ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs[1]

Kumar Abhishek, Sven Leyffer, and Jeffrey T. Linderoth

Mathematics and Computer Science Division

Preprint ANL/MCS-P1374-0906

March 28, 2008

[1]This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, and by the National Science and Engineering Research Council of Canada.

Contents

FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs*

Kumar Abhishek†, Sven Leyffer‡, and Jeffrey T. Linderoth§

March 28, 2008

Abstract

We describe a new solver for mixed integer nonlinear programs (MINLPs) that implements a linearization-based algorithm. The solver is based on the algorithm by Quesada and Grossmann, and avoids the complete solution of master mixed integer linear programs (MILPs) by adding new linearizations at open nodes of the branch-and-bound tree whenever an integer solution is found. The new solver, FilMINT, combines the MINTO branch-and-cut framework for MILP with filterSQP used to solve the nonlinear programs that arise as subproblems in the algorithm.

The MINTO framework allows us to easily extend cutting planes, primal heuristics, and other well-known MILP enhancements to MINLPs. We present detailed computational experiments that show the benefit of such advanced MILP techniques. We offer new suggestions for generating and managing linearizations that are shown to be efficient on a wide range of MINLPs. Comparisons to existing MINLP solvers are presented, that highlight the effectiveness of FilMINT.

Keywords: Mixed integer nonlinear programming, outer approximation, branch-and-cut.

AMS-MSC2000: 90C11, 90C30, 90C57.

---

*Argonne National Laboratory Preprint ANL/MCS-P1374-0906

†Department of Industrial and Systems Engineering, Lehigh University, 200 W. Packer Ave., Bethlehem, PA 18015, kua3@lehigh.edu

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, leyffer@mcs.anl.gov

§Department of Industrial and Systems Engineering, Lehigh University, 200 W. Packer Ave., Bethlehem, PA 18015, jtl3@lehigh.edu

1 Introduction

We consider the development of a new efficient solver for mixed integer nonlinear programs (MINLPs). Our interest is motivated by the rich collection of important applications that can be modeled as MINLPs, including nuclear core reload problems (Quist et al. [1998]), cyclic scheduling (Jain and Grossmann [1998]), trimloss optimization in the paper industry (Harjunkoski et al. [1988]), synthesis problems (Kocis and Grossmann [1988]), and layout problems (Castillo et al. [2005]). MINLP problems are conveniently expressed as

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) \leq 0, \\ & x \in X, y \in Y \cap \mathbb{Z}^p, \end{aligned} \tag{MINLP}$$

where $f : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $g : \mathbb{R}^{n\times p} \to \mathbb{R}^m$ are twice continuously differentiable functions, x and y are continuous and discrete variables, respectively, and X and Y are compact polyhedral subsets of $\mathbb{R}^n$ and $\mathbb{Z}^p$, respectively. In this paper, we focus on the case where the functions f and g_j are convex, so that by relaxing the restriction $y \in \mathbb{Z}^p$, a convex program is formed. The techniques we suggest may be applied as a heuristic in the case that one or more of the functions are nonconvex.

Methods for the solution of convex (MINLP) include the branch-and-bound method (Dakin [1965], Gupta and Ravindran [1985]), branch-and-cut (Stubbs and Mehrohtra [2002]), outer approximation (Duran and Grossmann [1986]), generalized Benders decomposition (Geoffrion [1972]), the extended cutting plane method (Westerlund and Pettersson [1995]), and LP/NLP-based branch-and-bound (Quesada and Grossmann [1992]). We refer the reader to Grossmann [2002] for a survey of solution techniques for MINLP problems.

Our aim is to provide a solver that is capable of solving MINLPs at a cost that is a small multiple of the cost of a comparable mixed integer linear program (MILP). In our view, the algorithm most likely to achieve this goal is LP/NLP-based branch and bound (LP/NLP-BB). This method is similar to outer approximation; but instead of solving an alternating sequence of MILP master problems and nonlinear programming (NLP) subproblems, it interrupts the solution of the MILP master whenever a new integer assignment is found, and solves an NLP subproblem. The solution of this subproblem provides new outer approximations that are added to the master MILP, and the solution of the updated MILP master continues. Thus only a single branch-and-cut tree is created and searched.

Our solver exploits recent advances in nonlinear programming and mixed integer linear programming to develop an efficient implementation of LP/NLP-BB. Our work is motivated by the observation of Leyffer [1993] that a simplistic implementation of this algorithm often outperforms nonlinear branch and bound and outer approximation by an order of magnitude.

Despite this clear advantage, however, there has been no implementation of LP/NLP-BB until the recent independent work by Bonami et al. [2008] and this paper. Our implementation, called FilMINT, is built on top of the mixed integer programming solver MINTO (Nemhauser et al. [1994]). Through MINTO, we are able to exploit a range of modern MILP features, such as enhanced branching and node selection rules, primal heuristics, preprocessing, and cut generation routines. To solve the NLP subproblems, we use filterSQP (R.Fletcher et al. [2002]), an active set solver with warm-starting capabilities that can take advantage of good initial primal and dual iterates.

Recently Bonami et al. [2008] have also developed a solver for MINLPs called Bonmin. While the two solvers share many of the same characteristics, our work differs from Bonmin in a number of significant ways. First, FilMINT differs from Bonmin in the way in which linearizations are added, and how these linearizations are managed. We derive additional linearizations at fractional integer points to strengthen the outer approximation, and we exploit cut management features in MINTO that allow us to keep a much smaller set of linearizations. Second, FilMINT uses an active-set solver for the NLP subproblems, which allows us to exploit warm-starting techniques that are not readily available for the interior-point code IPOPT (Wächter and Biegler [2006]) that is used in Bonmin. We also note, that MINTO's suite of advanced integer programming techniques is also different from that of CBC (Forrest [2004]), which is the MILP framework on which Bonmin is based. Finally, Bonmin is a hybrid between a branch-and-bound solver based on nonlinear relaxations and one based on polyhedral outer approximations, whereas FilMINT relies solely on linear underestimators. A comparison of performance of the two solvers is given in Section 5.

The paper is organized as follows: In the remainder of this section, we formally review the LP/NLP-based branch-and-bound algorithm, describe its implementation within MINTO's branch-and-cut framework, and outline the computational setup for our experiments. In Section 2, we report a set of careful experiments that show the effect of modern MILP techniques on an LP/NLP-based algorithm. In Section 3 we consider several ways to generate additional linearizations for the algorithm and how to use these mechanisms effectively via a cut management strategy. Section 4 gives a comparison of FilMINT to other MINLP solvers, and conclusions of the work are offered in Section 5.

1.1 LP/NLP-based Branch and Bound

LP/NLP-BB is a clever extension of outer approximation, which solves an alternating sequence of NLP subproblems and MILP master problems. The NLP subproblem (NLP(y^k)) is obtained by fixing the integer variables at y^k, and the MILP master problem accumulates linearizations (outer approximations of the feasible set, and underestimators of the objective function) from the solution of (NLP(y^k)).

LP/NLP-BB avoids solving multiple MILP master problems by interrupting the MILP tree search whenever an integer feasible solution $(\hat{x}, y^k)$ is found to solve the NLP subproblem (NLP(y^k)). The outer approximations from (NLP(y^k)) are then used to update the MILP master, and the MILP tree search continues. Thus, instead of solving a sequence of MILPs, as is the case in the outer-approximation method, only a single MILP tree search is required.

We characterize a node $n \equiv (l, u)$ of the branch-and-bound search tree by the bounds $\{(l, u)\}$ enforced on the integer variables y. Given bounds (l, u) on y, we define the NLP relaxation as

$$z_{\text{NLPR}(l,u)} = \underset{(x,y)\in X\times Y}{\text{minimize}}\{f(x, y) \mid g(x, y) \leq 0, l \leq y \leq u\}. \qquad \text{(NLPR}(l, u)\text{)}$$

If l and u are lower and upper bounds on the y variables for the original instance, then the optimal objective function value $z_{\text{NLPR}(l,u)}$ of (NLPR(l, u)) provides a lower bound on (MINLP); otherwise it provides a lower bound for the subtree whose parent node is $\{(l, u)\}$. In general, the solution to (NLPR(l, u)) may yield one or more nonintegral values for the integer decision variables y.

The NLP subproblem for fixed values of the integer decision variables y^k is defined as

$$z_{\text{NLP}(y^k)} = \underset{x\in X}{\text{minimize}}\{f(x, y^k) \mid g(x, y^k) \leq 0\}. \qquad \text{(NLP}(y^k)\text{)}$$

If (NLP(y^k)) is feasible, then it provides an upper bound to the problem (MINLP). If (NLP(y^k)) is infeasible, then the NLP solver detects the infeasibility and returns the solution to a feasibility problem. The form of the feasibility problem (NLPF(y^k)) that is solved by filterSQP is a minimization of the scaled ℓ_1 norm of the constraint violation:

$$\underset{x\in X}{\text{minimize}} \sum_{j=1}^{m} w_j g_j(x, y^k)^+ \qquad \text{(NLPF}(y^k)\text{)}$$

for some weights $w_j \geq 0$, not all zero. Here, $g_j(x, y^k)^+ = \max\{0, g_j(x, y^k)\}$ measures the violation of the nonlinear constraints.

From the solution to (NLP(y^k)) or (NLPF(y^k)), we can derive outer approximations for (MINLP). The convexity of the nonlinear functions imply that the linearizations about any point (x^k, y^k)) form an outer approximation of the feasible set and underestimate the objective function. Specifically, if we introduce an auxiliary variable η in order to replace the objective by a constraint, changing the objective to $\text{minimize } \eta$ and adding the constraint

$\eta \geq f(x, y)$, then the $m + 1$ inequalities (OA(x^k, y^k)) are valid for MINLP:

$$f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq \eta$$

$$g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0, \qquad \text{(OA}(x^k, y^k)\text{)}$$

where $\nabla g(x^k, y^k)^T = [\nabla g_1(x^k, y^k) : \ldots : \nabla g_m(x^k, y^k)]^T$ is the Jacobian of the nonlinear constraints.

The inequalities (OA(x^k, y^k)) are used to create a master MILP. Given a set of points $\mathcal{K} = \{(x^0, y^0), (x^1, y^1), \ldots, (x^{|\mathcal{K}|}, y^{|\mathcal{K}|}\}$, let $\mathcal{OA}(\mathcal{K})$ be the set of points satisfying the outer approximations at those points:

$$\mathcal{OA}(\mathcal{K}) = \{(\eta, x, y) \in \mathbb{R}^{1+n+p} \mid (x, y, \eta) \text{ satisfy OA}(x^k, y^k)\ \forall (x^k, y^k) \in \mathcal{K}\}.$$

An outer approximation master MILP for the original MINLP may be formed as

$$z_{\text{MP}(\mathcal{K})} = \underset{(\eta,x,y)}{\text{minimize}}\{\eta \mid (\eta, x, y) \in \mathcal{OA}(\mathcal{K}) \cap (\mathbb{R} \times X \times (Y \cap \mathbb{Z}^p))\}. \qquad \text{(MP}(\mathcal{K})\text{)}$$

If $\mathcal{K}$ in (MP($\mathcal{K}$)) contains an appropriately-defined finite set of points (these points will be identified by the LP/NLP-BB solution algorithm 1.1), then under mild conditions, $z_{\text{MINLP}} = z_{\text{MP}(\mathcal{K})}$ (Fletcher and Leyffer [1994], Bonami et al. [2008]).

LP/NLP-BB relies on solving the continuous relaxation to (MP($\mathcal{K}$)) and enforcing integrality of the y variables by branching. We label this problem as CMP($\mathcal{K}, l, u$).

$$\underset{(\eta,x,y)}{\text{minimize}}\{\eta \mid (\eta, x, y) \in \mathcal{OA}(\mathcal{K}) \cap (\mathbb{R} \times X \times (Y \cap [l, u]))\} \qquad \text{(CMP}(\mathcal{K}, l, u)\text{)}$$

The main algorithm underlying our work, LP/NLP-BB, is formally stated in pseudo-code form in Algorithm 1.1. The index k is used to track the number of times the relaxed master problem (CMP($\mathcal{K}, l, u$)) is solved, and the index bk is the value of k the last time the **(branch)** portion of the algorithm was executed. We defer discussion of the **(cut)** portion of the LP/NLP-BB algorithm until Section 3.

An important difference between the LP/NLP-BB algorithm and typical branch-and-bound algorithms is that after finding an integer feasible point in LP/NLP-BB, the corresponding node is resolved. This modification is necessary because the integer point is no longer feasible in the master problem after adding the linearizations about that point.

Solve NLPR(y^l, y^u) and let (η^0, x^0, y^0) be its solution **(initialize)**
if NLPR(y^l, y^u) is infeasible **then**
  Stop. MINLP is infeasible
else
  $\mathcal{K} \leftarrow \{(x^0, y^0)\}, \mathcal{L} \leftarrow \{(y^l, y^u, \eta^0)\}, UB \leftarrow \infty, k \leftarrow 0, bk \leftarrow 0$
end if
while $\mathcal{L} \neq \emptyset$ **do**
  Select and remove node (l, u, η) from $\mathcal{L}$ **(select)**
  $\eta^k \leftarrow \eta, k \leftarrow k + 1$
  Solve CMP$(\mathcal{K}, l, u)$ and let $(\eta^k, \hat{x}, y^k)$ be its solution. **(evaluate)**
  if CMP$(\mathcal{K}, l, u)$ is infeasible **OR** $\eta^k \geq UB$ **then**
    Fathom node (l, u, η^k). **Goto (select)**.
  end if
  if $y_k \in \mathbb{Z}^p$ **then**
    Solve NLP(y^k). **(update master)**
    if NLP(y^k) is feasible **then**
      $UB \leftarrow \min\{UB, z_{\text{NLP}(y^k)}\}$
      Remove all nodes in $\mathcal{L}$ whose parent objective value $\eta \geq UB$. **(fathom)**
      Let (x^k, y^k) be solution to NLP(y^k)
    else
      Let (x^k, y^k) be solution to NLPF(y^k)
    end if
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{(x^k, y^k)\}$. **Goto (evaluate).**
  else if Do additional linearizations **then**
    See Algorithm 3.1 **(cut)**
    Goto (evaluate)
  else
    Select b such that $y^k_b \notin \mathbb{Z}$. $bk \leftarrow k$. **(branch)**
    $\hat{u}_b \leftarrow \lfloor y^k_b \rfloor, \qquad \hat{u}_j \leftarrow u_j \quad \forall j \neq b$
    $\hat{l}_b \leftarrow \lceil y^k_b \rceil, \qquad \hat{l}_j \leftarrow l_j \quad \forall j \neq b$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(l, \hat{u}, \hat{\eta}^k)\} \cup \{(\hat{l}, u, \hat{\eta}^k)\}$
  end if
end while

Algorithm 1.1: LP/NLP-BB algorithm.

1.2 Implementation within the MINTO Framework

FilMINT is built on top of MINTO's branch-and-cut framework, using filterSQP to solve the NLP subproblems. MINTO provides *user application functions* through which the user can implement a customized branch-and-cut algorithm, and FilMINT is written entirely within these user application functions. No changes are necessary to the core MINTO library in order to implement the LP/NLP-BB algorithm. MINTO can be used with any LP solver that has the capability to modify and resolve linear programs and interpret their solutions. In our experiments, we use the `Clp` linear programming solver that is called through its `OsiSolverInterface`. Both `Clp` and the `OsiSolverInterface` are open-source tools available from COIN-OR (`http://www.coin-or.org`).

FilMINT obtains problem information from AMPL's ASL interface (Fourer et al. [1993], Gay [1997]). ASL also provides the user with gradient and Hessian information for nonlinear functions, which are required by the NLP solver and are used to compute the linearizations (OA(x^k, y^k)) required for LP/NLP-BB. FilMINT's NLP solver, filterSQP, is a sequential quadratic programming (SQP) method that employs a filter to promote global convergence from remote starting points. A significant advantage of using an active-set SQP method in this context is that the method can readily take advantage of good starting points. We use as the starting point the solution of corresponding the LP node, namely, $(\eta^k, \hat{x}, y^k)$. Another advantage of using filterSQP for implementing (LP/NLP-BB) is that filterSQP contains an automatic restoration phase that enables it to detect infeasible subproblems reliably and efficiently. The user need not create and solve the feasibility problem (NLPF(y^k)) explicitly. Instead, filterSQP returns the solution of (NLPF(y^k)) automatically.

The MINTO user application functions used by FilMINT are `appl_mps`, `appl_feasible`, `appl_primal`, and `appl_constraints`. A brief description of FilMINT's use of these functions is stated next.

- `appl_mps:` The MINLP instance is read. This corresponds to step **(initialize)** in Algorithm 1.1.

- `appl_feasible:` This user application function allows the user to verify that a solution to the active formulation satisfying the integrality conditions is feasible. When we generate an integral solution y^k for the master problem, the NLP subproblem (NLP(y^k)) is solved, and its solution provides an upper bound and a new set of outer approximation cuts.

- `appl_constraints`. This function allows the user to generate violated constraints. The solution of (NLP(y^k)) or (NLPF(y^k)) in `appl_feasible` generates new linearizations. These are stored and added to the master problem (CMP($\mathcal{K}, l, u$)) by this

method. This function is also used to implement NLP solves at fractional LP solutions, an enhancement that will be explained in Section 3.

- `appl_primal`. This function allows the user to communicate a new upper bound and primal solution to MINTO, if the solve of (NLP(y^k)) resulted in an improved feasible solution to (MINLP).

1.3 Computational Setup and Preliminary Implementation

In this section we describe the computational setup and provide an initial comparison of LP/NLP-BB to a standard MINLP branch-and-bound solver. Our aim is to explore the usefulness of the wide range of MILP techniques that MINTO offers in the context of solving MINLP problems. We believe that this study is of interest beyond the scope of the specific LP/NLP-BB algorithm since it may provide an indication of which MILP techniques are likely to be efficient in other methods for solving MINLPs, such as branch and bound using the relaxation (NLPR(l, u)). We carry out a set of carefully constructed computational experiments to discover the salient features of a MILP solver that have the biggest impact on solving MINLPs.

The test problems have been collected from the GAMS collection of MINLP problems (Bussieck et al. [2003]), the MacMINLP collection of test problems (Leyffer [2003]), and the collection on the website of IBM-CMU research group (Sawaya et al. [2006]). Since FilMINT accepts only AMPL as input, all GAMS models were converted into AMPL format. The test suite comprises 222 convex problems covering a wide range of applications.

The experiments have been run on a cluster of (identical) computers at Lehigh University. The cluster consists of 120 nodes of 64-bit AMD Opteron microprocessors. Each of the nodes has a CPU clockspeed of 1.8 GHz, and 2 GB RAM and runs on Fedora Core 2 operating system. All of the codes we tested were compiled by using the GNU suite of C, C++, and FORTRAN compilers.

The test suite of convex instances have been categorized as easy, moderate, or hard, based on the time taken using MINLP-BB, a branch-and-bound solver based on the relaxation (NLPR(l, u)) (Leyffer [1998]), to solve each instance. The easy convex instances take less than one minute to solve. Moderate convex instances take between one minute to one hour to solve. The hard convex instances are not solved in one hour. There are 100 easy, 37 moderate, and 85 hard instances in the test suite, and the names and characteristics of these instances can be found in the appendix.

Experiments have been conducted by running the test problems using FilMINT (with various features set on or off) for a time limit of four hours. We create performance profiles (see Dolan and Moré [2002]) to summarize and compare the runs on the same test suite using different solvers and options. For the easy and moderate problems, we use solution time as

a metric for the profiles. For the hard instances, however, the optimal solution is often not achieved (or even known). For these instances, we use a scaled solution value as the solver metric. We define the scaled solution value of solver s on instance i as $\rho_i^s = 1 + (z_i^s - z_i^*)/z_i^*$, where z_i^s is the best solution value obtained by solver s on instance i, and z_i^* is the best known solution value for instance i. The performance profile therefore indicates the quality of the solution found by a solver within four hours of CPU time.

We start by benchmarking a straightforward implementation of LP/NLP-BB (Algorithm 1.1) against MINLP-BB, an branch-and-bound algorithm that uses (NLPR(l, u)) to obtain a lower bound. The straightforward LP/NLP-BB implementation does not use any of MINTO's advanced MILP features, such as primal heuristics, cuts, and preprocessing. The algorithm branches on the the most fractional variable and selects the next node to be solved as the one with the smallest lower bound. The implementation is thus quite similar to one created in the Ph.D. work of Leyffer [1993]. We refer to this version of FilMINT as the *vanilla* version.

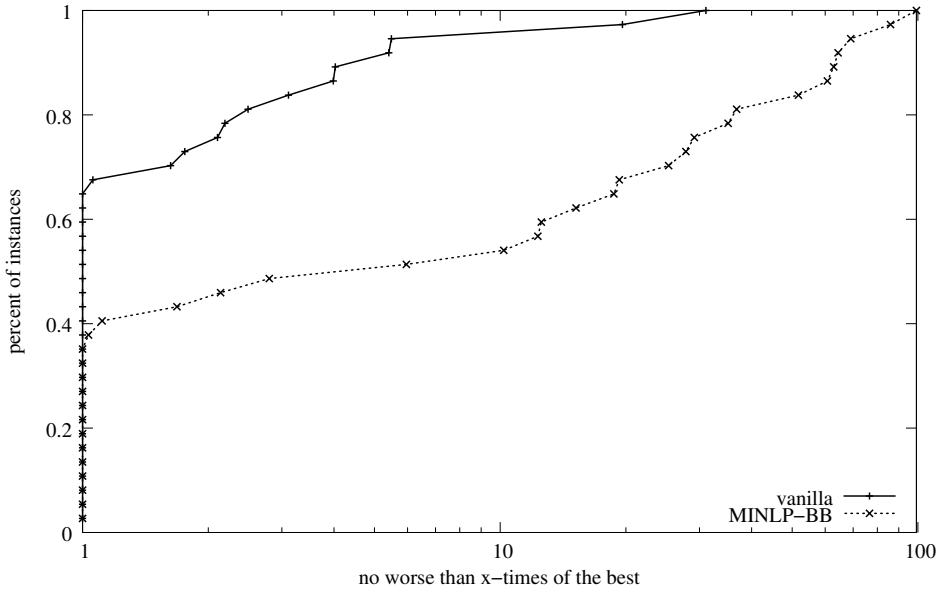

Figure 1: Performance profile comparing vanilla and MINLP-BB for moderate instances.

The performance profiles in Figures 1–2 compare the performance of the vanilla and MINLP-BB solvers for the moderate and hard instances in the test suite. For easy instances, the performance of FilMINT and MINLP-BB is quite similar. We drop these instances from our analysis and do not show any computational results for them (detailed results are available from the authors on request). The results for the moderate problems show a significant improvement of LP/NLP-BB compared to MINLP-BB. The results for the hard instances, however, show that this simplistic implementation of LP/NLP-BB is not competitive with MINLP-BB for difficult instances, especially for finding high quality feasible solutions. This

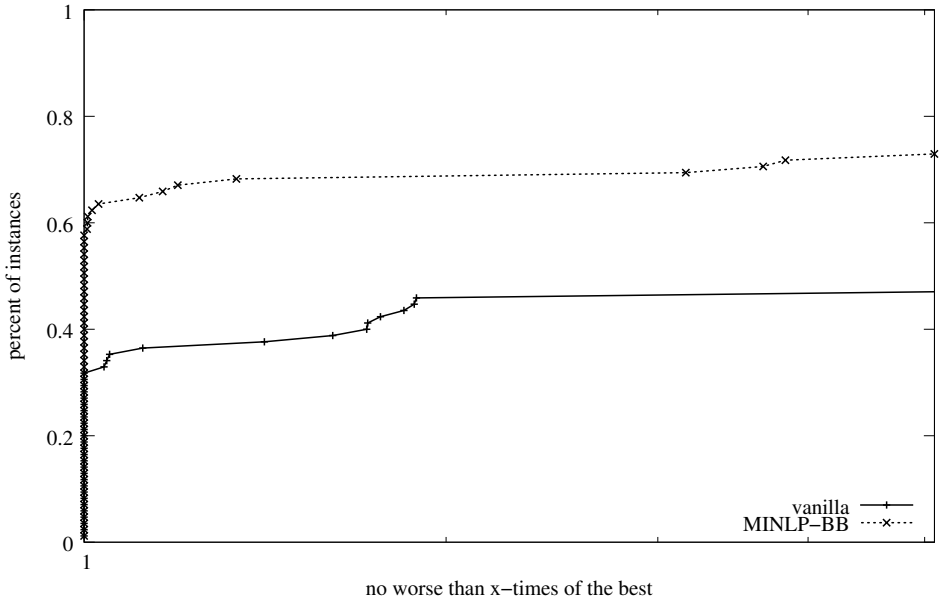

Figure 2: Performance profile comparing vanilla and MINLP-BB for hard instances.

observation motivates us to explore the use of advanced MILP features that can easily be switched on with MINTO.

The remainder of our computational experiment is divided into two parts. In the first part, we explore the effect of various MILP features such as cutting planes, heuristics, branching and node selection rules, and preprocessing. By turning on each feature individually, we obtain an indication of which MILP techniques have the biggest impact. The IP features that are found to work well in this part are then included in an intermediate version of our solver (called vanIP). In the second part, we build on this improved version of LP/NLP-BB by adding features that affect the generation and management of cuts and outer approximations. Each additional feature that appears to improve the performance is included in our final solver, called FilMINT. Finally, we benchmark FilMINT against to two MINLP solvers, MINLP-BB (Leyffer [1998]) and Bonmin (Bonami et al. [2008]).

2 Exploiting the MILP Framework

In this section we explore the benefits for the LP/NLP-BB algorithm of including standard MIP features such as cutting planes, heuristics, branching and node selection rules, and preprocessing. We conduct careful experiments to assess the impact of these features on the performance of the algorithm.

2.1 Cutting Planes, Preprocessing, and Primal Heuristics

Cutting planes have become an important tool in solving mixed integer programs. FilMINT uses the cut generation routines of MINTO to strengthen the formulation and cut off the fractional solution. If a fractional solution to the linear program CMP($\mathcal{K}, l, u$) is obtained, MINTO tries to exclude this solution with clique inequalities, implication inequalities, lifted knapsack covers, lifted GUB covers, and lifted simple generalized flow covers. The interested reader may examine the survey paper [Linderoth and Ralphs, 2005] and the references therein for a description of these cutting planes.

Another effective technique in modern algorithms for solving MILPs is preprocessing. By preprocessing the MILP, matrix coefficients and variable bounds can be improved, thereby increasing the bound obtained from the solution to the problem's relaxation. Savelsbergh [1994] explains the variety of preprocessing techniques used for solving MILPs. Many of these same techniques can be applied to the outer-approximation master problem MP($\mathcal{K}$). However, "dual" preprocessing techniques, which rely on inferring characteristics about an optimal solution, may not be valid when applied to MP($\mathcal{K}$) unless all integer points in $Y \cap \mathbb{Z}^p$ are included in $\mathcal{K}$. We therefore deactivate the dual processing features of MINTO in the implementation of FilMINT.

Primal heuristics for MILP aim to find good feasible solutions quickly. A high-quality solution, and the resultant upper bound on the optimal solution value, that is obtained at the beginning of the search procedure can significantly reduce the number of nodes that must be evaluated to solve a MILP. Feasible (integer-valued) solutions to the outer-approximation master MP($\mathcal{K}$) are doubly important to the LP/NLP-BB algorithm, since it is at these points where the problem (NLP(y^k)) is solved, resulting in additional linearizations that are added to MP($\mathcal{K}$). MINTO uses a diving-based primal heuristic to obtain feasible solutions at nodes of the branch-and-bound tree. In the diving heuristic, some integer variables are fixed and the linear program re-solved. The fixing and re-solving is iterated until either the an integral solution is found or the linear program becomes infeasible. FilMINT uses MINTO's primal heuristic to find feasible solutions to (MP($\mathcal{K}$)).

To quantify the effectiveness of MILP cutting planes, preprocessing, and primal heuristics in the context of solving MINLP, we conducted an experiment in which the vanilla LP/NLP-BB algorithm was augmented with each of the individual techniques. Figures 3–4 show the performance profiles for this experiment. In the figures, the lines MILPcuts, preprocess, and primal-heuristic graph the relative performance of the algorithm with only that feature enabled. The performance of the vanilla version of the algorithm is also depicted.

The results of the experiment indicate that the addition of each individual MILP technique provides a slight improvement over the vanilla algorithm on moderately difficult in-

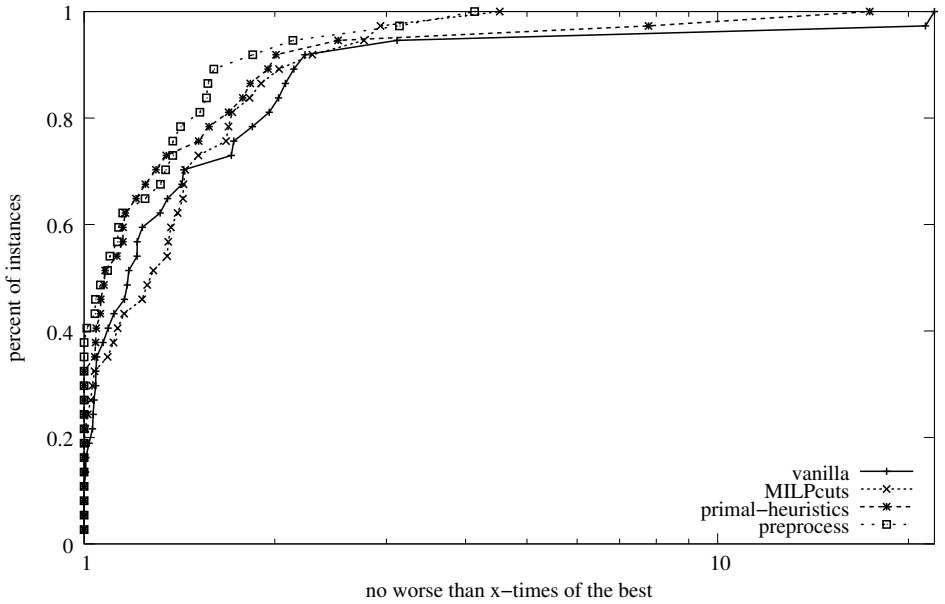

Figure 3: Relative Performance of MILP techniques for moderate instances.

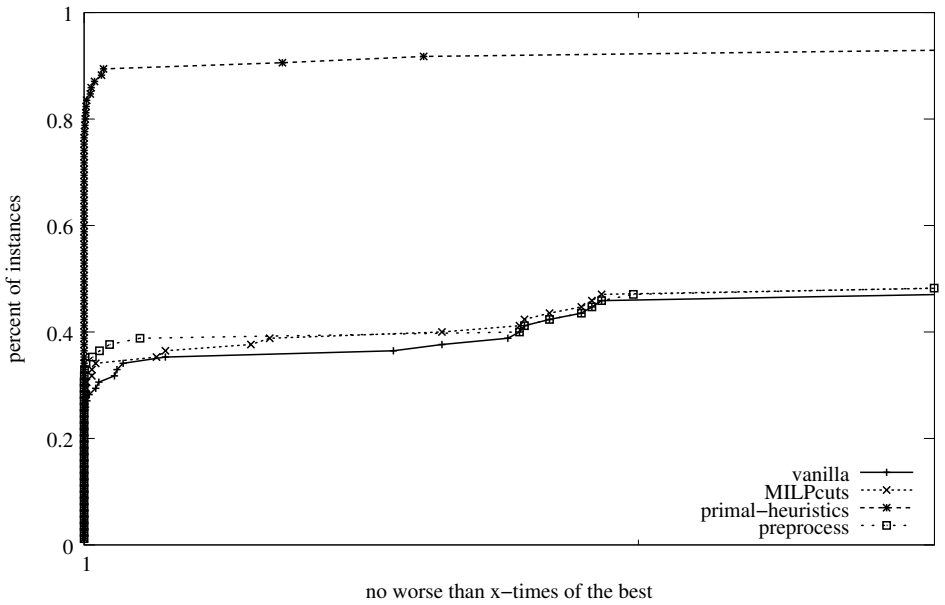

Figure 4: Performance profile showing the effect of MILP cuts, preprocessing and heuristics for hard instances.

stances. Figure 4 shows that the addition of primal heuristics has a very positive impact when solving the difficult instances in the test suite. Recall that our solution metric for the difficult instances was the (scaled) value of the best solution found on that instance, so it is not too surprising that primal heuristics performed so well in this case. It is, however, encouraging to note that heuristics designed to find integer feasible solutions to MILP can be applied on the linear master problem (MP($\mathcal{K}$)), and are often useful for finding high quality solutions to the MINLP. An emerging line of research is involved in developing primal heuristics especially tailored to MINLP [Bonami et al., 2006].

2.2 Branching and Node Selection Rules

Another advantage of building FilMINT within the MINTO framework is that it provides the same branching and node selection rules that MINTO provides. A branching scheme is specified by two rules: a branching variable selection rule and a node selection rule, and MINTO comes equipped with a variety of built-in strategies for each. The branching rules available in MINTO are are maximum fractionality, penalty-based, strong branching, pseudocost-based, an adaptive method combining the penalty and pseudocost-based methods, and SOS branching. The node selection rules available in MINTO are best bound, depth first, best projection, best estimate, and an adaptive method that searches depth-first until it is estimated that the current node will not lead to an optimal solution, whereupon the node with the best estimated solution is searched. The interested reader is referred to the paper of Linderoth and Savelsbergh [1999] for a description of these rules. The penalty-based and adaptive branching rules require access to the simplex tableau of the active linear program, and these methods were not available in MINTO for our specific choice of LP solver. The SOS-branching strategy is very problem specific, so we excluded it from consideration as a general branching rule. Thus, the branching rules that are of interest include maximum fractionality, strong-branching, and pseudo-cost based. The node selection rules that we tested were best bound, depth first, best estimate, and the adaptive rule. The vanilla algorithm uses by default maximum fractional branching and the best-bound node selection strategy.

To compare the methods, the vanilla version of the LP-NLP/BB algorithm was run using each branching variable and node selection method on all instances in the test suite. In this experiment, when testing the branching variable selection, the best-bound node selection rule was used, and when testing node selection, the most-fractional variable selection method was used.

Performance profiles of the computational experiments for the different branching rules tested are shown in Figures 5 and 6. The results show that pseudocost branching outperforms all other rules when used in the LP/NLP-BB algorithm. The performance gains are quite significant, but not unexpected, since similar performance gains over maximum frac-

tional branching are also seen for MILP [Linderoth and Savelsbergh, 1999]. The poor performance of strong branching was surprising. The default implementation of strong branching in MINTO can be quite time-consuming, and the time spent seems to be not worth the effort for our test instances.

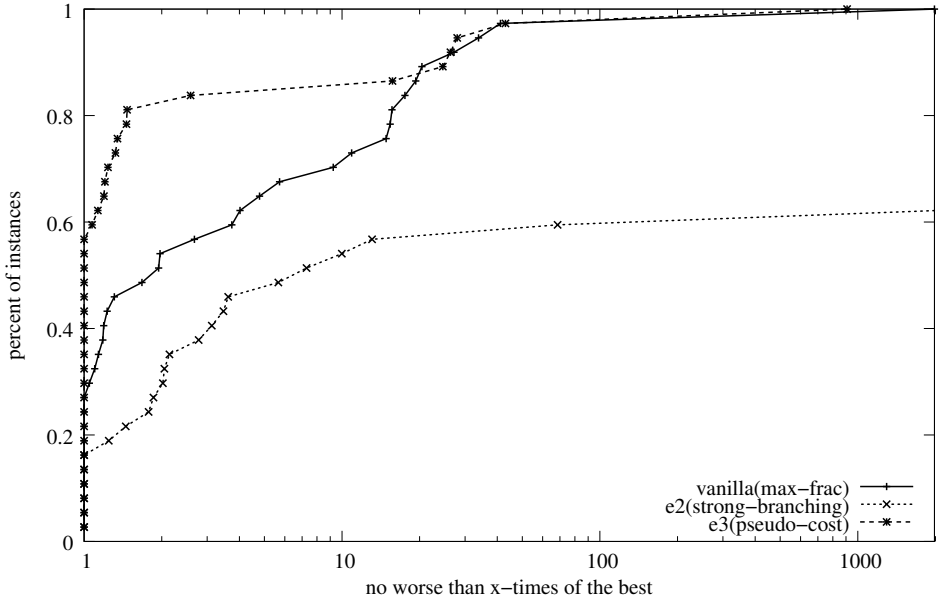

Figure 5: Relative Performance of Branching Rules for Moderate Instances.

Performance profiles of the results for the experiments dealing with node selection are given in Figures 7 and 8. The experimental results indicate that the adaptive node selection rule gives the biggest improvement over pure best-bound search, followed closely by the node selection rule based on best estimates. While the performance gains in terms for the moderate problem instances is good, the improvement for the hard problems is quite significant. This can be explained by the fact that feasible solutions are more likely to be found deep in the branch-and-bound tree, and our metric for hard instances is the scaled value of the best solution found.

2.3 Summary of MILP Features

The computational experiments helped us identify features from a MILP solver that would improve the efficiency of the LP/NLP-BB algorithm for solving (MINLP). Based on the experiments, we include MINTO's cutting planes, preprocessing, primal heuristics, pseudocost-based branching, and MINTO's adaptive node selection strategy as part of the default solver for subsequent experiments. However, since FilMINT is implemented directly in a MILP branch-and-cut framework, it is a simple matter to implement customized branching rules, node selection strategies, or cutting planes for specific problem classes instead of these de-

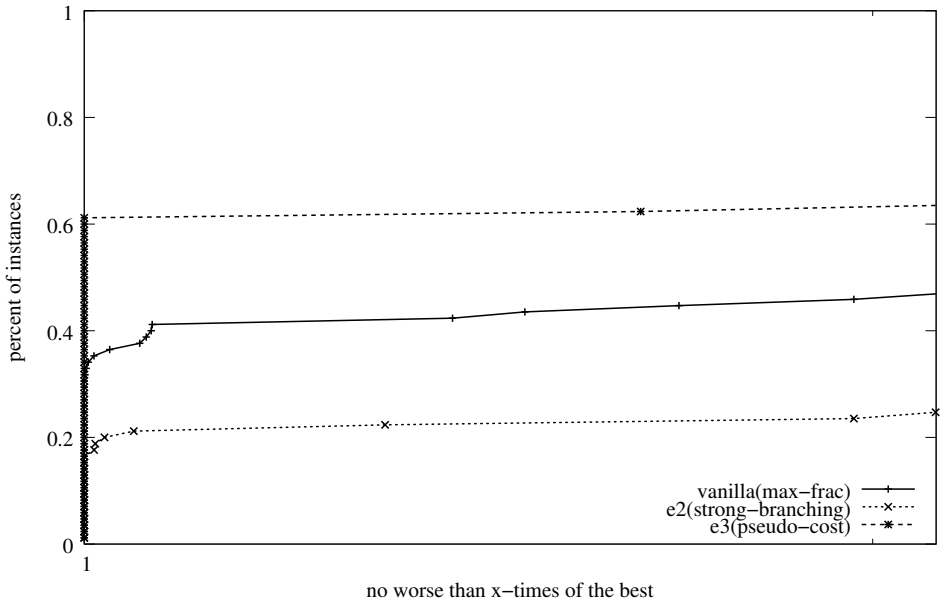

Figure 6: Relative Performance of Branching Rules for Difficult Instances.

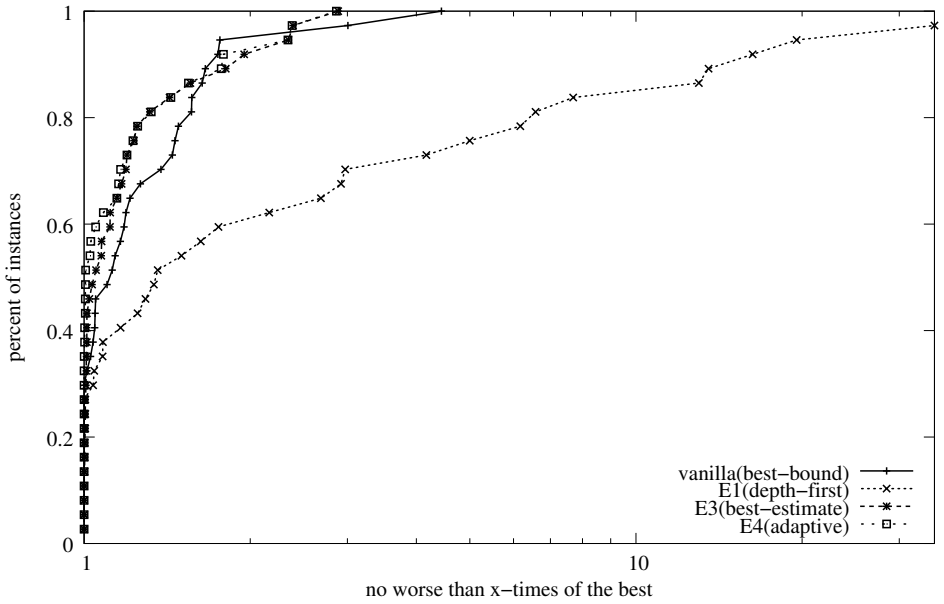

Figure 7: Relative Performance of Node Selection Rules for Moderate Instances.

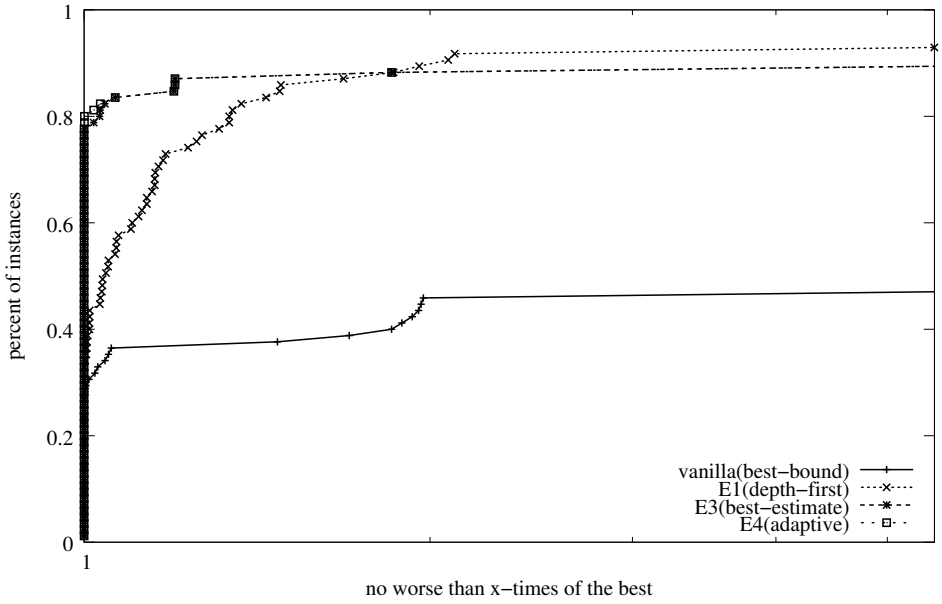

Figure 8: Relative Performance of Node Selection Rules for Difficult Instances.

fault rules.

Figures 9–10 show the cumulative effect of turning on all selected MILP-based features for both the moderate and difficult instances. In the performance profiles, the label vanIP refers to the solver with the MILP features turned on. The solver vanIP quite significantly outperforms the straightforward implementation of the LP/NLP-BB solver (vanilla).

3 Linearization Management

In the branch-and-cut algorithm for solving MILP, cuts are used to approximate the convex hull of integer solutions. In the LP/NLP-BB algorithm, linearizations of the constraints are used to approximate the feasible region of the NLP relaxation. For convex MINLPs, linearizations may be generated at *any* point and still give a valid outerapproximation of the feasible region, so we have at our disposal a mechanism for enhancing the LP/NLP-BB algorithm by adding many linear inequalities to the master problem (MP($\mathcal{K}$)). In the branch-and-cut algorithm for MILP, cutting planes like Gomory cuts, mixed-integer-rounding cuts, and disjunctive cuts are similar in the sense that it is easy to quickly generate a large number of linear inequalities to approximate the convex hull of integer solutions. In our implementation FilMINT, a fundamental design philosophy is to treat linearizations of the nonlinear feasible region in a manner similar to the way cutting planes are treated by a branch-and-cut algorithm for MILP.

In this section, we first discuss simple strategies for effectively managing the large number of inequalities an enhanced LP/NLP-BB algorithm may generate. Key to an effective

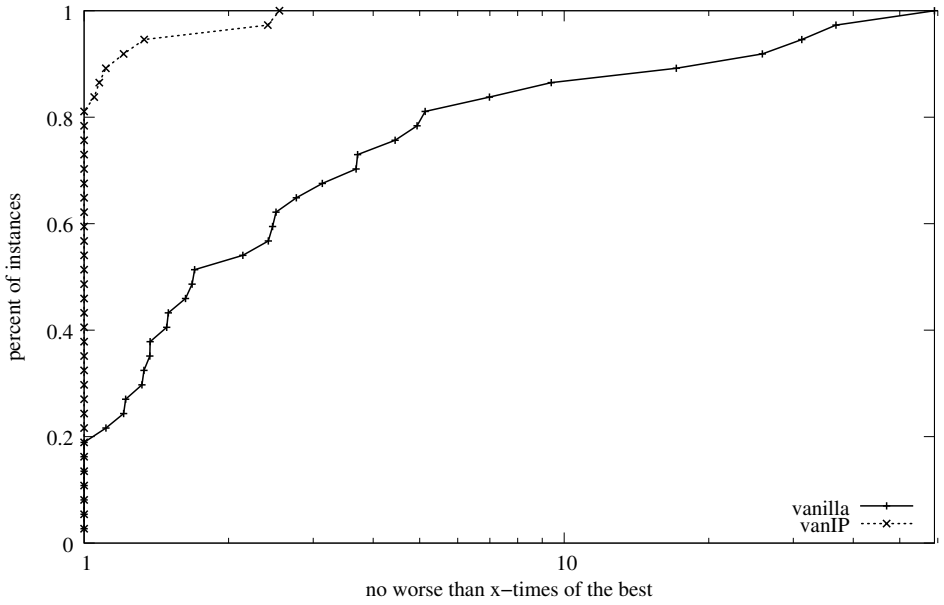

Figure 9: Relative Performance of MILP-Enabled Solver for Moderate Instances.

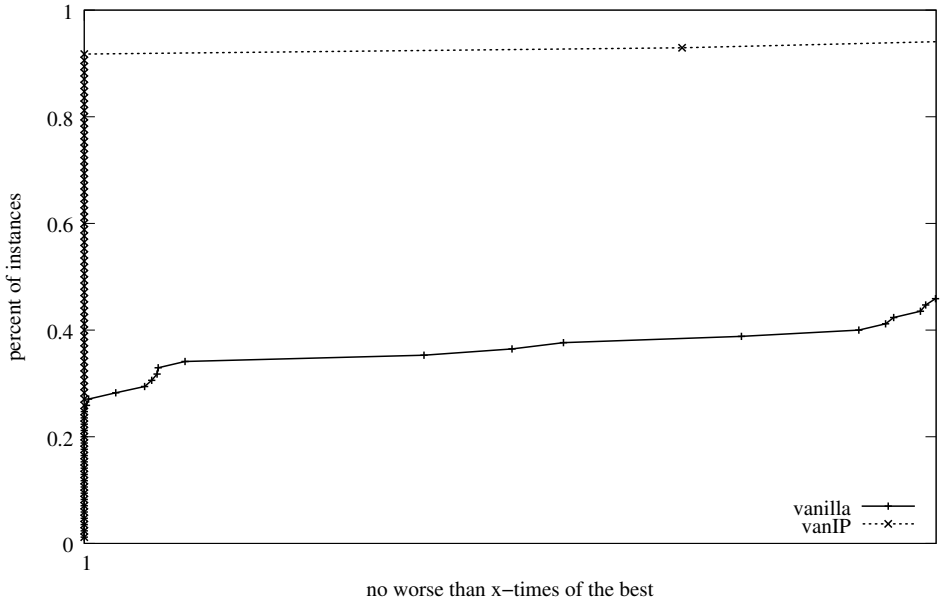

Figure 10: Relative Performance of MILP-Enabled Solver for Difficult Instances.

management strategy is a policy for deciding when inequalities should be added and removed from the master problem (MP($\mathcal{K}$)) and also when additional inequalities should be generated. The section concludes with a discussion of a general mechanism for generating linearizations that trades off the the quality of approximation of the nonlinear feasible region with the time required to obtain the linearization.

3.1 Linearization Addition and Removal

Adding linearizations to the master problem (MP($\mathcal{K}$)) increases the solution time of the linear program solved at each node and adds to the storage requirements of the algorithm. An effective implementation must then consider ways to be frugal about adding additional linearizations. Similar implementation issues arise in a branch-and-cut algorithm for MILP, and our techniques are based on this analogy.

One simple strategy for limiting the number of linear inequalities in the continuous relaxation of the master problem (CMP($\mathcal{K}, l, u$)) is to only add inequalities that are violated by the current solution to the linear program. Another simple strategy for controlling the size of (MP($\mathcal{K}$)) is to remove inactive constraints from the formulation. MINTO has a row-management feature that automatically performs this reduction. MINTO monitors the values of the dual variables at the end of every solution to a linear program. If the dual variable for a constraint is zero, implying that the constraint is inactive, for a fixed number of consecutive linear program solutions, then MINTO deactivates the constraint and places the inequality in an auxiliary data structure known as a cut pool. If a constraint in the cut pool later becomes violated, it is added back to the active formulation. MINTO has an environment variable, `MIOCUTDELBND`, which indicates the number of consecutive solutions for which a constraint can be inactive before it is removed. After conducting a few small-scale experiments, the value of `MIOCUTDELBND` was set to 15 in our implementation. Figure 11 is a performance profiles demonstrating the positive impact of including each of these simple linearization management features in a LP/NLP-BB algorithm for the moderately difficult instances in our test suite. The features did not have a significant impact for the difficult instances.

A more sophisticated approach to managing the size of the master problem (MP($\mathcal{K}$)) is to aggregate linearizations obtained from earlier points, rather than removing them. An effective way to perform the aggregation may be based on Benders cuts ([Geoffrion, 1972]). Summing the objective linearizations and the constraint linearizations weighted with the optimal NLP multipliers μ^k from the solution of (NLP(y^k)) yields the Benders cut:

$$\eta \geq f(x^k, y^k) + (\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k))^T (y - y^k).$$

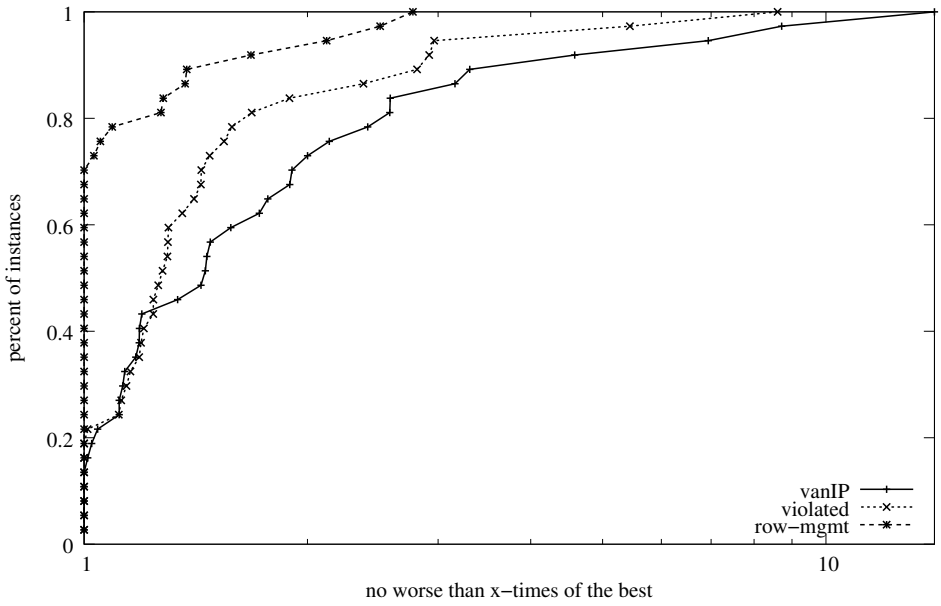

Figure 11: Performance profile showing the effect of row management for moderate instances.

This cut can be simplified by observing that the term

$$\nabla_y f(x^k, y^k)^T + (\mu^k)^T \nabla_y g(x^k, y^k) = \gamma^k$$

corresponds to the NLP multiplier γ^k corresponding to fixing the integer variables $y = y^k$ in (NLP(y^k)). The Benders cut is weaker than the outer approximations; on the other hand, it compresses information from $m + 1$ linear inequalities into a single cut. Given MINTO's tunable and automatic row management strategy, we did not see any specific need to employ such aggregation schemes in our first implementation of FilMINT, but we may reinvestigate this decision in subsequent software releases.

3.2 Cut or Branch?

In branch-and-cut, there is a fundamental implementation choice that must be made when confronted with an infeasible (fractional) solution: should the solution be eliminated by cutting or branching? Sophisticated implementations of branch-and-cut for solving MILP rely on a variety of *cut management* techniques for answering this question. The interested reader is directed to Atamtürk and Savelsbergh [2005] to see the options for controlling cut management in commercial MILP systems.

One cut management approach typically employed is to not add cutting planes at every node of the search tree. Rather, cutting planes are generated at certain nodes, for example at a fixed interval, with a bias towards generating cuts at nodes close to the root of the search

tree. For the LP/NLP-BB algorithm, the branching variable selection provides an additional motivation for generating linearizations very early in the search process. Most branching procedures are based on selecting a variable that will increase the objective function of the two child subproblems significantly. As pointed out by Forrest et al. [1974], branching on a variable that has little or no effect on the solution at subsequent nodes results in a *doubling* of the amount of work necessary to completely process that node. Thus, by the very nature of the branch and bound process, the branching decisions made at the top of the tree are the most crucial. In the LP/NLP-BB Algorithm 1.1, if few linearizations (OA(x^k, y^k)) are included in the master problem (MP($\mathcal{K}$)), the problem can be a very poor approximation of the true problem (MINLP), and branching decisions made based on this master problem may be very poor. Thus, it is quite important for the LP/NLP-BB algorithm that the master problem (MP($\mathcal{K}$)) obtain good linearization information early in the solution process.

A second cut management technique is to add cuts for multiple iterations (typically called *rounds*) at a given node. Algorithmic parameters control the number of rounds at a node.

Based on the analogy to cut management in branch-and-cut for MILP, there are three parameters that we use to control the linearization strategy of our implementation of the LP/NLP-BB algorithm. The first parameter β controls the likelihood that linearizations will be generated at a node. The strategy is biased so that linearizations are more likely to be generated at nodes high in the search tree. Specifically, the probability that linearizations are generated at a node is $\min\{\beta 2^{-d}, 1\}$. Note that in a complete search tree, there are 2^d nodes at level d, so the parameter β is the expected number of nodes at level d at which cuts are generated, if the search tree was complete. The second parameter K is used for detecting "tailing off" of the linearization procedure. If the percentage change in the solution value of the relaxed master problem (CMP($\mathcal{K}, l, u$)) is not at least K, then the algorithm decides to branch rather than to add more linearizations about the current solution point. The final parameter is responsible for ensuring that there is a reasonable balance between branching and cutting by constraint linearizations. Linearizations will be taken around at most M different points at any one node of the search tree. The parameters (β, K, M) are specific to the *type* of point about which we are generating linearizations, the details of which are explained next.

3.3 Linearization Generation Methods

A simple observation is that due to the convexity of the functions f and g in (MINLP), the outerapproximation inequalities (OA(x^k, y^k)) are valid regardless of the point (x^k, y^k) about which they are taken. This gives the LP/NLP-BB algorithm great flexibility in generating linearizations to approximate the feasible region of (MINLP). The tradeoff to consider is the time required to generate the linearization versus the quality/strength of the resulting

linearization. We examined three different ways to select points about which to add linearizations, spanning the spectrum of this tradeoff.

The first method simply linearizes the functions f and g about the fractional point $(\hat{x}, y^k)$ obtained as a solution to (CMP($\mathcal{K}, l, u$)) at the **evaluate** step of Algorithm 1.1. This point selection mechanism is called the *ECP-based method*, as it is analogous the Extended Cutting Plane method for solving (MINLP) proposed by Westerlund and Pettersson [1995], which is itself an extension of the cutting plane method of Kelley [1960] for solving convex programs. The ECP-based point selection has the advantage that it is extremely fast to generate linearizations, requiring only the evaluation of the gradient of the objective function and the Jacobian of the constraints at the specified point. However, the points about which linearizations are taken may be far from feasible, so the resulting linearizations may form a poor approximation of the nonlinear feasible region.

In the second point selection method, we fix the integer decision variables to the values obtained from the solution of (CMP($\mathcal{K}, l, u$)) ($y = y^k$), and the nonlinear program (NLP(y^k)) is solved to obtain the point about which to linearize. This method is called *fixfrac*, as it is similar in spirit to the original LP/NLP-based algorithm, but now integer decision variables y may be fixed at *fractional* values. The fixfrac method has the advantage of generating linearization about points that are closer to the feasible region than the ECP-based method, at the expense of solving the nonlinear program (NLP(y^k)).

In the third point selection method, no variables are fixed (save those that are fixed by the nodal subproblem), and the NLP relaxation (NLPR(l, u)) is solved to obtain a point about which to generate linearizations. This is called the *NLPR-based method*. The linearizations obtained from the NLPR-based method are the tightest of the three methods that we employ. However, it can be time-consuming to compute the linearizations.

The ECP, fixfrac, and NLPR methods differ in the set of variables fixed before solving a nonlinear program to determine the linearization point. The ECP-based method fixes *all* variables from the solution of (CMP($\mathcal{K}, l, u$)), the fixfrac method fixes only the integer decision variables y, and the NLPR-based method fixes no decision variables. Further research and empirical experiments will be required to determine if more efficient point selection methodologies exist.

The three classes of linearizations form a hierarchy, with ECP linearizations being the weakest/cheapest, and NLPR cuts being the strongest/most costly. The manner in which we generate linearizations in FilMINT exploits this hierarchy. First, ECP linearizations are generated until it appears that they are no longer useful. Next fixfrac linearizations are generated, and finally NLPR linearizations are generated. Our final strategy for producing points about which to generate linearizations combines all three methods and is given in pseudo-code form in Algorithm 3.1.

Let $d =$ depth of current node (l, u, η).
Let $r \in [0, 1]$ be a uniformly generated random number
if $r \leq \beta_{\text{ecp}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{ecp}}$ **AND** $n_{\text{ecp}} \leq M_{\text{ecp}}$ **then**
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\hat{x}, y^k)\}$. **(ecp)**
  $n_{\text{ecp}} \leftarrow n_{\text{ecp}} + 1$.
else if $r \leq \beta_{\text{ff}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{ff}}$ **AND** $n_{\text{ff}} \leq M_{\text{ff}}$ **then**
  Solve NLP(y^k). **(fixfrac)**
  if NLP(y^k) is feasible **then**
    Let $(\tilde{x}^k, y^k)$ be solution to NLP(y^k)
  else
    Let $(\tilde{x}^k, y^k)$ be solution to NLPF(y^k)
  end if
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\tilde{x}^k, y^k)\}$.
  $n_{\text{ff}} \leftarrow n_{\text{ff}} + 1$.
else if $r \leq \beta_{\text{nlpr}} 2^{-d}$ **AND** $(\eta^k - \eta^{k-1})/|\eta^{k-1}| \geq K_{\text{nlpr}}$ **AND** $n_{\text{nlpr}} \leq M_{\text{nlpr}}$ **then**
  Solve NLPR(l^k, u^k). **(nlpr)**
  Let $(\bar{x}^k, \bar{y}^k)$ be solution to NLPR(l^k, u^k)
  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\bar{x}^k, \bar{y}^k)\}$.
  $n_{\text{nlpr}} \leftarrow n_{\text{nlpr}} + 1$.
end if

Algorithm 3.1: Algorithm for generating linearizations.

Values for the parameters in Algorithm 3.1 to use in the default version of FilMINT were chosen after careful experimentation and are shown in Table 1. Note that $\beta_{\text{nlpr}} = 0$, so that NLPR linearizations are not added at nodes of the branch-and-cut tree by default in FilMINT. However, as stated in Algorithm 1.1, linearizations about the solution to the original nonlinear programming relaxation NLPR(y^l, y^u) are added to initialize the algorithm.

Table 1: Default Linearization Parameters

Method	β	M	K
ECP	10	10	0.1%
Fixfrac	1	10	0.1%
NLPR	0	1	0.1%

To demonstrate the effectiveness of the linearization-point selection methodologies, an experiment was conducted wherein the MILP-technique enhanced version of the LP/NLP-BB algorithm was further augmented with each of the additional linearizations techniques individually. Runs of the experiment can be viewed as implementing Algorithm 3.1, with $M = 0$ for the linearization point selectiom mechniams *not* under study. The cut management parameters (β, M, K) for the chosen linearization point selection method were the same as in Table 1, except for NLPR, in which a value of $\beta_{\text{nlpr}} = 1$ was used. Figures 12 and 13 show performance profiles of this experiment. The solver row-mgmt on the profile is the MIP-enhanced LP/NLP-BB algorithm without any additional linearizations, and the solver filmint is the default version of the FilMINT solver. The experiment conclusively demonstrates that linearizing about additional points is quite advantageous for the LP/NLP-BB algorithm. Also, using multiple mechanisms for choosing the points about which to generate linearizations, as is done in FilMINT, helps the algorithm's performance.

4 Comparison of MINLP Solvers

In this section, we report on an experiment comparing the performance of FilMINT with two other well-known solvers for MINLPs on our suite of test instances. FilMINT is compared to Bonmin [Bonami et al., 2008] and to MINLP-BB [Leyffer, 1998]. MINLP-BB is a branch-and-bound solver based that uses the nonlinear programming relaxation NLPR(l, u) to provide a lower bound on z_{MINLP}. The NLP subproblems in MINLP-BB are solved by filterSQP, the same solver that is used in FilMINT. The Bonmin solver consists of a suite of algorithms, and FilMINT is compared against the `I-Hyb` hybrid algorithm of Bonmin. The `I-Hyb` algorithm of Bonmin is an implementation of the LP/NLP-BB algorithm that has been augmented with two additional features. First, NLP relaxations NLPR(l, u) are solved occasionally (every $L = 10$ nodes) during the branch-and-bound search. Second, truncated MILP branch-and-bound enumerative procedures are performed in an effort to find integer-feasible solutions. The enumerative procedure performed by Bonmin is akin to

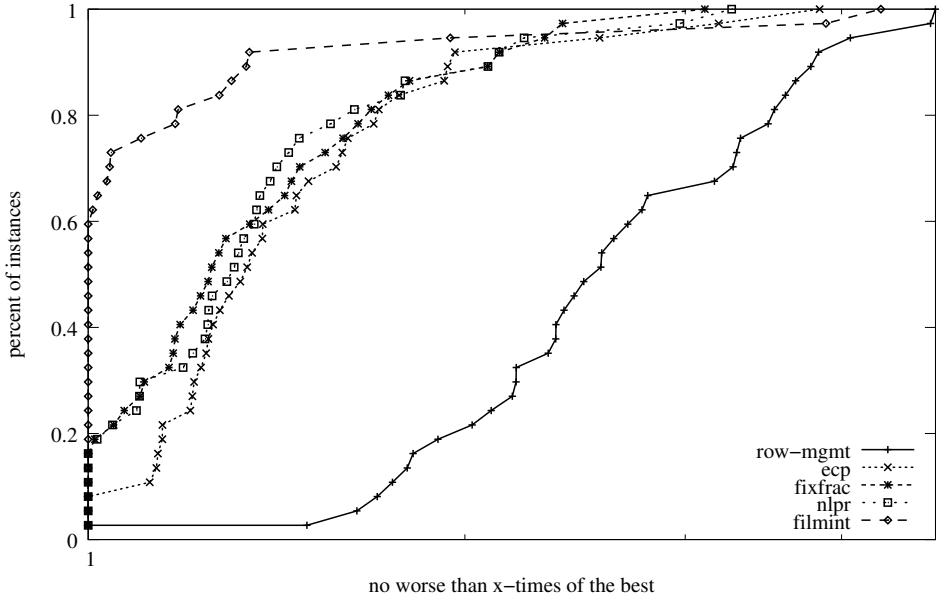

Figure 12: Performance Profile Comparing Linearization Point Selection Schemes on Moderate Instances

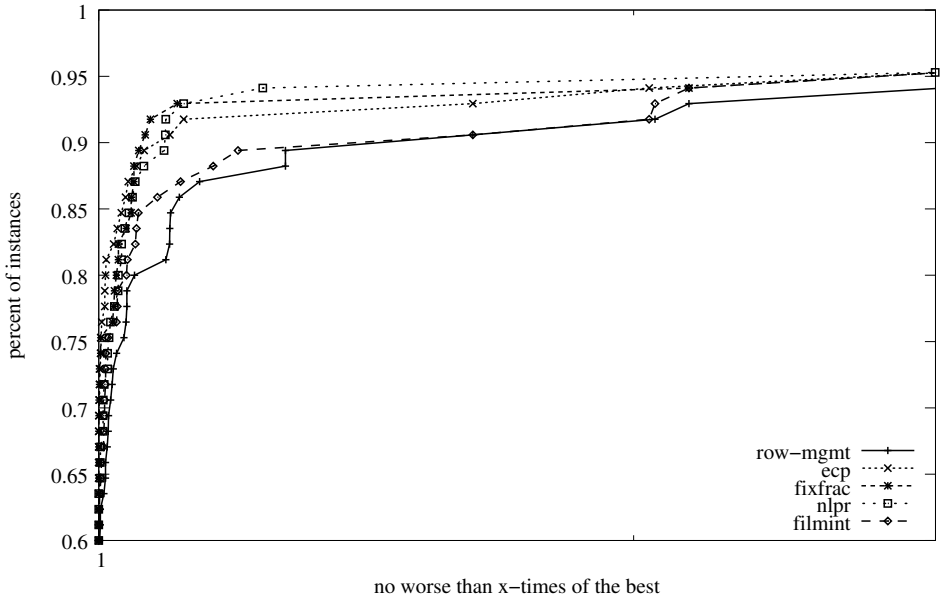

Figure 13: Performance Profile Comparing Linearization Point Selection Schemes on Difficult Instances

the diving-based primal heuristic found in MINTO v3.1 (and used by FilMINT).

Experiments were run using two different versions of the Bonmin software. The first version `Bonmin` is a version of the executable built from source code taken from the COIN-OR code repository during the summer of 2006. The second version, `Bonmin-v2`, refers to a more recent release of the software, (specifically version 0.1.4), available from `https://projects.coin-or.org/Bonmin/browser/releases`.

The parameters in the default version of FilMINT were optimized based on all previous experiments. Specifically, MINTO v3.1's default MIP features (preprocessing, pseudocost branching variable selection, adaptive node selection, cutting planes) were enabled. Linearizations in default FilMINT are generated, added, and removed in a manner described in Section 3, with the paramaters given in Table 1.

Figures 14 and 15 compare the solvers FilMINT, Bonmin (v1), Bonmin (v2), MINLP-BB, and the vanilla implementation of the LP/NLP-BB algorithm. The computational setup used for this experiment is the same that we have used for all our experiments, detailed in Section 1.3. All solvers were given a time limit of four hours for each instance. The profile reveals that for the moderate instances, FilMINT and Bonmin-v2 are clearly the most effective solvers. For the difficult instances, FilMINT and both versions of Bonmin are the most effective at finding high quality feasible solutions within the four-hour time limit. Tables showing the exact times taken for each instance are given in the Appendix. A total of 88 out of the 122 instances in our test suite are solved by one of the 5 solvers to provable optimality within the 4 hour time limit. Figure 16 is a performance profile (using time as the performance metric) comparing each solver's performance on this subset of the instances.

5 Conclusions

FilMINT is an implementation of the Quesada-Grossmann LP/NLP-BB algorithm for solving convex mixed integer nonlinear programs. By augmenting FilMINT with modern algorithmic techniques found in mixed integer programming software and by enhancing the linearization procedure of the algorithm, a robust and efficient solver was created. Continuing work aims an enhancing FilMINT's effectiveness, including new heuristic techniques and cutting planes. FilMINT is available for use on the NEOS Server at `http://neos.mcs.anl.gov/neos/solvers/minco:FilMINT/AMPL.html`.

Acknowledgments

The second author is supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

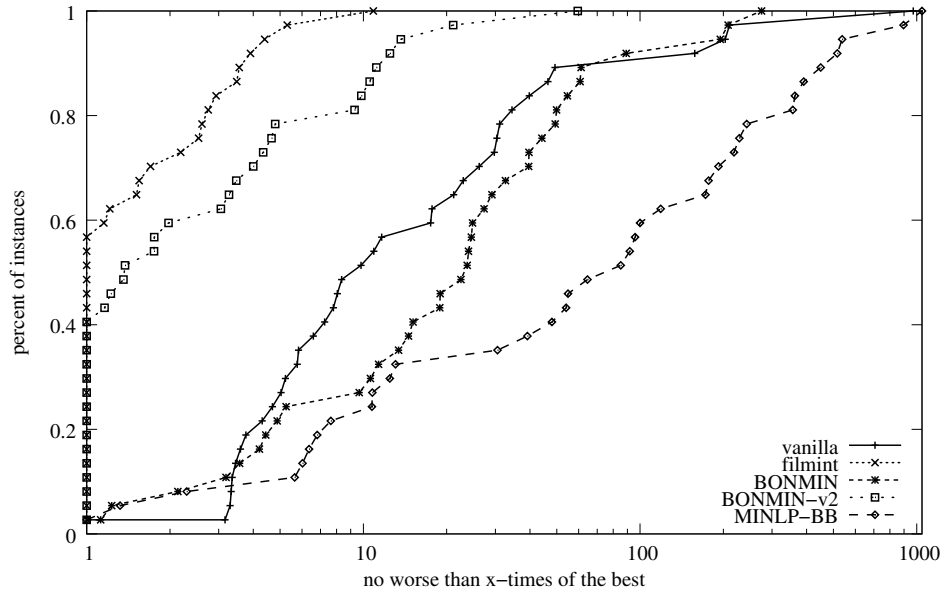

Figure 14: Performance Profile Comparing Solvers on Moderate Instances.

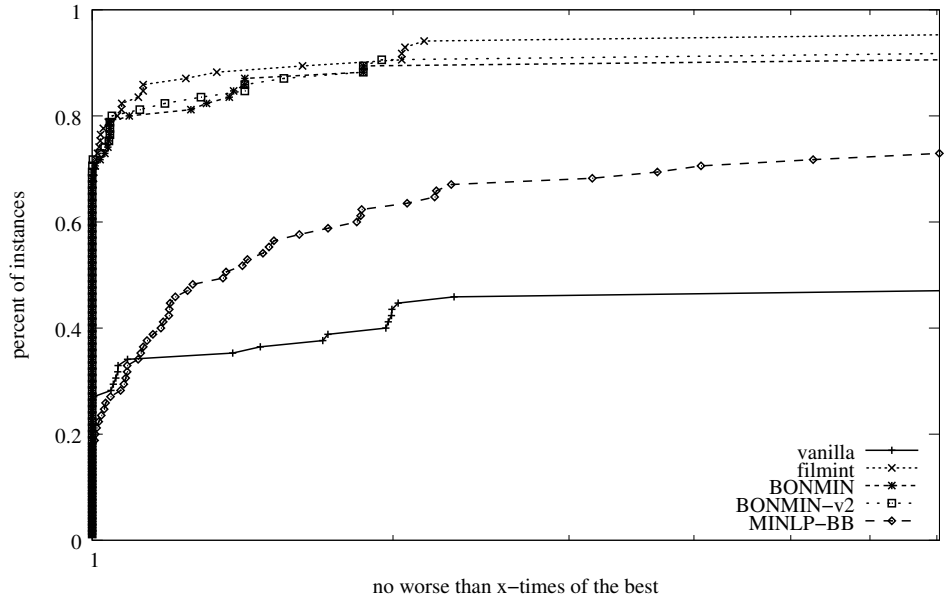

Figure 15: Performance Profile Comparing Solvers on Difficult Instances

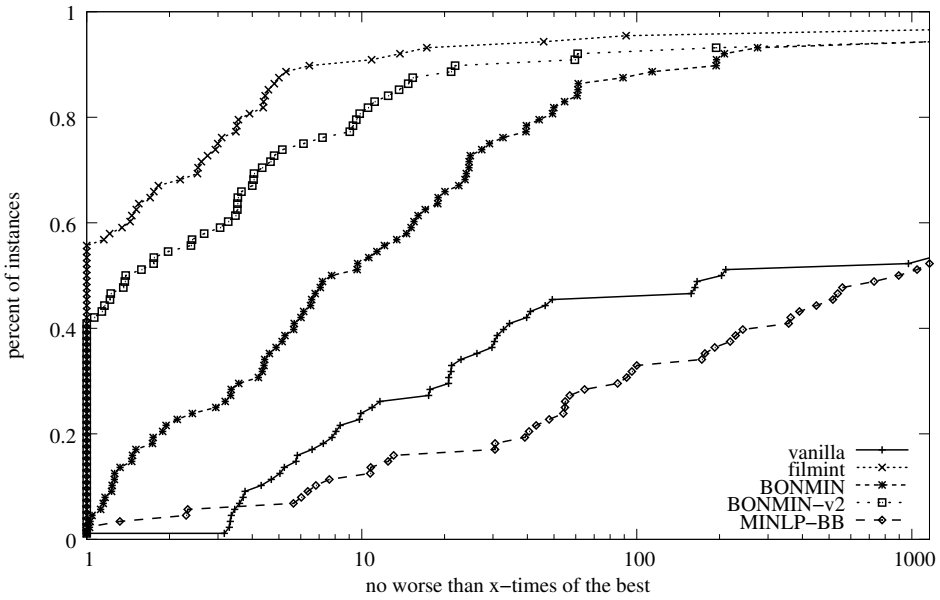

Figure 16: Performance Profile Comparing Solvers on Solved Instances

References

A. Atamtürk and M. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140:67–124, 2005.

P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 2008. to appear.

P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. Research Report RC23862 (W0602-029), IBM, 2006.

M. R. Bussieck, A. S. Drud, and A.Meeraus. MINLPLib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1), 2003.

I. Castillo, J. Westerlund, S. Emet, and T. Westerlund. Optimization of block layout deisgn problems with unequal areas: A comparison of milp and minlp optimization methods. *Computers and Chemical Engineering*, 30:54–69, 2005.

R. J. Dakin. A tree search algorithm for mixed programming problems. *Computer Journal*, 8:250–255, 1965.

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

M. A. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

J. Forrest. CBC, 2004. Available from `http://www.coin-or.org/`.

J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin. Practical solution of large scale mixed integer programming problems with UMPIRE. *Management Science*, 20:736–773, 1974.

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, 1993.

D. M. Gay. Hooking your solver to AMPL. Technical Report 97-4-06, Computing Sciences Research Center, Bell Laboratories, 1997.

A. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

I. E. Grossmann. Review of nonlinear mixed–integer and disjunctive programming techniques. *Optimization and Engineering*, 3:227–252, 2002.

O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

I. Harjunkoski, T. Westerlund, R. Pörn, and H. Skrifvars. Different transformations for solving non–convex trim loss problems by MINLP. *European J. Operational Research*, 105:594–603, 1988.

V. Jain and I. E. Grossmann. Cyclic scheduling of continuous parallel-process units with decaying performance. *AIChE Journal*, 44(7):1623–1636, 1998.

J. E. Kelley. The cutting plane method for solving convex programs. *Journal of SIAM*, 8(4): 703–712, 1960.

G. R. Kocis and I. E. Grossmann. Global optimization of nonconvex mixed–integer nonlinear programming (MINLP) problems in process synthesis. *Industrial Engineering Chemistry Research*, 27:1407–1421, 1988.

S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee, Dundee, Scotland, UK, 1993.

S. Leyffer. User manual for MINLP-BB, 1998. University of Dundee.

S. Leyffer. MacMINLP: Test problems for mixed integer nonlinear programming, 2003. `http://www-unix.mcs.anl.gov/~leyffer/macminlp`.

J. Linderoth and T. Ralphs. Noncommercial software for mixed-integer linear programming. In *Integer Programming: Theory and Practice*, pages 253–303. CRC Press Operations Research Series, 2005.

J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

I. Quesada and I. E. Grossmann. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

A. J. Quist, R. van Gemeert, J. E. Hoogenboom, T. Ílles, C. Roos, and T. Terlaky. Application of nonlinear optimization to reactor core fuel reloading. *Annals of Nuclear Energy*, 26: 423–448, 1998.

R.Fletcher, S. Leyffer, and P. Toint. On the global convergence of a Filter-SQP algorithm. *SIAM Journal on Optimization*, 13:44–59, 2002.

M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

N. W. Sawaya, C. D. Laird, and P. Bonami. A novel library of non-linear mixed-integer and generalized disjunctive programming problems. In preparation, 2006.

R. Stubbs and S. Mehrohtra. Generating convex polynomial inequalities for mixed 0-1 programs. *Journal of Global Optimization*, 24:311–332, 2002.

A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering*, 19(Supplement 1):131–136, 1995.