AMPL representation and solution of multiple stochastic programming formulations

Christian Valente, Victor Zverovich, Gautam Mitra, Robert Fourer
Agenda

• Classification of problems of interest
• Our approach to modelling
• AMPLsp
• Conclusions
About us..

• In the past years, **OptiRisk Systems** has been working closely with AMPL Inc. and has developed various products in the AMPL ecosystem
  – AMPL Studio (graphical interface)
  – AMPLCOM (library)
  – SPIne and SAMPL (extensions to AMPL)
  – FortSP (decomposition based solver)
  – AMPLDev (graphical interface)
  – AMPL API and AMPL IDE (as contractors)
Taxonomy of optimisation problems under uncertainty revised

- SP Problems
  - Distribution Problems
  - Recourse Problems
    - Problems with Chance Constraints
    - Problems with ICC
  - Stochastic Measures (e.g. EVPI and VSS)
- Wait and See
- Expected Value
- Two-stage
- Multi-stage

Introduction
Classification
Our approach to SP
AMPLsp
Conclusions
We concentrate on Two-Stage SP problems, with (integrated) chance constraints.
Model classes

- Expected Value
  \[ Z_{EV} \overset{\text{def}}{=} \min \bar{c}x \]
  where \( x \in \bar{F} \)
  and \( \bar{F} = \{ x | A x = \bar{b}, x \geq 0 \} \)

- Wait and See
  \[ Z_{WS} \overset{\text{def}}{=} E_{\omega} \left[ Z^{(\omega)} \right] \]
  where \( Z^{(\omega)} = \min c^{(\omega)}x \)
  and \( x \in F^{(\omega)} \)

- Here and now
  \[ Z_{HN} \overset{\text{def}}{=} \min \left[ E_{\omega}^x \left[ Q(x^{(\omega)}, \omega) \right] \right] \]
  subject to \( A x = \bar{b} \)
  \( x \geq 0 \)
  where \( x \in F \) and \( F = \bigcap f^{(\omega)} \)
  \[ D^{(\omega)} y^{(\omega)} = d^{(\omega)} + B^{(\omega)} x \]
  \( y^{(\omega)} \geq 0 \)
Chance Constraints

- Algebraic formulation:
  - Individual Chance Constraints
    \[ P(h_i(x, \xi) \geq 0) \geq p_i, \quad i \in I \]
  - Joint Chance Constraints
    \[ P((h_i(x, \xi) \geq 0, i \in I) \geq p \]
  where \( x \) and \( \xi \) are respectively decisions and random vectors, \( I \) is a set of indices of constraints in the given problem
Chance Constraints

- Practical Importance
  - Chance constraints provide a simple risk measure
  - Related to VaR
  - Applications in finance, energy production, water management, ...

- Can be expressed in any AMLs reformulating the problem by introducing extra constraints and binary variables
Integrated Chance Constraints

- Expected violation of constraint(s) $\leq$ shortfall $\beta_i$
- Individual ICC
  \[ E_\omega \left[ \eta_i(x, \omega)^- \right] \leq \beta_i, \quad \beta_i \geq 0, i \in I \]
- Joint ICC
  \[ E_\omega \left[ \max_{i \in I} \eta_i(x, \omega)^- \right] \leq \beta_i, \quad \beta_i \geq 0 \]

where $\eta_i(x, \omega)^-$ represents the violation that occurs in constraint $i$ under realisation $\omega$
Integrated Chance Constraints

• Practical Importance
  – ICCs represent a risk measure (closely related to CVaR or to SSD)
  – Computationally more tractable than chance constraints
  – Applications in finance, e.g. asset-liability management, portfolio choice models

• Can be expressed in any AML introducing extra constraints and continuous variables
Our approach to optimisation (under uncertainty)

• Maintain separation between the activities in optimisation:
  – Modelling
  – Instance generation
  – Solving

• Benefits
  – Easier specification of the algebraic model
  – Modularity makes software easier to maintain
  – Specialists can work in their own domain
SP Modelling process

**Classification**

**Introduction**

**Our approach to SP**

**AMPLsp**

**Conclusions**

**Predictive modelling**
- Which scenario generators?
- Model(s) of randomness

**Decision modelling**
- Which decision model?
- Decision Model

**Solution algorithms**
- Which solution method?
- Solution algorithm

**Runtime phase**
- SG Parameters
- Algebraic model
- Instance data
- Solver Controls

**Technical choices**
- Programming language
- Modelling language
- Instance format
- Solver to use

**Modelling language**
- Standalone vs library

**Solver to use**
Our approach to modelling

AML
- How to define the model at algebraic level

AMS
- What modelling system to use

Instance level format
- How to represent the model instance

Algebraic model
Modelling System
Instance representation

AMLsp
Conclusions
SP Instance representation

- SP problems have a specific block structure

\[
\begin{align*}
\min & \quad cx + c_{s1}y_{s1} + c_{s2}y_{s2} \\
Ax & \leq b \\
A_{s1,1}x + A_{s1,2}y_{s1} & \leq b_{s1} \\
A_{s2,1}x + A_{s2,2}y_{s2} & \leq b_{s2}
\end{align*}
\]

- When passed to a solver as a deterministic equivalent, this structure is lost

\[
\begin{align*}
\min & \quad \tilde{c}z \\
\tilde{A}z & \leq \tilde{b}
\end{align*}
\]

Where \(\tilde{A}, \tilde{b}, \tilde{c}\) and \(z\) are compositions of the respective vectors/matrices.
SP Instance Representation

• The structure can be exploited by solution algorithms

• At instance level we aim to communicate:

\[
\begin{align*}
\min & \quad cx + \tilde{c}y \\
A & \quad x \leq b \\
\tilde{A}_1 & \quad x + \tilde{A}_2y \leq \tilde{b}
\end{align*}
\]

\[
\tilde{c} = c_{s1}, c_{s2} \\
b = b_{s1}, b_{s2} \\
\tilde{A}_1 = A_{S1,1}, A_{S2,1} \\
\tilde{A}_2 = A_{S1,2}, A_{S2,2}
\]

where the tilde submatrices are then separately passed, scenario by scenario.

• Most of the elements in the sub-blocks are repeated -> only changes in respect to the tilde matrices are communicated
SP Instance Representation

- A well specified language for instance level representation has already been proposed and is used (SMPS)
- To be able to generate such format, the modelling system must be told the structure of the model we wish it to convey
- Following slides show our past and current approaches at this
Our approach to modelling (1)

### Algebraic model

<table>
<thead>
<tr>
<th>AML</th>
<th>AMS</th>
<th>MPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any modelling language, no specialized syntax (DEQ formulation)</td>
<td>Any modelling system for linear/non-linear optimisation</td>
<td>MPS like format (direct representation of the DEQ formulation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instance representation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **AML (Any Modelling Language)**: Any modelling language, no specialized syntax (DEQ formulation).
- **AMS (Any Modelling System)**: Any modelling system for linear/non-linear optimisation.
- **MPS (MPS like format)**: MPS like format (direct representation of the DEQ formulation). Additional disadvantages:
  - Replication of information
  - Loss of structure
<table>
<thead>
<tr>
<th>SAMPL</th>
<th>SPIfE</th>
<th>SMPS</th>
</tr>
</thead>
</table>
| - Specialized syntax for SP | - Preprocessor at algebraic language level  
- Generated a core model and the needed information using AMPL as a subsystem  
- Expressed model != solved model | - SMPS like format (compact representation, conveys the stochastic information separately) |
Our approach to modelling (3)

**Algebraic model**

**Modelling System**

**Instance representation**

**SAMPL**
- Specialized syntax for SP

**SAMPL**
- Reimplementation of AMPL
- Generated model efficiently
- Development independent from AMPL

**SMPS**
- SMPS like format (compact representation, conveys the stochastic information separately)
Our approach to modelling (4)

- **AMPLsp**
  - AMPL formulation following some guidelines

- **AMPL + smpswriter**
  - Official AMPL interpreter
  - Uses the solver module `smpswriter` to efficiently generate smps

- **SMPS**
  - SMPS like format (compact representation, conveys the stochastic information separately)
From SAMPL to AMPLsp

• We have been developing SAMPL, an extended version of AMPL with additional language constructs and models communication facilities.

• Focus of the language was:
  – Easy formulation of the classes of problems presented (e.g. no artificial variables for (I)CCP)
  – Efficient model instance generation
  – Efficient model solution: instances generated in SMPS-like format, which conveys the model structure, exploitable by various specialised algorithms
From SAMPL to AMPLsp

• SAMPL has always been separated from AMPL, first implemented as a pre-processor then as an alternative language interpreter
  – Two development teams and efforts
  – Not all AMPL facilities were implemented
  – Sync with new AMPL features

• **SAMPL is discontinued**, to be replaced by
  – AMPL with an intelligent reuse of existing constructs
  – smpswriter [https://github.com/ampl/mp](https://github.com/ampl/mp) (a new solver interface, able to write SMPS files)
Dakota model (deterministic)

set PROD;
set RESOURCE;

param Cost{RESOURCE};

param ProdReq{RESOURCE,PROD};

param Price{PROD};

param Budget;

param Demand{PROD};

var amountbuy{RESOURCE} >=0 ;

var amountprod{PROD}>=0, suffix stage 2;

var amountsell{PROD}>=0, suffix stage 2;

maximize wealth:
    sum{p in PROD} Price[p]*amountsell[p]-
    sum{r in RESOURCE} Cost[r]*amountbuy[r];
Dakota model (deterministic)

subject to

CBudget: \( \sum_{r \in \text{RESOURCE}} \text{Cost}[r] \cdot \text{amountbuy}[r] \leq \text{Budget} \);

CBalance\( \{r \in \text{RESOURCE}\} \):
\[
\text{amountbuy}[r] \geq \sum_{p \in \text{PROD}} \text{ProdReq}[r,p] \cdot \text{amountprod}[p];
\]

CProduction\( \{p \in \text{PROD}\} \):
\[
\text{amountsell}[p] \leq \text{amountprod}[p];
\]

CSales\( \{p \in \text{PROD}\} \):
\[
\text{amountsell}[p] \leq \text{Demand}[p];
\]
Dakota model (stochastic)

- The implementation is designed to have minimal impact on AMPL by reusing the representational power of the *nl* format and a few conventions.

- Preliminary declarations/conventions:
  
  ```
  function expectation;
  function random;
  suffix stage IN;
  ```

- Add scenario set and appropriate indexing to represent realizations:
  ```
  set SCEN;
  param Demand{PROD, SCEN};
  ```
Dakota model (stochastic)

- For every occurrence of the random parameter in the model we pass a placeholder.
- Parameter **Demand** becomes a variable (in the sense that its value will be determined after AMPL generates the model instance).

\[
\text{param Demand\{PROD\};} \quad \rightarrow \quad \text{var RandomDemand\{PROD\};}
\]

- An AMPL function allows the smpswriter to link the parameter values to its placeholder.

\[
\text{yield: random\{p in PROD\} (Demand[p],}
\quad \{s in SCEN\} \text{ RandomDemand[p,s])};
\]
Dakota model (stochastic)

- **Stage assignment**
  
  ```ampl
  var amountbuy{RESOURCE} >=0 ;
  var amountprod{PROD}>=0, suffix stage 2;
  var amountsell{PROD}>=0, suffix stage 2;
  ```

- **Objective as expectation**
  
  ```ampl
  maximize wealth:
  expectation(sum{p in PROD} Price[p]*amountsell[p])
  - sum{r in RESOURCE} Cost[r]*amountbuy[r];
  ```
Complete model

function expectation;
function random;
suffix stage IN;

set PROD;
set RESOURCE;
set SCEN;
param RandomDemand{PROD, SCEN};
var Demand{PROD};
yield: random({p in PROD} (RandomDemand[p],
    {s in SCEN} Demand[p,s]));
param Cost{RESOURCE};
param ProdReq{RESOURCE, PROD};
param Price{PROD};
param Budget;

var amountbuy{RESOURCE} >=0 ;
var amountprod{PROD}>=0, suffix stage 2;
var amountsell{PROD}>=0, suffix stage 2;
Complete model

\[
\text{maximize wealth :} \\
\text{expectation (sum\{p in PROD\} Price[p]*amountsell[p])} - \\
\text{sum\{r in RESOURCE\} Cost[r]*amountbuy[r];}
\]

subject to

\[\text{CBudget: sum\{r in RESOURCE\} Cost[r]*amountbuy[r] <= Budget;}\]

\[\text{CBalance\{r in RESOURCE\}: amountbuy[r] >= sum\{p in PROD\} ProdReq[r,p] * amountprod[p];}\]

\[\text{CProduction\{p in PROD\}: amountsell[p] <= amountprod[p];}\]

\[\text{CSales\{p in PROD\}: amountsell[p] <= RandomDemand[p];}\]
Process

Algebraic Model

AMPL

nl file

smpswriter

smps file

solver

Reading time = 0.015569 s.
Stage 1 has 1 row(s), 3 column(s), and 3 nonzero(s).
Stage 2 has 9 row(s), 6 column(s), and 21 nonzero(s).
Problem has 2 stage(s) and 3 scenario(s).

<table>
<thead>
<tr>
<th>Itn</th>
<th>Objective</th>
<th>Bound</th>
<th>Rel.Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1281.63</td>
<td>4577.78</td>
<td>2.57184</td>
</tr>
<tr>
<td>2</td>
<td>1281.63</td>
<td>4120</td>
<td>2.21466</td>
</tr>
<tr>
<td>3</td>
<td>1281.63</td>
<td>4120</td>
<td>2.21466</td>
</tr>
<tr>
<td>4</td>
<td>1281.63</td>
<td>2589.71</td>
<td>1.02064</td>
</tr>
<tr>
<td>5</td>
<td>1580.19</td>
<td>2446.55</td>
<td>0.54826</td>
</tr>
<tr>
<td>6</td>
<td>1580.19</td>
<td>2266.71</td>
<td>0.434451</td>
</tr>
<tr>
<td>7</td>
<td>1580.19</td>
<td>1790.67</td>
<td>0.133201</td>
</tr>
<tr>
<td>8</td>
<td>1580.19</td>
<td>1711.66</td>
<td>0.0831986</td>
</tr>
<tr>
<td>9</td>
<td>1580.19</td>
<td>1655.78</td>
<td>0.0478343</td>
</tr>
<tr>
<td>10</td>
<td>1580.19</td>
<td>1616.67</td>
<td>0.0230834</td>
</tr>
<tr>
<td>11</td>
<td>1616.67</td>
<td>1616.67</td>
<td>-2.81287e-016</td>
</tr>
</tbody>
</table>

Number of iterations = 11.
Master time = 0 s.
Recourse time = 0 s.
Optimal solution found, objective = 1616.67.

shell 'smpswriter.model';

shell 'fortsp model';
Availability

• Source code is available on github as part of the project ampl/mp: An open-source library for mathematical programming

  https://github.com/ampl/mp

• Find details in solvers/smparser
Further developments

• Syntax
  – Streamline the initial declarations
  – Improve the procedure to define a random parameter

• Functionalities
  – Incorporate CPPs and ICCPs

• Integration
  – Implement the smpswriter as a standard AMPL solver (no need to system calls)

• Variables
  – Allow second stage variables to be indexed over the scenario set
Conclusions

• Formulating SP problems with DEQ does not scale up
  - Spatial complexity
  - Computational complexity

• Efficient SP model generation and solution needs special tools in
  - Modelling – generating model instance
  - Solving – using specialised algorithms

• New implementation doesn’t suffer of the inherent problems of the previous
References