) proposed a cutting-plane approach for the first-stage problem (18). Their L-Shaped method builds respective cutting-plane models of the feasible domain $K = K_1 \cap \dots \cap K_S$ and of the expected recourse function $F = \sum_{i=1}^{S} p_i q_i$. We outline cutting-plane models and their relationship with decomposition. Let us denote the dual of $R_i(x)$ in (17) as

$$
D_i(x): \quad \begin{array}{rl} \max & \boldsymbol{z^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \\ \text{subject to} & W_i^T\boldsymbol{z} \leq \boldsymbol{q_i}, \end{array}
\tag{20}
$$

where $\boldsymbol{z}$ is a real-valued vector.

The feasible region is a convex polyhedron that we assumed nonempty. We will characterise this polyhedron by two finite sets of vectors: let U_i and V_i denote the sets of the extremal points and of the extremal rays, respectively, in case the polyhedron can be represented by these sets. To handle the general case, we require further formalism; let us add a slack vector $\boldsymbol{\gamma}$ of appropriate dimension, and use the notation $[W_i^T, I](\boldsymbol{z}, \boldsymbol{\gamma}) = W_i^T\boldsymbol{z} + \boldsymbol{\gamma}$. Given a composite vector $(\boldsymbol{z}, \boldsymbol{\gamma})$ of appropriate dimensions, let support $(\boldsymbol{z}, \boldsymbol{\gamma})$ denote the set of those column-vectors of the composite matrix $[W_i^T, I]$ that belong to non-zero$(\boldsymbol{z}, \boldsymbol{\gamma})$-components. Using these, let

$U_i = \{\boldsymbol{z} | W_i^T\boldsymbol{z} + \boldsymbol{\gamma} = \boldsymbol{q_i}, \boldsymbol{\gamma} \geq 0$, support $(\boldsymbol{z}, \boldsymbol{\gamma})$ is a linearly independent set $\}$,

$V_i = \{\boldsymbol{z} | W_i^T\boldsymbol{z} + \boldsymbol{\gamma} = 0, \boldsymbol{\gamma} \geq 0, \|(\boldsymbol{z}, \boldsymbol{\gamma})\| = 1$ support $(\boldsymbol{z}, \boldsymbol{\gamma})$ is a minimal dependent set $\}$,

where a minimal dependent set is a set that is not linearly independent; but is minimal in the sense that having discarded any of its elements, the remaining elements compose a linearly independent set.

U_i is the set of the basic feasible solutions of problem (20) and hence it is a finite set. Finiteness of V_i can be proven in a similar manner: Given a minimal dependent subset of the columns of the matrix $[W_i^T, I]$, there are no more than 2 vectors in V_i that have the given subset as support. The feasible domain of the dual problem $D_i(x)$ in (20) can be represented as convex combinations of U_i-elements added to cone-combinations of V_i-elements.

We have $\boldsymbol{x} \in K_i$ if and only if the dual problem $D_i(x)$ has a finite optimum, that is,

$$\boldsymbol{v_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le 0 \text{ holds for every } \boldsymbol{v_i} \in V_i$$

In this case, the optimum of $D_i(x)$ is attained at an extremal point, and can be computed as

$$\begin{array}{ll} \text{min} & \vartheta_i \\ \text{subject to} & \vartheta_i \in \mathbb{R} \\ & \boldsymbol{u_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le \vartheta_i \quad (\boldsymbol{u_i} \in U_i) \end{array}$$

By the linear programming duality theorem, the optimum of the above problem is equal to $q_i(x)$; hence the first-stage problem (18) is written as

$$\begin{array}{ll} \text{min} & \boldsymbol{c^T x} + \sum_{i=1}^{S} p_i\vartheta_i \\ \text{subject to} & \boldsymbol{x} \in X, \ \ \vartheta_i \in \mathbb{R} \ \ (i = 1, \dots, S) \\ & \boldsymbol{v_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le 0 \ \ (\boldsymbol{v_i} \in V_i, i = 1, \dots, S) \\ & \boldsymbol{u_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le \vartheta_i \ \ (\boldsymbol{u_i} \in U_i, i = 1, \dots, S) \end{array} \tag{21}$$

This we call the *disaggregated form*. The *aggregated form* is stated as

$$\begin{array}{ll} \text{min} & \boldsymbol{c^T x} + \vartheta \\ \text{subject to} & \boldsymbol{x} \in X, \ \ \vartheta \in \mathbb{R} \\ & \boldsymbol{v_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le 0 \ \ (\boldsymbol{v_i} \in V_i, i = 1, \dots, S) \\ & \sum_{i=1}^{S} p_i\boldsymbol{u_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \le \vartheta \ \ \big((\boldsymbol{u_1}, \dots, \boldsymbol{u_S}) \in \mathcal{U}\big) \end{array} \tag{22}$$

where $\mathcal{U} \subset U_1 \times \ldots \times U_S$ is a subset that contains an element for each facet in the graph of the polyhedral convex function F; formally, we have

$$F(\boldsymbol{x}) = \sum_{i=1}^{S} \left\{ p_i \max_{\boldsymbol{u_i} \in U_i} \boldsymbol{u_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x}) \right\} = \max_{(\boldsymbol{u_1}, \dots, \boldsymbol{u_S}) \in \mathcal{U}} \sum_{i=1}^{S} p_i \boldsymbol{u_i^T}(\boldsymbol{h_i} - T_i\boldsymbol{x})$$

Cutting-plane methods can be devised on the basis of either the disaggregated formulation (21) or the aggregated formulation (22). These are processed by iterative methods that build respective cutting-plane models of the feasible set K and the expected recourse function F. Cuts at a given iterate $\widehat{x}$ are generated by solving the dual problems $D_i(\widehat{x})(i = 1, \dots, S)$. Dual problems with unbounded objectives yield *feasibility cuts* that are used to construct a model of K. Dual problems with optimal solutions yield *optimality cuts* that are used to construct a model of F.

In its original form, the L-Shaped method of Van Slyke and Wets (1969) works on the aggregated problem. A multicut version that works on the disaggregated problem was proposed by Birge and Louveaux (1988).

There is a close relationship between decomposition and cutting-plane approaches. It turns out that the following approaches yield methods that are in principle identical:

- cutting-plane method for either the disaggregated problem (21) or the aggregated problem (21),
- DantzigWolfe decomposition (Dantzig and Wolfe (1960)) applied to the dual of the deterministic equivalent problem (19),
- Benders decomposition (Benders (1962)) applied to the deterministic equivalent problem (19).

Cutting-plane formulations have the advantage that they give a clear visual illustration of the procedure. A state-of-the-art overview of decomposition methods can be found in Ruszczyński (2003).

B.2.1 Aggregated versus disaggregated formulations

The difference between the aggregated and the disaggregated problem formulations may result in a substantial difference in the efficiency of the solution methods. By using disaggregated cuts, more detailed information is stored in the master problem, hence the number of the master iterations is reduced in general. This is done at the expense of larger master problems.

Birge and Louveaux (1997) conclude that the multicut approach is in general more effective when the number of the scenarios is not significantly larger than the number of the constraints in the first-stage problem. This conclusion is based on their own numerical results (Birge and Louveaux (1988)) and those of Gassmann (1990).

B.3 Regularisation and trust region methods

It is observed that successive iterations do not generally produce an orderly progression of solutions - in the sense that while the change in objective value from one iteration to the next may be very small, even zero, a wide difference may exist between corresponding values of the first-stage variables. This feature of zigzagging in cutting plane methods is the consequence of using a linear approximation. Improved methods were developed that use quadratic approximation: proximal point method by Rockafellar (1976), and bundle methods by Kiwiel (1985) and Lemaréchal (1978). These methods construct a sequence of *stability centers* together with the sequence of the iterates. When computing the next iterate, roaming away from the current stability center is penalised.

Another approach is the trust region methods, where a trust region is constructed around the current stability center, and the next iterate is selected from this trust region.

B.4 Regularised decomposition

The Regularised Decomposition (RD) method of Ruszczyński (1986) is a bundle-type method applied to the minimisation of the sum of polyhedral convex functions over a convex polyhedron, hence this method fits the disaggregated problem (19). The RD method lays an emphasis on keeping the master problem as small as possible. (This is achieved by an effective constraint reduction strategy.) A recent discussion of the RD method can be found in Ruszczyński (1986).

Ruszczyński and Świętanowski (1997) implemented the RD method, and solved two-stage stochastic programming problems, with a growing scenario set. Their test results show that the RD method is capable of handling large problems.

B.5 Regularisation: The level method

A more recent development in convex programming is the level method of Lemaréchal et al. (1995). This is a special bundle-type method that uses level sets of the model functions for regularisation. Let us consider the problem

$$\begin{array}{ll} \min & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in Y \end{array} \tag{23}$$

where $Y \subset \mathbb{R}^n$ is a convex bounded polyhedron, and f a real-valued convex function, Lipschitzian relative to Y. The level method is an iterative method, a direct generalization of the classical cutting-plane method. A *cutting-plane model* of f is maintained using function values and subgradients computed at the known iterates. Let $\underline{f}$ denote the current model function; this is the upper cover of the linear support functions drawn at the known iterates. Hence $\underline{f}$ is a polyhedral convex lower approximation of f. The level sets of the model function are used for regularisation.

Let $\hat{x}$ denote the current iterate. Let F^* denote the minimum of the objective values in the known iterates. Obviously F^* is an upper bound for the optimum of (23).

Let $\underline{F} := \min_{\boldsymbol{x} \in Y} \underline{f}(\boldsymbol{x})$ denote the minimum of the current model function over the feasible polyhedron. Obviously $\underline{F}$ is a lower bound for the optimum of (23).

If the gap $F^* - \underline{F}$ is small, then the algorithm stops. Otherwise let us consider the level set of the current model function belonging to the level $(1-\lambda)F^* + \lambda\underline{F}$ where $0 < \lambda < 1$ is a fixed parameter. Using formulas, the current level set is

$$\hat{Y} = \left\{ \boldsymbol{x} \in F \;\middle|\; \underline{f}(\boldsymbol{x}) \leq (1-\lambda)F^* + \lambda\underline{F} \right\}.$$

The next iterate is obtained by *projecting* the current iterate onto the current level set. Formally, the next iterate is an optimal solution of the convex quadratic programming problem $\min \|x - \hat{x}\|^2$ subject to $x \in \hat{Y}$.

Lemaréchal et al. (1995) give the following efficiency estimate: To obtain a gap smaller than ϵ, it suffices to perform

$$\kappa \left(\frac{DL}{\epsilon} \right)^2 \tag{24}$$

iterations, where D is the diameter of the feasible polyhedron, L is a Lipschitz constant of the objective function, and κ is a constant that depends only on the parameter of the algorithm.

Applying the level method to our first-stage problem (18), let $f(\boldsymbol{x}) := \boldsymbol{c}^{\boldsymbol{T}}\boldsymbol{x} + F(\boldsymbol{x})$ and $Y = X \bigcap K$. The feasible domain is not known explicitly (except for problems with relatively complete recourse). Hence, in general, we must construct a cutting-plane model of Y using feasibility cuts. The level method must be adapted accordingly: the objective value can only be computed for feasible iterates. We clearly obtain a finite procedure because the set of the possible cuts is finite. (We never discard cuts in our implementation. Though there are means of bundle reduction for the level method, we did not implement them because the level method solved our test problems in but a few iterations.)

Remark 1 In case of relatively complete recourse no feasibility cuts are needed, and the efficiency estimate (24) applies. This estimate is essentially different from the classic finiteness results obtained when a polyhedral convex function is minimised by a cutting-plane method. Finiteness results are based on enumeration. The straightforward finiteness proof assumes that basic solutions are found for the model problems, and that there is no degeneracy. (These assumptions facilitate bundle reduction.)

An interesting finiteness proof that allows for nonbasic solutions is presented in Ruszczyński (2006). This is based on the enumeration of the *cells* (i.e., polyhedrons, facets, edges, vertices) that the linear pieces of the objective function define.

Remark 2 In general, the constants L and D in (24) are not easily derived from the problem data. Upper bounds of the Lipschitz constant L are proposed in Fábián and Szőke (2007) for the case of special two-stage stochastic programming problems, e.g., those having network recourse (SP problems where the second-stage subproblems are network flow problems). But even with constants of reasonable magnitudes, the estimate (24) generally yields bounds too large for practical purposes.

However, the level method performs much better in practice than the estimate (24) implies. Nemirovski (2005) [chapter 5.3.2] observes the following experimental fact: when solving a problem with n variables, every n steps add a new accurate digit in our estimate of the optimum. This observation is confirmed by those experiments reported in Fábián and Szőke (2007), where the level method is applied for the solution of two-stage stochastic programming problems with relatively complete recourse. That paper also reports on solving the problems with growing scenario sets. According to the results presented, there is no relationship connecting the number of the scenarios and the number of the levelmaster iterations required (provided the number of the scenarios is large enough).

Remark 3 There are extensions of the level method that particularly fit in with a two-stage stochastic problem solver.

The constrained level method of Lemaréchal et al. (1995) is a primal-dual method that solves convex problems involving constraint functions. The first-stage problem (18) can be formulated using a constraint function instead of the set constraints $\boldsymbol{x} \in K_i (i = 1, \ldots, S)$.

A measure of the second-stage infeasibility can be used as a constraint function; namely, the expectation of the infeasibility in the corresponding second-stage problems. Fábián and Szőke (2007) applied the constrained level method to this constraint function formulation of the first-stage problem. The advantage of this approach is that regularisation extends to feasibility issues. This approach requires extending the expected recourse function beyond K.

Fábián (2000) proposed inexact versions of the level method and the constrained level method. The inexact methods use approximate data to construct models of the objective and constraint functions. At the beginning of the procedure, a rough approximation is used, and the accuracy is gradually increased as the optimum is approached. Solving the first-stage problem with an inexact method facilitates a progressive approximation of the distribution. Moreover we can work with approximate solutions of the second stage problems. Numerical results of Fábián and Szőke (2007) show that this progressive approximation framework is effective: although the number of the master iterations is larger than in the case of the exact method, there is a substantial reduction in the solution time of the second-stage problems.

A different approach of using inexact bundle methods for two-stage stochastic programming problems is proposed by Oliveira et al. (2011).

Remark 4 The level method can also be implemented for problems with unbounded domain Y. A set of initial cuts is then needed to make the master objective bounded from below, just like with a pure cutting-plane method.

The constant D is never actually used in course of the level method; it is used only in the convergence proof and in the theoretical efficiency estimate (24). In case the objective function f is polyhedral, then finiteness of the procedure follows from the finiteness of the set of the possible cuts.

Remark 5 An interesting feature of the level method is that the parameter λ is fixed. This is in contrast with other bundle-type methods that need continual tuning of the parameters in course of operation.

B.6 Regularisation: Trust-region

The box-constrained trust-region method of Linderoth and Wright (2003) solves the disaggregated problem (21), and uses a special trust-region approach.

Trust-region methods construct a sequence of stability centers together with the sequence of the iterates. Trust regions are constructed around the stability centers, and the next iterate is selected from the current trust region. Linderoth and Wright construct box-shaped trust regions, hence the resulting master problems remain linear. The size of the trust region is continually adapted on the basis of the quality of the current solution.

C Known Weaknesses

1. Execution time is heavily dependent on the total number of scenarios. If this is very large the deterministic equivalent solvers become impossible to use and Benders' solver may become too lengthy for the user's satisfaction.

2. A procedure to select a useful subset of scenarios by importance or by Monte-Carlo sampling has been programmed but is not yet tested to any extent.

3. The Benders' solver is designed for Markovian stochastic data in which the interaction between stages in the constraint matrix forms a stair pattern. This means that any one stage is constrained directly by the previous stage decisions only and not by decisions earlier than the previous stage. Two-stage problems are Markovian. Non-Markovian multi-stage problems should also be solvable with Benders, but, to our knowledge, no mathematical proof has been published.

4. In rare cases Benders' solver may be halted by a cycling status with the true optimum solution not yet reached (see description of the `--ben-theta-lower` option). Cycling may also result from attempt to solve a non-Markovian problem, or from degeneracy if a feasible solution is difficult to find.

D Examples of Use

The following is a simple example of a 4-stage problem:

Core File (MYSP.cor)

```
NAME          MYSP
ROWS
  N Z
  G R1
  G R2
  G R3
  L R4
  G R5
  G R6
  L R7
  G R8
  G R9
  L R10
COLUMNS
    X1        Z                  3.0   R1                  1.0
    X1        R2                 1.0
    X2        Z                  2.0   R1                  1.0
    Y1        Z                -15.0   R2                 -3.0
    Y1        R3                 1.0   R4                  1.0
    Y1        R5                 2.0
    Y2        Z                -12.0   R5                 -1.0
    Y2        R6                 1.0   R7                  1.0
    Y2        R8                 3.0
    Y3        Z                 -4.0   R8                 -1.0
    Y3        R9                 1.0   R10                 1.0
RHS
    RHS1      R3                 3.2   R4                  4.0
    RHS1      R6                 3.2   R7                  7.0
    RHS1      R9                 1.0   R10                 1.0
ENDATA
```

Time File (MYSP.tim)

```
TIME          MYSP
PERIODS
    X1        R1                       STAGE001
    Y1        R2                       STAGE002
    Y2        R5                       STAGE003
    Y3        R8                       STAGE004
ENDATA
```

Stoch File (MYSP.sto)

```
STOCH          MYSP
SCENARIOS      DISCRETE
 SC SCEN0001  'ROOT'    0.125          STAGE001
    RHS1       R3            3.2
    RHS1       R4            4.0
    RHS1       R6            3.2
    RHS1       R7            7.0
    RHS1       R9            4.0
    RHS1       R10           8.0
 SC SCEN0002  SCEN0001  0.125          STAGE004
    RHS1       R9            3.0
    RHS1       R10           6.0
 SC SCEN0003  SCEN0001  0.125          STAGE003
    RHS1       R6            4.8
    RHS1       R7            9.0
    RHS1       R9            9
    RHS1       R10           12.0
 SC SCEN0004  SCEN0003  0.125          STAGE004
    RHS1       R9            6.4
    RHS1       R10           18.0
 SC SCEN0005  SCEN0001  0.125          STAGE002
    RHS1       R3            1.2
    RHS1       R4            4.0
    RHS1       R6            2.2
    RHS1       R7            7.5
    RHS1       R9            3.0
    RHS1       R10           6.0
 SC SCEN0006  SCEN0005  0.125          STAGE004
    RHS1       R9            4.0
    RHS1       R10           9.0
 SC SCEN0007  SCEN0005  0.125          STAGE003
    RHS1       R6            2.8
    RHS1       R7            6.0
    RHS1       R9            5.2
    RHS1       R10           12.0
 SC SCEN0008  SCEN0007  0.125          STAGE004
    RHS1       R9            4.4
    RHS1       R10           8.0
ENDATA
```

Command

The following command causes all forms of output to be generated and the solution written to `MYSP.sol`:

```
fortsp --solve-ev --solve-ws --compute-evpi --compute-vss --sp-alg=benders \
       --ben-fffb --vss-fstage --solver=cplex --sol-file=MYSP.sol MYSP
```

Solution File (MYSP.sol)

Solution file generated from the run is as follows:

```
HN
Obj         -149
Optimal LP solution
Variables
  Name     Index Stage         Value        Rscos          Lob          Upb
  X1           0     1            12            0            0          inf
  X2           1     1             0            2            0          inf
Constraints
  Name     Index Stage        SPrice       RowAct          Lhs          Rhs
  R1           0     1            -0           12            0          inf
END
ExpVal
Obj         -152
Optimal LP solution
Variables
  Name     Index Stage         Value        Rscos          Lob          Upb
  X1           0     1            12            0            0          inf
  X2           1     1             0            2            0          inf
Constraints
  Name     Index Stage        SPrice       RowAct          Lhs          Rhs
  R1           0     1            -0           12            0          inf
END
WS
WS Scenario = 1
Obj         -140, Prob = 0.125
Optimal LP solution
Variables
  Name     Index Stage         Value        Rscos          Lob          Upb
  X1           0     1            12            0            0          inf
  X2           1     1             0            2            0          inf
Constraints
  Name     Index Stage        SPrice       RowAct          Lhs          Rhs
  R1           0     1            -0           12            0          inf
END

and the WS solutions for 7 more scenarios...
```

The solution shown above includes only 1st stage values and comprises:

- The WS solution for each scenario

- The WS summary result
- The EV solution
- The HN solution
- Values of the stochastic measures EVPI and VSS

The stochastic measures EVPI and VSS are both zero in this simple example since the values of EEV and WS are both equal to the HN objective. Note that if the option `--vss-fix-fstage` had been omitted, then the EEV would in fact have been infeasible and VSS would have been infinite. Theoretically $VSS = 0$ is not correct but this value is probably more important to a user than $VSS = infinity$.

Solver Output

```
IBM ILOG License Manager: "IBM ILOG Optimization Suite for Academic
Initiative" is accessing CPLEX 12 with option(s): "e m b q ".
Reading time = 0.000943 s.
Stage 1 has 1 rows, 2 columns, and 2 nonzeros.
Stage 2 has 3 rows, 1 columns, and 4 nonzeros.
Stage 3 has 3 rows, 1 columns, and 4 nonzeros.
Stage 4 has 3 rows, 1 columns, and 4 nonzeros.
Problem has 4 stages, 8 scenarios, and 62 random elements.
Solution time = 0.025982 s.
Tried aggregator 1 time.
LP Presolve eliminated 10 rows and 5 columns.
All rows and columns eliminated.
Presolve time =    0.00 sec.
Optimal solution found, objective = -149.
EVPI = 0
VSS = 0
```

Dimensions of the problems to be solved, run times and stochastic measures are reported. The output from the external solver (CPLEX) is suppressed for subproblems owing to the large volume that would be shown otherwise.

Solving the Deterministic Equivalent Problem

In the following command only the HN solution is requested (default), using the deterministic equivalent algorithm with implicit nonanticipativity. Data is the same as before.

```
fortsp --sp-alg=deteq --solver=cplex --sol-file=MYSP.sol MYSP
```

Solution File (MYSP.sol)

```
HN
Obj         -149
Optimal LP solution
```

```
Variables
  Name     Index Stage         Value         Rscos           Lob           Upb
  X1           0     1            12             0             0           inf
  X2           1     1             0             0             0           inf
Constraints
  Name     Index Stage        SPrice        RowAct           Lhs           Rhs
  R1           0     1             0           -12             0           inf
END
```

Solver Output

```
IBM ILOG License Manager: "IBM ILOG Optimization Suite for Academic
Initiative" is accessing CPLEX 12 with option(s): "e m b q ".
Reading time = 0.000922 s.
Stage 1 has 1 rows, 2 columns, and 2 nonzeros.
Stage 2 has 3 rows, 1 columns, and 4 nonzeros.
Stage 3 has 3 rows, 1 columns, and 4 nonzeros.
Stage 4 has 3 rows, 1 columns, and 4 nonzeros.
Problem has 4 stages, 8 scenarios, and 62 random elements.
DEP has 43 rows, 16 columns, and 58 nonzeros.
Tried aggregator 1 time.
LP Presolve eliminated 43 rows and 16 columns.
All rows and columns eliminated.
Presolve time =    0.00 sec.

Total real time on 2 threads =    0.00 sec.
Number of iterations = 0.
Solution time = 0.002468 s.
Optimal solution found, objective = -149.
```

The output from the external LP solver (CPLEX) for the deterministic equivalent problem is preserved as can be seen above.

E Performance on Test Models

The results presented in this section are based on the computational study by Zverovich et al. (2012).

E.1 Experimental Setup

The computational experiments were performed on a Linux machine with 2.4 GHz Intel CORE i5 M520 CPU and 6 GiB of RAM. Deterministic equivalents were solved with CPLEX 12.1 dual simplex and barrier optimisers. Crossover to a basic solution was disabled for the barrier optimiser and the number of threads was limited to 1. For other CPLEX options the default values were used.

The times are reported in seconds with times of reading input files included. For simplex and IPM the times of constructing deterministic equivalent problems are also included though it should be noted that they only amount to small fractions of the total. CPLEX linear and quadratic programming solver was used to solve master problem and subproblems in the implementation of Benders decomposition with and without level regularisation. All the test problems were presented in SMPS format introduced by Birge et al. (1987).

The first-stage solution of the expected value problem was taken as a starting point for the decomposition methods. The values of the parameters are specified below.

- Benders decomposition with regularisation by the level method:
  $\lambda = 0.5$,
- Regularised decomposition:
  $\sigma = 1, \gamma = 0.9$.
- Trust region method based on l_∞ norm:
  $\Delta = 1, \Delta_{hi} = 10^3$ (except for the saphir problems where $\Delta_{hi} = 10^9$), $\xi = 10^{-4}$.

E.2 Data Sets

We considered test problems which were drawn from four different sources described in Table 11. Table 12 gives the dimensions of these problems.

Most of the benchmark problems have stochasticity only in the right-hand side (RHS). Notable exception is the SAPHIR family of problems which has random elements both in the RHS and the constraint matrix.

Table 12: Dimensions of test problems

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
fxm	6	92	114	238	343	1520	2172	12139
	16	92	114	238	343	3900	5602	31239
fxmev	1	92	114	238	343	330	457	2589
pltexpa	6	62	188	104	272	686	1820	3703
	16	62	188	104	272	1726	4540	9233

Table 12: Dimensions of test problems (continued)

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
stormg2	8	185	121	528	1259	4409	10193	27424
	27	185	121	528	1259	14441	34114	90903
	125	185	121	528	1259	66185	157496	418321
	1000	185	121	528	1259	528185	1259121	3341696
airl-first	25	2	4	6	8	152	204	604
airl-second	25	2	4	6	8	152	204	604
airl-randgen	676	2	4	6	8	4058	5412	16228
assets	100	5	13	5	13	505	1313	2621
	37500	5	13	5	13	187505	487513	975021
4node	1	14	52	74	186	88	238	756
	2	14	52	74	186	162	424	1224
	4	14	52	74	186	310	796	2160
	8	14	52	74	186	606	1540	4032
	16	14	52	74	186	1198	3028	7776
	32	14	52	74	186	2382	6004	15264
	64	14	52	74	186	4750	11956	30240
	128	14	52	74	186	9486	23860	60192
	256	14	52	74	186	18958	47668	120096
	512	14	52	74	186	37902	95284	239904
	1024	14	52	74	186	75790	190516	479520
	2048	14	52	74	186	151566	380980	958752
	4096	14	52	74	186	303118	761908	1917216
	8192	14	52	74	186	606222	1523764	3834144
	16384	14	52	74	186	1212430	3047476	7668000
	32768	14	52	74	186	2424846	6094900	15335712
4node-base	1	16	52	74	186	90	238	772
	2	16	52	74	186	164	424	1240
	4	16	52	74	186	312	796	2176
	8	16	52	74	186	608	1540	4048
	16	16	52	74	186	1200	3028	7792
	32	16	52	74	186	2384	6004	15280
	64	16	52	74	186	4752	11956	30256
	128	16	52	74	186	9488	23860	60208
	256	16	52	74	186	18960	47668	120112
	512	16	52	74	186	37904	95284	239920
	1024	16	52	74	186	75792	190516	479536
	2048	16	52	74	186	151568	380980	958768
	4096	16	52	74	186	303120	761908	1917232
	8192	16	52	74	186	606224	1523764	3834160
	16384	16	52	74	186	1212432	3047476	7668016
	32768	16	52	74	186	2424848	6094900	15335728
4node-old	32	14	52	74	186	2382	6004	15264
chem	2	38	39	46	41	130	121	289
chem-base	2	38	39	40	41	118	121	277
lands	3	2	4	7	12	23	40	92
lands-blocks	3	2	4	7	12	23	40	92

Table 12: Dimensions of test problems (continued)

		Stage 1		Stage 2		Deterministic Equivalent		
Name	Scen	Rows	Cols	Rows	Cols	Rows	Cols	Nonzeros
env-aggr	5	48	49	48	49	288	294	876
env-first	5	48	49	48	49	288	294	876
env-loose	5	48	49	48	49	288	294	876
env	15	48	49	48	49	768	784	2356
	1200	48	49	48	49	57648	58849	177736
	1875	48	49	48	49	90048	91924	277636
	3780	48	49	48	49	181488	185269	559576
	5292	48	49	48	49	254064	259357	783352
	8232	48	49	48	49	395184	403417	1218472
	32928	48	49	48	49	1580592	1613521	4873480
env-diss-aggr	5	48	49	48	49	288	294	876
env-diss-first	5	48	49	48	49	288	294	876
env-diss-loose	5	48	49	48	49	288	294	876
env-diss	15	48	49	48	49	768	784	2356
	1200	48	49	48	49	57648	58849	177736
	1875	48	49	48	49	90048	91924	277636
	3780	48	49	48	49	181488	185269	559576
	5292	48	49	48	49	254064	259357	783352
	8232	48	49	48	49	395184	403417	1218472
	32928	48	49	48	49	1580592	1613521	4873480
phone1	1	1	8	23	85	24	93	309
phone	32768	1	8	23	85	753665	2785288	9863176
stocfor1	1	15	15	102	96	117	111	447
stocfor2	64	15	15	102	96	6543	6159	26907
rand0	2000	50	100	25	50	50050	100100	754501
	4000	50	100	25	50	100050	200100	1508501
	6000	50	100	25	50	150050	300100	2262501
	8000	50	100	25	50	200050	400100	3016501
	10000	50	100	25	50	250050	500100	3770501
rand1	2000	100	200	50	100	100100	200200	3006001
	4000	100	200	50	100	200100	400200	6010001
	6000	100	200	50	100	300100	600200	9014001
	8000	100	200	50	100	400100	800200	12018001
	10000	100	200	50	100	500100	1000200	15022001
rand2	2000	150	300	75	150	150150	300300	6758501
	4000	150	300	75	150	300150	600300	13512501
	6000	150	300	75	150	450150	900300	20266501
	8000	150	300	75	150	600150	1200300	27020501
	10000	150	300	75	150	750150	1500300	33774501
saphir	50	32	53	8678	3924	433932	196253	1136753
	100	32	53	8678	3924	867832	392453	2273403
	200	32	53	8678	3924	1735632	784853	4546703
	500	32	53	8678	3924	4339032	1962053	11366603
	1000	32	53	8678	3924	8678032	3924053	22733103

Table 11: Sources of test problems

Source	Reference	Comments
1. POSTS collection	Holmes (1995)	Two-stage problems from the (PO)rtable (S)tochastic programming (T)est (S)et (POSTS)
2. Slptestset collection	Ariyawansa and Felt (2004)	Two-stage problems from the collection of stochastic LP test problems
3. Random problems	Kall and Mayer (1998)	Artificial test problems generated with pseudo random stochastic LP problem generator GENSLP
4. SAMPL problems	König et al. (2007), Valente et al. (2008)	Problems instantiated from the SAPHIR gas portfolio planning model formulated in Stochastic AMPL (SAMPL)

It should be noted that the problems generated with GENSLP do not possess any internal structure inherent in real-world problems. However they are still useful for the purposes of comparing scale-up properties of algorithms.

E.3 Computational Results

The computational results are presented in Tables 13 and 14. Iter denotes the number of iterations. For decomposition methods this is the number of master iterations.

Finally we present the results in the form of performance profiles. The performance profile for a solver is defined by Dolan and Moré (2002) as the cumulative distribution function for a performance metric. We use the ratio of the solving time versus the best time as the performance metric. Let P and M be the set of problems and the set of solution methods respectively. We define by $t_{p,m}$ the time of solving problem $p \in P$ with method $m \in M$. For every pair (p, m) we compute performance ratio

$$r_{p,m} = \frac{t_{p,m}}{\min\{t_{p,m} | m \in M\}},$$

If method m failed to solve problem p the formula above is not defined. In this case we set $r_{p,m} := \infty$.

The cumulative distribution function for the performance ratio is defined as follows:

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

We calculated performance profile of each considered method on the whole set of test problems. These profiles are shown in Figure 6. The value of $\rho_m(\tau)$ gives the probability that method m solves a problem within a ratio τ of the best solver. For example according to Figure 6 level method was the first in more than 30% of cases and solved all the problems within a ratio 6 of the best time.

The notable advantages of performance profiles over other approaches to performance comparison are as follows. Firstly, they minimize the influence of a small subset of problems on the benchmarking process. Secondly, there is no need to discard solver failures. Thirdly, performance profiles provide a visualisation of large sets of test results as we have in our case.

As can be seen from Figure 6, while in most cases the performance of CPLEX barrier optimiser is better it was not able to solve some of the problems. Several large instances were not solved due to high memory requirements of constructing and solving deterministic equivalent. Other failures were caused by numerical difficulties. The performance profile of pure Benders decomposition is dominated by the the level method profile.

Table 13: Performance of DEP solution methods and level regularisation

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
fxm	6	0.06	1259	0.05	17	0.15	20	18417.1
	16	0.22	3461	0.13	23	0.15	20	18416.8
fxmev	1	0.01	273	0.01	14	0.13	20	18416.8
pltexpa	6	0.01	324	0.03	14	0.02	1	-9.47935
	16	0.01	801	0.08	16	0.02	1	-9.66331
stormg2	8	0.08	3649	0.25	28	0.16	20	15535200
	27	0.47	12770	2.27	27	0.31	17	15509000
	125	5.10	70177	8.85	57	0.93	17	15512100
	1000	226.70	753739	137.94	114	6.21	21	15802600
airl-first	25	0.01	162	0.01	9	0.03	17	249102
airl-second	25	0.00	145	0.01	11	0.03	17	269665
airl-randgen	676	0.25	4544	0.05	11	0.22	18	250262
assets	100	0.02	494	0.02	17	0.03	1	-723.839
	37500	1046.85	190774	6.37	24	87.55	2	-695.963
4node	1	0.01	110	0.01	12	0.06	21	413.388
	2	0.01	196	0.01	14	0.10	42	414.013
	4	0.01	326	0.02	17	0.11	45	416.513
	8	0.03	825	0.05	18	0.10	45	418.513
	16	0.06	1548	0.11	17	0.15	44	423.013
	32	0.16	2948	0.40	15	0.22	51	423.013
	64	0.72	7185	0.44	17	0.36	54	423.013
	128	2.30	12053	0.50	26	0.47	50	423.013
	256	7.69	31745	1.05	30	0.87	48	425.375
	512	57.89	57200	2.35	30	2.12	51	429.963
	1024	293.19	133318	5.28	32	3.95	53	434.112
	2048	1360.60	285017	12.44	36	7.82	49	441.738
	4096	-	-	32.67	46	9.12	46	446.856
	8192	-	-	53.82	45	22.68	55	446.856
	16384	-	-	113.20	46	45.24	52	446.856
	32768	-	-	257.96	48	127.86	62	446.856

Table 13: Performance of DEP solution methods and level regularisation (continued)

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
4node-base	1	0.01	111	0.01	11	0.04	16	413.388
	2	0.01	196	0.01	14	0.06	29	414.013
	4	0.01	421	0.02	14	0.07	30	414.388
	8	0.03	887	0.04	15	0.10	35	414.688
	16	0.06	1672	0.11	17	0.10	30	414.688
	32	0.15	3318	0.40	15	0.16	37	416.6
	64	0.49	7745	0.36	13	0.22	33	416.6
	128	1.58	17217	0.33	19	0.35	37	416.6
	256	4.42	36201	0.81	23	0.53	31	417.162
	512	22.44	80941	2.20	29	1.45	37	420.293
	1024	141.91	187231	5.21	32	3.33	41	423.05
	2048	694.89	337082	11.12	32	6.13	42	423.763
	4096	-	-	27.03	37	10.60	39	424.753
	8192	-	-	51.29	40	24.99	48	424.775
	16384	-	-	177.81	73	47.31	41	424.775
	32768	-	-	242.91	48	102.29	49	424.775
4node-old	32	0.20	3645	0.49	18	0.09	20	83094.1
chem	2	0.00	29	0.00	11	0.03	15	-13009.2
chem-base	2	0.00	31	0.00	11	0.05	14	-13009.2
lands	3	0.00	21	0.00	9	0.02	10	381.853
lands-blocks	3	0.00	21	0.00	9	0.02	10	381.853
env-aggr	5	0.01	117	0.01	12	0.04	16	20478.7
env-first	5	0.01	112	0.01	11	0.02	1	19777.4
env-loose	5	0.01	112	0.01	12	0.02	1	19777.4
env	15	0.01	321	0.01	16	0.05	15	22265.3
	1200	1.38	23557	1.44	34	1.73	15	22428.9
	1875	2.90	36567	2.60	34	2.80	15	22447.1
	3780	11.21	73421	7.38	40	5.47	15	22441
	5292	20.28	102757	12.19	42	7.67	15	22438.4
	8232	62.25	318430	-	-	12.58	15	22439.1
	32928	934.38	1294480	-	-	75.67	15	22439.1
env-diss-aggr	5	0.01	131	0.01	9	0.05	22	15963.9
env-diss-first	5	0.01	122	0.01	9	0.04	12	14794.6
env-diss-loose	5	0.01	122	0.01	9	0.03	5	14794.6
env-diss	15	0.01	357	0.02	13	0.10	35	20773.9
	1200	1.96	26158	1.99	50	2.80	35	20808.6
	1875	4.41	40776	3.63	53	4.49	36	20809.3
	3780	16.94	82363	9.32	57	8.87	36	20794.7
	5292	22.37	113894	16.17	66	12.95	38	20788.6
	8232	70.90	318192	-	-	22.49	41	20799.4
	32928	1369.97	1296010	-	-	112.46	41	20799.4
phone1	1	0.00	19	0.01	8	0.02	1	36.9
phone	32768	-	-	50.91	26	48.23	1	36.9

Table 13: Performance of DEP solution methods and level regularisation (continued)

		DEP - Simplex		DEP - IPM		Level		Optimal
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Value
stocfor1	1	0.00	39	0.01	11	0.03	6	-41132
stocfor2	64	0.12	2067	0.08	17	0.12	9	-39772.4
rand0	2000	373.46	73437	9.41	33	6.10	44	162.146
	4000	1603.25	119712	34.28	62	10.06	32	199.032
	6000	-	-	48.84	60	21.17	51	140.275
	8000	-	-	56.89	49	28.86	50	170.318
	10000	-	-	98.51	71	52.31	71	139.129
rand1	2000	-	-	39.97	24	52.70	74	244.159
	4000	-	-	92.71	28	72.30	59	259.346
	6000	-	-	158.24	32	103.00	58	297.563
	8000	-	-	228.68	34	141.81	65	262.451
	10000	-	-	320.10	39	181.98	63	298.638
rand2	2000	-	-	102.61	22	145.22	65	209.151
	4000	-	-	225.71	24	170.08	42	218.247
	6000	-	-	400.52	28	369.35	52	239.721
	8000	-	-	546.98	29	369.01	44	239.158
	10000	-	-	754.52	32	623.59	52	231.706
saphir	50	269.17	84727	-	-	341.86	43	129505000
	100	685.50	152866	-	-	700.44	46	129058000
	200	-	-	549.45	167	-	-	141473000
	500	-	-	-	-	608.48	44	137871000
	1000	-	-	-	-	804.11	46	133036000

Table 14: Performance of decomposition methods

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
fxm	6	0.08	25	0.15	20	0.09	22	0.05	5
	16	0.09	25	0.15	20	0.11	22	0.07	5
fxmev	1	0.08	25	0.13	20	0.08	22	0.05	5
pltexpa	6	0.02	1	0.02	1	0.02	1	0.03	1
	16	0.02	1	0.02	1	0.02	1	0.03	1
stormg2	8	0.14	23	0.16	20	0.08	9	0.10	10
	27	0.47	32	0.31	17	0.18	10	0.23	11
	125	1.73	34	0.93	17	0.50	8	0.89	12
	1000	11.56	41	6.21	21	3.38	6	7.30	11
airl-first	25	0.04	16	0.03	17	0.03	6	0.03	10
airl-second	25	0.02	10	0.03	17	0.02	4	0.03	5
airl-randgen	676	0.22	18	0.22	18	0.22	6	0.29	6
assets	100	0.02	1	0.03	1	0.03	1	0.02	1
	37500	87.68	2	87.55	2	172.23	2	114.38	1

Table 14: Performance of decomposition methods (continued)

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
4node	1	0.03	24	0.06	21	0.03	8	0.03	15
	2	0.04	38	0.10	42	0.02	16	0.05	29
	4	0.04	41	0.11	45	0.03	14	0.05	19
	8	0.07	64	0.10	45	0.03	13	0.05	16
	16	0.11	67	0.15	44	0.04	12	0.05	13
	32	0.23	100	0.22	51	0.05	10	0.07	13
	64	0.27	80	0.36	54	0.08	11	0.12	14
	128	0.39	74	0.47	50	0.15	11	0.19	14
	256	0.95	71	0.87	48	0.20	7	0.29	9
	512	3.72	92	2.12	51	0.46	7	0.62	9
	1024	5.14	70	3.95	53	0.42	3	1.23	10
	2048	11.78	83	7.82	49	1.30	4	1.22	5
	4096	18.46	89	9.12	46	2.79	3	2.03	4
	8192	46.56	106	22.68	55	9.87	3	6.59	4
	16384	99.00	110	45.24	52	38.28	3	27.50	4
	32768	194.68	122	127.86	62	299.85	3	222.61	4
4node-base	1	0.03	31	0.04	16	0.03	21	0.03	14
	2	0.04	44	0.06	29	0.03	19	0.05	19
	4	0.06	58	0.07	30	0.04	20	0.07	34
	8	0.05	47	0.10	35	0.04	19	0.08	28
	16	0.08	56	0.10	30	0.06	21	0.11	28
	32	0.17	63	0.16	37	0.07	13	0.18	22
	64	0.23	61	0.22	33	0.17	19	0.30	21
	128	0.39	65	0.35	37	0.34	19	0.63	23
	256	0.89	66	0.53	31	0.45	11	1.81	26
	512	3.27	84	1.45	37	1.84	14	4.98	29
	1024	9.57	115	3.33	41	5.53	13	9.17	17
	2048	19.72	142	6.13	42	21.82	13	31.08	21
	4096	38.51	174	10.60	39	85.68	12	146.50	18
	8192	133.45	290	24.99	48	354.05	14	-	-
	16384	164.07	175	47.31	41	1430.72	13	-	-
	32768	314.31	191	102.29	49	-	-	-	-
4node-old	32	0.08	30	0.09	20	0.04	7	0.09	10
chem	2	0.04	7	0.03	15	0.03	13	0.04	19
chem-base	2	0.02	6	0.05	14	0.02	13	0.04	22
lands	3	0.02	8	0.02	10	0.02	5	0.03	17
lands-blocks	3	0.01	8	0.02	10	0.02	5	0.03	17
env-aggr	5	0.02	3	0.04	16	0.02	3	0.03	5
env-first	5	0.02	1	0.02	1	0.02	1	0.02	1
env-loose	5	0.01	1	0.02	1	0.02	1	0.02	1
env	15	0.04	3	0.05	15	0.03	3	0.03	5
	1200	0.34	3	1.73	15	0.48	3	0.76	5
	1875	0.57	3	2.80	15	0.90	3	1.50	5
	3780	1.26	3	5.47	15	2.48	3	3.79	5
	5292	1.96	3	7.67	15	4.51	3	5.89	5
	8232	3.70	3	12.58	15	10.67	3	12.54	5
	32928	39.88	3	75.67	15	211.90	3	212.05	5

Table 14: Performance of decomposition methods (continued)

		Benders		Level		TR		RD	
Name	Scen	Time	Iter	Time	Iter	Time	Iter	Time	Iter
env-diss-aggr	5	0.03	9	0.05	22	0.03	9	0.03	17
env-diss-first	5	0.02	14	0.04	12	0.02	4	0.03	4
env-diss-loose	5	0.03	15	0.03	5	0.02	4	0.02	4
env-diss	15	0.05	27	0.10	35	0.05	18	0.07	12
	1200	1.13	24	2.80	35	2.25	18	3.45	19
	1875	2.50	29	4.49	36	5.52	19	4.52	15
	3780	5.04	29	8.87	36	20.23	19	8.98	11
	5292	8.14	34	12.95	38	40.39	17	17.90	13
	8232	14.21	35	22.49	41	119.88	16	99.19	23
	32928	79.52	35	112.46	41	-	-	-	-
phone1	1	0.02	1	0.02	1	0.02	1	0.02	1
phone	32768	48.34	1	48.23	1	73.45	1	73.75	1
stocfor1	1	0.02	6	0.03	6	0.02	2	0.02	2
stocfor2	64	0.10	7	0.12	9	0.18	14	0.23	18
rand0	2000	10.42	80	6.10	44	30.33	9	93.78	16
	4000	19.97	69	10.06	32	82.75	8	591.45	14
	6000	41.82	108	21.17	51	275.97	9	-	-
	8000	65.51	127	28.86	50	423.51	9	-	-
	10000	153.07	230	52.31	71	871.00	10	-	-
rand1	2000	265.14	391	52.70	74	155.81	12	361.54	17
	4000	587.22	502	72.30	59	508.18	11	-	-
	6000	649.58	385	103.00	58	937.74	11	-	-
	8000	917.24	453	141.81	65	1801.43	9	-	-
	10000	1160.62	430	181.98	63	-	-	-	-
rand2	2000	1800.00	818	145.22	65	334.36	12	794.31	17
	4000	1616.56	414	170.08	42	813.49	11	-	-
	6000	-	-	369.35	52	-	-	-	-
	8000	-	-	369.01	44	-	-	-	-
	10000	-	-	623.59	52	-	-	-	-
saphir	50	733.37	128	341.86	43	578.87	110	-	-
	100	1051.89	123	700.44	46	-	-	-	-
	200	-	-	-	-	-	-	-	-
	500	1109.48	122	608.48	44	1283.97	99	-	-
	1000	1444.17	124	804.11	46	-	-	-	-

E.4 Choice of IPM Solver

In order to justify the use of CPLEX IPM in our experiments we compared its performance to another state-of-the-art interior point solver. The second solver, HOPDM (Gondzio, 1995; Colombo and Gondzio, 2008), is an implementation of the infeasible primal-dual interior point method. Depending on the problem, it uses either the normal equations with Schur complement or augmented system factorisation.

The results summarised in Table 15 show that while it took HOPDM on average less iterations to solve a problem, CPLEX was faster in our benchmarks. This can be explained by

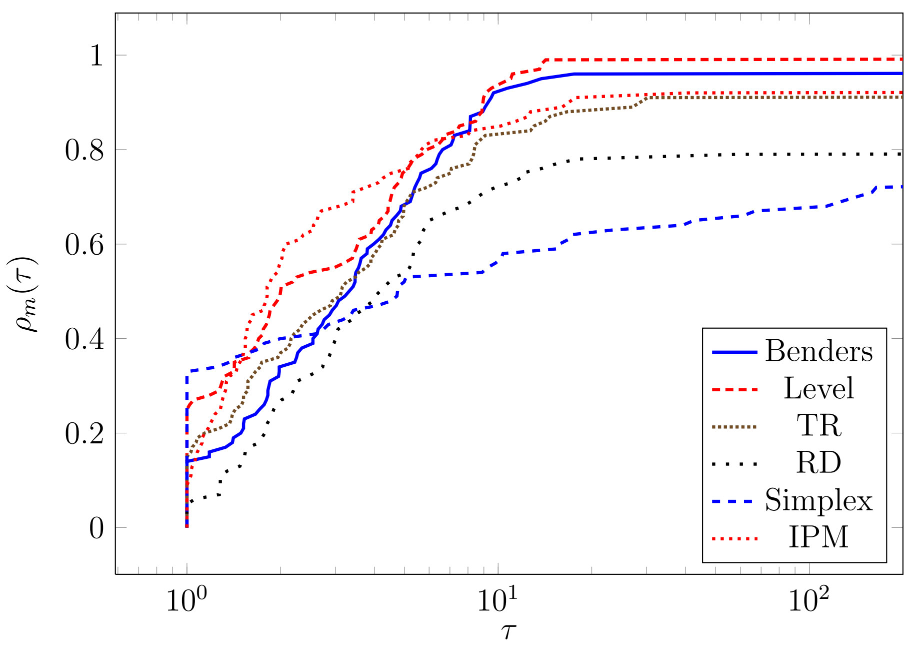


Figure 6: Performance profiles

the latter being better optimised to the underlying hardware. In particular, CPLEX uses high performance Intel Math Kernel Library which is tuned for the hardware we were using in the tests.

Table 15: Summary of CPLEX and HOPDM performance

	CPLEX	HOPDM
Average Iterations	29	21
Average Time	56.66	170.50
Solved Problems	87	78

E.5 Comments on Scale-Up Properties and on Accuracy

We performed a set of experiments recording the change in the gap between lower and upper bounds on objective function in the decomposition methods. The results are shown in Figures 7 – 10.

The computational results given in the previous section where obtained using the relative stopping tolerance $\varepsilon = 10^{-5}$ for the Benders decomposition with and without regularisation by the level method, i.e. the method terminated if $(z^* - z_*)/(|z_*| + 10^{-10}) \leq \varepsilon$, where z_* and z^* are, respectively, lower and upper bounds on the value of the objective function. The stopping criteria in the trust region algorithm and regularised decomposition are different because these methods do not provide global lower bound. Therefore ε was set to a lower value of 10^{-6} with the following exceptions that were made to achieve the desirable precision:

- env-diss with 8232 scenarios: $\varepsilon = 10^{-10}$ in RD,
- saphir: $\varepsilon = 10^{-10}$ in RD and TR.

For CPLEX barrier optimiser the default complementarity tolerance was used as a stopping criterion.

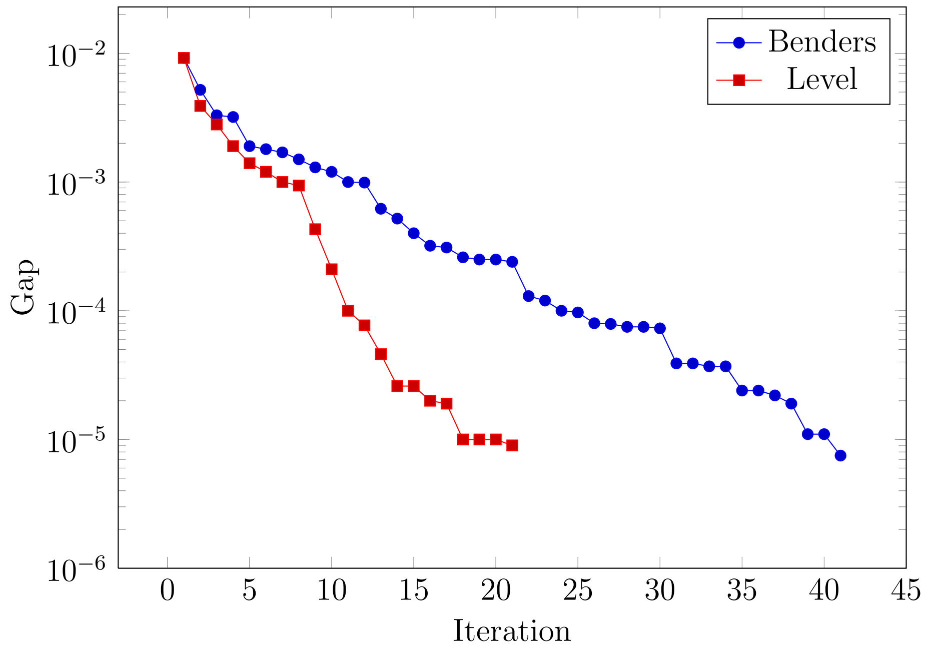


Figure 7: Gap between lower and upper bounds for storm-1000 problem

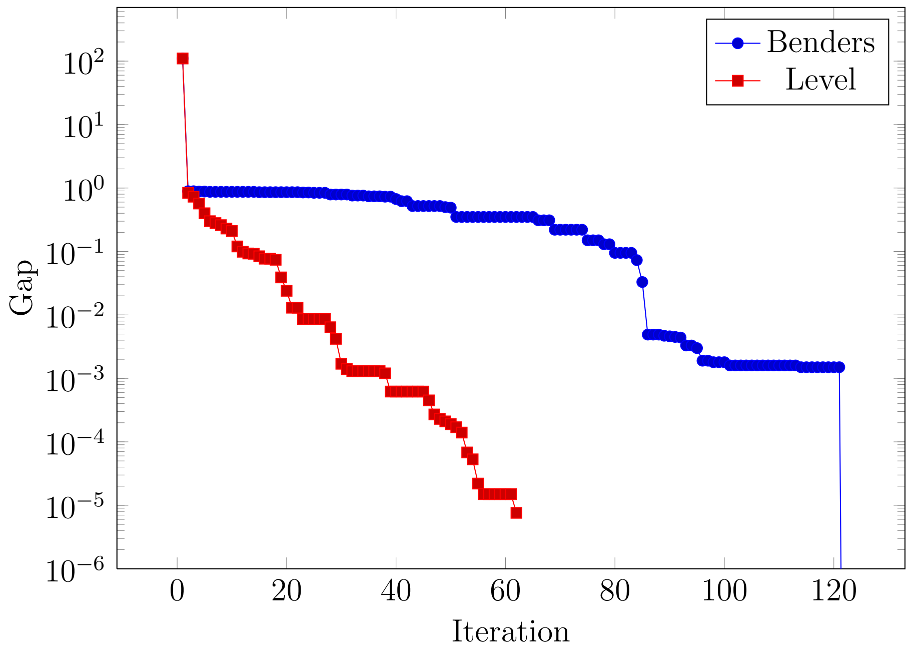


Figure 8: Gap between lower and upper bounds for 4node-32768 problem

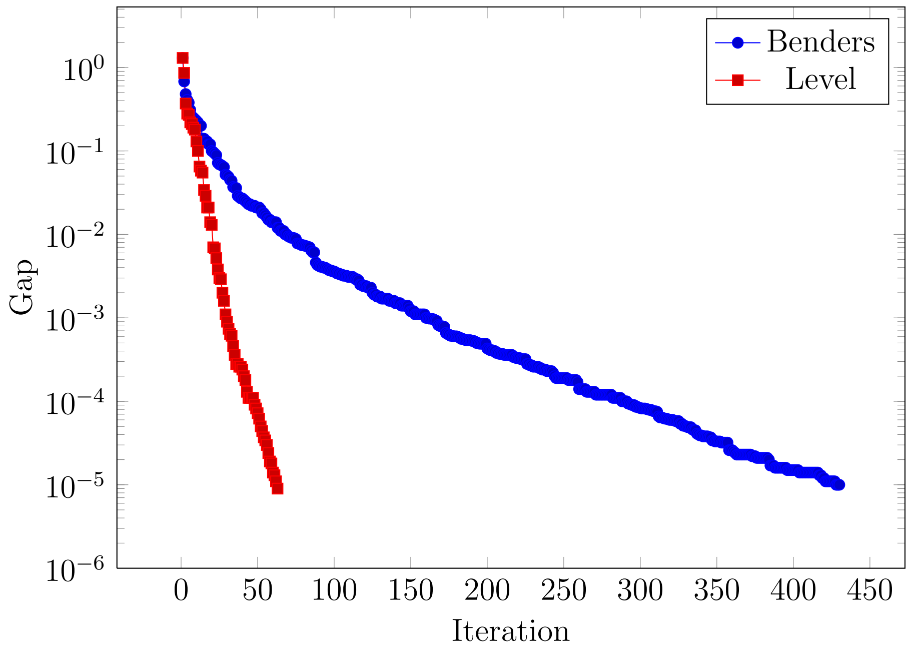


Figure 9: Gap between lower and upper bounds for rand1-10000 problem

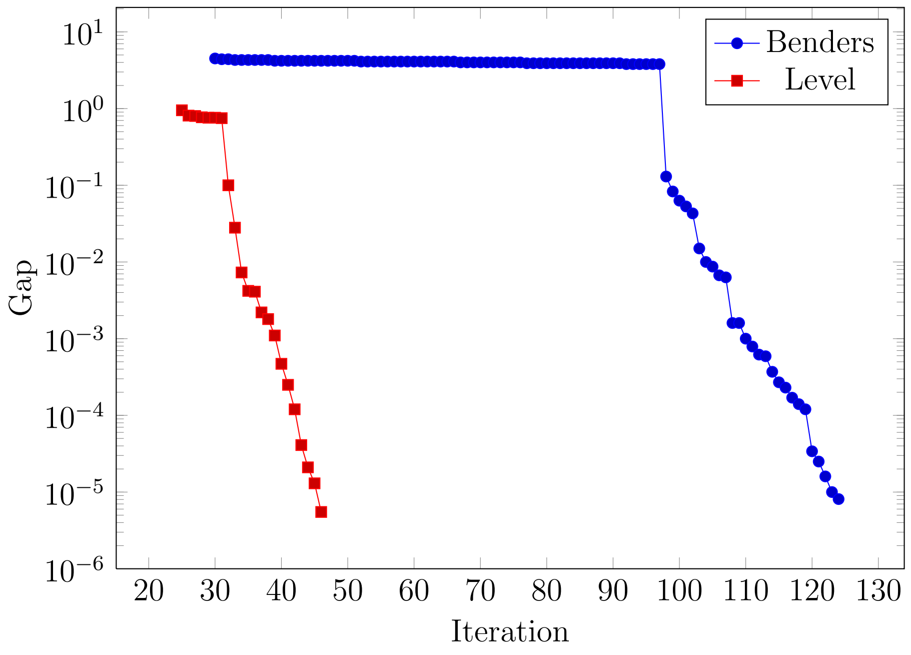


Figure 10: Gap between lower and upper bounds for saphir-1000 problem

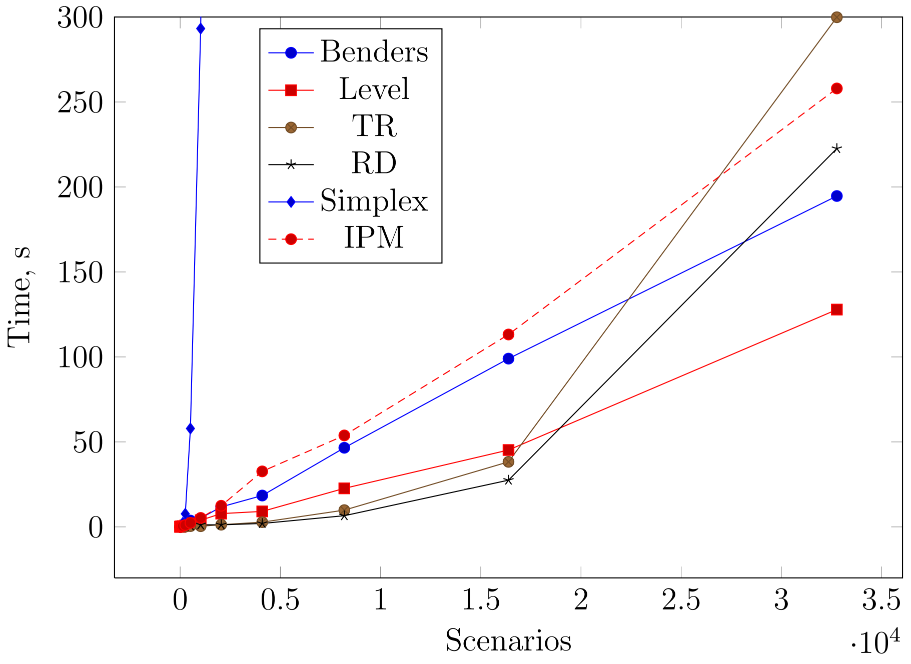


Figure 11: Time vs the number of scenarios on the 4node problems