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

September 30, 2007

[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 W-31-109-ENG-38, 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§

September 30, 2006

Abstract

We describe a new solver for mixed integer nonlinear programs (MINLPs) that implements a linearization-based algorithm in a branch-and-cut framework. The framework includes cutting planes, primal heuristics, and other well-known techniques for solving mixed integer linear programs (MILPs). The solver FilMINT (Filter-Mixed INTeger optimizer) combines the MINTO branch-and-cut framework for MILP with filterSQP used to solve the nonlinear programs that arise as subproblems in the algorithm. In contrast to the traditional outer-approximation algorithm, the algorithm implemented by FilMINT avoids the complete solution of master MILPs by adding new linearizations at open nodes of the branch-and-bound tree whenever an integer solution is found. We present detailed computational experiments that show the benefit of introducing advanced MILP techniques into such a framework. Further, we demonstrate how to use the framework to add and manage linearizations that arise in the algorithm. Comparisons to existing solvers for MINLPs are presented, highlighting 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

Recently, researchers have expressed renewed interest in developing efficient solvers for mixed integer nonlinear programming (MINLP) problems. This 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

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g_j(x,y) \leq 0, \quad j = 1,\ldots,m, \\ & x \in X, y \in Y \cap \mathbb{Z}^p, \end{aligned} \tag{MINLP}$$

where f, g_j are twice continuously differentiable functions, and x and y are continuous and discrete variables, respectively:

$$\begin{aligned} X &\stackrel{\text{def}}{=} \{x \mid x \in \mathbb{R}^n, Dx \leq d\}, \\ Y &\stackrel{\text{def}}{=} \{y \mid y \in \mathbb{R}^p, Ay \leq a, y^l \leq y \leq y^u\}. \end{aligned}$$

In this paper we concentrate on the case where f, g_j are convex. The case where f and g_j are nonconvex or where nonlinear equality constraints are present is beyond the scope of the present paper. We note, however, that our techniques can be applied as a heuristic in such cases, or can form the basis of more sophisticated deterministic techniques based on convex underestimators (Tawarmalani and Sahinidis [2002]).

Methods for the solution of (MINLP) include the branch-and-bound method (Dakin [1965], Gupta and Ravindran [1985]), branch-and-cut (Stubbs and Mehrotra [2002]), outer approximation (Duran and Grossman [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 recent 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.

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. [2005] and this paper.

Our implementation, called FilMINT, is built on top of the mixed integer programming solver MINTO (Nemhauser et al. [1994]). By using MINTO's branch-and-cut framework, 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 (Fletcher and Leyffer [2002], 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. [2005] have also developed a solver for MINLPs called Bonmin. While the two solvers share many of the same charateristics, our work differs from that of Bonami et al. [2005] in a number of significant ways. First, the solver Bonmin is truly a hybrid between a branch-and-bound solver based on nonlinear relaxations and one based on polyhedral outer approximations. FilMINT implements solely the LP/NLP-BB algorithm, because MINTO's branch-and-cut framework restricts us to obtaining lower bounds only from the solution of a *linear* program. Another important distinction between Bonmin and FilMINT is the frequency with which linearizations are created and the management of the linearizations. 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. Specifically, MINTO and CBC's branching and node selection rules are different, MINTO has an advanced preprocessing engine, and different classes of cutting planes are employed by each solver. The last important distinction is FilMINT's use of an active set solver, 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 Bonami et al. [2005].

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 MIP techniques on an LP/NLP-based algorithm. In Section 3 we consider several ways to create and manage the linearizations generated in the algorithm. In Section 4 we show how the NLP solutions can be exploited, and in Section 5 we compare our solver to other MINLP solvers.

1.1 LP/NLP-based Branch and Bound

In this section we formally define the underlying algorithm. 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) 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 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, only a single MILP tree search is required.

To precisely define LP/NLP-BB and our subsequent enhancements, we first make some definitions. We characterize a node $(l, u, \hat{\eta})$ of the branch-and-bound search tree by bounds $\{(l, u)\}$ enforced on the integer variables y and the objective value $\hat{\eta}$. Given bounds (l, u) on y, we define the NLP relaxation of MINLP as

$$\begin{aligned} z_{\text{NLPR}(l,u)} = \text{minimize} \quad & f(x, y) \\ \text{subject to} \quad & g_j(x, y) \leq 0 \quad j = 1, \ldots, m, \\ & x \in X, y \in Y, \\ & l \leq y \leq u. \end{aligned} \tag{NLPR(l, u))}$$

If $l \leq y^l$ and $u \geq y^u$, 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)) yields one or more nonintegral values for the integer variables y.

The NLP subproblem for a fixed y (say y^k) is defined as

$$\begin{aligned} z_{\text{NLP}(y^k)} = \text{minimize} \quad & f(x, y^k) \\ \text{subject to} \quad & g_j(x, y^k) \leq 0 \quad j = 1, \ldots, m, \\ & x \in X. \end{aligned} \tag{NLP(y^k))}$$

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 some feasibility problem for fixed y^k. The form of the feasibility problem (NLPF(y^k)) that is solved by filterSQP is

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m} w_j |g_j(x, y^k)|, \\ \text{subject to} \quad & x \in X. \end{aligned} \tag{NLPF(y^k))}$$

This problem can be interpreted as the minimization of a scaled ℓ_1 norm of the constraint violation.

From the solution to (NLP(y^k)) or (NLPF(y^k)), we can derive valid linear inequalities for (MINLP). The convexity of the nonlinear functions imply that the linearizations about any point (x^k, y^k)) form an outer approximation (OA) of the feasible set and underestimate the objective function. Specifically, if we introduce a dummy variable η in order to replace the objective by a constraint, that is $\text{minimize } \eta \text{ subject to} \eta \geq f(x, y)$, then the inequalities (OA(x_k, y_k)) are valid for MINLP:

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

(OA(x_k, y_k))

$$g_j(x^k, y^k) + \nabla g_j(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad j = 1, \ldots, m.$$

The inequalities (OA(x_k, y_k)) are used to create a master MILP. Given a set of points $\mathcal{K} = \{(x^0, y^0), (x^1, y^1), \ldots, (x^{|\mathcal{K}|}, y^{|\mathcal{K}|}\}$, we form the outer approximation master problem as

$$\begin{aligned} z_{\text{MP}(\mathcal{K})} = \text{minimize} \quad & \eta \\ \text{subject to} \quad & f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq \eta \quad \forall (x^k, y^k) \in \mathcal{K} \quad \text{(MP}(\mathcal{K})) \\ & g_j(x^k, y^k) + \nabla g_j(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad \forall (x^k, y^k) \in \mathcal{K} \quad j = 1, \ldots, m \\ & x \in X, y \in Y \cap \mathbb{Z}^p. \end{aligned}$$

If $\mathcal{K}$ in (MP($\mathcal{K}$)) contains *all* integer points in $Y \cap \mathbb{Z}^p$, and a constraint qualification holds, then $z_{\text{MINLP}} = z_{\text{MP}(\mathcal{K})}$ (Fletcher and Leyffer [1994], Bonami et al. [2005]).

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($\mathcal{K}, l, u$).

$$\begin{aligned} \text{minimize} \quad & \eta \\ \text{subject to} \quad & f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq \eta \quad \forall (x^k, y^k) \in \mathcal{K} \quad \text{(CMP}(\mathcal{K}, l, u)) \\ & g_j(x^k, y^k) + \nabla g_j(x^k, y^k)^T \begin{bmatrix} x - x^k \\ y - y^k \end{bmatrix} \leq 0 \quad \forall (x^k, y^k) \in \mathcal{K} \quad j = 1, \ldots, m \\ & x \in X, y \in Y, l \leq y \leq u \end{aligned}$$

The main algorithm underlying our work, LP/NLP-BB, is now formally stated in pseudo-code form in Algorithm 1.1.

Solve NLPR(y^l, y^u) and let $(\hat{\eta}^0, x^0, y^0)$ be its solution **(initialize)**
if NLPR(y^l, y^u) is infeasible **then**
  Stop. MINLP is infeasible
else
  $\mathcal{K} \leftarrow \{(x^0, y^0)\}, \mathcal{L} \leftarrow \{(y^l, y^u, \hat{\eta}^0)\}, UB \leftarrow \infty$
end if
while $\mathcal{L} \neq \emptyset$ **do**
  Select and remove node $(l^k, u^k, \hat{\eta}^k)$ from $\mathcal{L}$ **(select)**
  Solve CMP$(\mathcal{K}, l^k, u^k)$ and let $(\hat{\eta}^k, \hat{x}, y^k)$ be its solution. **(evaluate)**
  if CMP$(\mathcal{K}, l^k, u^k)$ is infeasible **OR** $\hat{\eta}^k \geq UB$ **then**
    Do nothing.
  else 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 $\hat{\eta}^k \geq UB$. **(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
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{(x^k, y^k)\}$. **Go To (evaluate).**
  else
    Select b such that $y^k_b \notin \mathbb{Z}$. **(branch)**
    $\hat{u}_b \leftarrow \lfloor y^k_b \rfloor, \qquad \hat{u}_j \leftarrow u^k_j \quad \forall j \neq b$
    $\hat{l}_b \leftarrow \lceil y^k_b \rceil, \qquad \hat{l}_j \leftarrow l^k_j \quad \forall j \neq b$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{(l^k, \hat{u}, \hat{\eta}^k)\} \cup \{(\hat{l}, u^k, \hat{\eta}^k)\}$
  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` LP 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, $(\hat{\eta}^k, \hat{x}, y^k)$. Another advantage of using filterSQP for implementing (LP/NLP-BB) is that filterSQP contains an automatric 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.

Figure 1 shows a flowchart of the LP/NLP-BB algorithm and the MINTO application functions used by FilMINT. We note that, for the sake of simplicity, the figure does not show all the details of the algorithm.

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.
- **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 more detail later.

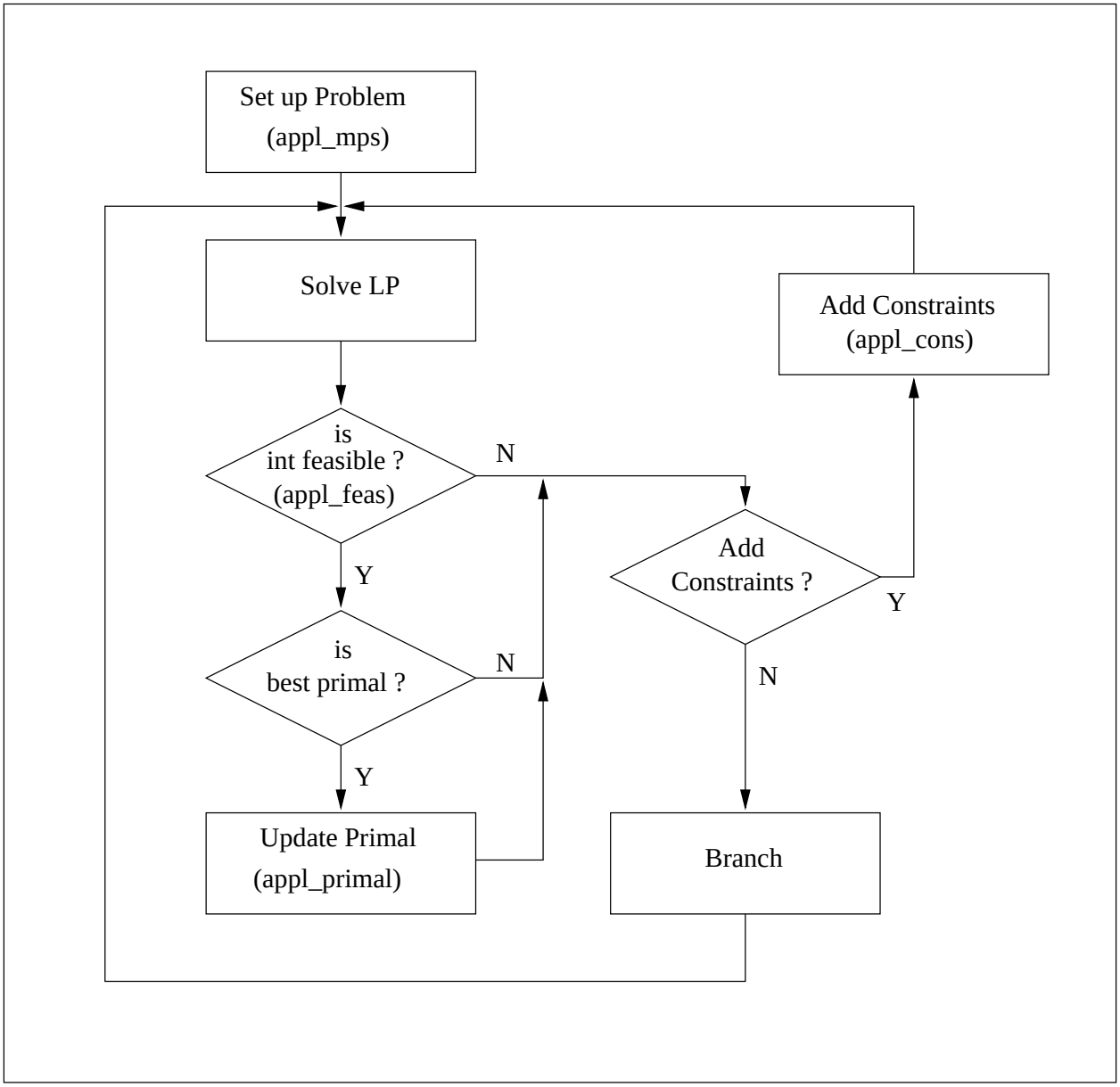

Figure 1: MINTO's implementation of branch and cut along with application functions.

- **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 Base Case

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 LP/NLP-BB and that it provides an indication of which MILP techniques are likely to be efficient in other methods for solving MINLPs, such as branch and bound. 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 246 convex problems covering a wide range of applications, including cyclic scheduling problems, trimloss problems, synthesis problems, and layout problems.

The experiments have been run on a Beowulf cluster of computers at Lehigh University. The Beowulf 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 Fedore Core 2 operating system. All of the codes we tested were compiled by using the GNU (vXXX) suite of C, C++, and FORTRAN compilers.

The test suite of convex problems have been categorized as easy, moderate, or hard, based on the time taken using MINLP-BB, a nonlinear branch-and-bound solver (Leyffer [1998]), to solve these problems. The easy convex problems take less than one minute to solve. Moderate convex problems take between one minute to one hour to solve. The hard convex problems are not solved in one hour. 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

$$\rho_i^s = 1 + \frac{z_i^s - z_i^*}{z_i^*},$$

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. The version does not use any of MINTO's advanced MILP features, such as primal heuristics, cuts, and preprocessing, and uses only maximum fractional branching and a best-bound node selection strategy. This setup is similar to the LP/NLP-BB solver implemented by Leyffer [1993]. We refer to this version of FilMINT as the *vanilla* version.

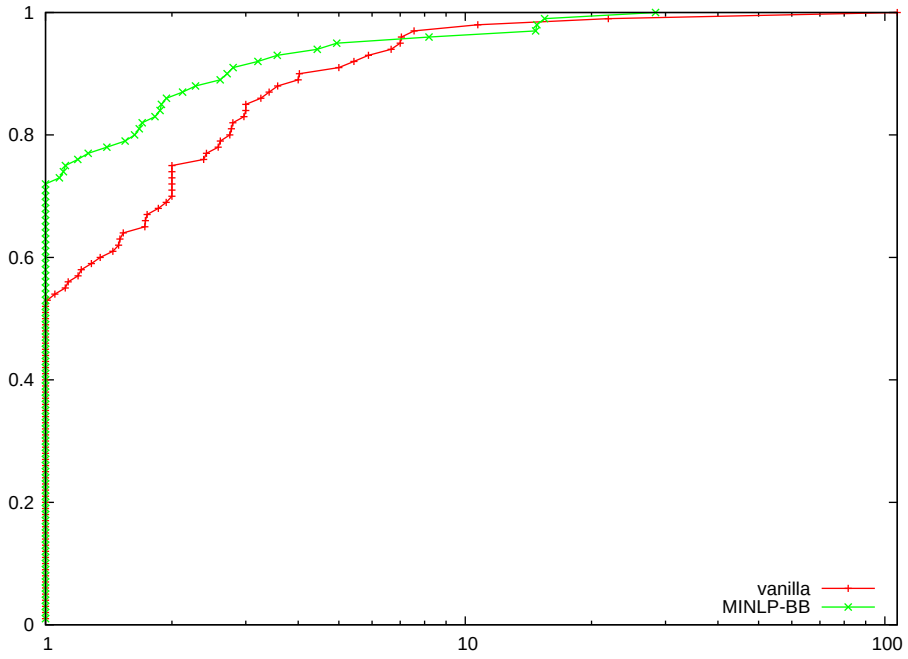

Figure 2: Performance profile comparing vanilla and MINLP-BB for easy convex instances.

The performance profiles in Figures 2–4 compare the performance of vanilla with MINLP-BB. The profiles for easy convex instances show only a small difference between FilMINT and MINLP-BB, and we drop these problem instances from the remainder of our comparison (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 the nonlinear branch-and-bound method for hard problems. This obser-

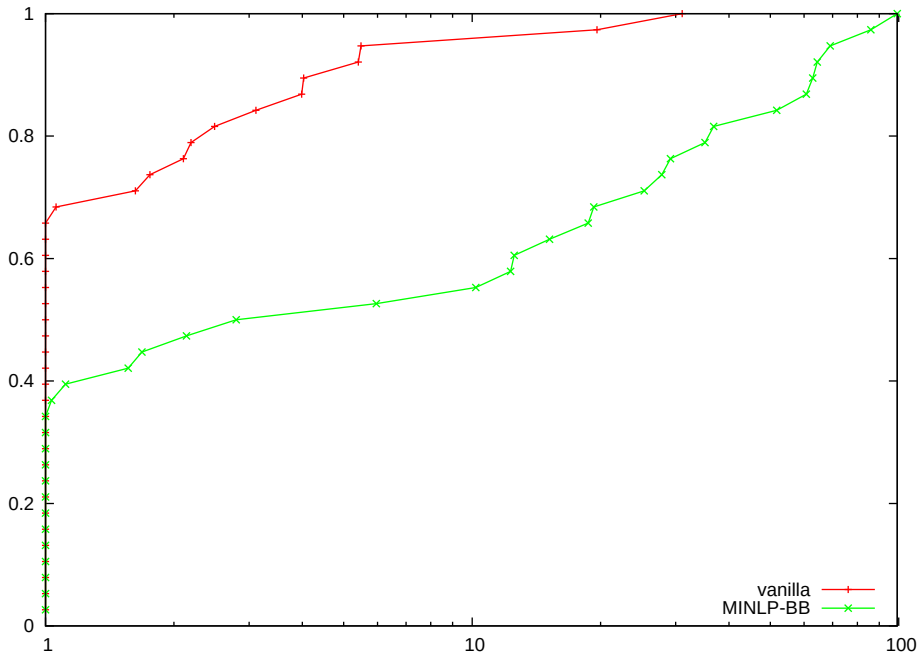

Figure 3: Performance profile comparing vanilla and MINLP-BB for moderate convex instances.

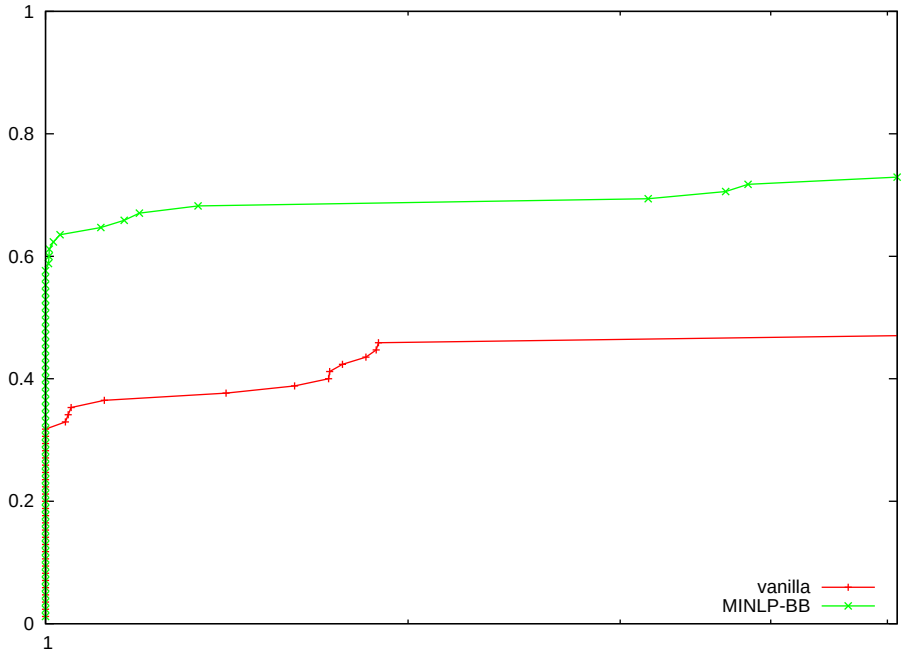

Figure 4: Performance profile comparing vanilla and MINLP-BB for hard convex instances.

vation motivates us to explore the use of advanced MIP 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 MIP 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 MIP techniques have the biggest impact. The IP features that are found to work well in this part are then included in an intermediate version (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 now included in turn. Finally, we benchmark FilMINT against to two MINLP solvers, namely, MINLP-BB (Leyffer [1998]) and BONMIN (Bonami et al. [2005]).

2 Exploiting the MILP Framework

In this section we explore the benefits of now-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 solution of the MINLP problem (MINLP) and the OA master MILP (MP($\mathcal{K}$)).

2.1 Cutting Planes and Preprocessing for the Master Problem

Cutting planes have become an important tool in solving mixed integer programs. Cuts are generated either independently of any problem structure (Gomory's mixed integer cuts, mixed integer rounding) or by using some special local structure in the problem (knapsack covers, implication cuts, clique inequalities, flow covers, generalized upper-bound (GUB) covers, etc.) FilMINT uses the cut generation routines of MINTO to strengthen the formulation and cut off the fractional solution. After a linear program is solved and a fractional solution is obtained, MINTO tries to exclude these solutions by searching the implication and clique table for violated inequalities and by searching for violated lifted knapsack covers, violated lifted GUB covers, and violated lifted simple generalized flow covers. Lifted knapsack covers are derived from pure 0-1 constraints. Lifted GUB cover inequalities have the same form but are derived from a structure consisting of a single knapsack constraint and a set of nonoverlapping generalized upper-bound constraints.

Another important technique for solving MILPs is preprocessing. Preprocessing techniques try to reduce the size of coefficients and the bounds on variables. They also help to identify infeasibility, redundancy, and fix variables. Primal preprocessing on the master

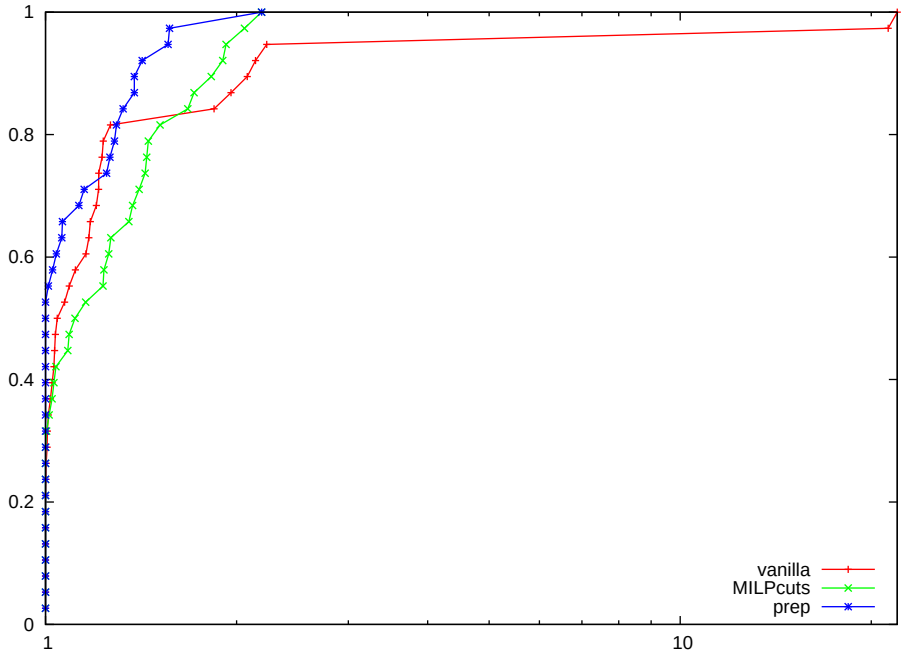

Figure 5: Performance profile showing the effect of MILP cuts and preprocessing for moderate convex instances.

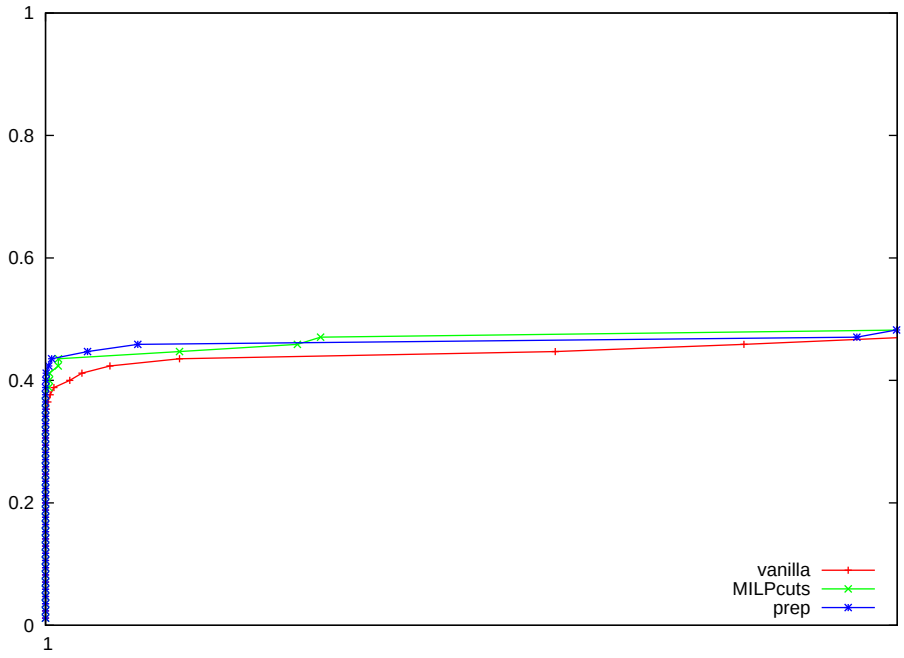

Figure 6: Performance profile showing the effect of MILP cuts and preprocessing for hard convex instances.

problem (MP($\mathcal{K}$)) can help reduce the size of the problem and make it simpler to solve. Problems that contain special structure, such as knapsack inequalities or clique inequalities, can be solved much more efficiently by using preprocessing techniques. Unfortunately, dual preprocessing of the master problem is likely to be invalid for the (MINLP) problem.

Figures 5–6 show the performance profiles for the runs with cutting planes turned on (labeled "MILPcuts"), and with preprocessing turned on (labeled "prep"). Both are compared to the vanilla implementation of LP/NLP-BB. The graphs show that both cutting planes and preprocessing provide similar improvements over the vanilla implementation and help in solving MINLPs, though the performance gains are relatively moderate. We also observe that cutting planes play a more important role as the integrality in the model increases.

2.2 Primal Heuristics

Primal heuristics aim to find good, but not necessarily optimal, solutions quickly. A good solution obtained in the beginning of the search procedure in the branch-and-cut framework reduces the number of nodes that need to be evaluated and the time to solve the problem. For hard MILP problems, even a good feasible solution might be difficult to obtain. Although there are several heuristics for specific classes of problems, they are not very useful in a general-purpose black-box MILP solver. Some of the primal heuristics that can be used for solving a general MILP include rounding, fixing- and diving-based heuristics, local branching, and relaxation induced neighbourhood search (RINS). MINTO uses a rounding-based heuristic to get feasible solutions. It also uses a fixing and diving-based primal heuristic to obtain feasible solutions quickly. MINTO also allows the user to have a certain control of the heuristic behavior through a set of parameters affecting the visit of the branching tree, the frequency of application of the internal heuristics, and so forth.

The solution obtained by the primal heuristic, say y^k, is used by FilMINT to fix the integer/binary variables and solve (NLP(y^k)). An optimal solution to (NLP(y^k)) provides a valid upper bound for the original problem as well as for the MILP formulation at the node. For hard MINLP problems, getting a feasible solution is important because it enables us to create new NLP subproblems that generate new linearizations to tighten the formulation.

The impact of turning on primal heuristics is shown by the performance profiles in Figures 7–8. The label "primal-heuristics" refers to the solver with primal heuristics turned on. We see that primal heuristics have a big impact on the solution scheme of the (MINLP) problem, especially for the hard convex instances. We explain this performance gain by the fact that primal heuristics generate more integer solutions and therefore have a greater impact on the performance metric we have chosen for the harder problems.

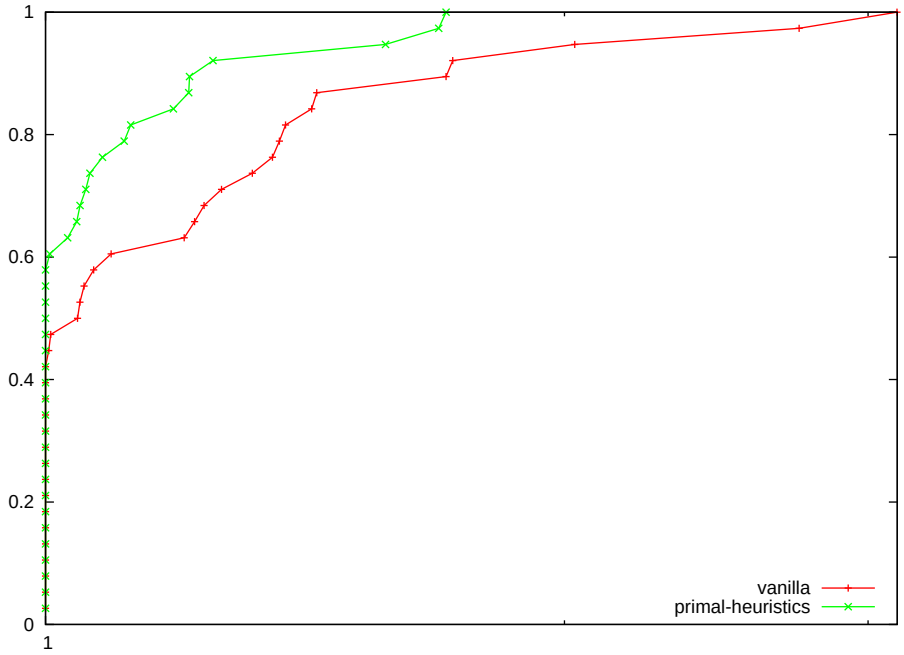

Figure 7: Performance profile showing the effect of primal heuristics for moderate convex instances.

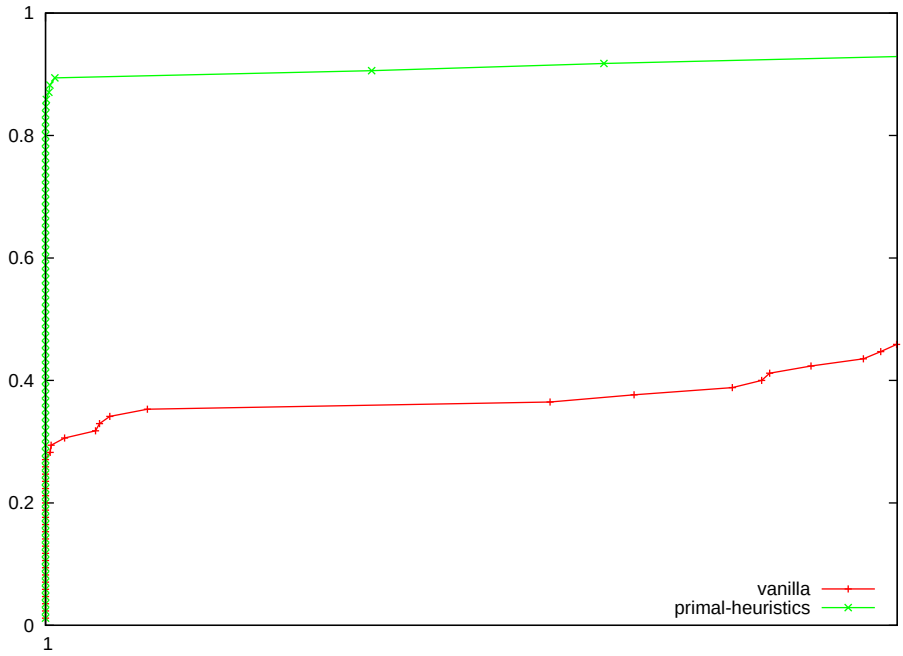

Figure 8: Performance profile showing the effect of primal heuristics for hard convex instances.

2.3 Branching and Node Selection Rules

Another advantage of building FilMINT within the MINTO framework is that it provides us with 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. The branching rules available are maximum fractionality (e0), penalty based (e1), strong branching (e2), pseudo-cost based (e3), adaptive (e4), and SOS branching (e5). The different node selection rules are best bound (E0), depth first (E1), best projection (E2), best estimate (E3), and adaptive (E4). We note that integer feasible solutions are more likely to be found deeper in the tree. Extensive computational experiments have been done to find good branching and node selection rules. Since the effectiveness of branching and node selection depends on the structure of the problem, some branching scheme is better for some problem classes, while some other is better for others. However, the branching rules that are of interest include maximum fractionality (e0), strong-branching (e2), and pseudo-cost based (e3). The node selection rules that are investigated in more details include best bound (E0), depth first (E1), best estimate (E3), and the adaptive rule (E4). Vanilla, by default, uses maximum fractional branching and the best-bound node selection strategy.

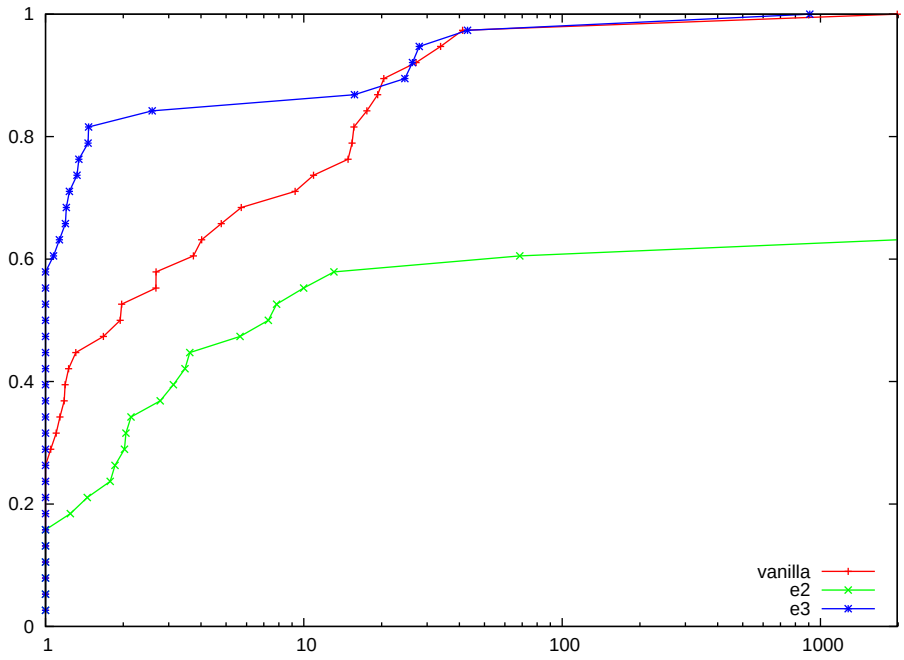

Figure 9: Performance profile comparing different branching rules for moderate convex instances.

The results of the computational experiments for different branching rules are shown in Figures 9–10. The results show that pseudo-cost branching outperforms all other rules. The

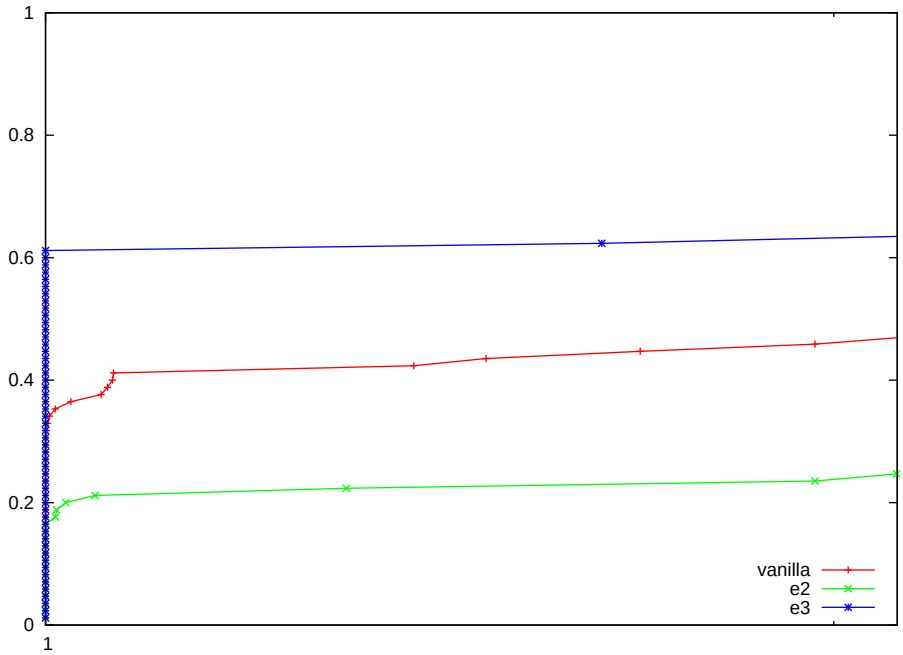

Figure 10: Performance profile comparing different branching rules for hard convex instances.

performance gains are quite stunning, but not unexpected, given the experience with maximum fractional and pseudo-cost branching in MILP (Linderoth and Savelsbergh [1997]).

The results of the experiments dealing with node selection are given in Figures 11–12. Here, the adaptive node selection rule gives the biggest improvement compared to the vanilla version, followed closely by the node selection rule based on best estimates. The performance gains in terms for the moderate problem instances is quite small, but the improvement for the hard problems is significant.

2.4 Summary of MIP Features

The computational experiments in this stage helped us to identify features in the MILP framework that can be used to solve the (MINLP) problem more effectively. Based on our experiments, we include MINTO's cutting planes, preprocessing, primal heuristics, pseudo-cost-based branching, and MINTO's adaptive node selection strategy as part of the default solver for the next stage of experiments. Figures 13–14 show the cumulative effect of turning on all these MILP-based features.

The label "vanIP" refers to the solver with these features turned on. The solver vanIP clearly outperforms a standard LP/NLP-BB solver (vanilla). The most significant improve-

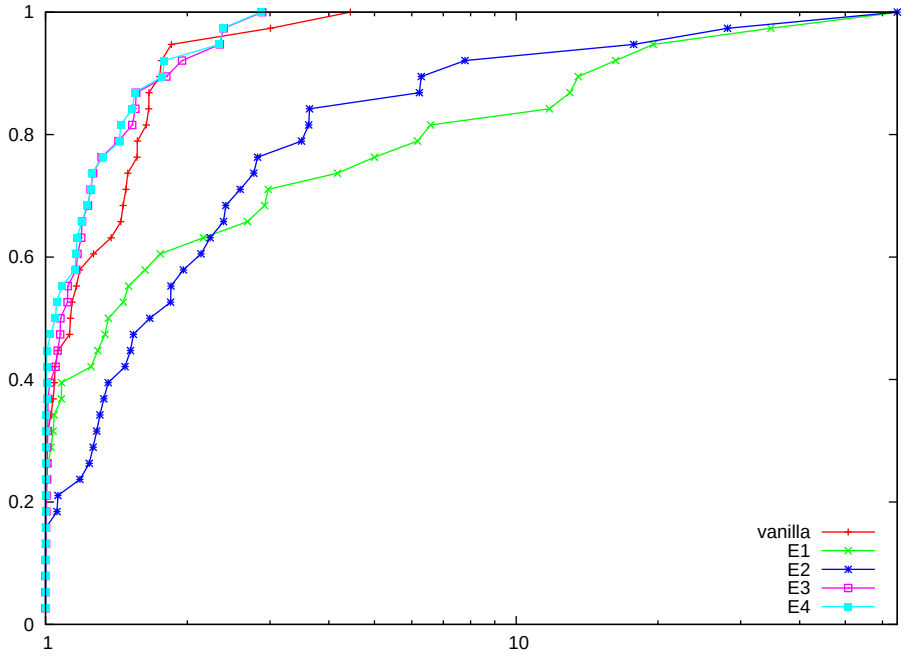

Figure 11: Performance profile comparing different node selection rules for moderate convex instances.

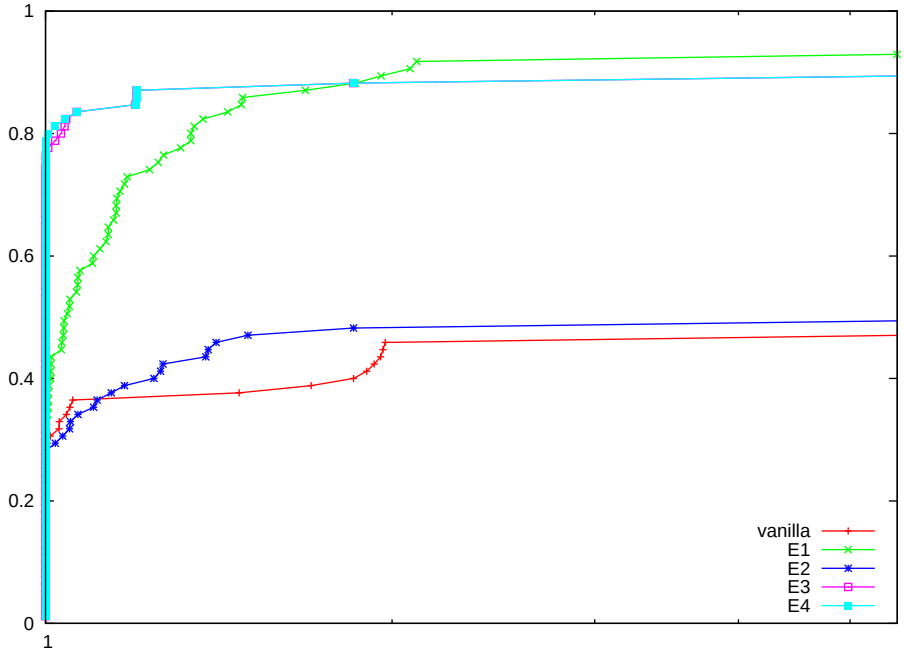

Figure 12: Performance profile comparing different node selection rules for hard convex instances.

ment was found to be from better branching rules and node selection strategies. It is noteworthy that other branching rules and node selection strategies can be used by filMINT, which may be important for certain problem instances.

In Section 5 we compare the vanilla solver and vanIP with MINLP-BB and BONMIN, as well as with the final FilMINT default solver. In the next section, we consider different methods for dealing with linearization generation and management.

3 Linearization Generation and Management

The linearizations (OA(x_k, y_k)) are obtained by using gradient information from the solution of a NLP subproblem at point (x^k, y^k). We note that linearizations approximate the nonlinear, convex, feasible region as defined in the problem (MINLP). In contrast, cutting planes in the MILP framework approximate the convex hull of integer points in the problem. We think of linearizations in our solution scheme in the same way as cutting planes. We add linearizations with the aim of tightening the formulation and improving the lower bounds. Linearizations are obtained in different ways in our framework and are explained in greater detail in this section.

Linearization management in FilMINT is parametrized in terms of parameters that control the frequency and the number of cuts added at the given stage of the tree search. Such an approach is akin to the way cutting planes are handled in any generic MILP solver, including MINTO. We handle cutting planes obtained from the MILP framework and those obtained from the nonlinear functions in the same way, adding information about the nonlinearity and integrality in the problem, while trying to draw a fine balance so as to use these cuts effectively. To see how good our cuts are and whether the cut management helps, we run a careful set of experiments to demonstrate the effectiveness of the approach. We next explain the different ways of adding and managing linearizations that we have explored.

3.1 Adding Only Violated Linearizations

When linearizations are obtained from the solution of an NLP subproblem, we do not add all of them directly to the master problem. Instead, we check whether the linearizations are violated by the optimal LP solution at that node and add only the violated linearizations. This approach keeps the size of the formulation manageable, which improves the solution time for each LP. Figures 13–14 show the impact of adding only violated linearizations. The label "violated" refers to the solver with violated cuts turned on, on top of the solver vanIP. The plots show that adding only violated cuts can result in a moderate improvement to the solver. We include this feature in the next round of experiments, dealing with other linearization

related schemes.

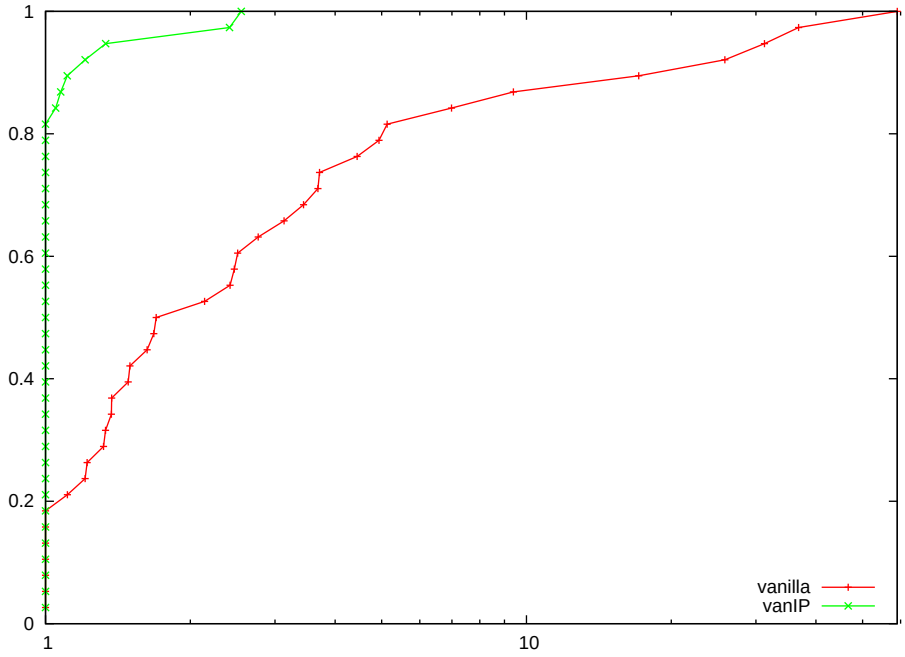

Figure 13: Performance profile comparing vanilla with solver vanIP and the effect of violated cuts for moderate convex instances.

3.2 Managing Linearizations and Other Inequalities

Having too many linearizations or other inequalities clearly can be inefficient. A large number of linearizations make the LP problem bulky and significantly increase the solution time and the storage requirements. A larger number of linearizations also increases the potential for encountering degeneracy and may further increase the number of pivots taken by the LP solver. The problem size can be reduced by temporarily removing the inactive constraints from the formulation. We therefore attempt to manage the linearizations added to the master problem by turning on MINTO's row management feature.

MINTO monitors the values of the dual variables at the end of every iteration to see whether the corresponding global constraint is active. If the dual variable for a constraint is zero, implying that the constraint is inactive, for a few iterations, then MINTO deactivates the constraint and puts it back in its cut pool. If these constraint become violated later, they are added back to the active formulation. MINTO has an environment variable, MIOCUTDELBND, which indicates the number of consecutive iterations a constraint can be inactive

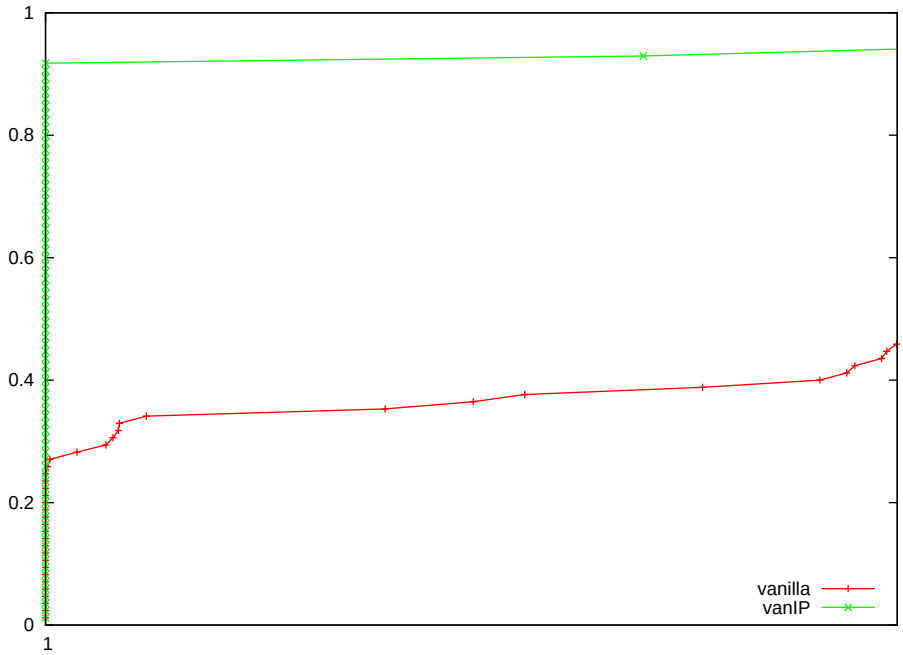

Figure 14: Performance profile comparing vanilla with solver vanIP and the effect of violated cuts for hard convex instances.

in the active formulation before it is removed. After conducting a few small-scale experiments, we set the value of MIOCUTDELBND to 15

Figures 15–16 show the impact of turning on MINTO's row management with MIOCUTDELBND set to 15. The label "row-mgmt" refers to the solver with MINTO's row management turned on, on top of the solver, violated. Turning on row management for moderate problems results in a huge improvement in the solution times. For hard problems in the test suite, we see that for 74% of the problems row management gives a better solution, compared to only 60% for the solver violated. The improvement of row management for the hard problems is not as dramatic. We expect that this is largely due to the different metric that we use to measure success (namely, closeness to the best integer solution). We include row management in the next round of experiments.

3.3 Generating Linearizations at Fractional LP Solution

A disadvantage of the LP/NLP-BB approach is that linearizations are generated only from the solution of the NLP subproblem fixed at an integer solution, y^k. As long as no integer LP solution is found by the branch-and-cut procedure, no nonlinear information in terms of linearizations is added to the master problem.

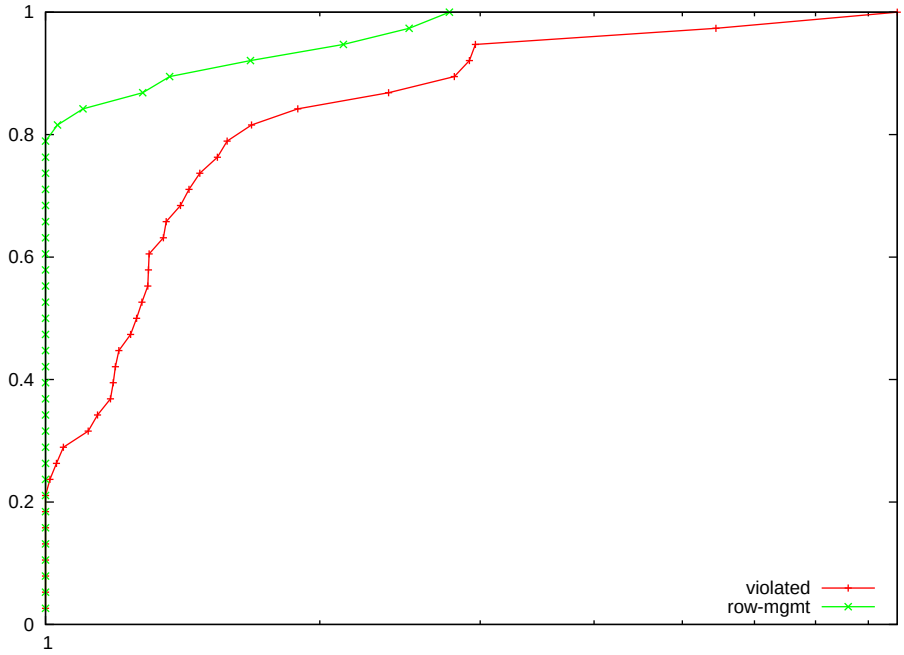

Figure 15: Performance profile showing the effect of row management for moderate convex instances.

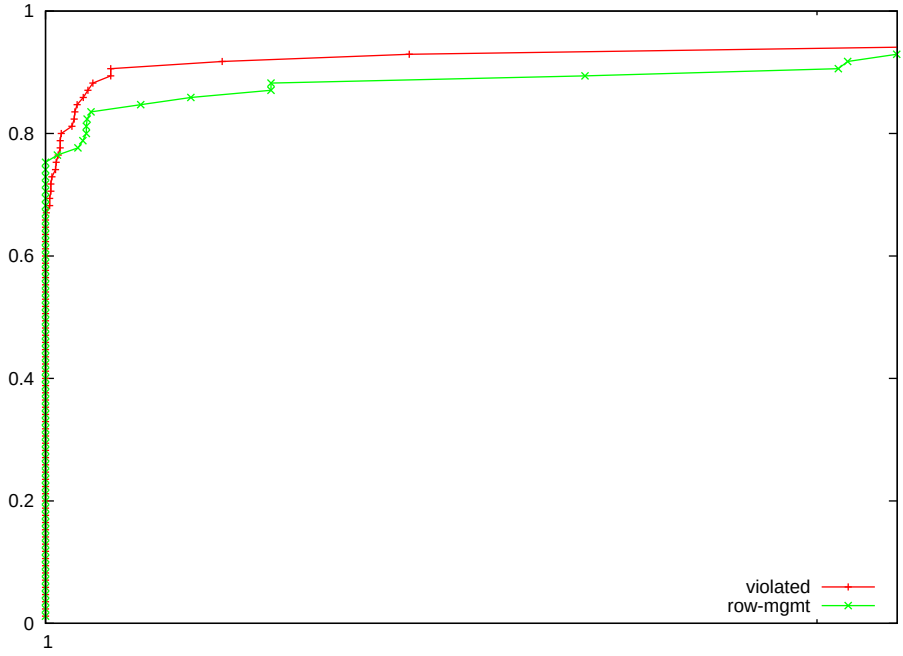

Figure 16: Performance profile showing the effect of row management for hard convex instances.

We believe that it is important to add nonlinear information as early as possible to the MILP master problem to help the solver make better branching decisions and to find better MINLP feasible solutions earlier. There are two ways to achieve this goal. The first is to apply primal heuristics as discussed in Section 2.2, and the second is to generate linearization from fractional integer variables. Thus, we solve NLP subproblems with variables fixed at the fractional LP solution y^k and generate linearizations at the solution of this NLP. The linearizations obtained in this case are the same as (OA(x_k, y_k)) except that the values of the integer variables y^k are no longer integral. We note that these linearizations are valid because the problem is convex. However, we cannot use the solution of the NLP subproblem to update the upper bound for the problem.

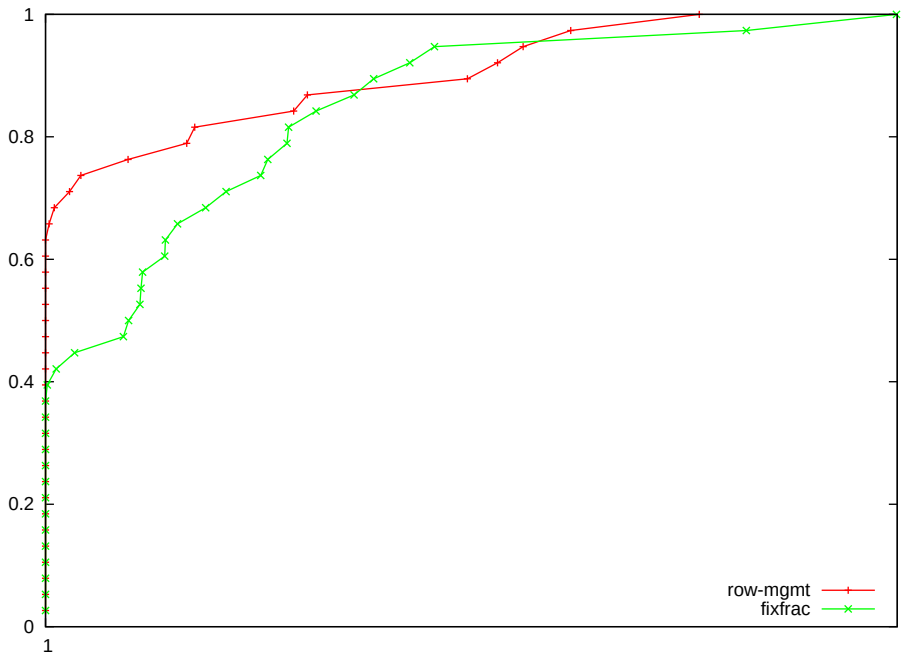

Figure 17: Effect of adding linearization by using fractional LP solution for moderate convex instances.

Figures 17–18 show the impact of adding linearizations obtained by using a fractional LP solution. The label "fixfrac" refers to this solver built on top of the solver row-mgmt. The profiles show that this procedure helps the solution scheme for the problems in our test suite. We next explain the cut generation parameters that we use to control the cut generation procedure.

We note that some of the linearizations generated are more effective if added at an early stage of the tree enumeration, whereas some are better if added at a later stage. By managing the cut generation procedure intelligently, one can achieve significant benefits from

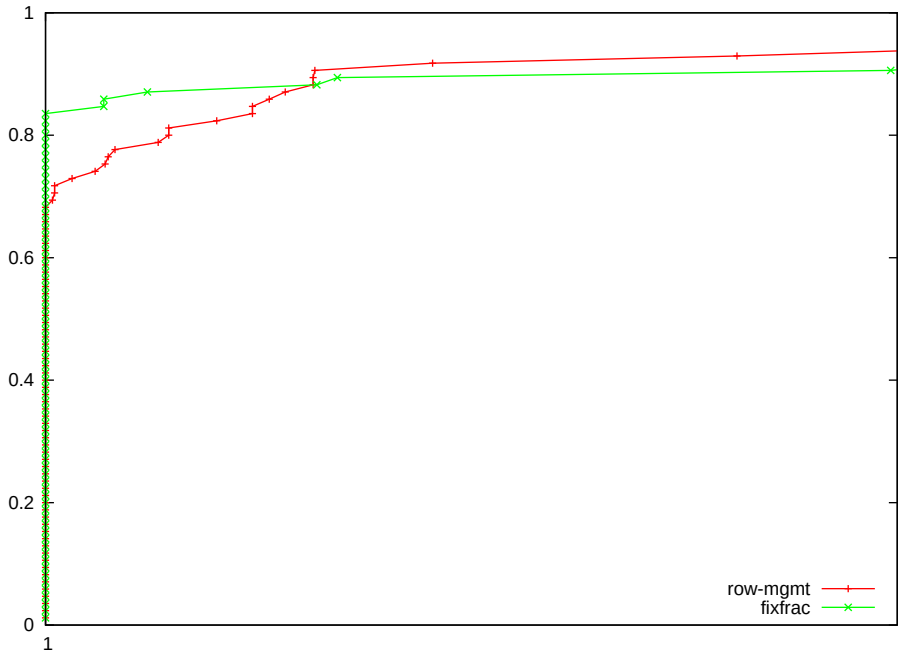

Figure 18: Effect of adding linearization by using fractional LP solution for hard convex instances.

these linearizations. We deal with the linearization generation procedure in the same way as cutting planes in a MILP framework. Cut generation is parametrized in terms of two important parameters. The first, α, is the tail factor that decides how many rounds of linearization generation should be done at the current node. The second, κ, is the skip factor that decides how many nodes to skip before invoking the cut generation procedure. The tail factor decides whether the LP solution improves enough so as to justify another round of cut generation at that node. To calculate the tail factor, we compare the current LP solution with the previous few LP solutions (typically three) and see whether we are improving the lower bounds enough. The percentage improvement in the lower bounds also depends on the depth of the tree. Based on our experimental results, we set α to 10 for the first 1,000 nodes that are enumerated, and 50 for the later nodes.

The skip factor also depends on the depth of the tree enumeration. We feel that cuts added in the early stage of the enumeration are more effective in reducing the gap and helping reduce the search tree. Therefore, we keep the skip factor much smaller at an early stage and increase the skip factor as we go deeper down the tree. Based on experiments, we set κ to 10 for the first 100 nodes, and 100 for the later nodes. We have tried to fine-tune these parameters so that the cut generation procedure works well on a large set of problems on average. We also have an upper limit on the number of linearizations that are generated at a

particular round of cut generation.

3.4 Obtaining Extended Cutting-Plane-Based Linearizations

Solving NLP subproblems typically takes more time than solving LP relaxations. Thus, we consider generating additional linearizations at fractional solutions of the LP relaxation, because the necessary gradient and function evaluations are relatively cheap and the linearizations can still tighten the master problem. This procedure is similar to the extended cutting plane method by Westerlund and Pettersson [1995]. Figures 19–20 show the impact of adding extended-cutting-plane-based linearizations. The label "ecp" refers to the solver with ECP-based linearizations on top of the solver fixfrac. Our results show that ECP-based linearizations do not fare well on average. We are investigating this situatiuon further by trying different cut generation schemes.

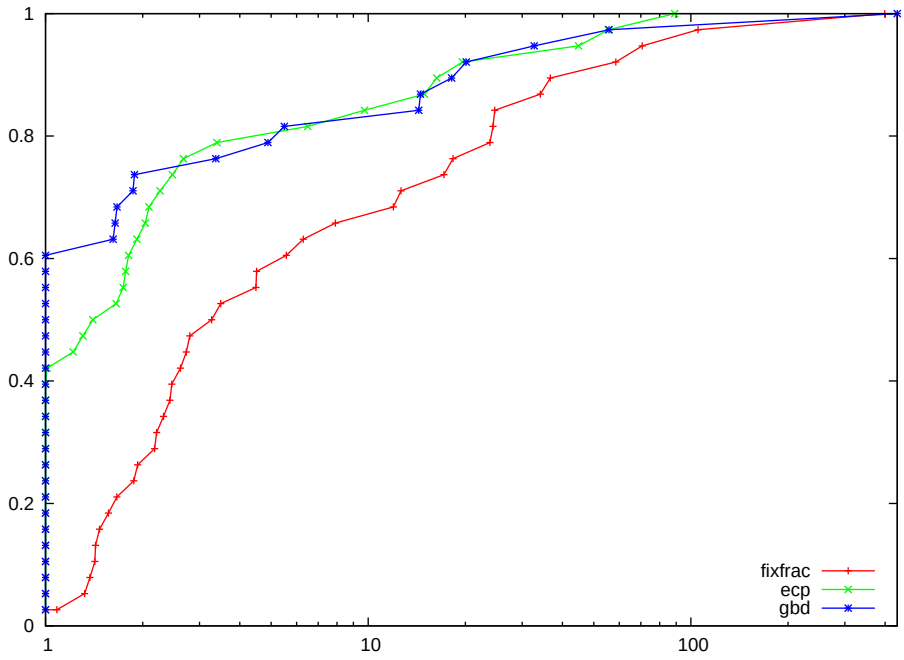

Figure 19: Performance profile of ECP- and GBD-based cuts for moderate convex instances.

3.5 Obtaining Benders-Cut-Based Linearization

Another approach to managing the size of the LP relaxation is suggested by the fact that a single Benders cut is a relaxation of the corresponding outer approximation cuts. Summing the objective linearizations and the constraint linearizations weighted with the optimal NLP

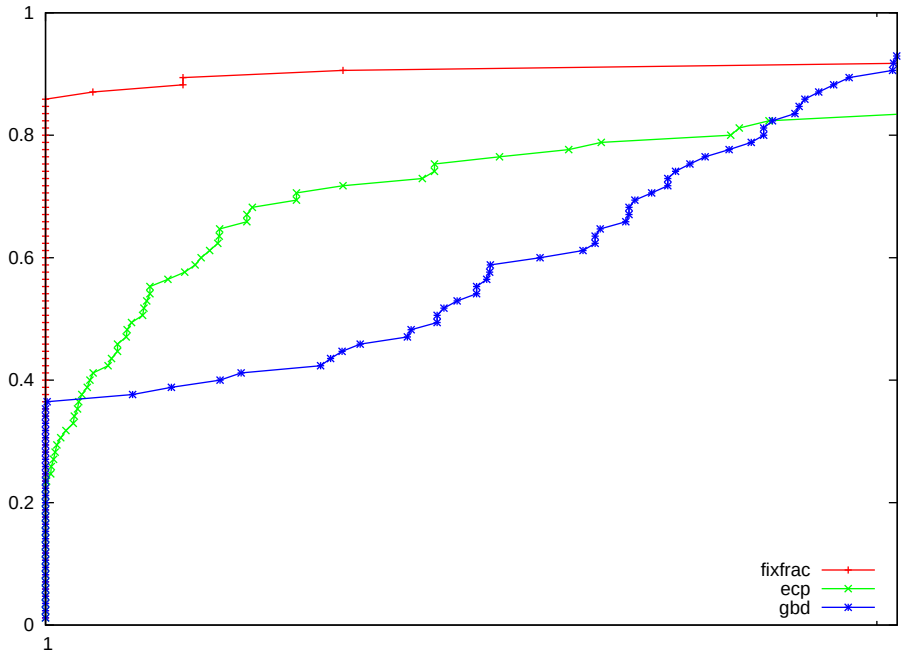

Figure 20: Performance profile of ECP- and GBD-based cuts for hard convex instances.

multipliers μ^k yields the Benders cut:

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

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)). Clearly, the Benders cut is weaker than the outer approximations; on the other hand, it compresses $m + 1$ linear inequalities into a single cut.

Figures 19–20 show the impact of adding Benders-cut-based linearizations. The label "gbd" refers to the solver with GBD-based linearizations on top of the solver fixfrac. We see that Benders cuts do well for problems in the moderate set. However, they do not fare well for hard problems. In the future, we will investigate other uses of Benders cuts to make them work better for hard problems.

4 Exploiting the Solution of NLP Relaxation

The branch-and-bound algorithm for problem (MINLP) solves a sequence of NLP relaxations (NLPR(l, u)) at the nodes. The solution of the NLP relaxation at a node provides

valuable information that can be used in the LP/NLP-BB as well. As mentioned, we solve an NLP relaxation at the root node to obtain initial linearizations to tighten the master formulation (MP($\mathcal{K}$)).

Solving a relaxed NLP subproblem at any node helps us to get tighter bounds for the MINLP problem at that node. Since solving the relaxation of the NLP may take considerable execution time, we solve it only occasionally, say after every k nodes in the branch-and-cut tree.

The solution of the relaxed NLP subproblem, being tighter, can be used to prune the node if it is greater than the best-known upper bound. We note that solving NLP relaxations at every node makes the algorithm behave like a nonlinear branch-and-bound algorithm. This provides us with an algorithm that integrates the approach of a classical branch and bound algorithm with that of a classical LP/NLP branch-and-bound algorithm. We solve the NLP relaxations every 10 nodes in our experiments. Figures 21–22 show the impact of this approach. The label "`nlpr`" refers to the solver with the addition of NLP relaxations on top of the solver fixfrac. Even though the improvement is only marginal, we include this method as part of our default settings for FilMINT.

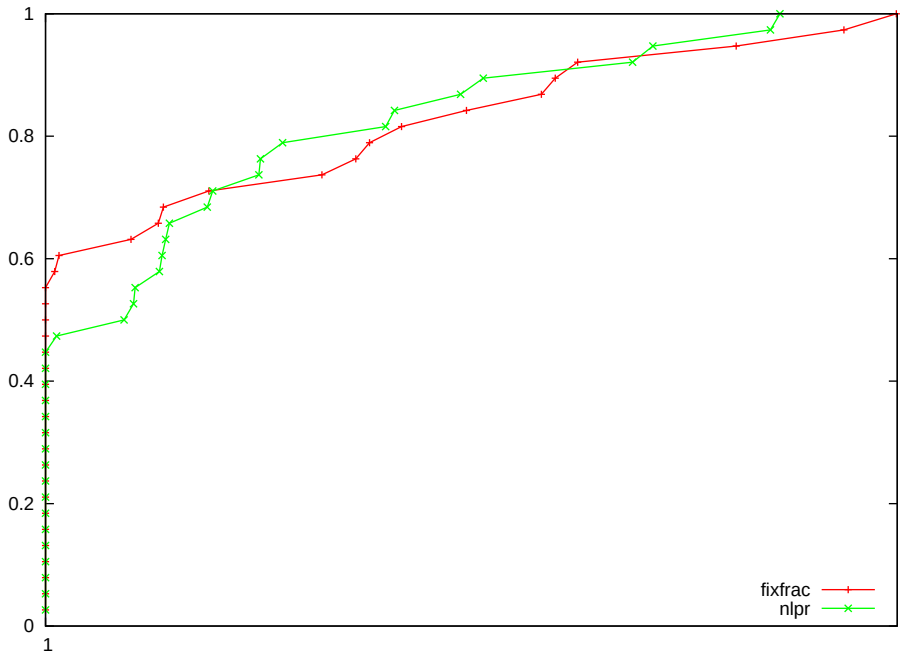

Figure 21: Effect of solving NLP relaxations for moderate convex instances.

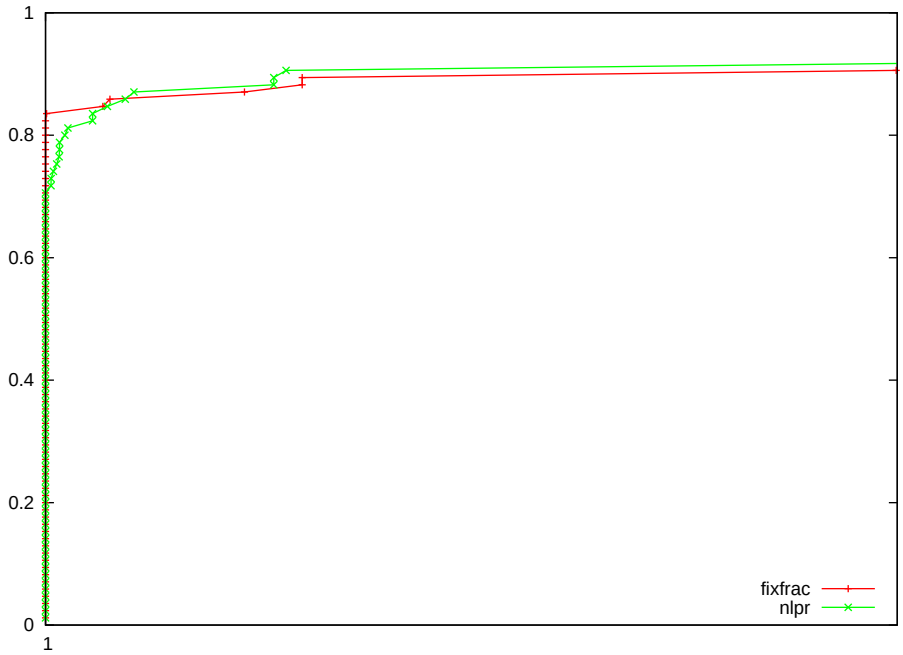

Figure 22: Effect of solving NLP relaxations for hard convex instances.

5 Analysis of Computational Experiments

Based on the first stage of experiments, we include as part of the default solver MINTO's cutting planes, preprocessing, primal heuristics, pseudo-cost-based branching, and adaptive node selection strategy. The computational experiments in the second stage helped us to identify linearization-based methods and generation schemes that solve the problems in our test suite more efficiently. Based on the experiments, we include as part of the default solver violated cuts, row management, and methods dealing with solving NLP at fractional LP solution and solving (NLPR(l, u)) problems at certain nodes. We also identify some parameters dealing with linearization generation.

Figures 23–24 compare the progress made by the solver to that of the default version, vanilla. The label "filmint" now refers to the final default settings for our solver. We observe that the largest improvement comes from adding the IP features to the vanilla LP/NLP-BB method. We also observe a significant improvement using MINTO's row management.

We have compared FilMINT to a nonlinear branch-and-bound based solver, MINLP-BB (Leyffer [1998]), and the hybrid version of bonmin (Bonami et al. [2005]). Because FilMINT is developed within a branch-and-cut framework, it provides flexibility for researching different classes of cuts that can benefit the solution scheme. MINLP-BB also uses the same NLP solver, filterSQP, for solving NLP problems. Therefore, the comparison

shows how the two methodologies work. We choose BONMIN for the comparison because of its similarities to our framework. This comparison will also show the effect of using a different IP and NLP solver (BONMIN uses an interior-point method). MINLP-BB and BONMIN are run for the same set of instances for the same time (four hours). The hybrid algorithm of BONMIN was run with the default hybrid settings. The performance profile (see Figures 23–24) shows that FilMINT is an order of magnitude faster on average than MINLP-BB, and 2-4 times faster than BONMIN for the moderate convex problems. For the hard convex problems, FilMINT outperforms BONMIN, and both solvers are orders of magnitude better than MINLP-BB.

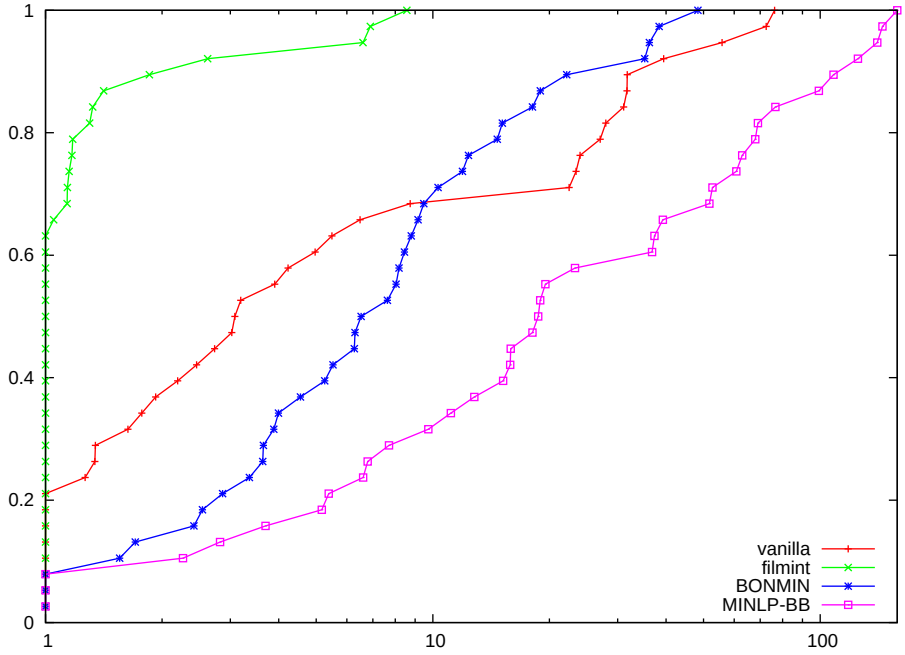

Figure 23: Performance profile comparing FilMINT with MINLP-BB and BONMIN for moderate convex instances.

6 Conclusions

To solve MINLP problems, we introduce a new solver, FilMINT, based on an LP/NLP methodology in a branch-and-cut framework. We investigate new ways of adding and managing linearizations and show their effectiveness in solving MINLP problems. By carefully choosing MILP and linearization-based features, and by using existing software components, we show how a framework such as the one proposed may be used to derive a flexible and powerful methodology to solve hard convex MINLP problems. The framework provides a

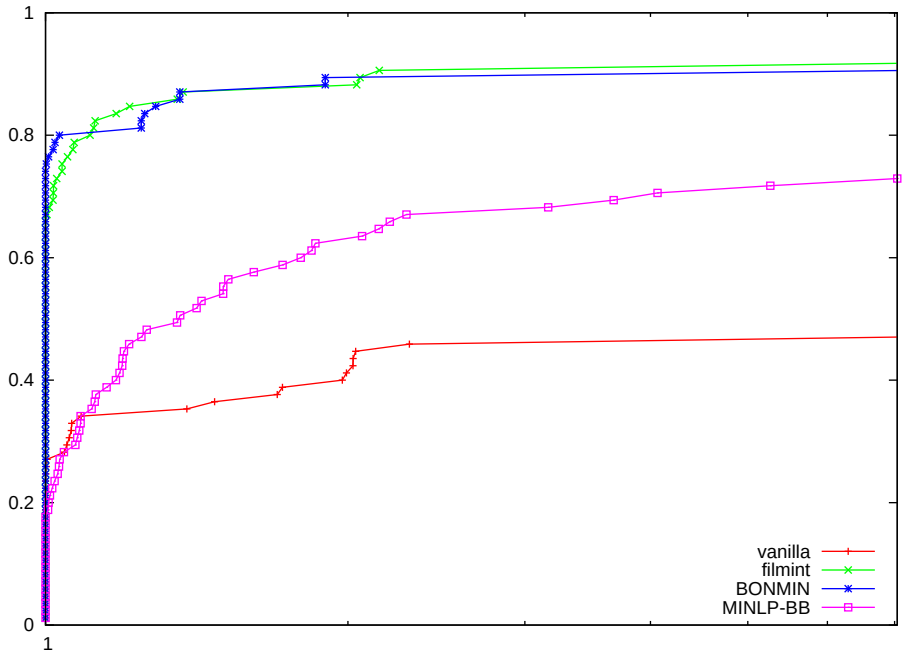

Figure 24: Performance profile comparing FilMINT with MINLP-BB and BONMIN for hard convex instances.

means to further investigate cutting planes that can be useful in solving the problems much faster. We compare our solver to two existing MINLP solvers and improve on both on average by a factor of 2-4.

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 W-31-109-ENG-38.

References

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. Report RC23771 (W0511-023), IBM Research, 2005.

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. Grossman. 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.

R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–270, 2002.

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

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

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, South San Francisco, 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 Journal of 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.

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*. Ph.D. 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 M. Savelsbergh. A computational study of search strategies in mixed integer programming. Technical Report TLI-97-12, Georgia Institute of Technology, 1997. Available from `http://akula.isye.gatech.edu/~jeff/papers/branch.ps`.

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.

N. W. Sawaya, C. D. Laird, and P. Bonami. A novel library of non-linear mixed-integer and generalized disjunctive programming problems. In preparation. Instances available at `http://egon.cheme.cmu.edu/ibm/page.htm`, 2006.

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

M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, Boston, 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*, 106:25–57, 2006.

T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering Supplement*, 19:S131–S136, 1995. ESCAPE–95.