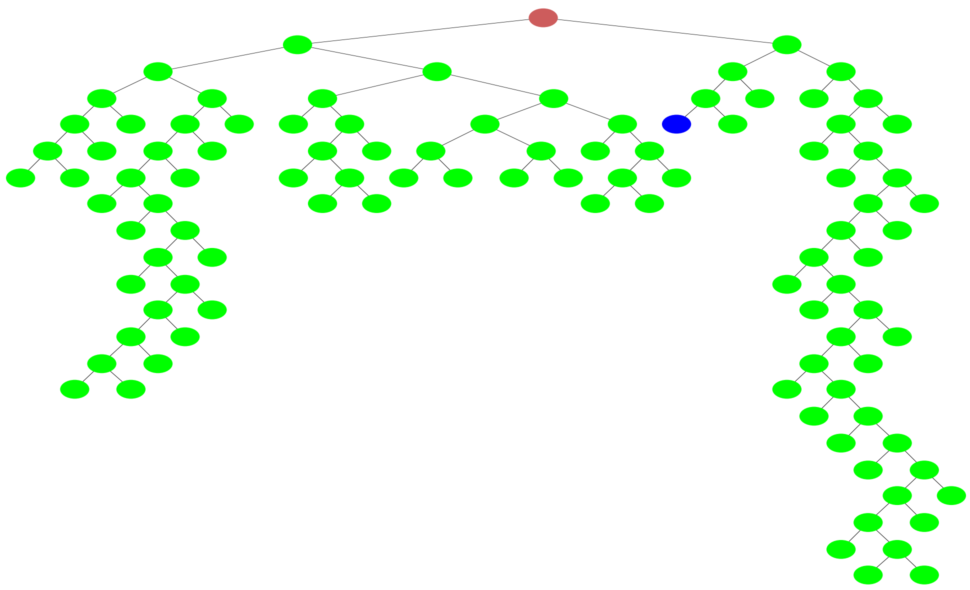
The SYMPHONY Callable Library for Mixed-Integer Linear Programming

Menal Guzelsoy and Ted Ralphs
Industrial and Systems Engineering
Lehigh University



INFORMS Annual Meeting, San Francisco, CA, November 16, 2005

Outline of Talk

- Overview of SYMPHONY
- Background
  - Duality
  - Sensitivity analysis
  - Warm starting
  - Bicriteria and parametric programming
- Implementation
  - Warm Starting
  - Sensitivity Analysis
  - Bicriteria Solve
- Examples
- Computational Experiments

Goals of the Project

- This work is part of a larger effort to develop strategies for real-time integer programming.
- The goal is to make integer programming a tactical decision-making tool.
- Toolbox
  - Sensitivity analysis
  - Warm starting
  - Parametric analysis
  - Heuristics
  - Parallel/grid/on-demand computing
- The Holy Grail: Solve difficult integer programs in "real time" in the presence of uncertain data.

A 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.
- SYMPHONY is part of the Computational Infrastructure for Operations Research (COIN-OR) libraries (`www.coin-or.org`).
- New features give SYMPHONY the look and feel of an LP solver.
- This talk will focus on these new features,
- This talk is based on the (not officially released) SYMPHONY 5.1, available from the COIN-OR CVS server.

SYMPHONY Features

- Core solution methodology is branch and cut.
  - Hybrid depth-first/best-first search strategy.
  - Strong branching mechanism.
  - Primal heuristic from CBC.
  - Customizable through parameters and callbacks.
- Cuts can be generated with COIN-ORs Cut Generation Library.

- Cliques	- Lift and Project	- Probing
- Flow Covers	- Reduce and Split	- Simple Rounding
- Gomory	- Mixed-integer Rounding	- Two-slope MIR
- Knapsack Cover	- Off Hole	- Problem-specific

- User interfaces
  - Native C callable
  - Open Solver Interface C++
  - FLOPC++ modeling language
  - MPS, AMPL/GMPL, LPFML file formats.

What's Available

- Packaged releases from `www.branchandcut.org`
- Current source at CVS on `www.coin-or.org`.
- An extensive user's manual on-line and in PDF.
- A tutorial illustrating the development of a custom solver step by step.
- Configuration and compilation files for supported architectures
  - Single-processor Linux, Unix, or Windows
  - Distributed memory parallel (PVM)
  - Shared memory parallel (OpenMP)
- Source code for SYMPHONY solvers.
  - Generic MILP
  - Multicriteria MILP
  - Multicriteria Knapsack
  - Traveling Salesman Problem
  - Vehicle Routing Problem
  - Mixed Postman Problem
  - Set Partitioning Problem
  - Matching Problem
  - Network Routing

Mathematical Programming Duality

- For an optimization problem

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

called the *primal problem*, an optimization problem

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

such that $w \leq z$ is called a *dual problem*.

- It is a *strong dual* if $w = z$.
- Uses for the dual problem
  - Bounding
  - Deriving optimality conditions
  - Sensitivity analysis
  - Warm starting

Duals for ILP: 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)
- j. Lasserre '05

Duals for Linear Optimization Problems

- 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 $c \in \mathbb{R}^n$.
- How do we derive a dual? Consider the following more formal notion of duality for linear optimization problems (Wolsey).

$$\begin{aligned} w &= \max_{g:\mathbb{R}^m \to \mathbb{R}} \{g(b) \mid g(Ax) \leq c^\top x, x \geq 0\} && (1) \\ &= \max_{g:\mathbb{R}^m \to \mathbb{R}} \{g(b) \mid g(d) \leq z(d), d \in \mathbb{R}^m\}, && (2) \end{aligned}$$

where $z(d) = \min_{x \in \mathcal{F}(d)} c^\top x$ is the *value function* and $\mathcal{F}(d) = \{x \in X \mid Ax = d, x \geq 0\}$.

- If $X = \mathbb{R}^n$, then an optimal dual function is the usual LP dual.
- If $X = \mathbb{Z}^n$, then an optimal dual function is more difficult to construct.

Dual Solutions from Primal Algorithms

- In LP, an optimal dual function arises naturally as a by-product of the simplex algorithm.
- The *optimal basis* yields optimal primal and dual solutions and a *certificate of optimality*.
- Sensitivity analysis and warm starting procedures for LP are based on the associated optimality conditions.
- We can extend this to ILP by considering the implicit certificate of optimality associated with branch and bound.

Dual Solutions for ILP from Branch and Bound

- Let $\mathcal{P}_1, \ldots, \mathcal{P}_s$ be a partition of $\mathcal{P}$ into (nonempty) subpolyhedra.
- Let LP_i be the linear program $\min_{x^i \in \mathcal{P}_i} c^\top x^i$ 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

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

  where γ_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*.
- This is the certificate constructed by branch and bound.

Sensitivity Analysis

- The function

$$L(d) = \min\{c_{B^i}(B^i)^{-1}d + \gamma_i \mid 1 \le i \le s\},$$

 provides an optimal solution to (2).

- The corresponding upper bounding function is

$$U(c) = \min\{c_{B^i}(B^i)^{-1}b + \beta_i \mid 1 \le i \le s, \hat{x}^i \in \mathcal{P}^I\}$$

- 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.

Example: Using Sensitivity Analysis

- SYMPHONY will calculate bounds after changing the objective or right-hand side vectors.

```
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);
}
```

Limitations

- 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.
- Another issue is that the quality of the bounds may degrade quickly outside the allowable range.
- Two options for getting improved bounds:
  - Reactive: Continue solving from a "warm start."
  - Proactive: Perform a parametric analysis.

Warm Starting

- Why is warm starting important for ILP??
- 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.
- What exactly does "warm starting" mean?

Warm Starting Information

- Most optimization algorithms can be viewed as iterative procedures for constructing a certificate of optimality, often based on duality.
- By providing a candidate certificate from a previous computation, the procedure can sometimes be accelerated.
- In linear programming, the initial certificate is a starting basis, which can be iteratively modified if it does not satisfy optimality conditions.
- The corresponding concept in ILP is a *starting partition*, which yields a computable dual function.
- A starting partition can be obtained from a previous branch and bound calculation.
- Unlike the LP case, however, a single branch and bound tree yields a wide range of possible starting partitions.
- It is not obvious which one to choose.

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.
- A global cut pool can be saved and reused if desired.

Warm Starting Procedures

- 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)

Constructing the Warm Start

- Given a branch and bound tree, any subtree can yield a starting partition.
- A partition that is too fine-grained may not be useful.
- The greater the change in problem data, the less useful information can be obtained from the tree.
- Options for constructing a starting partition.
  - Take the first n nodes.
  - Take all nodes above a given level in the tree.
  - Take the first $p\%$ of the nodes.
  - Try to construct a partition using information about how the problem was modified.

Example; Using Warm Starting (Parameter Modification)

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

```
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();
}
```

Example: Using Warm Starting (Problem Modification)

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

```
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();
}
```

Example: Using Warm Starting

- 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.

	CPU Time	Tree Nodes
Generate warm start	28	100
Solve orig problem (from warm start)	3	118
Solve mod problem (from scratch)	24	122
Solve mod problem (from warm start)	6	198

Parametric Analysis

- For global sensitivity analysis, we need to solve parametric programs.
- SYMPHONY includes 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.

Example: Bicriteria ILP

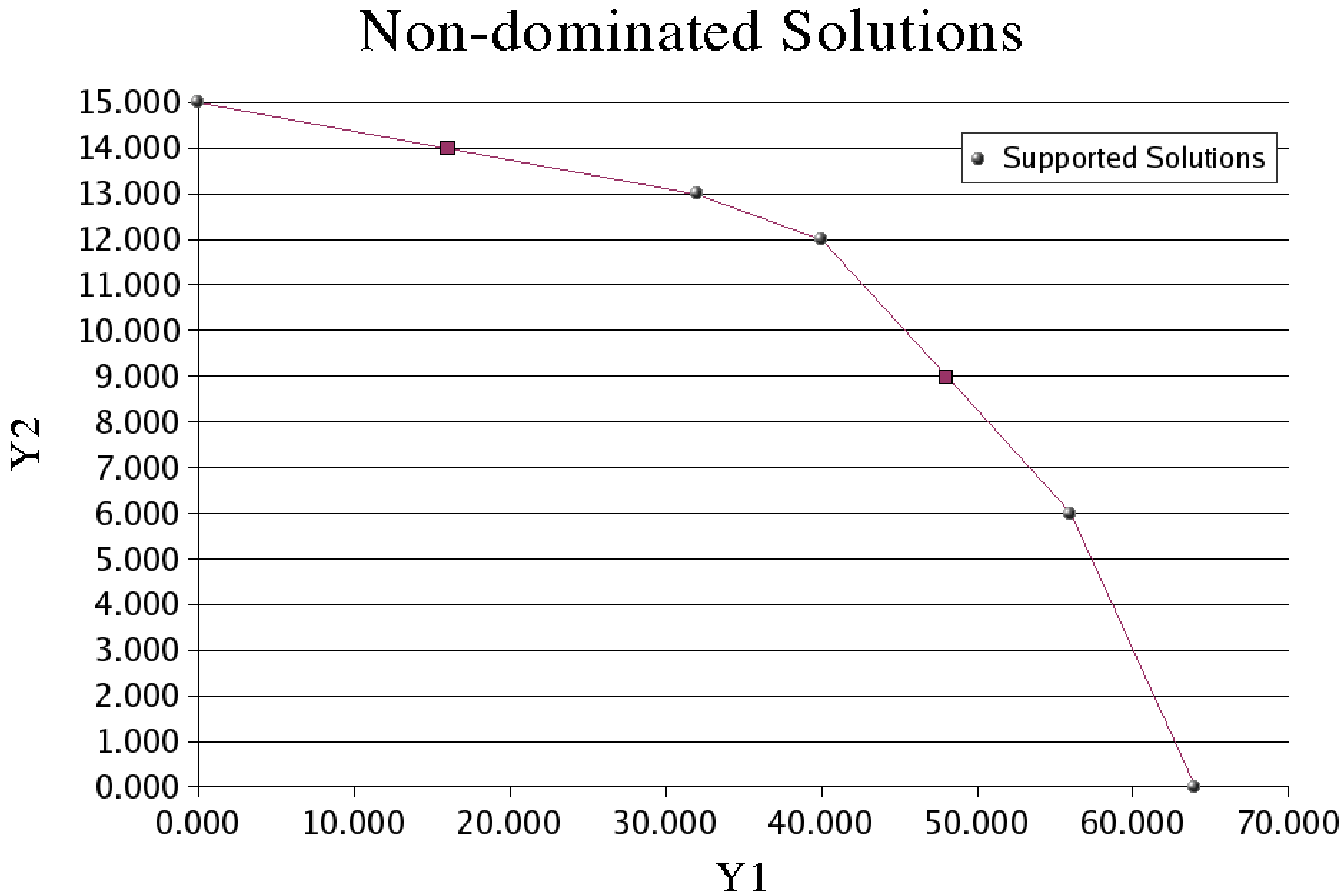
- Consider the following bicriteria ILP:

$$
\begin{aligned}
\text{vmax} \quad & [8x_1, x_2] \\
\text{s.t.} \quad & 7x_1 + x_2 \le 56 \\
& 28x_1 + 9x_2 \le 252 \\
& 3x_1 + 7x_2 \le 105 \\
& x_1, x_2 \ge 0
\end{aligned}
$$

- The following code solves this model.

```
int main(int argc, char **argv)
{
   OsiSymSolverInterface si;
   si.parseCommandLine(argc, argv);
   si.setObj2Coeff(1, 1);
   si.loadProblem();
   si.multiCriteriaBranchAndBound();
}
```

Example: Pareto Outcomes for Example



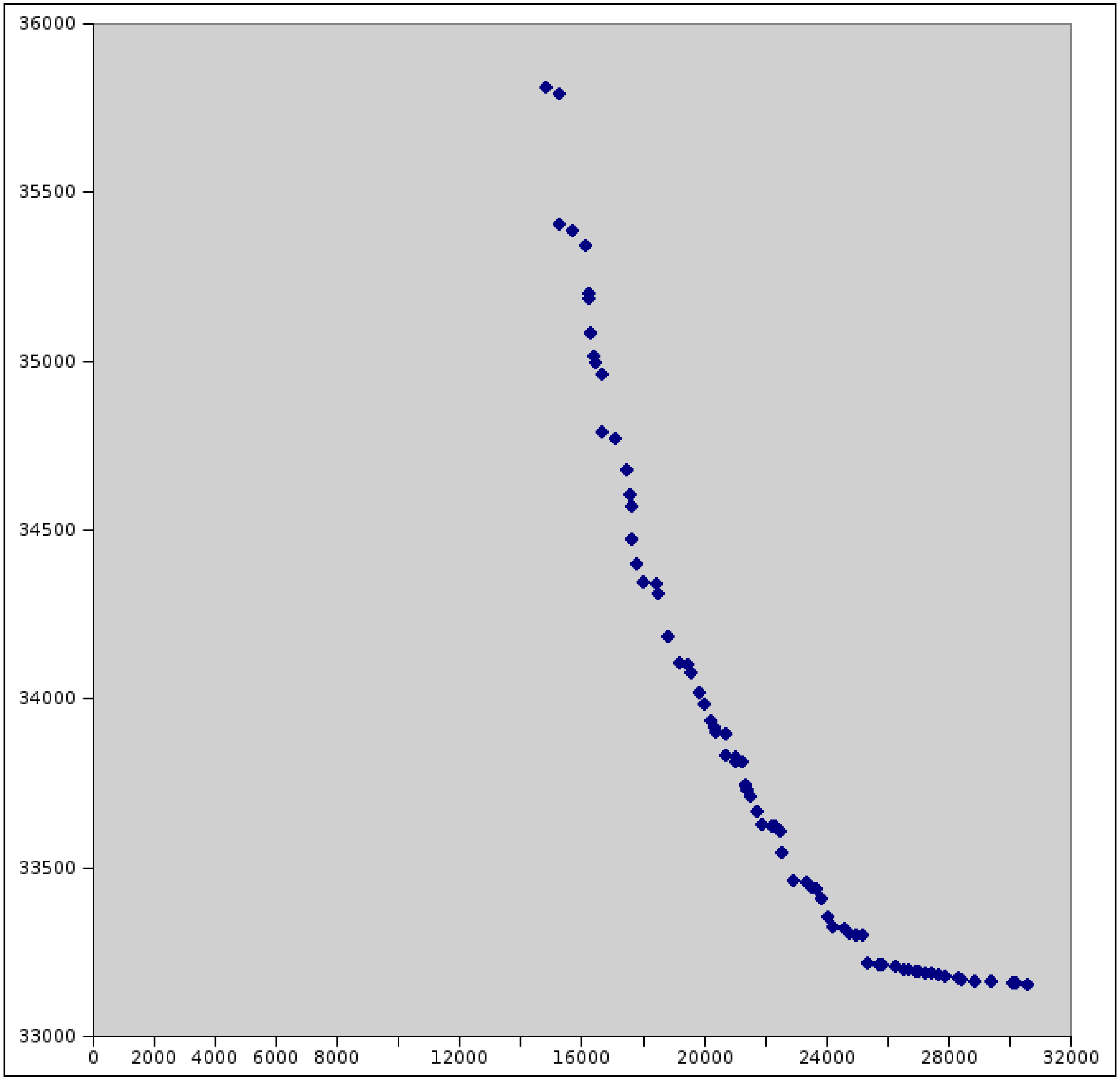
Example: Bicriteria Solver

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

θ range	$p(\theta)$	x_1^*	x_2^*
$(-\infty, 1.333)$	64	8	0
$(1.333, 2.667)$	$56 + 6\theta$	7	6
$(2.667, 8.000)$	$40 + 12\theta$	5	12
$(8.000, 16.000)$	$32 + 13\theta$	4	13
$(16.000, \infty)$	15θ	0	15

Example: Graph of Price Function

Example: Pareto Outcomes for a Network Design Problem



Using Warm Starting: Change in the Objective Function

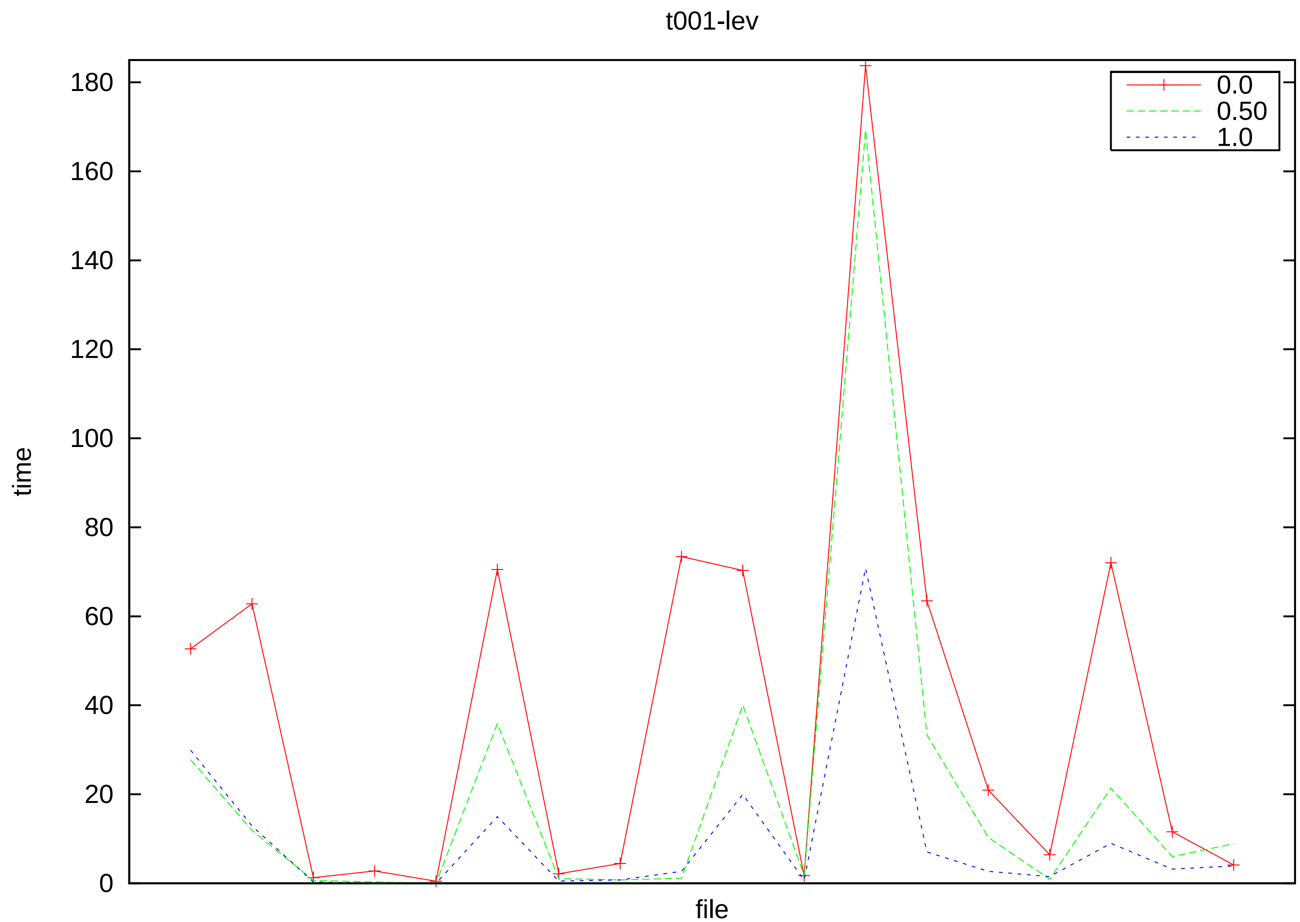


Table 1: Warm start after 1% modification on a random subset of objective coefficients of random size. Warm start consists of nodes above the $r\%$ level of the tree, $r \in \{0, 50, 100\}$

Using Warm Starting: Change in the Objective Function

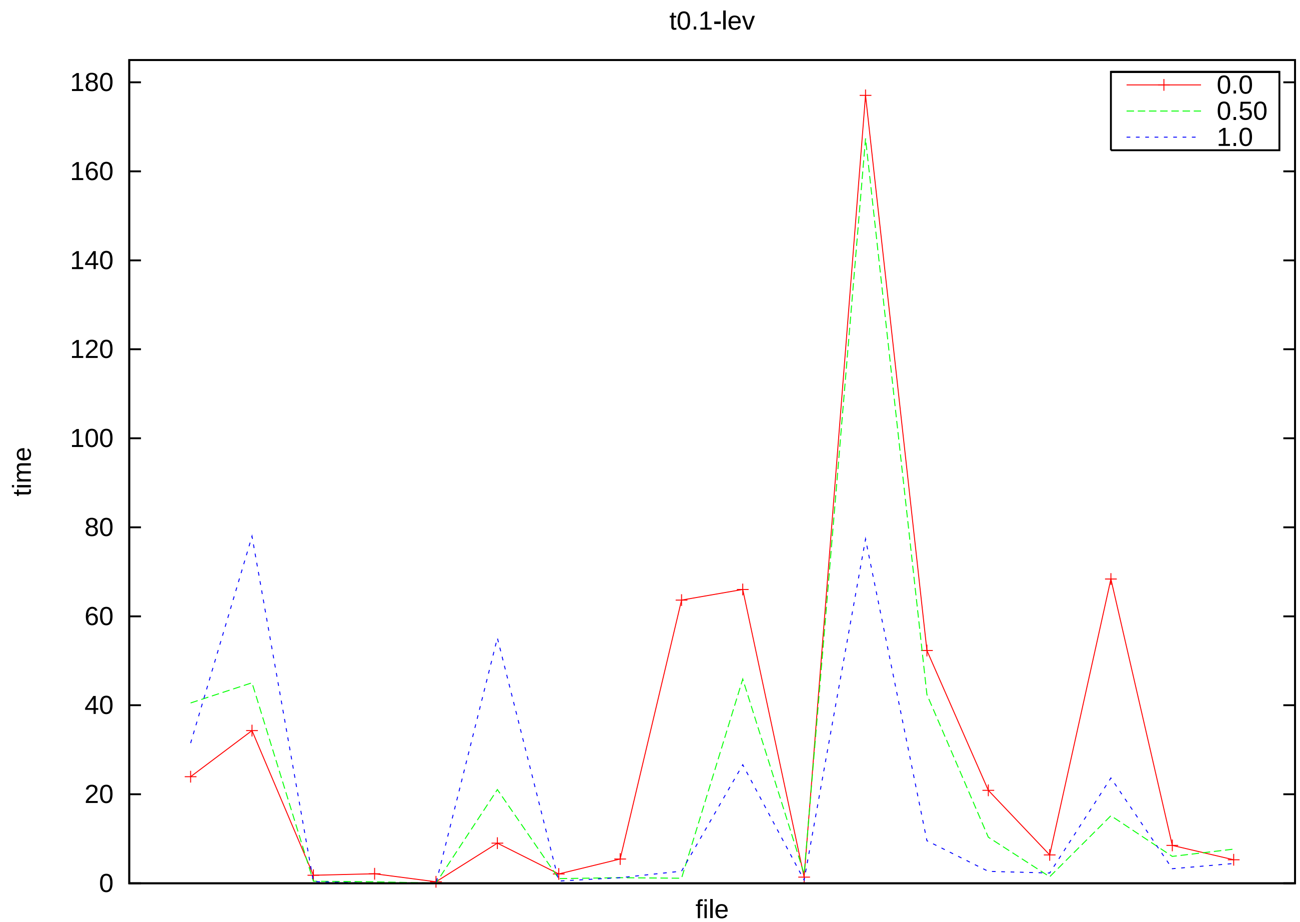


Table 2: Warm start after 10% modification on a random subset of objective coefficients of random size. Warm start consists of nodes above the $r\%$ level of the tree, $r \in \{0, 50, 100\}$

Using Warm Starting: Change in the Objective Function

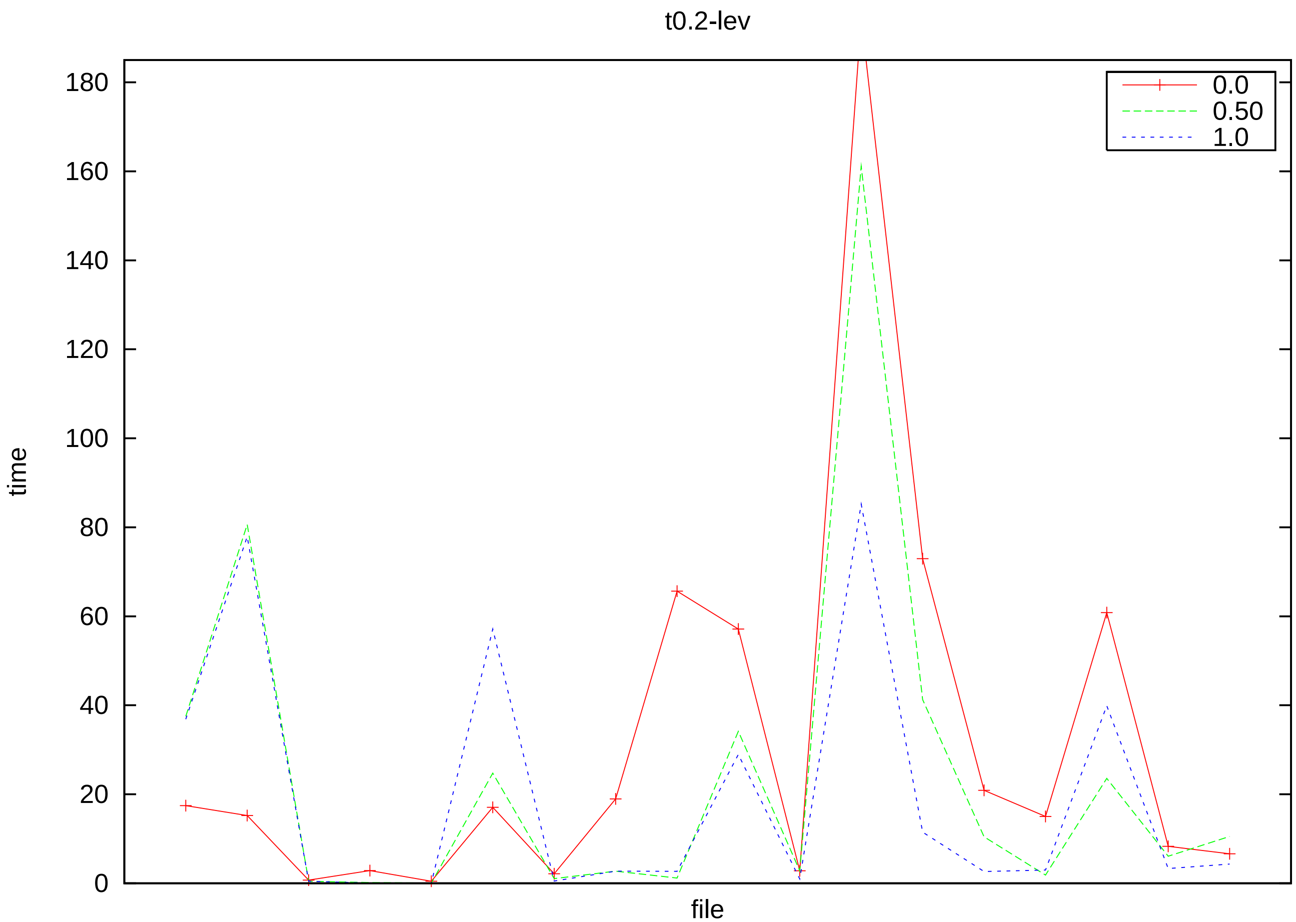
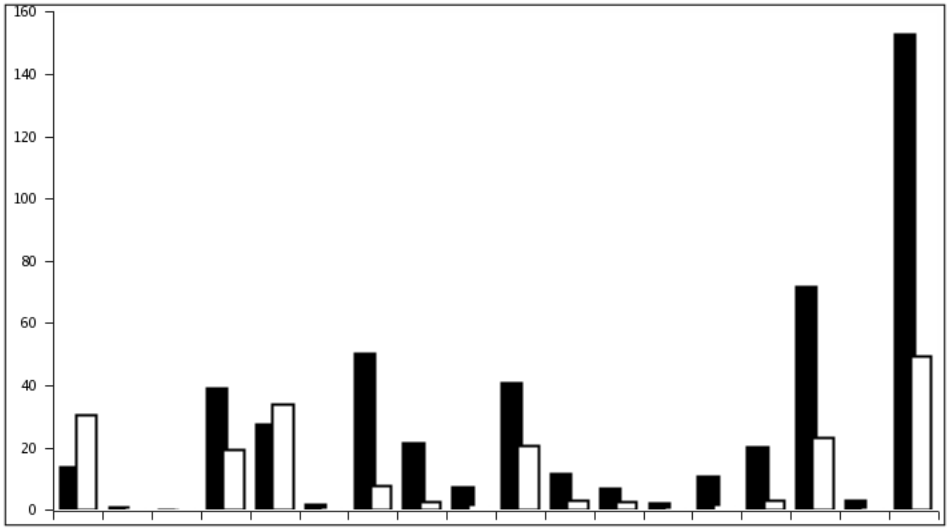
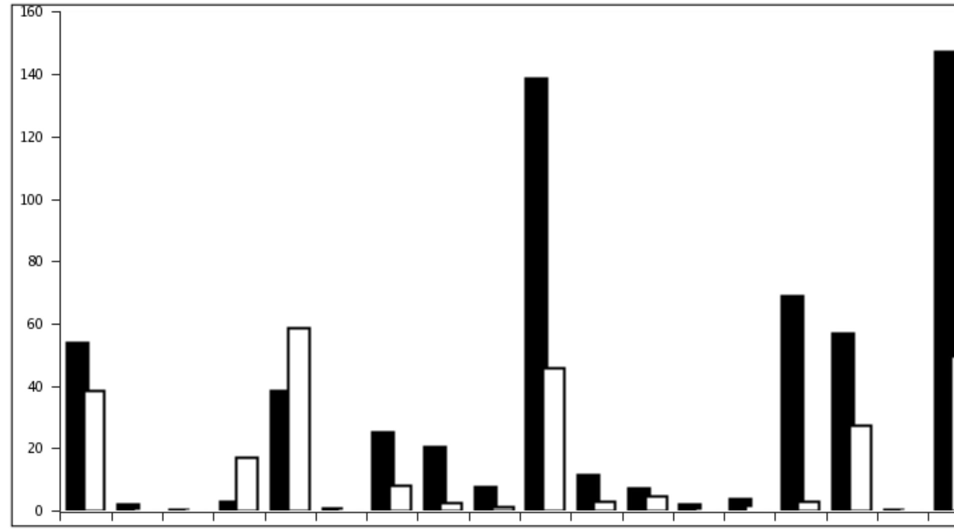


Table 3: Warm start after 20% modification on a random subset of objective coefficients of random size and use the nodes above the $r\%$ level of the tree, $r \in \{0, 50, 100\}$

Using Warm Starting: Change in the Objective Function





Black: without warm starting
White: with warm starting

Table 4: Warm start after random perturbation of $+/-10\%$ on a random subset of objective coefficients of size $0.1n$ (left) and of size $0.2n$ (right)

Using Warm Starting: Change in the Right-hand Side

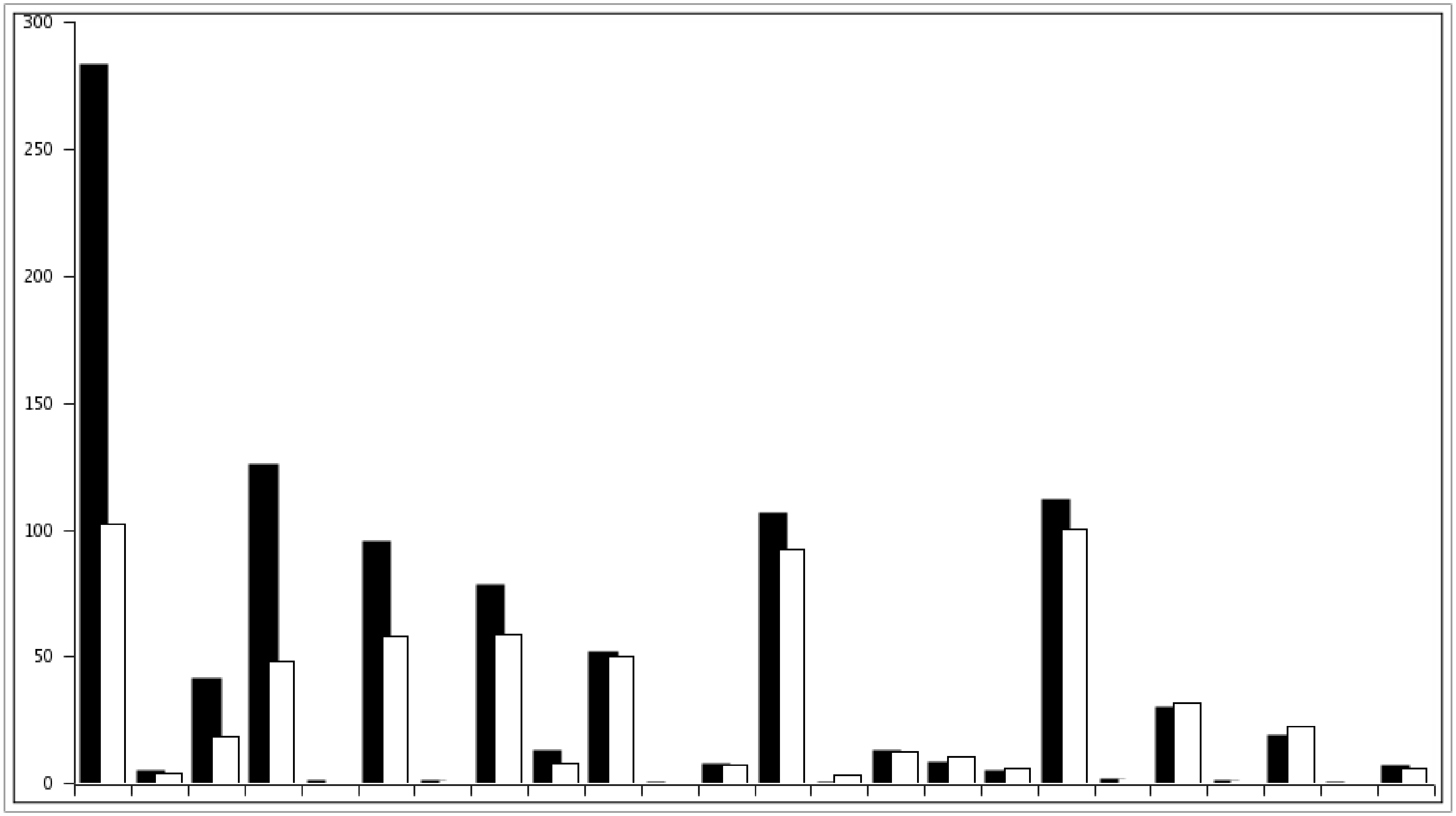
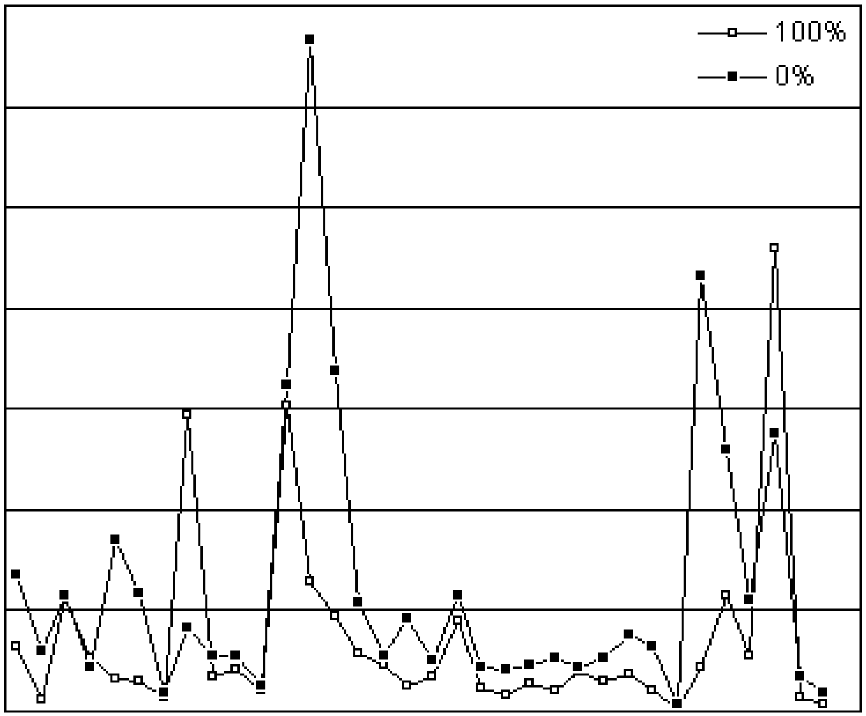


Table 5: Change rhs b of a knapsack problem between $b/2$ and $3b/2$ and warm start using the nodes above the 25% level of the tree.

Using Warm Starting: Bicriteria Optimization



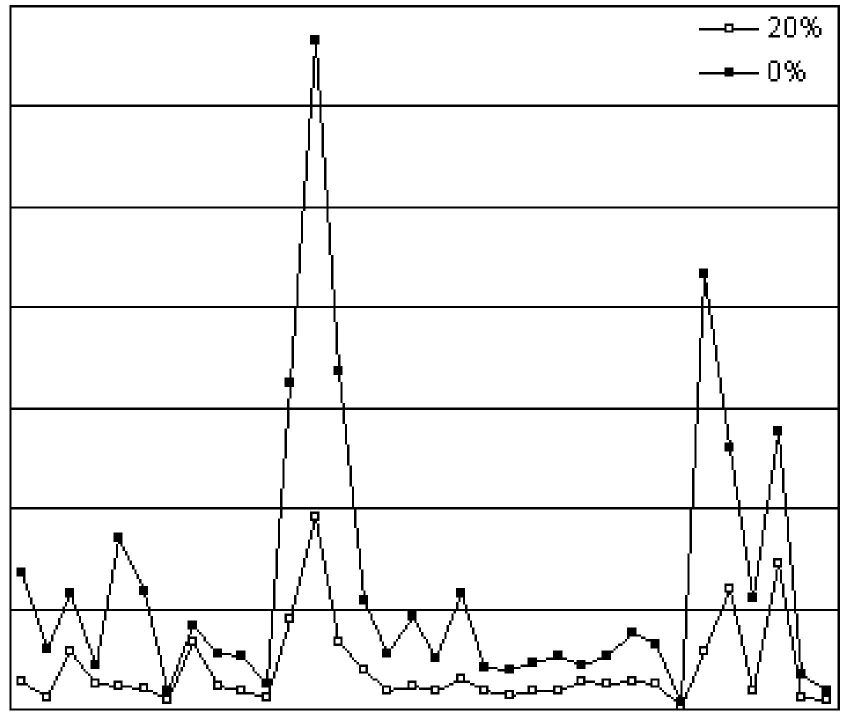


Table 6: Results of using warm starting to solve bicriteria optimization problems.

Extensions: Reduced Cost Fixing

For an ILP problem, let f be a feasible subadditive dual function and let $\hat{z}^{IP}$ be an upper bound on z^{IP}. If $c_k - f(a_k) > 0$ and

$$v = \left\lceil \frac{\hat{z}^{IP} - f(b)}{c_k - f(a_k)} \right\rceil > 0$$

for a column k, then there is an optimal solution x^* with $x_k^* \leq v - 1$.

- It is possible to obtain a *feasible* subadditive dual function if the problem is solved by branch and bound.
- f is still feasible to $\mathcal{P}(\tilde{b})$,i.e., when $b \rightarrow \tilde{b}$.
- Using reduced cost fixing over this function will preprocess/tighten the variable bounds before warm starting.

Extensions: Sensitivity Analysis for Branch and Cut Algorithm

- We can extend Wolsey's MILP sensitivity analysis for branch and bound algorithm to branch and cut to get a rough lower bound to modified problem with $b \longrightarrow \tilde{b}$.
- The algorithm basically calculates a lower bound for each tree node assuming the same tree was used to solve $\mathcal{P}(\tilde{b})$, and gathers those bounds to get a lower bound to $\mathcal{P}(\tilde{b})$.
- We can get a rough lower bound for each node of the branch and cut tree and follow the rest of the algorithm.

Extensions: Sensitivity Analysis for Branch and Cut Algorithm

Let the LP relaxation for node k be

$$
\begin{aligned}
\mathcal{P}^t(b) = min \;\; & cx \\
s.t. \;\; \sum_{j=1}^{n} a_j x_j &\geq b \quad (\gamma^k) \\
\sum_{j=1}^{n} h_j^k x_j &\geq r^k \quad (\pi^k) \\
x &\geq l^k \quad (\underline{\theta}^k) \\
-x &\geq -u^k \quad (\overline{\theta}^k) \\
x &\geq 0
\end{aligned}
$$

with the associated dual feasible vectors where the constraint set $\sum_{j=1}^{n} h_j^k x_j \geq r^k$ represents the cuts added so far.

Extensions: Lower Bound for $\mathcal{P}^t(\tilde{b})$

Then for any feasible solution x to $\mathcal{P}^t(\tilde{b})$(without the cut set):

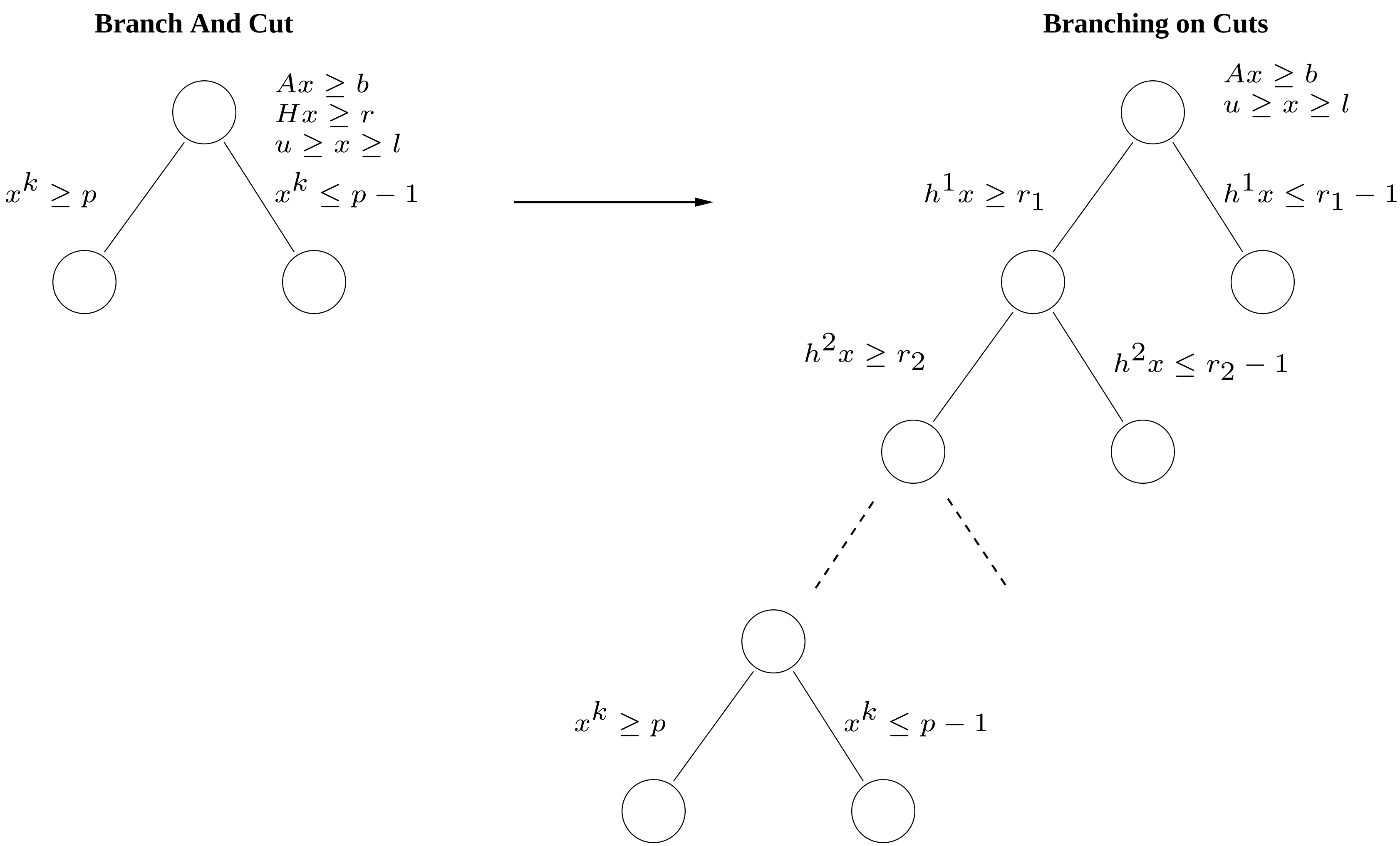
$$\begin{aligned} cx = \sum c_j x_j &\geq \gamma^k \sum a_j x_j + \pi^k \sum h_j^k x_j + \sum \underline{\theta}_j^k x_j - \sum \overline{\theta}_j^k x_j \\ &\geq \gamma^k \tilde{b} + \pi^k \sum h_j^k x_j + \underline{\theta}^k l^k - \overline{\theta}^k u^k \\ &\geq \gamma^k \tilde{b} + \pi^k \tilde{r}^k + \underline{\theta}^k l^k - \overline{\theta}^k u^k \end{aligned}$$

where

$$\tilde{r}^k = \sum h_j^k y_j \;\; and \;\; y_j = \left\{ \begin{array}{ccc} l_j^k & if & \pi^k h_j^k \geq 0 \\ u_j^k & & o.w \end{array} \right\}$$

May be useful for the problems with bounded variables, for instance, binary problems.

Extensions: Branching on Cuts



Is it possible to get a strong (not necessarily subadditive) dual formulation from branching on cuts tree, similar to Wolsey's formulation for branch and bound?

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.
- This work has only scratched the surface of what can be done.
- We need to learn much more about how these methods behave in practice.
- In future work, we plan on refining SYMPHONY's capabilities and employing them within a larger distributed computational framework..