How to solve mathematical programs

- Use a mathematical workbench like MATLAB, MATHEMATICA, MAPLE, R.

- Use a modeling language to convert the theoretical model to a computer usable representation and employ an out-of-the-box general solver to find solutions.

- Use a framework that already has many general algorithms available and only implement problem specific parts, e.g., separators or upper bounding.

- Develop everything yourself, maybe making use of libraries that provide high-performance implementations of specific algorithms.

Thorsten Koch
“Rapid Mathematical Programming”
Technische Universität, Berlin, Dissertation, 2004
How to solve mathematical programs

▶ Use a mathematical workbench like MATLAB, MATHEMATICA, MAPLE, R.

Advantages: easy if familiar with the workbench
Disadvantages: restricted, not state-of-the-art

How to solve mathematical programs

▶ Use a modeling language to convert the theoretical model to a computer usable representation and employ an out-of-the-box general solver to find solutions.

Advantages: flexible on modeling side, easy to use, immediate results, easy to test different models, possible to switch between different state-of-the-art solvers
Disadvantages: algoritmical restrictions in the solution process, no upper bounding possible

How to solve mathematical programs

▶ Use a framework that already has many general algorithms available and only implement problem specific parts, e.g., separators or upper bounding.

Advantages: allow to implement sophisticated solvers, high performance bricks are available, flexible
Disadvantages: view imposed by designers, vendor specific hence no trans-ferability,

How to solve mathematical programs

▶ Develop everything yourself, maybe making use of libraries that provide high-performance implementations of specific algorithms.

Advantages: specific implementations and max flexibility
Disadvantages: for extremely large problems, bounding procedures are more crucial than branching
Modeling Languages

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Solver</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIMMS</td>
<td><a href="http://www.aimms.com">www.aimms.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>AMPL</td>
<td><a href="http://www.ampl.com">www.ampl.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>GAMS</td>
<td><a href="http://www.gams.com">www.gams.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>LINGO</td>
<td><a href="http://www.lindo.com">www.lindo.com</a></td>
<td>fixed</td>
<td>commercial</td>
</tr>
<tr>
<td>LPL (Linear Programming Language)</td>
<td><a href="http://www.virtual-optima.com">www.virtual-optima.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>MINOPT</td>
<td><a href="http://www.optimization.anl/nminopt">www.optimization.anl/nminopt</a></td>
<td>open</td>
<td>mixed</td>
</tr>
<tr>
<td>MIP (Mathematical Programming Language)</td>
<td><a href="http://www.zib.de/optimization/soplex">www.zib.de/optimization/soplex</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>OMNI</td>
<td><a href="http://www.omega.com">www.omega.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>OPL (Optimization Programming Language)</td>
<td><a href="http://www.mpe.com">www.mpe.com</a></td>
<td>open</td>
<td>commercial</td>
</tr>
<tr>
<td>GNU-MP</td>
<td><a href="http://www.gnu.org/software/glpk">www.gnu.org/software/glpk</a></td>
<td>fixed</td>
<td>free</td>
</tr>
<tr>
<td>ZIMPL</td>
<td><a href="http://www.zib.de/koch/zimpl">www.zib.de/koch/zimpl</a></td>
<td>open</td>
<td>free</td>
</tr>
</tbody>
</table>

Thorsten Koch
“Rapid Mathematical Programming”
Technische Universität, Berlin, Dissertation, 2004

Outline

1. An Overview of Software for MIP

2. ZIBOpt

ZIBOpt

- Zimpl is a little algebraic Modeling language to translate the mathematical model of a problem into a linear or (mixed-) integer mathematical program expressed in .lp or .mps file format which can be read and (hopefully) solved by a LP or MIP solver.

- Scip is an IP-Solver. It solves Integer Programs and Constraint Programs: the problem is successively divided into smaller subproblems (branching) that are solved recursively. Integer Programming uses LP relaxations and cutting planes to provide strong dual bounds, while Constraint Programming can handle arbitrary (non-linear) constraints and uses propagation to tighten domains of variables.

- SoPlex is an LP-Solver. It implements the revised simplex algorithm. It features primal and dual solving routines for linear programs and is implemented as a C++ class library that can be used with other programs (like SCIP). It can solve standalone linear programs given in MPS or LP-Format.
Callable libraries
How to construct a problem instance in SCIP

- SCIPcreate(), // create a SCIP object
- SCIPcreateProb() // build the problem
- SCIPcreateVar() // create variables
- SCIPaddVar() // add them to the problem
- SCIPcreateConsLinear(),
- SCIPaddConsLinear()
- SCIPreleaseCons() // after finishing.
- SCIPsolve()
- SCIPreleaseVar() releases variable pointers
- SCIP_CALL() // exception handling

SCIPsetIntParam(scip, "display/memused/status", 0) == set display \nmemused status 0
SCIPprintStatistics() == display statistics

Sudoku into Exact Hitting Set

Exact Covering: Set partitioning with \( \vec{c} = \vec{1} \)

- \( \vec{A} = 1, 4, 7; \)
- \( \vec{B} = 1, 4; \)
- \( \vec{C} = 4, 5, 7; \)
- \( \vec{D} = 3, 5, 6; \)
- \( \vec{E} = 2, 3, 6, 7; \)
- \( \vec{F} = 2, 7. \)

\[
\begin{array}{cccccc}
A & B & C & D & E & F \\
\hline
\text{min} & & & & & \\
1 & 1 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 & 1 & 1 \\
4 & 1 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 1 & 1 & 0 \\
6 & 0 & 0 & 0 & 1 & 1 \\
7 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

The dual of Exact Covering is the Exact Hitting Set

- \( \vec{G} = 1, 3, 5, 6. \)

\[
\begin{array}{cccccc}
A & B & C & D & E & F \\
\hline
\text{max} & & & & & \\
1 & 1 & 0 & 0 & 0 & 1 \\
2 & 1 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 1 & 1 & 0 & 0 \\
6 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]