The SYMPHONY Callable Library for Mixed-Integer Linear Programming

Ted Ralphs and Menal Guzelsoy
Industrial and Systems Engineering
Lehigh University

INFORMS Computing Society Conference, Annapolis, MD, Thursday, January 6, 2005
Outline of Talk

• Introduction to SYMPHONY

• A little bit of theory
  – Duality
  – Sensitivity analysis
  – Warm starting
  – Parametric analysis

• A little bit of computation
  – Implementation in SYMPHONY
  – Examples
  – Computational results
A Really Brief Overview of SYMPHONY

- SYMPHONY is an open-source software package for solving and analyzing mixed-integer linear programs (MILPs).

- SYMPHONY can be used in three distinct modes.
  - **Black box solver**: Solve generic MILPs (command line or shell).
  - **Callable library**: Call SYMPHONY from a C/C++ code.
  - **Framework**: Develop a customized solver or callable library.

- Available as part of the [Computational Infrastructure for Operations Research (COIN-OR)](http://www.coin-or.org) (www.coin-or.org).

- Packaged releases available for download on [www.branchandcut.org](http://www.branchandcut.org).

- The new interface and features of SYMPHONY give it the look and feel of an LP solver.

- This talk will focus on these new features—for detailed information on using SYMPHONY, please attend yesterday’s SYMPHONY tutorial :).
A Really Brief Introduction to Duality

- For an optimization problem

\[ z = \min \{ f(x) \mid x \in X \}, \]

called the \textit{primal problem}, an optimization problem

\[ w = \max \{ g(u) \mid u \in U \} \]

such that \( w \leq z \) is called a \textit{dual problem}.

- It is a \textit{strong dual} if \( w = z \).

- Uses for the dual problem
  - Bounding
  - Deriving optimality conditions
  - Sensitivity analysis
  - Warm starting
Some Previous Work

- R. Gomory (and W. Baumol) ('60–'73)
- G. Roodman ('72)
- E. Johnson (and Burdet) ('72–'81)
- R. Jeroslow (and C. Blair) ('77-'85)
- A. Geoffrion and R. Nauss ('77)
- D. Klein and S. Holm ('79–'84)
- L. Wolsey (and L. Schrage) ('81–'84)
- ...
- D. Klabjan ('02)
Duals for ILP

• Let $\mathcal{P} = \{ x \in \mathbb{R}^n \mid Ax = b, x \geq 0 \}$ nonempty for $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$.

• We consider the (bounded) pure integer linear program $\min_{x \in \mathcal{P} \cap \mathbb{Z}^n} c^\top x$ for $c \in \mathbb{R}^n$.

• The most common dual for this ILP is the well-known Lagrangian dual.
  
  – The Lagrangian dual is not generally strong.
  – Blair and Jeroslow discussed how to make the Lagrangian dual strong by for ILP by introducing a quadratic penalty term.

• How do we derive a strong dual? Consider the following more formal notion of dual (Wolsey).

$$
\begin{align*}
  w_{IP}^g & = \max_{g : \mathbb{R}^m \to \mathbb{R}} \{ g(b) \mid g(Ax) \leq c^\top x, x \geq 0 \} \\
  & = \max_{g : \mathbb{R}^m \to \mathbb{R}} \{ g(b) \mid g(d) \leq z_{IP}(d), d \in \mathbb{R}^m \},
\end{align*}
$$

where $z_{IP}(d) = \min_{x \in \mathcal{P}^I(d)} c^\top x$ is the value function and $\mathcal{P}^I(d) = \{ x \in \mathbb{Z}^n \mid Ax = d, x \geq 0 \}$.
Dual Solutions from Primal Algorithms

- Sensitivity analysis and warm starting procedures for LP are based on optimality conditions arising from LP duality.

- The optimal basis contains all the information needed to construct optimal primal and dual solutions.

- This information can be obtained as a by-product of the primal simplex algorithm.

- We extend this to ILP by considering the implicit optimality conditions associated with branch and bound.
Dual Solutions for ILP from Branch and Bound

• An extension of the optimality conditions for LP to ILP is straightforward.
• Let $\mathcal{P}_1, \ldots, \mathcal{P}_s$ be a partition of $\mathcal{P}$ into (nonempty) subpolyhedra.
• Let $LP_i$ be the linear program $\min_{x^i \in \mathcal{P}_i} c^T x^i$ associated with the subpolyhedron $\mathcal{P}_i$.
• Let $B^i$ be an optimal basis for $LP_i$.
• Then the following is a valid lower bound

$$L = \min \{c_{B^i}(B^i)^{-1}b + \gamma_i | 1 \leq i \leq s\},$$

where $\gamma_i$ is the constant factor associated with the nonbasic variables fixed at nonzero bounds.

• A similar function yields an upper bound.
• A partition that yields equal lower and upper bounds is called an optimal partition.
Sensitivity Analysis for ILP

- The function

\[ L(d) = \min\{c_{B_i}(B^i)^{-1}d + \gamma_i \mid 1 \leq i \leq s\}, \]

provides an optimal solution to (2).

- The corresponding upper bounding function is

\[ U(c) = \min\{c_{B_i}(B^i)^{-1}b + \beta_i \mid 1 \leq i \leq s, \hat{x}^i \in P^I\} \]

- These functions can be used for local sensitivity analysis, just as one would do in linear programming.
  - For changes in the right-hand side, the lower bound remains valid.
  - For changes in the objective function, the upper bound remains valid.
  - One can also add cuts and variables.

- One can compute an “allowable range” for changes to the instance data. as the intersection of the ranges for each member of the partition.
Sensitivity Analysis in SYMPHONY

- Using the functions on the previous slide, SYMPHONY can calculate bounds after changing the objective or right-hand side vectors.

```c
int main(int argc, char **argv)
{
    OsiSymSolverInterface si;
    si.parseCommandLine(argc, argv);
    si.loadProblem();
    si.setSymParam(OsiSymSensitivityAnalysis, true);
    si.initialSolve();
    int ind[2];
    double val[2];
    ind[0] = 4; val[0] = 7000;
    ind[1] = 7; val[1] = 6000;
    lb = si.getLbForNewRhs(2, ind, val);
    ub = si.getUbForNewRhs(2, ind, val);
}
```
A Few Caveats

- The method presented only applies to pure branch and bound.
- Cut generation complicates matters.
- Fixing by reduced cost also complicates matters.
- Have to deal with infeasibility of subproblems.
- These issues can all be addressed, but the methodology is more involved.
- **Question**: What happens outside the allowable range?
- **Answers**:
  - Continue solving from a “warm start.”
  - Perform a parametric analysis.
Warm Starting

• **Question**: What is “warm starting”? 

• **Question**: Why are we interested in it?

• There are many examples of algorithms that solve a sequence of related ILPs.
  – Decomposition algorithms 
  – Stochastic ILP 
  – Parametric/Multicriteria ILP 
  – Determining irreducible inconsistent subsystem 

• For such problems, warm starting can potentially yield big improvements.

• Warm starting is also important for performing sensitivity analysis outside of the allowable range.
Warm Starting Information

• **Question**: What is “warm starting information”?

• Many optimization algorithms can be viewed as iterative procedures for satisfying a set of optimality conditions, often based on duality.

• These conditions provide a measure of “distance from optimality.”

• Warm starting information can be seen as additional input data that allows an algorithm to quickly get “close to optimality.”

• In linear and integer linear programming, the *duality gap* is the usual measure.

• A starting basis can reduce the initial duality gap in LP.

• The corresponding concept in ILP is a *starting partition*.

• It is not at all obvious what makes a good starting partition.

• The most obvious choice for a starting partition is to use the optimal partition from a previous computation.
Warm Starts for MILP

• To allow resolving from a warm start, we have defined a SYMPHONY warm start class, which is derived from CoinWarmStart.

• The class stores a snapshot of the search tree, with node descriptions including:
  – lists of active cuts and variables,
  – branching information,
  – warm start information, and
  – current status (candidate, fathomed, etc.).

• The tree is stored in a compact form by storing the node descriptions as differences from the parent.

• Other auxiliary information is also stored, such as the current incumbent.

• A warm start can be saved at any time and then reloaded later.

• The warm starts can also be written to and read from disk.
Warm Starting Procedure

• After modifying parameters
  – If only parameters have been modified, then the candidate list is recreated and the algorithm proceeds as if left off.
  – This allows parameters to be tuned as the algorithm progresses if desired.

• After modifying problem data
  – Currently, we only allow modification of rim vectors.
  – After modification, all leaf nodes must be added to the candidate list.
  – After constructing the candidate list, we can continue the algorithm as before.

• There are many opportunities for improving the basic scheme, especially when solving a known family of instances (Geoffrion and Nauss)
Using Warm Starting (Parameter Modification)

- The following example shows a simple use of warm starting to create a dynamic algorithm.

```c
int main(int argc, char **argv)
{
    OsiSymSolverInterface si;
    si.parseCommandLine(argc, argv);
    si.loadProblem();
    si.setSymParam(OsiSymFindFirstFeasible, true);
    si.setSymParam(OsiSymSearchStrategy, DEPTH_FIRST_SEARCH);
    si.initialSolve();
    si.setSymParam(OsiSymFindFirstFeasible, false);
    si.setSymParam(OsiSymSearchStrategy, BEST_FIRST_SEARCH);
    si.resolve();
}
```
Using Warm Starting (Problem Modification)

- The following example shows how to warm start after problem modification.

```c
int main(int argc, char **argv)
{
    OsiSymSolverInterface si;
    CoinWarmStart ws;
    si.parseCommandLine(argc, argv);
    si.loadProblem();
    si.setSymParam(OsiSymNodeLimit, 100);
    si.initialSolve();
    ws = si.getWarmStart();
    si.resolve();
    si.setObjCoeff(0, 1);
    si.setObjCoeff(200, 150);
    si.setWarmStart(ws);
    si.resolve();
}
```
Using Warm Starting: Generic Mixed-Integer Programming

• Applying the code from the previous slide to the MIPLIB 3 problem p0201, we obtain the results below.

• Note that the warm start doesn’t reduce the number of nodes generated, but does reduce the solve time dramatically.

<table>
<thead>
<tr>
<th></th>
<th>CPU Time</th>
<th>Tree Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate warm start</td>
<td>28</td>
<td>100</td>
</tr>
<tr>
<td>Solve orig problem (from warm start)</td>
<td>3</td>
<td>118</td>
</tr>
<tr>
<td>Solve mod problem (from scratch)</td>
<td>24</td>
<td>122</td>
</tr>
<tr>
<td>Solve mod problem (from warm start)</td>
<td>6</td>
<td>198</td>
</tr>
</tbody>
</table>
Using Warm Starting: Generic Mixed-Integer Programming

- Here, we show the effect of using warm starting to solve generic MILPs whose objective functions have been perturbed.

- The coefficients were perturbed by a random percentage between $\alpha$ and $-\alpha$ for $\alpha = 1, 10, 20$.

Table 1: Results of using warm starting to solve multi-criteria optimization problems.
Using Warm Starting: Stochastic Integer Programming

<table>
<thead>
<tr>
<th>Problem</th>
<th>Tree Size Without WS</th>
<th>Tree Size With WS</th>
<th>% Gap Without WS</th>
<th>% Gap With WS</th>
<th>CPU Without WS</th>
<th>CPU With WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>storm8</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>14.75</td>
<td>8.71</td>
</tr>
<tr>
<td>storm27</td>
<td>5</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>69.48</td>
<td>48.99</td>
</tr>
<tr>
<td>storm125</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>322.58</td>
<td>176.88</td>
</tr>
<tr>
<td>LandS27</td>
<td>71</td>
<td>69</td>
<td>-</td>
<td>-</td>
<td>6.50</td>
<td>4.99</td>
</tr>
<tr>
<td>LandS125</td>
<td>37</td>
<td>29</td>
<td>-</td>
<td>-</td>
<td>15.72</td>
<td>12.72</td>
</tr>
<tr>
<td>LandS216</td>
<td>39</td>
<td>35</td>
<td>-</td>
<td>-</td>
<td>30.59</td>
<td>24.80</td>
</tr>
<tr>
<td>dcap233_200</td>
<td>39</td>
<td>61</td>
<td>-</td>
<td>-</td>
<td>256.19</td>
<td>120.86</td>
</tr>
<tr>
<td>dcap233_300</td>
<td>111</td>
<td>89</td>
<td>0.387</td>
<td>-</td>
<td>1672.48</td>
<td>498.14</td>
</tr>
<tr>
<td>dcap233_500</td>
<td>21</td>
<td>36</td>
<td>24.701</td>
<td>14.831</td>
<td>1003</td>
<td>1004</td>
</tr>
<tr>
<td>dcap243_200</td>
<td>37</td>
<td>53</td>
<td>0.622</td>
<td>0.485</td>
<td>1244.17</td>
<td>1202.75</td>
</tr>
<tr>
<td>dcap243_300</td>
<td>64</td>
<td>220</td>
<td>0.0691</td>
<td>0.0461</td>
<td>1140.12</td>
<td>1150.35</td>
</tr>
<tr>
<td>dcap243_500</td>
<td>29</td>
<td>113</td>
<td>0.357</td>
<td>0.186</td>
<td>1219.17</td>
<td>1200.57</td>
</tr>
<tr>
<td>sizes3</td>
<td>225</td>
<td>165</td>
<td>-</td>
<td>-</td>
<td>789.71</td>
<td>219.92</td>
</tr>
<tr>
<td>sizes5</td>
<td>345</td>
<td>241</td>
<td>-</td>
<td>-</td>
<td>964.60</td>
<td>691.98</td>
</tr>
<tr>
<td>sizes10</td>
<td>241</td>
<td>429</td>
<td>0.104</td>
<td>0.0436</td>
<td>1671.25</td>
<td>1666.75</td>
</tr>
</tbody>
</table>
For global sensitivity analysis, we need to solve parametric programs.

Along with Saltzman and Wiecek, we have developed an algorithm for determining all Pareto outcomes for a bicriteria MILP.

The algorithm consists of solving a sequence of related ILPs and is asymptotically optimal.

Such an algorithm can be used to perform global sensitivity analysis by constructing a “slice” of the value function.

Warm starting can be used to improve efficiency.
Bicriteria MILPs

• The general form of a bicriteria (pure) ILP is

\[ \text{vmax}\, [cx, dx], \]
\[ \text{s.t.}\quad Ax \leq b, \]
\[ x \in \mathbb{Z}^n. \]

• Solutions don't have single objective function values, but pairs of values called outcomes.

• A feasible \( \hat{x} \) is called efficient if there is no feasible \( \bar{x} \) such that \( cx \geq c\hat{x} \) and \( dx \geq d\hat{x} \), with at least one inequality strict.

• The outcome corresponding to an efficient solution is called Pareto.

• The goal of a bicriteria ILP is to enumerate Pareto outcomes.
Example: Bicriteria ILP

• Consider the following bicriteria ILP:

\[
\begin{align*}
\text{vmax} & \quad \{8x_1, x_2\} \\
\text{s.t.} & \quad 7x_1 + x_2 \leq 56 \\
& \quad 28x_1 + 9x_2 \leq 252 \\
& \quad 3x_1 + 7x_2 \leq 105 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

• The following code solves this model.

```c
int main(int argc, char **argv)
{
    OsiSymSolverInterface si;
    si.parseCommandLine(argc, argv);
    si.loadProblem();
    si.setObj2Coeff(1, 1);
    si.multiCriteriaBranchAndBound();
}
```
Example: Pareto Outcomes for Example

Non-dominated Solutions

Supported Solutions

Y2

Y1
Example: Bicriteria Solver

- By examining the supported solutions and break points, we can easily determine $p(\theta)$, the optimal solution to the ILP with objective $8x_1 + \theta$.

<table>
<thead>
<tr>
<th>$\theta$ range</th>
<th>$p(\theta)$</th>
<th>$x_1^*$</th>
<th>$x_2^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(-\infty, 1.333)$</td>
<td>64</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>$(1.333, 2.667)$</td>
<td>$56 + 6\theta$</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>$(2.667, 8.000)$</td>
<td>$40 + 12\theta$</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>$(8.000, 16.000)$</td>
<td>$32 + 13\theta$</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>$(16.000, \infty)$</td>
<td>$15\theta$</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>
Example: Graph of Price Function
Table 2: Results of using warm starting to solve bicriteria optimization problems.
Conclusion

• We have briefly introduced the issues surrounding warm starting and sensitivity analysis for integer programming.

• An examination of early literature has yielded some ideas that can be useful in today’s computational environment.

• We presented a new version of the SYMPHONY solver supporting warm starting and sensitivity analysis for MILPs.

• We have also demonstrated SYMPHONY’s multicriteria optimization capabilities.

• This work has only scratched the surface of what can be done.

• In future work, we plan on refining SYMPHONY’s warm start and sensitivity analysis capabilities.

• We will also provide more extensive computational results.