ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Solving Mixed-Integer Nonlinear Programs by QP-Diving

Ashutosh Mahajan, Sven Leyffer, and Christian Kirches

Mathematics and Computer Science Division

Preprint ANL/MCS-P2071-0312

March 26, 2012

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Contents

Solving Mixed-Integer Nonlinear Programs by QP-Diving*

Ashutosh Mahajan,† Sven Leyffer,‡ and Christian Kirches§

March 26, 2012

Abstract

We present a new tree-search algorithm for solving mixed-integer nonlinear programs (MINLPs). Rather than relying on computationally expensive nonlinear solves at every node of the branch-and-bound tree, our algorithm solves a quadratic approximation at every node. We show that the resulting algorithm retains global convergence properties for convex MINLPs, and we present numerical results on a range of test problems. Our numerical experience shows that the new algorithm allows us to exploit warm-starting techniques from quadratic programming, resulting in a reduction in solve times for convex MINLPs by orders of magnitude on some classes of problems.

Keywords: Mixed-Integer Nonlinear Programming, Sequential Quadratic Programming.

AMS-MSC2010: 90C11, 90C30, 90C55.

1 Introduction

We solve mixed-integer nonlinear programming (MINLP) problems of the form

$$\begin{cases} \underset{x}{\text{minimize}} & f(x), \\ \text{subject to} & c(x) \le 0, \\ & x \in X, \\ & x_i \in \mathbb{Z},\ \forall i \in I, \end{cases} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable convex functions, $X \subset \mathbb{R}^n$ is a bounded polyhedral set, and $I \subseteq \{1, \ldots, n\}$ is the index set of integer variables. We also denote by x_I the subvector of integer variables, and we refer to problem (1.1) as a convex MINLP.

*Preprint ANL/MCS-P2071-0312

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

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

§Interdisciplinary Center for Scientific Computing (IWR), Ruprecht-Karls-Universität Heidelberg, 69120 Heidelberg, Germany, christian.kirches@iwr.uni-heidelberg.de.

MINLPs arise in a wide variety of applications, such as the efficient management of electricity transmission (Bacher, 1997; Momoh et al., 1997), including transmission expansion (Romero et al., 2002; Garver, 1997), transmission switching (Bartholomew et al., 2008; Hedman et al., 2008), and contingency analysis and blackout prevention of electric power systems (Bienstock and Mattia, 2007; Donde et al., 2005). MINLPs also arise in the design of water distribution networks (Bragalli et al., 2006; Karuppiah and Grossmann, 2006), operational reloading of nuclear reactors (Quist et al., 1998), minimization of the environmental impact of utility plants (Eliceche et al., 2007), wireless bandwidth allocation (Bhatia et al., 2006; Sheikh and Ghafoor, 2010; Costa-Montenegro et al., 2007), selective filtering (Sinha et al., 2002; Soleimanipour et al., 2002), network topology design (Bertsekas and Gallager, 1987; Boorstyn and Frank, 1977; Chi et al., 2008), optical network performance optimization (Elwalid et al., 2006), portfolio optimization (Bienstock, 1996; Jobst et al., 2001), block layout design in the manufacturing and service sectors (Castillo et al., 2005), and integrated design and control of chemical processes (Flores-Tlacuahuac and Biegler, 2007). Recent novel applications include the optimal response to catastrophic oil spills such as the Deepwater oil spill in the Gulf of Mexico (You and Leyffer, 2010, 2011), minimum-cost designs for reinforced concrete structures that satisfy building code requirements (Guerra et al., 2009), and optimal response to a cyber attack (Goldberg et al., 2012; Altunay et al., 2011). More applications and background material can be found in monographs and survey papers (Floudas, 1995; Grossmann and Kravanja, 1997; Grossmann, 2002; Lee and Leyffer, 2011).

Methods for MINLP include nonlinear branch-and-bound (Dakin, 1965; Gupta and Ravindran, 1985), outer approximation (Duran and Grossmann, 1986; Fletcher and Leyffer, 1994), extended cutting-plane method (Westerlund and Pettersson, 1995), Benders decomposition (Geoffrion, 1972), and branch-and-cut methods (Stubbs and Mehrohtra, 2002; Quesada and Grossmann, 1992). These algorithmic developments have resulted in a number of robust implementations of branch-and-bound MINLP-BB (Leyffer, 1998) and SBB (Bussieck and Drud, 2001); the hybrid solver BONMIN (Bonami et al., 2008); the branch-and-cut solver FilMINT (Abhishek et al., 2010); and the new toolkit MINOTAUR (Mahajan et al., 2011).

In this paper, we present a new variant of nonlinear branch-and-bound. Our main motivation is the lack of hot-start capabilities in nonlinear solvers. We loosely define a hot-start as a solve that does not require any factorizations but takes advantage of factorizations at the parent node and rank-one updates. In fact, it is almost impossible to effectively hot-start any NLP solver. This is in stark contrast to the situation in linear programming (LP), where factors of the LP basis are readily available and can be used to re-solve LPs. The main reason why there are no hot-starts for NLP solvers is that the factors are outdated as soon as a step is taken because the Hessian and Jacobian matrices are nonlinear and not constant. This lack of hot-starts affects both active-set and interior-point methods and has a negative effect on the efficiency of nonlinear branch-and-bound solvers compared with LP-based solvers (Quesada and Grossmann, 1992). We illustrate this point in Table 1. The table shows the CPU times for a full sweep of strong-branching (Dribeek, 1966) re-solves for a small number of representative MINLP problems. In the table, No. Ints gives the number of integer variables, and NLP, QP-Cold, and QP-Hot give the CPU time for the cumulative strong-branching solves, where NLP refers to re-solving NLPs, QP-Warm refers to taking one QP-step of a sequential quadratic programming (QP) method, and QP-Hot refers to QP solves that reuse the previous basis factors. These experiments were carried out with a version of MINLP-BB (Fletcher and Leyffer, 2005) using filterSQP (Leyffer, 1998; Fletcher and Leyffer, 1998) as the nonlinear solver and BQPD (Fletcher, 1995) as the QP solver. Re-using the factors clearly provides a significant advantage, an observation consistent with recent results on inexact strong branching (Bonami et al., 2011).

Table 1: Comparison of NLP and QP strong branching times.

Problem	No. Ints	NLP	QP-Cold	QP-Hot
stockcycle	480	4.08	3.32	0.532
RSyn0805H	296	78.7	69.8	1.94
SLay10H	180	18.0	17.8	1.25
Syn30M03H	180	40.9	14.7	2.12

The results in Table 1 motivate us to consider a nonlinear branch-and-bound algorithm that exploits the QP solver's hot-start capabilities as much as possible. Our approach is related to early branching ideas (Borchers and Mitchell, 1994; Leyffer, 2001) but provides a more radical break with traditional branch-and-bound techniques. In particular, our new approach exploits hot-starting techniques of QP solvers such as BQPD (Fletcher, 1995), whereas the early branching approach just uses cold starts to solve updated QPs, corresponding to QP-Cold in Table 1.

The remainder of this paper is organized as follows. In Section 2, we describe the proposed new branch-and-bound algorithm, and in Section 3 discuss extensions. In Section 4 we present an implementation of our idea in MINOTAUR (Mahajan et al., 2011), a toolkit for solving MINLPs. We compare our approach to a standard implementation of branch-and-bound on a range of test problems from the literature.

2 QP-Diving for Mixed-Integer Nonlinear Programs

We propose a new algorithm for MINLPs that is based on the traditional branch-and-bound approach but no longer requires the solution of NLP subproblems at every node. Instead, we observe that it is sufficient to solve QP approximations at most nodes. We expect our new algorithm to benefit from the savings shown in Table 1 by searching the tree using QP solves that can be hot-started. We show below how to adjust the branch-and-bound rules to ensure convergence for convex MINLPs. In particular, we indicate how we need to handle infeasible QPs and those with integer solutions, and we show how bounds can be exploited during the tree search.

Our algorithm starts by solving the NLP relaxation of (1.1). If the relaxation is infeasible, so is the MINLP. If the solution is integer, then we have also solved the MINLP. Otherwise, we let the solution of the root node be $\hat{x}$, and we construct a QP approximation around $\hat{x}$, denoted by (QP($\hat{x}, l, u$)). A short graphical description of our algorithm compared with standard branch-and-bound is given in Figure 1.

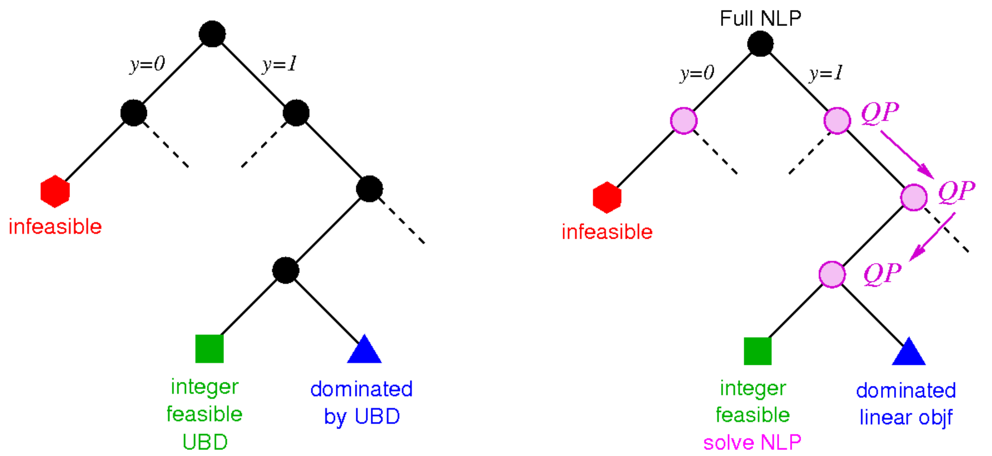

Figure 1: Illustration of QP-diving for MINLPs. The left image shows a traditional nonlinear branch-and-bound solver that traverses the tree by solving NLPs. The right image shows a mixed tree with nonlinear solves (black) and QP solves (purple).

A node in the branch-and-bound tree is uniquely defined by a set of bounds, (l, u), on the integer variables. At every node (l, u) of the branch-and-bound tree, we define the following QP approximation,

$$\left\{ \begin{array}{ll} \underset{x,\eta}{\text{minimize}} & \eta + \frac{1}{2}(x-\hat{x})^T \hat{H}(x-\hat{x}) \\ \text{subject to} & \hat{c} + \nabla \hat{c}^T (x-\hat{x}) \le 0 \\ & \hat{f} + \nabla \hat{f}^T (x-\hat{x}) \le \eta \\ & \eta \le U - \epsilon \\ & x \in X, \quad l \le x_I \le u, \end{array} \right. \tag{QP($\hat{x}, l, u$)}$$

where $\hat{H}$ approximates the Hessian of the Lagrangian at $\hat{x}$. At the root node we have $l = -\infty$ and $u = \infty$. The motivation for adding the variable η and equivalently rewriting the linear objective as a constraint will be explained below. We also define the NLP relaxation at a node (l, u) as

$$\left\{ \begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & c(x) \le 0 \\ & x \in X, \quad l \le x_I \le u. \end{array} \right. \tag{NLP(l, u)}$$

We can now describe the rules of the new tree-search algorithm.

Infeasible Nodes. The convexity of the constraints $c(x) \le 0$ implies that their linearizations are outer approximations of the feasible set. Thus, if these linearizations are infeasible in (QP($\hat{x}, l, u$)), it follows that (NLP(l, u)) is also infeasible, and the node can be pruned.

Upper Bounds on QP Nodes. We note that, in general, a QP approximation can both underestimate and overestimate a nonlinear objective, for example, the quadratic approximation of $f : \mathbb{R} \to \mathbb{R} = -\sqrt{x}$, at $x = 1$ underestimates $f(0)$ but overestimates $f(2)$. Hence, we cannot prune nodes based on the objective function value of QP. However, we can borrow a trick from outer approximation (Duran and Grossmann, 1986) and replace the linear part of the QP objective by a new variable, η. We linearize the objective around $\hat{x}$ and then add the following two inequalities to the QP:

$$\eta \leq U - \epsilon \ \text{ and } \ \eta \geq \hat{f} + \nabla \hat{f}^T (x - \hat{x}), \tag{2.2}$$

where U is the upper bound that is updated during the tree search (see next paragraph) and $\epsilon > 0$ is the optimality tolerance. Adding (2.2) to the QP ensures that we can possibly detect whether a node is dominated by an upper bound. Whenever a QP is infeasible and (2.2) is either active or violated at the solution of the corresponding phase-1 problem, the node can be pruned because it would exceed the upper bound. In practice, we do not distinguish infeasible problems due to upper bounds or constraint inconsistency.

Integer Feasible Nodes. If the solution of (QP($\hat{x}, l, u$)) is integral, then we must solve (NLP(l, u)), because the solution of the QP is, in general, not a solution of the NLP. If the solution of (NLP(l, u)) is integral, then we either obtain a new upper bound or prune the node because its solution is dominated by the incumbent.

Branching. If the solution x' of (QP($\hat{x}, l, u$)) or (NLP(l, u)) is feasible but not integral, then we branch on any nonintegral variable, say x'_i. The branching can be done in ways similar to those used in mixed-integer linear programming (Achterberg et al., 2004). Next, we define the branching subroutine that adds two new child problems to the stack of unsolved ones.

Subroutine: $\mathcal{S} \leftarrow$ **BranchOnVariable** $(x'_i, l, u, \mathcal{S})$ *// Branch on a non-integral x'_i for $i \in I$*
Set $u_i^- = \lfloor x'_i \rfloor, l^- = l$ and $l_i^+ = \lceil x'_i \rceil, u^+ = u$.
Add QP($\hat{x}, l^-, u^-$) and QP($\hat{x}, l^+, u^+$) to $\mathcal{S} = \mathcal{S} \cup \{\text{QP}(\hat{x}, l^-, u^-), \text{QP}(\hat{x}, l^+, u^+)\}$.

Only when we find an integer point do we need to solve an NLP problem. The remainder of the tree is searched by using QPs. In particular, it follows that the Hessian and Jacobian matrices remain unchanged during the tree search, thus allowing us to take advantage of the factorization of the augmented system matrix during the QP solves. This observation motivates the term QP-diving, because most savings are realized during dives down the tree when we add bounds to the QP. On the other hand, when we backtrack, it is typically more efficient to refactor the augmented system because the dense reduced Hessian matrix changes significantly. The complete QP-diving branch-and-bound algorithm is described in Algorithm 1. Proposition 2.1 formally establishes convergence of this algorithm for convex MINLPs.

QP-Diving for MINLP

Choose a tolerance $\epsilon > 0$, and set $U = \infty$.
Initialize the stack of open problems $\mathcal{S} = \emptyset$.
Solve the NLP relaxation of (1.1), and let the solution be $\hat{x}$.
Add (QP($\hat{x}, -\infty, \infty$)) to the stack: $\mathcal{S} = \mathcal{S} \cup \{\text{QP}(\hat{x}, -\infty, \infty)\}$.
while $\mathcal{S} \neq \emptyset$ **do**
  Remove a problem (QP($\hat{x}, l, u$)) from the stack: $\mathcal{S} = \mathcal{S} - \{\text{QP}(\hat{x}, l, u)\}$.
  Solve (QP($\hat{x}, l, u$)) and let its solution be x'.
  if *(QP($\hat{x}, l, u$)) is infeasible* **then**
    Node can be pruned: infeasible or dominated by upper bound.
  else if x'_I *integral* **then**
    Solve (NLP(l, u)) and let its solution be $\tilde{x}$.
    if *(NLP(l, u)) infeasible or* $\tilde{f} > U$ **then**
      Node can be pruned: infeasible or dominated by upper bound.
    else if $\tilde{x}_I$ *integral* **then**
      Update incumbent solution: $U = \tilde{f}$, $x^* = \tilde{x}$
    else
      BranchOnVariable($\tilde{x}_i, l, u, \mathcal{S}$)
  else
    BranchOnVariable($x'_i, l, u, \mathcal{S}$)

Algorithm 1: QP-diving for MINLP

Proposition 2.1. *Consider solving (1.1) by using Algorithm 1. Assume that the problem functions f and c are convex and twice continuously differentiable, and that X is a bounded polyhedral set. Then, it follows that Algorithm 1 terminates at an optimal solution after searching a finite number of nodes or with an indication that (1.1) has no solution.*

Proof. The proof is similar to that of convergence of nonlinear branch-and-bound for convex MINLPs. We first observe that every branching step splits the parent problem into two child nodes and that the union of the feasible sets of the two child nodes contains all integer feasible points of the parent node. Our assumption that X is a bounded and polyhedral set ensures that the branching process is finite. As long as we branch, we create more subproblems, but we do not eliminate any node that might contain the solution. Hence, we need to consider only what happens when nodes are pruned and show that we cannot remove a node that contains the optimal solution. To this end, we consider the following two cases.

Case (i): QP at a node, (QP($\hat{x}, l, u$)), is infeasible. The convexity of $f(x)$ and $c(x)$ implies that the linearizations are outer approximations, and it therefore follows that either (NLP(l, u)) is infeasible or its solution is dominated by the upper bound, U. In both cases, the node was pruned correctly.

Case (ii): x'_I is an integral solution of (QP($\hat{x}, l, u$)), and we solve (NLP(l, u)) letting the solution be denoted by $\tilde{x}$. If (NLP(l, u)) is infeasible or if $f(\tilde{x}) > U$, then we prune the node because no better solution can be found in this subtree. Otherwise, if $\tilde{x}_I$ is integral, then we obtain a new incumbent and also prune this node. If $\tilde{x}_I$ is not integral, then we choose an integer variable and branch on it. □

The proof shows that in effect it does not matter what problem we solve at intermediate nodes as long as infeasibility of linear constraints is detected correctly (which is similar in spirit to the algorithm of Quesada and Grossmann (1992)). Because our original problem is nonlinear, we cannot rely on the solution of the QP to provide an upper bound or to even be feasible in (1.1). Hence, we must solve (NLP(l,u)) at a node where (QP$(\hat{x},l,u)$) is integer feasible.

It may happen, however, that even though the solution to (QP$(\hat{x},l,u)$) is integer, the solution of (NLP(l,u)) is fractional; that is why we must branch after solving this NLP. This mechanism, however, has the side effect that the solution of (QP$(\hat{x},l,u)$) will be feasible in one of the child nodes on (NLP(l,u)). This observation does not contradict the finite termination of QP-diving (we would simply branch again after another NLP).

3 Extensions of QP-Diving

Even though Algorithm 1 is finite in theory, several concerns exist about its performance. We discuss these concerns in this section, together with possible remedies to mitigate their effect.

1. **Larger Search Trees.** In our numerical results we observe that the search tree generated by QP-diving can be orders of magnitude larger than the search tree required by nonlinear branch-and-bound. There are three reasons for this effect. First, at some nodes, where (NLP(l,u)) is infeasible or its lower bound is higher than the incumbent value, the QP approximation (QP$(\hat{x},l,u)$) may be feasible. Second, in cases where the solution of (QP$(\hat{x},l,u)$) is integral, (NLP(l,u)) may not be integral, and we branch on its solution. This implies that the solution of (QP$(\hat{x},l,u)$) may be feasible in one of the child nodes on (NLP(l,u)). Third, whenever we obtain a new upper bound by solving an NLP, the only way we can exploit this bound is by updating U for all QPs on the search tree. In particular, we cannot prune any nodes on the tree by comparing the QP objective value with the new upper bound.

2. **Accuracy of the QP Approximation.** The QP approximation is likely to change significantly as we search the tree. In particular, the active nonlinear constraints will change as binary variables switch units on and off. We have observed that many multipliers of the nonlinear constraints are zero at the root node, resulting in a Hessian approximation that takes only partial nonlinear information into account. In addition, the linearizations clearly change as we move down tree.

We can reduce the tree size by tightening the QP approximations. If the solution to (QP$(\hat{x},l,u)$), say x', violates the nonlinear constraints

$$c(x) \leq 0 \quad \text{or} \quad f(x) \leq \eta \tag{3.3}$$

by more than a certain (predetermined) value, then we add an outer approximation cut to separate x' and ensure that it will not be feasible in any QP in the corresponding subtree. This approach is similar to that of Quesada and Grossmann (1992) and Abhishek et al. (2010). Thus, we add cuts of the form

$$\eta \geq f^{(k)} + \nabla f^{(k)^T}(x - x^{(k)}) \tag{3.4}$$
$$0 \geq c^{(k)} + \nabla c^{(k)^T}(x - x^{(k)}) \tag{3.5}$$

for some $k = 1, \dots, K$. We add these inequalities in our algorithm for points obtained after solving NLPs and QPs. However, one concern is that this approach increases the size and solution cost of the QP. We therefore add only those constraints that are either binding or are violated. When new constraints are added, we can still hot-start BQPD. In our implementation, however, we prefer to reload the problem and provide the previous solution as the starting point.

Another concern is that when we add linearizations to tighten, the Hessian matrix is no longer consistent. In particular, η now becomes a piecewise linear function involving several supporting hyperplanes to $f(x)$ from previous approximation points. This piecewise nature should also be reflected in the Hessian. We can achieve this goal by updating the Hessian matrix to construct a convex combination of previous Hessians

$$H = \sum_{k=1}^{K} \lambda_k H^{(k)}, \quad \text{where} \quad \sum_{k=1}^{K} \lambda_k = 1,$$

where $H^{(k)}$ is the Hessian of the Lagrangian at $x^{(k)}$ and where $\lambda_k \geq 0$ are the QP multipliers. We regard practical rules for creating a new QP or updating the Hessian as an important aspect of future work.

Two approaches can be used to address the accuracy of the QP approximation. To tackle the difficulty arising when dual multipliers of all nonlinear constraints are zero, we change all the multipliers at zero to a small, fixed, specific positive value. The correctness of the algorithm is not affected by changing the multipliers as long as they are nonnegative. We capture second-order information about the constraints by this process, which would be lost otherwise. To address the difficulty accuracy of the linear and quadratic approximation, we can update the QP approximations during the tree search. For example, after backtracking we can update the NLP and create a new QP approximation about the solution of that QP. However, updating the QP in this way would mean that we can no longer use hot-starts, and we have therefore not implemented this approach.

We note that all extensions retain the convergence properties of the basic algorithm that rely only on finiteness of the tree search and convexity of the problem functions. Next, we present numerical comparisons of our algorithm with regular branch-and-bound methods.

4 Numerical Experiments

We have implemented QP-diving in MINOTAUR (Mahajan et al., 2011), a flexible toolkit written in C++ for solving MINLP problems. We compare compare QP-diving with two branch-and-bound versions of BONMIN (Bonami et al., 2008), namely, using IPOPT (Wächter and Biegler, 2006) and filterSQP (Fletcher and Leyffer, 1998) as NLP solvers. One benefit of using QP approximations is that some MILP-based cut-generation techniques, such as flow-cover inequalities (Gu et al., 1999) and knapsack inequalities (Marchand and Wolsey, 1999), can be used directly in QP-diving. We have not experimented with this option because MINOTAUR currently does not have routines to generate these cuts.

All experiments were performed on a Linux workstation with a 2.6 GHz Intel Xeon processor, 8 MB cache, and 32 GB RAM. All software was compiled by using GNU compiler version 4.1.2. BONMIN version 1.5.1 was built with IPOPT version 3.10.1 and HSL libraries. MINOTAUR was compiled by using the same version of IPOPT. The test problems were taken from the IBM/CMU collection of test problems (CMU, 2012), and all solvers were run with a time limit of two hours. When using filterSQP for QP-diving, we had to reload `BQPD` after every call to solve NLP, because filterSQP also calls `BQPD` internally and changes some statically defined variables in it. All zero dual variables of the initial NLP relaxation were assigned a value of 0.5.

Important metrics from our computational tests are tabulated in Appendix A. Table A.1 reports the time taken to solve each instance and the number of nodes explored. We also report in Table A.2 the number of NLPs or QPs solved and the time used to solve them by three variants of MINOTAUR (two branch-and-bound methods and QP-diving). We graphically represent the data using what we call extended performance-profiles, described next.

4.1 Extended Performance-Profiles

A performance profile (Dolan and Moré, 2002) is a graph representing the relative performance of different solvers according to a specific metric measured for a given set of problem instances. We extend the notion of performance profiles to provide a more informative picture of the results. Since running time is usually the metric of choice, we will use it in the remaining discussion. Other metrics, such as the number of iterations, can also be used with the same method.

Suppose we are given a set $\mathcal{I}$ of problem instances, a set S of solvers, and measured value of solution times, say $t_{p,s}$, for each $p \in \mathcal{I}$ and $s \in S$. The *performance ratio* is defined as

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S\}}, \tag{4.6}$$

and a related distribution function $\rho_s(\tau)$ is defined for each solver $s \in S$ as

$$\rho_s(\tau) = \frac{\text{size}\{p \in \mathcal{I} \mid r_{p,s} \le \tau\}}{|S|}. \tag{4.7}$$

Obviously, $\rho_s(\tau) \in [0, 1]$ is an increasing function of τ; in fact, it is cumulative distribution function for the performance ratio. Dolan and Moré (2002) noted that if the set S is suitably large and representative of the class of problems it contains, $\rho_s(\tau)$ is the probability that solver s is at most τ times slower than the best-performing solver on a randomly selected instance of that class. In particular, $\rho_s(1)$ is the probability that solver s is the fastest for a problem instance, and $\rho_s(4)$ is the probability that solver s can solve an instance at most four times slower than any other solver.

One limitation of the performance profile is that it does not tell us the probability that a given solver s is *faster* than any other solver by a given factor τ. It gives us only the probability that it is *slower* by at most a factor τ. We overcome this limitation by changing the definition of performance ratio (4.6) to

$$\hat{r}_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S, i \neq s\}} \tag{4.8}$$

and that of the distribution function $\rho_s(\tau)$ to

$$\hat{\rho}_s(\tau) = \frac{\text{size}\{p \in \mathcal{I} \mid \hat{r}_{p,s} \leq \tau\}}{|S|}. \tag{4.9}$$

We note that by definition (4.6), $r_{p,s} \geq 1$ for all $p \in \mathcal{I}, s \in S$. Consequently, $\rho_s(\tau) = 0$ when $\tau < 1$. However, $\hat{r}_{p,s}$ can assume values less than one if solver s is the fastest solver for the instance p. It also implies $\hat{\rho}_s(\tau)$ is not necessarily zero when $\tau < 1$. In fact, it follows for our definitions that when $\tau < 1$, then $\hat{\rho}_s(\tau)$ is precisely the probability that solver s is faster than any other solver in S by at least factor $1/\tau$. For example, $\hat{\rho}_s(0.25)$ is the probability that solver s is four times faster than any other solver in S on a given instance.

Thus, our definition extends the existing definition of performance profile. This fact follows from observing that $\text{size}\{p \in \mathcal{I} \mid \hat{r}_{p,s} \leq \tau\} = \text{size}\{p \in \mathcal{I} \mid r_{p,s} \leq \tau\}$ for $\tau \geq 1$, and thus $\rho_s(\tau) = \hat{\rho}_s(\tau)$ when $\tau \geq 1$, and motivates the term "extended performance-profiles."

We now consider examples of the two profiles constructed from our experiments; see Figure 2. Observe that the extended performance-profile looks exactly the same as the performance profile for $\tau \geq 1$. In addition, the extended performance-profile indicates that solver "QP-diving-IPOPT" is faster than the other two solvers by a factor of 2 or more on almost 30% of the instances (or equivalently, that the probability of its being at least twice as fast as the other two solvers is 0.3).

Before we discuss the results of our computational experiments, we comment on the characteristics of extended performance-profiles.

- The extended performance-profile is, like the performance profile, a nondecreasing, piecewise constant function, continuous from the right at each breakpoint.
- The value $\hat{\rho}_s(1)$ is the probability that solver s is faster than all other solvers in S.
- $\lim_{\tau \searrow 0} \hat{\rho}_s(\tau)$ is the probability that out of all solvers in S, s alone solves an instance.
- If $\hat{r}_{p,s} \in (0, r_M]$ and if $\hat{r}_{p,s}$ is r_M only when instance p is not solved by solver s, then $\hat{\rho}_s(r_M)$ is the probability that solver s solves an instance.

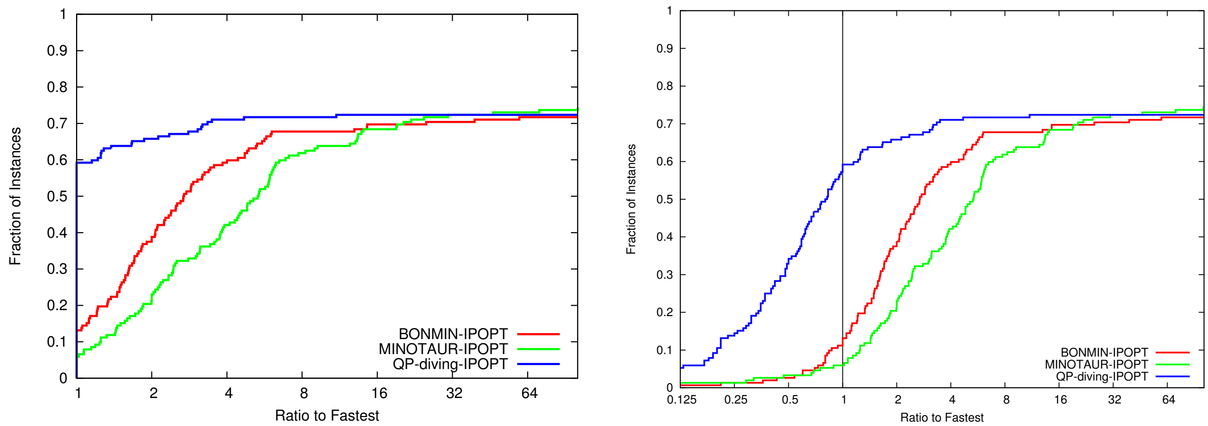

Figure 2: Performance profile (left) and extended performance-profile (right) generated by using the same data.

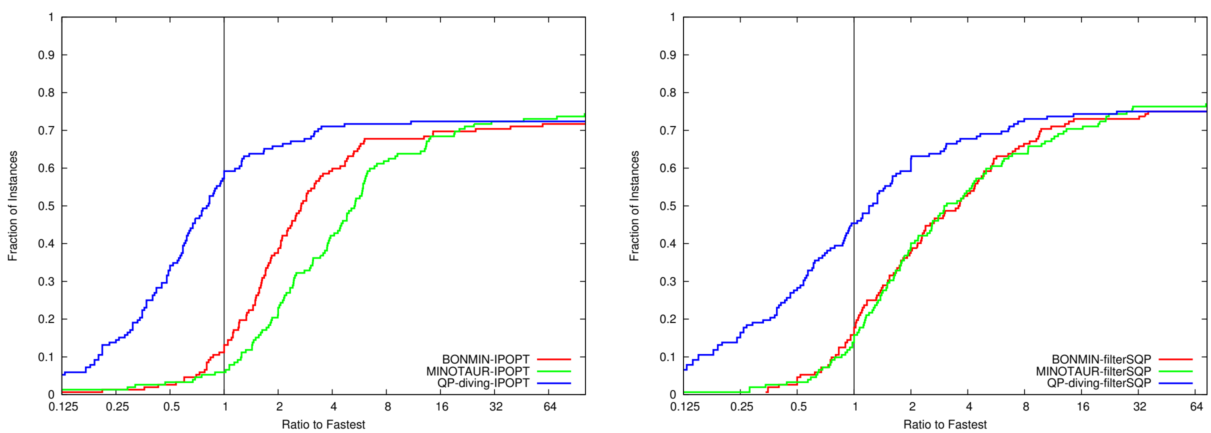

Figure 3: Extended performance-profile comparing time in nonlinear branch-and-bound with QP-diving using IPOPT (left) and filterSQP (right) as the NLP solver.

4.2 Computational Performance of QP-Diving

Our results are summarized in Figures 3–6. The two profiles in Figure 3 compare the CPU time of QP-diving with the CPU time for the two branch-and-bound implementations. The left figure shows the results for IPOPT and the right for filterSQP. We observe that when IPOPT is used as the NLP solver, QP-diving is the fastest solver on nearly 60% of all the instances and on 80% of all instances that are solved in two hours. It is at least two times faster than any other solver on 30% of all instances. The filterSQP solver speeds the performance of the branch-and-bound algorithms of BONMIN and MINOTAUR, but QP-diving still performs significantly better than the two. It is fastest among the three on 45% of all instances, and at least two times faster on 25%. A comparison of all six tests, shown in Figure 4, confirms the superior performance of QP-diving.

Figure 5 shows the comparison based on the number of nodes generated during the tree search. We observe that the trees generated by QP-diving are often an order of magnitude larger than the

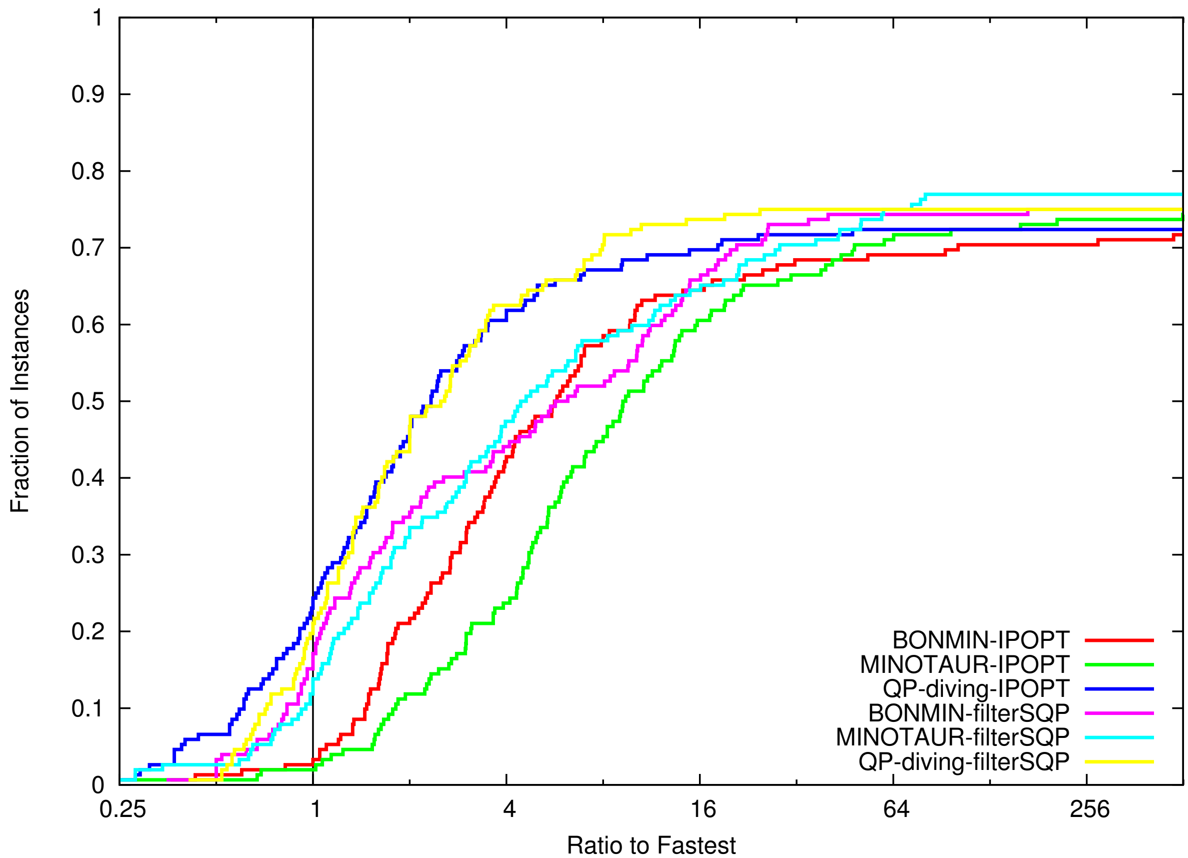

Figure 4: Extended performance-profile comparing time in nonlinear branch-and-bound with time in QP-diving.

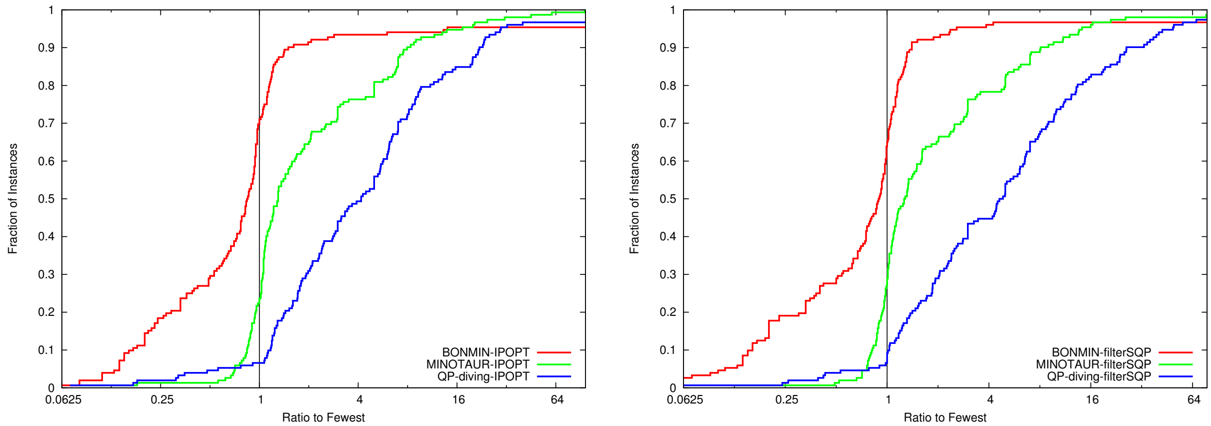

Figure 5: Extended performance-profile comparing number of nodes in nonlinear branch-and-bound with QP-diving using IPOPT (left) and filterSQP (right) as the NLP solver.

trees for the two branch-and-bound solvers. The reasons for this inefficiency are discussed in Section 3. In spite of processing a larger number of nodes, QP-diving outperforms branch-and-bound in terms of CPU time. The reason for this success is that QP-diving solves every node significantly faster. For example, the results for `RSyn0805M02M` (see Table A.2) show that even though the QP-diving tree is about five times larger than the regular tree, the QP warm-starting makes it faster overall. This improved per node performance can be seen in Figure 6, where we compare time spent per NLP node by the three solvers IPOPT, filterSQP, and `BQPD`. For the first two, we used the time reported by MINOTAUR branch-and-bound. For `BQPD`, we used the time reported by QP-diving with filterSQP. The time for `BQPD` also includes that spent in reloading QPs every time cuts are added or filterSQP was called, which we think can be improved significantly. We observe that `BQPD` is at least 16 times faster than both NLP solvers on 12% of all instances, 8 times faster on 30%, and 4 times faster on 68%. We expect to see a reduction in tree size and further improvement in solution time once we experiment with updated QP approximations during the tree search.

The number of cuts generated is tabulated in Table A.3. When we solved with filterSQP, 18 problems did not see any cuts, 72 saw fewer than 20 cuts, and only 17 saw more than 100 cuts. We generated more than a thousand cuts for trimloss instances, which suggests that removing unnecessary cuts and better cut management may be useful for some difficult instances.

5 Conclusions and Future Work

We have presented a new branch-and-bound algorithm that searches the tree by using QP approximations that can be warm-started at every node. We observe favorable numerical performance compared with a standard nonlinear branch-and-bound algorithm. Our algorithm and its implementation can be improved in several ways, however. We have already mentioned the problem of updating the objective function of the QP. Valid inequalities for the integer-constrained QP are

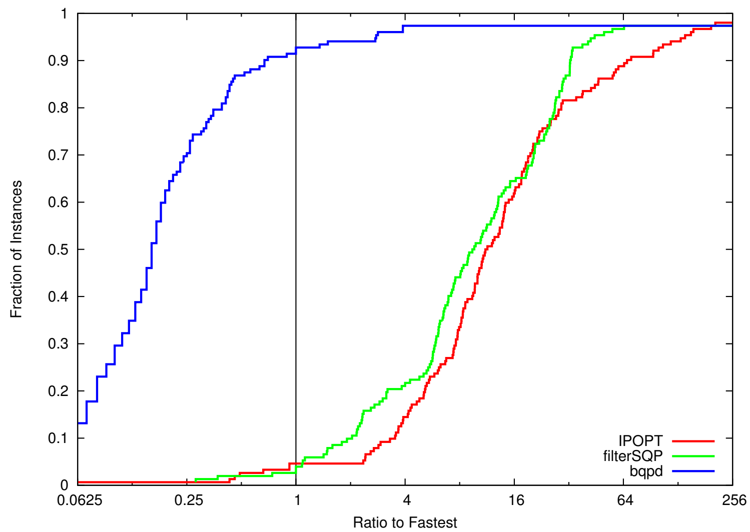

Figure 6: Extended performance-profile comparing time spent per call to NLP solvers.

valid for the MINLP as well. These inequalities can be added along with the linearizations we mentioned to get even tighter relaxations. On the implementation side large improvements are also possible. By redesigning the interfaces for QP solvers, we can avoid loading the whole problem when we add new inequalities. Methods for warm-starting QPs not just after bound changes but also when the objective is changed or when constraints are removed or added will also benefit.

An extension of our work is to consider the algorithm in the context of heuristics. One example would be a local branching heuristic (Fischetti and Lodi, 2003), which can be interpreted as a trust-region approach. If all integer variables are binary, then we can interpret local branching as adding a trust region around a current binary point x^* to (QP($\hat{x}, l, u$)):

$$\|x_I^* - x_I\|_1 \le \Delta_I, \quad \Leftrightarrow \quad \sum_{i \in I : x_i^* = 0} x_i + \sum_{i \in I : x_i^* = 1} (1 - x_i) \le \Delta_I,$$

where $\Delta \ge 1$ is an integer that corresponds to the number of bit changes from x^* to x. Similarly, we can add a trust region around the continuous variables of the form

$$\|x_C^* - x_C\|_\infty \le \Delta_C,$$

where C is the index set of continuous variables and $\Delta_C > 0$ is the continuous trust region radius. In this case, our approach resembles the SMIQP approach of Exler and Schittkowski (2007). Other heuristics, such as diving and a feasibility pump, can be extended to solve only QPs in a similar way.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. This work was also supported by the U.S. Department of Energy through grant DE-FG02-05ER25694.

References

Abhishek, K., Leyffer, S., and Linderoth, J. T. (2010). FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal on Computing*, 22:555–567. DOI:10.1287/ijoc.1090.0373.

Achterberg, T., Koch, T., and Martin, A. (2004). Branching rules revisited. *Operations Research Letters*, 33:42–54.

Altunay, M., Leyffer, S., Linderoth, J. T., and Xie, Z. (2011). Optimal security response to attacks on open science grids. *Computer Networks*, 55:61–73.

Bacher, R. (10-12 December 1997). The Optimal Power Flow (OPF) and its solution by the interior point approach. EES-UETP Madrid, Short Course.

Bartholomew, E. F., O'Neill, R. P., and Ferris, M. C. (2008). Optimal transmission switching. *IEEE Transactions on Power Systems*, 23:1346–1355.

Bertsekas, D. and Gallager, R. (1987). *Data Networks*. Prentice-Hall, Endlewood Cliffs, NJ.

Bhatia, R., Segall, A., and Zussman, G. (2006). Analysis of bandwidth allocation algorithms for wireless personal area networks. *Wireless Networks*, 12:589–603.

Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74:121–140.

Bienstock, D. and Mattia, S. (2007). Using mixed-integer programming to solve power grid blackout problems. *Discrete Optimization*, 4:115–141.

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

Bonami, P., Lee, J., Leyffer, S., and Waechter, A. (2011). More branch-and-bound experiments in convex nonlinear integer programming. Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division.

Boorstyn, R. and Frank, H. (1977). Large-scale network topological optimization. *IEEE Transactions on Communications*, 25:29–47.

Borchers, B. and Mitchell, J. E. (1994). An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21:359–368.

Bragalli, C., D'Ambrosio, C., Lee, J., Lodi, A., and Toth, P. (2006). An MINLP solution method for a water network problem. In *Algorithms - ESA 2006 (14th Annual European Symposium. Zurich, Switzerland, September 2006, Proceedings)*, pages 696–707. Springer.

Bussieck, M. R. and Drud, A. (2001). SBB: A new solver for mixed integer nonlinear programming. Talk, OR 2001, Section *Continuous Optimization*.

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

Chi, K., Jiang, X., Horiguchi, S., and Guo, M. (2008). Topology design of network-coding-based multicast networks. *IEEE Transactions on Mobile Computing*, 7(4):1–14.

CMU (2012). CMU-IBM open source MINLP project. http://egon.cheme.cmu.edu/ibm/page.htm.

Costa-Montenegro, E., González-Castaño, F. J., Rodriguez-Hernández, P. S., and Burguillo-Rial, J. C. (2007). Nonlinear optimization of IEEE 802.11 mesh networks. In *ICCS 2007, Part IV*, pages 466–473, Springer Verlag, Berlin.

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

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

Donde, V., Lopez, V., Lesieutre, B., Pinar, A., Yang, C., and Meza, J. (2005). Identification of severe multiple contingencies in electric power networks. In *Proceedings 37th North American Power Symposium*. LBNL-57994.

Dribeek, N. J. (1966). An algorithm for the solution of mixed integer programming problems. *Management Science*, 12:576–587.

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

Eliceche, A. M., Corvalán, S. M., and Martínez, P. (2007). Environmental life cycle impact as a tool for process optimisation of a utility plant. *Computers and Chemical Engineering*, 31:648–656.

Elwalid, A., Mitra, D., and Wang, Q. (2006). Distributed nonlinear integer optimization for data-optical internetworking. *IEEE Journal on Selected Areas in Communications*, 24(8):1502–1513.

Exler, O. and Schittkowski, K. (2007). A trust region SQP algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1:269–280.

Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming*, 98:23–47.

Fletcher, R. (1995). User manual for BQPD. University of Dundee.

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

Fletcher, R. and Leyffer, S. (1998). User manual for filterSQP. University of Dundee Numerical Analysis Report NA-181.

Fletcher, R. and Leyffer, S. (2005). MINLP (AMPL input). http://www-neos.mcs.anl.gov/neos/solvers/MINCO:MINLP-AMPL.

Flores-Tlacuahuac, A. and Biegler, L. T. (2007). Simultaneous mixed-integer dynamic optimization for integrated design and control. *Computers and Chemical Engineering*, 31:648–656.

Floudas, C. (1995). *Nonlinear and Mixed–Integer Optimization*. Topics in Chemical Engineering. Oxford University Press, New York.

Garver, L. L. (1997). Transmission network estimation using linear programming. *IEEE Transactions on Power Apparatus Systems*, 89:1688–1697.

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

Goldberg, N., Leyffer, S., and Safro, I. (2012). Optimal response to epidemics and cyber attacks in networks. Preprint ANL/MCS-1992-0112, Argonne National Laboratory, Mathematics and Computer Science Division.

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

Grossmann, I. E. and Kravanja, Z. (1997). Mixed–integer nonlinear programming: A survey of algorithms and applications. In L.T. Biegler, T.F. Coleman, A. C. and Santosa, F., editors, *Large–Scale Optimization with Applications, Part II: Optimal Design and Control*, Springer, New York.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W. P. (1999). Lifted flow covers for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467.

Guerra, A., Newman, A. M., and Leyffer, S. (2009). Concrete structure design using mixed-integer nonlinear programming with complementarity constraints. Preprint ANL/MCS-P1869-1109, Argonne National Laboratory, Mathematics and Computer Science Division. To appear in SIAM Journal on Optimization.

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

Hedman, K. W., O'Neill, R. P., Fisher, E. B., and Oren, S. S. (2008). Optimal transmission switching - sensitivity analysis and extensions. *IEEE Transactions on Power Systems*, 23:1469–1479.

Jobst, N. J., Horniman, M. D., Lucas, C. A., and Mitra, G. (2001). Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. *Quantitative Finance*, 1:489–501.

Karuppiah, R. and Grossmann, I. E. (2006). Global optimization for the synthesis of integrated water systems in chemical processes. *Computers and Chemical Engineering*, 30:650–673.

Lee, J. and Leyffer, S., editors (2011). *Mixed Integer Nonlinear Programming*, IMA Volume in Mathematics and its Applications. Springer, New York.

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

Leyffer, S. (2001). Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization & Applications*, 18:295–309.

Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., and Munson, T. (2011). MINOTAUR: a toolkit for solving mixed-integer nonlinear optimization. wiki-page. http://wiki.mcs.anl.gov/minotaur.

Marchand, H. and Wolsey, L. (1999). The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15–33.

Momoh, J., Koessler, R., Bond, M., Stott, B., Sun, D., Papalexopoulos, A., and Ristanovic, P. (1997). Challenges to optimal power flow. *IEEE Transaction on Power Systems*, 12:444–455.

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

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

Romero, R., Monticelli, A., Garcia, A., and Haffner, S. (2002). Test systems and mathematical models for transmission network expansion planning. *IEE Proceedings – Generation, Transmission and Distrbution.*, 149(1):27–36.

Sheikh, W. and Ghafoor, A. (2010). An optimal bandwidth allocation and data droppage scheme for differentiated services in a wireless network. *Wireless Communications and Mobile Computing*, 10(6):733–747.

Sinha, R., Yener, A., and Yates, R. D. (2002). Noncoherent multiuser communications: Multistage detection and selective filtering. *EURASIP Journal on Applied Signal Processing*, 12:1415–1426.

Soleimanipour, M., Zhuang, W., and Freeman, G. H. (2002). Optimal resource management in wireless multimedia wideband CDMA systems. *IEEE Transactions on Mobile Computing*, 1(2):143–160.

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

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

Westerlund, T. and Pettersson, F. (1995). A cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering*, 19:s131–s136.

You, F. and Leyffer, S. (2010). Oil spill response planning with MINLP. *SIAG/OPT Views-and-News*, 21(2):1–8.

You, F. and Leyffer, S. (2011). Mixed-integer dynamic optimization for oil-spill response planning with integration of a dynamic oil weathering model. *AIChe Journal*. Published online: DOI: 10.1002/aic.12536.

A Numerical Results

Table A.1: Time used (seconds) and nodes processed in branch-and-bound. "-1" denotes a failure.

	BONMIN-IPOPT		MINOTAUR-IPOPT		QP-diving-IPOPT		BONMIN-fSQP		MINOTAUR-fSQP		QP-diving-fSQP	
Instance	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes
BatchS101006M	25.86	442	30.93	389	21.25	2383	12.16	446	7.18	397	47.68	5221
BatchS121208M	54.09	600	72.94	601	39.39	4355	31.71	596	90.48	3003	115.64	9585
BatchS151208M	162.81	1792	207.49	1687	145.42	9117	133.83	2229	176.62	4373	409.66	27389
BatchS201210M	222.72	1594	271.37	1366	149.90	8199	194.60	1862	578.76	9627	1173.91	64279
CLay0203H	7.90	212	15.00	192	0.61	107	2.39	219	1.72	289	0.25	85
CLay0203M	3.40	225	10.57	467	4.37	279	0.27	232	0.31	373	0.24	234
CLay0204H	75.17	1477	46.37	1409	5.17	1688	29.85	1516	30.01	1914	2.70	1185
CLay0204M	15.76	1802	35.51	1937	8.26	3073	3.27	1467	0.90	1442	5.90	6254
CLay0205H	-1	-1	714.61	12062	92.37	17409	568.35	11697	570.44	12519	86.75	21646
CLay0205M	262.02	18832	391.68	20010	109.93	59550	66.61	16422	22.54	13188	105.14	116801
CLay0303H	28.26	385	51.26	444	0.72	186	5.10	360	6.38	941	0.53	191
CLay0303M	7.04	388	33.92	799	13.12	950	0.70	394	1.25	1123	0.76	734
CLay0304H	202.71	1837	347.30	1906	3.42	645	76.01	2669	159.19	8503	2.18	629
CLay0304M	147.23	2966	925.04	19829	689.51	25178	14.42	4618	89.63	28440	41.39	28074
CLay0305H	2405.67	15549	1952.17	16072	95.76	18574	806.10	15011	850.92	16318	101.71	23094
CLay0305M	554.33	19186	794.91	31440	7200.04	620687	87.78	19371	110.16	25761	1275.79	820646
FLay02H	0.07	0	0.14	7	0.07	7	0.05	0	0.03	7	0.04	7
FLay02M	0.03	0	0.07	7	0.05	7	0.01	0	0.01	7	0.03	7
FLay03H	1.54	100	1.76	106	0.96	127	0.88	110	0.54	104	0.39	127
FLay03M	0.41	98	0.73	103	0.34	117	0.07	102	0.06	103	0.10	125
FLay04H	54.55	2714	51.19	2331	20.93	2745	47.98	2664	35.87	2325	13.90	2775
FLay04M	13.16	2810	21.01	2517	4.78	2711	1.88	2728	1.54	2608	1.36	2721
FLay05H	2739.25	90780	2518.04	76380	1747.91	122253	5536.38	109186	3706.54	83741	1519.38	127595

Instance	BONMIN-IPOPT CPU	BONMIN-IPOPT Nodes	MINOTAUR-IPOPT CPU	MINOTAUR-IPOPT Nodes	QP-diving-IPOPT CPU	QP-diving-IPOPT Nodes	BONMIN-fSQP CPU	BONMIN-fSQP Nodes	MINOTAUR-fSQP CPU	MINOTAUR-fSQP Nodes	QP-diving-fSQP CPU	QP-diving-fSQP Nodes
FLay05M	556.31	96586	571.63	59691	270.13	113867	142.98	106922	100.71	84926	94.78	111250
FLay06H	7201.08	180728	7200.03	173959	7200.01	596312	7200.37	71586	7200.01	81871	7200.14	717942
FLay06M	7201.98	586634	7200.01	634593	7200.00	3093407	7206.75	3666524	7200.00	3962640	7200.00	4804475
RSyn0805H	3.11	44	3.21	57	0.55	109	7.77	44	10.44	124	1.95	219
RSyn0805M	53.11	2703	408.52	22561	8.78	4755	19.62	2481	23.91	3186	8.57	5955
RSyn0805M02H	13.30	14	36.84	194	2.76	85	69.02	14	139.70	199	5.03	89
RSyn0805M02M	2809.94	44330	4157.82	69540	2663.91	282407	2743.36	43878	4099.54	70603	2668.97	330549
RSyn0805M03H	21.22	10	120.68	312	6.28	93	63.97	12	172.55	97	16.84	73
RSyn0805M03M	7199.95	65907	6085.55	64611	7200.03	398622	7199.92	43561	7200.06	58109	7200.02	483792
RSyn0805M04H	20.18	2	120.45	192	6.32	55	64.28	0	179.41	77	8.99	49
RSyn0805M04M	7199.99	37865	7200.12	46303	7200.07	215853	7199.94	21449	7200.12	31897	7200.04	227642
RSyn0810H	3.66	34	4.26	70	0.70	99	3.03	34	5.51	121	0.56	89
RSyn0810M	298.52	14633	2049.95	100471	375.49	219901	82.88	11551	53.74	10341	396.65	237115
RSyn0810M02H	19.23	44	41.22	195	13.22	306	114.66	42	576.28	689	43.23	327
RSyn0810M02M	7200.64	105227	7200.05	108984	7200.02	975065	7200.52	95389	7200.08	102498	7200.01	948103
RSyn0810M03H	67.43	146	214.11	731	39.51	571	319.07	164	2010.71	1313	174.89	836
RSyn0810M03M	7200.46	56490	7200.11	65639	7200.07	884521	7200.19	34041	7200.17	48425	7200.01	808281
RSyn0810M04H	37.21	8	322.60	485	10.48	71	177.53	10	266.34	71	13.57	65
RSyn0810M04M	7200.29	31746	7200.05	38581	7200.02	631915	7199.78	16386	7200.27	24582	7200.02	615599
RSyn0815H	5.21	54	6.16	91	0.87	111	12.17	54	18.40	109	1.43	117
RSyn0815M	7200.15	244605	396.07	18470	1250.07	439858	85.97	8017	84.19	8832	375.44	190051
RSyn0815M02H	14.14	10	65.57	245	3.00	47	62.59	10	110.23	120	6.78	97
RSyn0815M02M	7200.44	53730	7200.07	107343	7200.03	1064957	7200.54	75213	7200.02	84632	7200.01	988018
RSyn0815M03H	61.27	72	285.04	840	21.26	264	381.09	92	1653.43	888	55.19	544
RSyn0815M03M	7200.38	32474	7200.04	60940	7200.06	676874	7200.22	28656	7200.16	35430	7200.02	745400
RSyn0815M04H	49.71	12	385.68	543	27.41	157	291.28	10	1182.66	257	54.97	186
RSyn0815M04M	7200.02	13731	7200.26	34888	7200.07	552280	7199.90	14598	7200.32	19509	7200.02	589701
RSyn0820H	6.08	73	5.31	82	1.10	117	21.16	71	12.59	82	8.87	475
RSyn0820M	7200.13	270531	7200.01	345926	1648.57	548869	989.05	73774	1085.10	83481	935.49	395093
RSyn0820M02H	23.38	42	85.29	285	9.25	175	184.18	28	552.07	382	24.70	183
RSyn0820M02M	7200.84	75454	7200.07	92194	7200.03	1506283	7200.72	67735	7200.11	69394	7200.05	1415701
RSyn0820M03H	98.72	194	509.82	1541	36.89	465	945.25	232	2970.74	1302	100.05	1022
RSyn0820M03M	7200.56	39705	7200.09	51504	7200.02	930856	7199.94	24395	7200.07	31997	7200.01	874957

Instance	BONMIN-IPOPT		MINOTAUR-IPOPT		QP-diving-IPOPT		BONMIN-fSQP		MINOTAUR-fSQP		QP-diving-fSQP	
	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes
RSyn0820M04H	122.01	91	978.26	1821	74.80	389	1115.70	96	4408.82	1050	203.62	425
RSyn0820M04M	7200.37	21127	7200.35	29535	7200.02	617815	7199.71	12986	7200.15	16999	7200.02	643890
RSyn0830H	5.33	32	5.45	60	5.10	175	26.29	38	24.26	90	8.68	175
RSyn0830M	7201.61	225135	7200.03	293387	3873.49	1391071	2612.48	151510	2485.99	144708	3299.63	1259559
RSyn0830M02H	31.37	75	139.55	567	23.66	313	380.70	71	307.04	193	77.94	315
RSyn0830M02M	7200.92	67073	7200.07	80827	7200.01	1562498	7200.48	44335	7200.07	43121	7200.03	1424931
RSyn0830M03H	75.57	106	369.02	760	68.89	473	997.47	104	569.31	166	221.39	503
RSyn0830M03M	7200.53	33898	7200.04	39718	7200.02	779706	7199.98	19537	7200.09	18439	-1	-1
RSyn0830M04H	317.03	446	888.47	1222	1011.40	4305	4172.53	450	2114.26	400	3093.12	3901
RSyn0830M04M	7200.47	19749	7200.19	22183	-1	-1	7199.54	9861	7200.63	9071	7200.19	611048
RSyn0840H	3.74	14	3.32	21	0.65	27	16.91	14	9.69	21	1.16	27
RSyn0840M	7201.45	207598	7200.04	228884	7200.01	4014276	7202.24	349553	7200.02	344393	7200.01	4223628
RSyn0840M02H	25.85	36	79.71	219	9.67	101	252.88	36	215.84	89	25.87	101
RSyn0840M02M	7200.84	56582	7200.07	67698	7200.10	1241062	7200.29	35471	7200.15	35989	7200.06	1385347
RSyn0840M03H	89.87	109	533.62	1053	149.08	655	1119.33	96	1445.13	320	725.18	757
RSyn0840M03M	7200.55	31476	7200.08	41184	7200.02	679923	7199.64	13447	7200.30	14924	7200.02	753282
RSyn0840M04H	325.73	423	4319.30	7312	765.52	2382	3955.58	465	3774.68	616	2272.66	2599
RSyn0840M04M	7200.35	19738	7200.32	20077	-1	-1	7199.90	7996	7200.72	6969	-1	-1
SLay04H	0.64	35	1.04	37	0.21	42	0.37	35	0.33	37	0.26	84
SLay04M	0.44	35	0.56	37	0.09	39	0.02	35	0.03	37	0.07	39
SLay05H	1.40	59	2.38	61	0.52	76	1.62	59	1.36	61	0.37	69
SLay05M	0.68	59	1.10	61	0.17	73	0.07	59	0.07	61	0.10	61
SLay06H	3.64	110	4.96	98	1.27	126	6.23	114	4.84	116	1.28	142
SLay06M	1.90	110	2.05	99	0.36	122	0.21	110	0.18	98	0.20	101
SLay07H	8.11	241	10.78	200	4.86	353	20.56	214	12.69	153	3.62	314
SLay07M	3.63	241	3.95	199	0.82	212	0.69	241	0.53	198	0.46	179
SLay08H	13.82	269	19.41	287	8.94	461	47.06	249	36.45	314	8.57	424
SLay08M	5.22	265	6.65	303	1.78	402	1.26	261	1.23	304	1.67	383
SLay09H	27.22	438	33.62	385	33.12	1103	134.18	456	94.30	573	36.36	1122
SLay09M	9.30	387	11.83	397	5.94	924	2.73	367	3.05	512	5.48	847
SLay10H	494.31	7902	332.40	5268	3646.12	41735	2063.80	6561	904.73	5308	6354.45	60083
SLay10M	107.63	6682	96.06	4703	293.45	15621	56.37	6700	43.10	4779	343.12	14333
Syn05H	0.04	0	0.05	3	0.05	5	0.01	0	0.01	1	0.02	1

Instance	BONMIN-IPOPT CPU	BONMIN-IPOPT Nodes	MINOTAUR-IPOPT CPU	MINOTAUR-IPOPT Nodes	QP-diving-IPOPT CPU	QP-diving-IPOPT Nodes	BONMIN-fSQP CPU	BONMIN-fSQP Nodes	MINOTAUR-fSQP CPU	MINOTAUR-fSQP Nodes	QP-diving-fSQP CPU	QP-diving-fSQP Nodes
Syn05M	0.08	6	0.12	11	0.04	11	0.01	6	0.02	11	0.02	11
Syn05M02H	0.04	0	0.15	3	0.05	3	0.02	0	0.06	3	0.05	3
Syn05M02M	0.35	12	0.48	19	0.14	19	0.05	12	0.07	19	0.06	19
Syn05M03H	0.07	0	0.27	5	0.07	5	0.03	0	0.13	5	0.06	5
Syn05M03M	0.54	18	1.01	27	0.17	31	0.22	18	0.24	29	0.11	31
Syn05M04H	0.08	0	0.47	7	0.11	7	0.05	0	0.21	5	0.08	5
Syn05M04M	1.23	32	1.93	41	0.32	45	0.39	32	0.47	43	0.18	45
Syn10H	0.07	0	0.08	3	0.04	3	0.01	0	0.03	3	0.02	3
Syn10M	0.33	30	0.43	35	0.07	35	0.02	30	0.03	35	0.04	35
Syn10M02H	0.09	0	0.40	5	0.09	5	0.10	0	0.55	3	0.09	3
Syn10M02M	3.76	246	5.44	230	1.79	321	1.56	246	1.56	237	0.88	321
Syn10M03H	0.32	0	0.85	9	0.16	9	0.15	0	0.75	9	0.18	9
Syn10M03M	19.82	874	21.50	683	8.84	1353	9.71	874	8.71	670	4.49	1355
Syn10M04H	0.47	0	1.56	13	0.29	13	0.74	0	1.55	11	0.32	13
Syn10M04M	59.10	1946	66.44	1686	26.40	2871	33.83	1932	31.04	1588	14.15	2957
Syn15H	0.09	0	0.09	3	0.06	3	0.03	0	0.06	3	0.04	3
Syn15M	0.67	70	1.07	74	0.28	79	0.11	70	0.12	72	0.10	79
Syn15M02H	0.17	0	0.30	3	0.15	3	0.10	0	0.31	3	0.12	3
Syn15M02M	11.39	466	17.23	635	5.51	839	5.49	466	4.46	410	2.71	837
Syn15M03H	0.46	0	0.65	5	0.27	5	0.21	0	0.93	5	0.28	5
Syn15M03M	67.71	1704	55.40	1392	41.98	3233	36.37	1696	36.36	1465	22.48	3341
Syn15M04H	0.63	0	1.27	7	0.52	7	0.47	0	1.81	7	0.49	7
Syn15M04M	299.98	4814	356.42	6571	840.46	15095	183.36	5064	191.09	4460	293.76	15387
Syn20H	0.12	0	0.21	5	0.09	5	3.61	0	0.50	6	0.10	5
Syn20M	5.94	596	7.81	651	2.07	815	1.26	596	1.24	651	0.71	815
Syn20M02H	0.56	2	1.20	7	0.33	7	1.35	0	2.17	7	0.45	7
Syn20M02M	498.89	16653	406.48	15913	199.99	38587	260.33	16782	237.87	15378	135.95	39563
Syn20M03H	1.99	2	2.61	13	0.58	13	97.33	0	12.31	16	3.01	15
Syn20M03M	7200.13	126172	5881.06	135042	7200.18	287472	3844.12	140565	4385.84	137783	4256.88	448763
Syn20M04H	3.58	2	4.51	15	1.10	19	20.09	0	22.94	21	3.79	19
Syn20M04M	7200.61	69274	7200.04	99467	7200.01	552797	7200.95	149542	7200.04	124616	7200.04	762908
Syn30H	0.51	2	0.35	5	0.28	11	0.90	2	0.31	5	0.19	9
Syn30M	24.47	1914	60.76	3901	28.17	15591	23.38	5141	17.87	3920	16.68	16483

	BONMIN-IPOPT		MINOTAUR-IPOPT		QP-diving-IPOPT		BONMIN-fSQP		MINOTAUR-fSQP		QP-diving-fSQP	
Instance	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes
Syn30M02H	1.12	0	1.52	5	0.93	11	3.64	0	2.85	5	1.55	11
Syn30M02M	7201.10	202183	7200.03	203103	7200.08	1257156	7200.96	185612	7200.04	196983	7200.01	1489820
Syn30M03H	5.83	10	4.10	9	2.57	25	28.78	14	13.86	12	6.61	27
Syn30M03M	7201.28	123115	7200.03	130809	7200.02	1093625	7200.68	88018	7200.06	89027	7200.03	1263041
Syn30M04H	18.71	31	10.79	22	10.56	73	108.11	35	34.97	17	23.63	73
Syn30M04M	7201.08	80796	7200.09	67919	7200.01	836133	7200.57	50312	7200.16	49464	7200.14	961396
Syn40H	1.18	10	1.01	12	0.57	21	3.79	10	1.69	12	0.92	25
Syn40M	851.70	59580	833.07	45521	403.55	292019	413.97	59634	314.49	45411	396.31	296455
Syn40M02H	3.42	10	3.19	9	1.86	19	25.03	8	24.96	20	9.90	53
Syn40M02M	7201.46	171069	7200.01	176652	7200.03	1390625	7200.96	112484	7200.05	116418	7200.01	1756859
Syn40M03H	16.39	46	12.38	39	10.82	90	160.67	49	49.92	35	37.29	98
Syn40M03M	7201.09	85865	7200.03	87678	7200.05	992775	7200.59	52030	7200.10	52745	7200.03	1243779
Syn40M04H	62.97	104	43.38	83	145.01	711	459.25	62	172.28	41	302.58	396
Syn40M04M	7200.91	57541	7200.10	62975	-1	-1	7200.40	29991	7200.19	26165	7200.02	662289
Water0202	299.77	24	223.68	35	7304.44	4	-1	-1	1.08	1	1.68	1
Water0202R	4.02	26	1.72	28	3.65	29	0.54	20	0.40	29	4.21	37
Water0303	524.80	56	561.02	82	8562.32	4	-1	-1	1.10	1	1.75	1
Water0303R	457.27	292	31.38	112	109.29	201	16.05	122	4.49	95	110.58	227
fo7	7200.72	271097	7200.00	283924	-1	-1	2615.16	294162	3596.99	497909	-1	-1
fo7_2	5805.85	161263	7200.02	268795	7200.01	4062083	1565.95	162450	3027.16	326312	7200.01	4212973
fo8	7200.76	172196	7200.07	247645	-1	-1	-1	-1	7200.01	409149	7200.01	3356737
fo9	7201.99	273067	7200.03	194816	7200.01	2637864	7201.80	243346	7200.13	227236	7200.00	3043648
o7	7200.86	189167	7200.01	289784	7200.01	4058444	7204.46	1055747	-1	-1	-1	-1
o7_2	7200.78	216050	7200.01	282117	7200.01	6404740	-1	-1	-1	-1	7200.01	5483071
trimloss12	-1	-1	7200.06	136388	7200.09	24091	7200.72	26839	7200.03	62618	7200.18	26855
trimloss2	-1	-1	4.97	425	0.40	385	0.13	392	0.12	441	0.37	385
trimloss4	-1	-1	7200.02	435726	7200.01	2370039	-1	-1	1907.07	1134951	7200.01	2180303
trimloss5	-1	-1	7200.09	434847	7200.03	334711	7204.10	1250587	7200.01	1541004	7200.05	324996
trimloss6	-1	-1	7200.04	348337	7200.01	107098	7202.42	250565	7200.02	321350	7200.28	107726
trimloss7	-1	-1	7200.01	336927	7200.08	108284	7201.93	249881	7200.02	313041	7200.28	105972

Table A.2: NLPs or QPs processed in branch-and-bound and the time (seconds) spent in it. "-1" denotes a failure.

	MINOTAUR-IPOPT			MINOTAUR-filterSQP			QP-diving-filterSQP		
Instance	CPU	NLPs	CPU/100NLPs	CPU	NLPs	CPU/100NLPs	CPU	QPs	CPU/100QPs
BatchS101006M	30.88	527	5.8596	7.13	554	1.2870	40.14	5594	0.7176
BatchS121208M	72.84	815	8.9374	90.11	3734	2.4132	107.2	10304	1.0404
BatchS151208M	207.25	1901	10.9022	176.09	5100	3.4527	375.7	28183	1.3331
BatchS201210M	271.15	1628	16.6554	577.41	10654	5.4197	1120.7	65141	1.7204
CLay0203H	15	244	6.1475	1.7	344	0.4942	0.09	194	0.0464
CLay0203M	10.55	513	2.0565	0.29	423	0.0686	0.03	375	0.0080
CLay0204H	46.33	1571	2.9491	29.94	2081	1.4387	2.07	1509	0.1372
CLay0204M	35.45	2059	1.7217	0.84	1581	0.0531	1.58	8583	0.0184
CLay0205H	713.83	12412	5.7511	569.81	12883	4.4230	82.99	22470	0.3693
CLay0205M	390.72	20354	1.9196	21.94	13540	0.1620	45.06	140913	0.0320
CLay0303H	51.25	520	9.8558	6.34	1008	0.6290	0.25	383	0.0653
CLay0303M	33.9	851	3.9835	1.22	1187	0.1028	0.12	1053	0.0114
CLay0304H	347.2	2096	16.5649	158.86	8687	1.8287	1.44	1015	0.1419
CLay0304M	923.94	19993	4.6213	88.73	28597	0.3103	9.1	38449	0.0237
CLay0305H	1951.1	16486	11.8349	849.74	16717	5.0831	97.89	24029	0.4074
CLay0305M	792.88	31821	2.4917	109.03	26151	0.4169	373.1	1002552	0.0372
FLay02H	0.14	15	0.9333	0.01	15	0.0667	0	24	0.0000
FLay02M	0.07	15	0.4667	0	15	0.0000	0	25	0.0000
FLay03H	1.76	130	1.3538	0.52	128	0.4063	0.13	206	0.0631
FLay03M	0.72	127	0.5669	0.04	127	0.0315	0.02	204	0.0098
FLay04H	51.11	2411	2.1199	35.81	2399	1.4927	8.44	3224	0.2618
FLay04M	20.94	2583	0.8107	1.47	2688	0.0547	0.58	3189	0.0182
FLay05H	2513.1	76634	3.2794	3702.66	84023	4.4067	904.94	142240	0.6362
FLay05M	569.58	59954	0.9500	98.36	85202	0.1154	64.33	125540	0.0512
FLay06H	7185.93	174393	4.1205	7193.19	82323	8.7378	5956.76	734123	0.8114
FLay06M	7161.57	635047	1.1277	7074.95	3963082	0.1785	6158.91	5122187	0.1202
RSyn0805H	3.2	95	3.3684	10.42	164	6.3537	0.76	300	0.2533
RSyn0805M	406.97	23005	1.7691	23.79	3394	0.7009	6.03	6173	0.0977
RSyn0805M02H	36.8	560	6.5714	139.65	501	27.8743	2.12	246	0.8618
RSyn0805M02M	4150.67	70446	5.8920	4093.27	71488	5.7258	2508.75	331699	0.7563
RSyn0805M03H	120.63	1126	10.7131	172.49	365	47.2575	4.23	263	1.6084

Instance	MINOTAUR-IPOPT			MINOTAUR-filterSQP			QP-diving-filterSQP		
	CPU	NLPs	CPU/100NLPs	CPU	NLPs	CPU/100NLPs	CPU	QPs	CPU/100QPs
RSyn0805M03M	6078.43	65989	9.2113	7192.53	59402	12.1082	6628.28	485316	1.3658
RSyn0805M04H	120.39	694	17.3473	179.35	243	73.8066	4.62	203	2.2759
RSyn0805M04M	7193.41	47909	15.0147	7195.06	33399	21.5427	6799.86	229228	2.9664
RSyn0810H	4.25	114	3.7281	5.48	177	3.0960	0.35	140	0.2500
RSyn0810M	2040.41	100911	2.0220	53.32	10515	0.5071	282.43	237331	0.1190
RSyn0810M02H	41.19	536	7.6847	576.15	1431	40.2621	9.07	548	1.6551
RSyn0810M02M	7186.22	110066	6.5290	7187.64	103536	6.9422	7018.74	949349	0.7393
RSyn0810M03H	213.98	1601	13.3654	2010.41	2149	93.5510	47.53	1356	3.5052
RSyn0810M03M	7189.96	67025	10.7273	7190.87	49661	14.4799	6329.3	810206	0.7812
RSyn0810M04H	322.45	1587	20.3182	266.28	269	98.9888	8.26	250	3.3040
RSyn0810M04M	7192.47	40083	17.9439	7194.63	26100	27.5656	6343.76	617732	1.0269
RSyn0815H	6.15	155	3.9677	18.37	181	10.1492	0.73	204	0.3578
RSyn0815M	394.48	18963	2.0803	83.83	9062	0.9251	282.57	190368	0.1484
RSyn0815M02H	65.52	719	9.1127	110.18	278	39.6331	3.12	256	1.2188
RSyn0815M02M	7187.53	108565	6.6205	7191.1	85846	8.3767	6378.43	990508	0.6440
RSyn0815M03H	284.91	1740	16.3741	1653.21	1669	99.0539	33.12	880	3.7636
RSyn0815M03M	7189.91	62566	11.4917	7193.52	37022	19.4304	6064.74	748126	0.8107
RSyn0815M04H	385.54	1645	23.4371	1182.54	725	163.1090	23.54	393	5.9898
RSyn0815M04M	7192.08	36862	19.5108	7195.89	21127	34.0602	6061.37	592420	1.0232
RSyn0820H	5.3	128	4.1406	12.57	128	9.8203	3.38	648	0.5216
RSyn0820M	7162.67	346452	2.0674	1081.21	83777	1.2906	696.71	405745	0.1717
RSyn0820M02H	85.24	855	9.9696	551.97	975	56.6123	6.74	366	1.8415
RSyn0820M02M	7186.05	93568	7.6800	7189.61	70692	10.1703	6180.68	1417434	0.4360
RSyn0820M03H	509.54	2615	19.4853	2970.4	2174	136.6329	74.21	1593	4.6585
RSyn0820M03M	7190.74	53334	13.4825	7193.56	33565	21.4317	5806.79	877908	0.6614
RSyn0820M04H	977.88	3185	30.7027	4408.46	1965	224.3491	58.4	698	8.3668
RSyn0820M04M	7193.04	31689	22.6989	7195.6	18949	37.9735	6016.09	646899	0.9300
RSyn0830H	5.44	110	4.9455	24.22	144	16.8194	2.05	306	0.6699
RSyn0830M	7171.37	294015	2.4391	2477.78	145092	1.7077	2920.96	1266561	0.2306
RSyn0830M02H	139.47	1121	12.4416	306.97	345	88.9768	15.63	520	3.0058
RSyn0830M02M	7184.2	82405	8.7182	7191.55	44667	16.1004	5436.52	1427348	0.3809
RSyn0830M03H	368.86	1754	21.0296	569.22	362	157.2431	55.26	771	7.1673
RSyn0830M03M	7189.91	41858	17.1769	7194.69	20483	35.1252	-1	-1	-1.0000

	MINOTAUR-IPOPT			MINOTAUR-filterSQP			QP-diving-filterSQP		
Instance	CPU	NLPs	CPU/100NLPs	CPU	NLPs	CPU/100NLPs	CPU	QPs	CPU/100QPs
RSyn0830M04H	888.16	2318	38.3158	2114.08	644	328.2733	743.75	4704	15.8110
RSyn0830M04M	7193.09	25065	28.6977	7197.24	11517	62.4923	5952.88	614417	0.9689
RSyn0840H	3.3	63	5.2381	9.66	63	15.3333	0.4	74	0.5405
RSyn0840M	7171.87	229566	3.1241	7177.94	344857	2.0814	4696.04	4224267	0.1112
RSyn0840M02H	79.65	549	14.5082	215.78	239	90.2845	6.74	245	2.7510
RSyn0840M02M	7184.91	69530	10.3335	7192.54	37799	19.0284	5276.23	1387837	0.3802
RSyn0840M03H	533.36	1863	28.6291	1444.99	520	277.8827	134.64	1134	11.8730
RSyn0840M03M	7189.63	43598	16.4907	7195.67	17168	41.9133	5696.8	756594	0.7530
RSyn0840M04H	4317.67	8612	50.1355	3774.39	848	445.0932	669.91	3146	21.2940
RSyn0840M04M	7192.69	23261	30.9217	7197.47	10003	71.9531	-1	-1	-1.0000
SLay04H	1.04	85	1.2235	0.32	85	0.3765	0.16	163	0.0982
SLay04M	0.56	85	0.6588	0.02	85	0.0235	0.01	98	0.0102
SLay05H	2.36	141	1.6738	1.35	141	0.9574	0.24	163	0.1472
SLay05M	1.09	141	0.7730	0.06	141	0.0426	0.02	150	0.0133
SLay06H	4.94	218	2.2661	4.81	236	2.0381	0.92	297	0.3098
SLay06M	2.04	219	0.9315	0.16	218	0.0734	0.08	240	0.0333
SLay07H	10.76	368	2.9239	12.64	321	3.9377	2.91	545	0.5339
SLay07M	3.93	367	1.0708	0.5	366	0.1366	0.22	379	0.0580
SLay08H	19.37	511	3.7906	36.34	538	6.7546	7.03	722	0.9737
SLay08M	6.61	527	1.2543	1.18	528	0.2235	1.02	677	0.1507
SLay09H	33.54	673	4.9837	94.11	861	10.9303	30.91	1601	1.9307
SLay09M	11.78	685	1.7197	2.97	800	0.3713	4.24	1277	0.3320
SLay10H	331.22	5628	5.8852	902.93	5668	15.9303	5920.53	66844	8.8572
SLay10M	95.23	5063	1.8809	42.29	6044	0.6997	321.5	16212	1.9831
Syn05H	0.05	5	1.0000	0	1	0.0000	0	0	0.0000
Syn05M	0.12	17	0.7059	0.01	17	0.0588	0	20	0.0000
Syn05M02H	0.14	9	1.5556	0.04	9	0.4444	0	10	0.0000
Syn05M02M	0.48	43	1.1163	0.05	43	0.1163	0.01	50	0.0200
Syn05M03H	0.27	13	2.0769	0.11	13	0.8462	0.01	14	0.0714
Syn05M03M	1.01	74	1.3649	0.22	76	0.2895	0.03	84	0.0357
Syn05M04H	0.46	17	2.7059	0.19	13	1.4615	0.01	14	0.0714
Syn05M04M	1.92	105	1.8286	0.45	101	0.4455	0.08	111	0.0721
Syn10H	0.07	5	1.4000	0.01	5	0.2000	0	6	0.0000

	MINOTAUR-IPOPT			MINOTAUR-filterSQP			QP-diving-filterSQP		
Instance	CPU	NLPs	CPU/100NLPs	CPU	NLPs	CPU/100NLPs	CPU	QPs	CPU/100QPs
Syn10M	0.43	49	0.8776	0.02	49	0.0408	0.01	53	0.0189
Syn10M02H	0.4	17	2.3529	0.54	15	3.6000	0.02	15	0.1333
Syn10M02M	5.42	363	1.4931	1.54	349	0.4413	0.31	458	0.0677
Syn10M03H	0.84	25	3.3600	0.73	25	2.9200	0.05	26	0.1923
Syn10M03M	21.46	991	2.1655	8.67	907	0.9559	2.78	1679	0.1656
Syn10M04H	1.55	33	4.6970	1.53	31	4.9355	0.12	34	0.3529
Syn10M04M	66.34	2220	2.9883	30.94	1982	1.5610	10.42	3502	0.2975
Syn15H	0.09	5	1.8000	0.04	5	0.8000	0.01	6	0.1667
Syn15M	1.06	100	1.0600	0.1	100	0.1000	0.03	120	0.0250
Syn15M02H	0.3	9	3.3333	0.29	9	3.2222	0.04	10	0.4000
Syn15M02M	17.19	789	2.1787	4.43	540	0.8204	1.44	1074	0.1341
Syn15M03H	0.64	13	4.9231	0.91	13	7.0000	0.11	14	0.7857
Syn15M03M	55.33	1680	3.2935	36.27	1841	1.9701	13.53	4194	0.3226
Syn15M04H	1.26	17	7.4118	1.79	17	10.5294	0.23	18	1.2778
Syn15M04M	355.99	7260	4.9034	190.81	5110	3.7341	136.79	21796	0.6276
Syn20H	0.2	11	1.8182	0.48	20	2.4000	0.01	14	0.0714
Syn20M	7.77	685	1.1343	1.21	685	0.1766	0.31	961	0.0323
Syn20M02H	1.19	31	3.8387	2.15	31	6.9355	0.12	33	0.3636
Syn20M02M	405.57	16393	2.4740	237.14	15806	1.5003	102.94	41735	0.2467
Syn20M03H	2.59	43	6.0233	12.28	46	26.6957	0.33	47	0.7021
Syn20M03M	5868.72	135790	4.3219	4376.53	138450	3.1611	2807.26	504477	0.5565
Syn20M04H	4.5	51	8.8235	22.91	77	29.7532	0.68	57	1.1930
Syn20M04M	7190.6	100530	7.1527	7186.2	125636	5.7199	5167.99	795034	0.6500
Syn30H	0.34	13	2.6154	0.29	13	2.2308	0.06	28	0.2143
Syn30M	60.57	4049	1.4959	17.73	4068	0.4358	12.7	17256	0.0736
Syn30M02H	1.51	29	5.2069	2.82	29	9.7241	0.49	57	0.8596
Syn30M02M	7181.16	203929	3.5214	7185.45	197799	3.6327	5620.78	1516954	0.3705
Syn30M03H	4.08	49	8.3265	13.83	56	24.6964	1.71	88	1.9432
Syn30M03M	7183.31	132063	5.4393	7187.22	90149	7.9726	4983.96	1280041	0.3894
Syn30M04H	10.76	82	13.1220	34.93	75	46.5733	6.02	169	3.5621
Syn30M04M	7188.7	69533	10.3385	7190.64	50946	14.1142	5270.33	972788	0.5418
Syn40H	1	30	3.3333	1.68	30	5.6000	0.23	67	0.3433
Syn40M	830.75	45807	1.8136	312.89	45697	0.6847	379.56	297141	0.1277

	MINOTAUR-IPOPT			MINOTAUR-filterSQP			QP-diving-filterSQP		
Instance	CPU	NLPs	CPU/100NLPs	CPU	NLPs	CPU/100NLPs	CPU	QPs	CPU/100QPs
Syn40M02H	3.18	43	7.3953	24.93	80	31.1625	3.76	244	1.5410
Syn40M02M	7176.68	177780	4.0368	7184.83	117500	6.1147	5266.25	1788507	0.2944
Syn40M03H	12.35	99	12.4747	49.88	87	57.3333	7.75	205	3.7805
Syn40M03M	7186.28	89366	8.0414	7191.61	54197	13.2694	5126.67	1252050	0.4095
Syn40M04H	43.34	229	18.9258	172.22	181	95.1492	79.05	1190	6.6429
Syn40M04M	7187.6	64973	11.0624	7193.96	28049	25.6478	5190.23	672185	0.7721
Water0202	222.04	49	453.1429	0.04	1	4.0000	0	0	0.0000
Water0202R	1.61	42	3.8333	0.28	43	0.6512	0.32	70	0.4571
Water0303	559.17	110	508.3364	0.04	1	4.0000	0	0	0.0000
Water0303R	30.92	140	22.0857	4.04	123	3.2846	6.36	308	2.0649
fo7	7179.72	284178	2.5265	3576.27	498163	0.7179	-1	-1	-1.0000
fo7_2	7181.68	269067	2.6691	3013.1	326560	0.9227	6985.57	4213347	0.1658
fo8	7179.75	248009	2.8950	7164.85	409499	1.7497	6995.81	3357262	0.2084
fo9	7177.68	195312	3.6750	7181.28	227694	3.1539	7036.66	3044255	0.2311
o7	7180.44	290050	2.4756	-1	-1	-1.0000	-1	-1	-1.0000
o7_2	7182.01	282389	2.5433	-1	-1	-1.0000	6930.25	5483470	0.1264
trimloss12	7161.7	138748	5.1617	7188.03	64071	11.2189	6074.63	30266	20.0708
trimloss2	4.94	467	1.0578	0.09	472	0.0191	0.03	460	0.0065
trimloss4	7169.9	436232	1.6436	1860.57	1135146	0.1639	4810.83	2181575	0.2205
trimloss5	7153.55	435671	1.6420	7116.22	1541287	0.4617	5881.42	327623	1.7952
trimloss6	7167.93	350113	2.0473	7162.59	321839	2.2255	6275.52	110754	5.6662
trimloss7	7155.98	338703	2.1128	7164.74	313530	2.2852	6302.24	108985	5.7827

Table A.3: Number of cuts added in QP-diving with IPOPT and filterSQP solvers. "-1" denotes a failure.

Instance	IPOPT	fSQP	Instance	IPOPT	fSQP
BatchS101006M	35	40	RSyn0815M02H	7	8
BatchS121208M	19	67	RSyn0815M02M	38	38
BatchS151208M	114	19	RSyn0815M03H	14	13
BatchS201210M	22	8	RSyn0815M03M	64	58
CLay0203H	145	141	RSyn0815M04H	9	11
CLay0203M	37	49	RSyn0815M04M	71	77
CLay0204H	308	286	RSyn0820H	6	7
CLay0204M	58	62	RSyn0820M	22	22
CLay0205H	504	597	RSyn0820M02H	11	11
CLay0205M	79	86	RSyn0820M02M	42	38
CLay0303H	234	238	RSyn0820M03H	17	23
CLay0303M	49	87	RSyn0820M03M	59	59
CLay0304H	387	371	RSyn0820M04H	20	22
CLay0304M	93	126	RSyn0820M04M	78	75
CLay0305H	591	753	RSyn0830H	2	2
CLay0305M	96	198	RSyn0830M	14	14
FLay02H	10	10	RSyn0830M02H	8	8
FLay02M	8	12	RSyn0830M02M	20	19
FLay03H	21	23	RSyn0830M03H	12	10
FLay03M	20	24	RSyn0830M03M	27	-1
FLay04H	34	36	RSyn0830M04H	19	16
FLay04M	37	35	RSyn0830M04M	-1	36
FLay05H	104	85	RSyn0840H	1	1
FLay05M	114	88	RSyn0840M	15	17
FLay06H	182	143	RSyn0840M02H	2	2
FLay06M	228	216	RSyn0840M02M	22	25
RSyn0805H	1	2	RSyn0840M03H	10	16
RSyn0805M	11	12	RSyn0840M03M	43	43
RSyn0805M02H	10	8	RSyn0840M04H	14	12
RSyn0805M02M	28	30	RSyn0840M04M	-1	-1
RSyn0805M03H	6	5	SLay04H	8	23
RSyn0805M03M	35	36	SLay04M	7	6
RSyn0805M04H	3	3	SLay05H	8	8
RSyn0805M04M	56	51	SLay05M	8	7
RSyn0810H	1	0	SLay06H	15	18
RSyn0810M	10	10	SLay06M	14	15
RSyn0810M02H	8	11	SLay07H	44	47
RSyn0810M02M	33	35	SLay07M	29	25
RSyn0810M03H	13	12	SLay08H	59	53
RSyn0810M03M	51	52	SLay08M	52	56
RSyn0810M04H	19	18	SLay09H	103	105
RSyn0810M04M	79	84	SLay09M	93	97
RSyn0815H	1	8	SLay10H	1405	1343
RSyn0815M	22	21	SLay10M	794	762

Instance	IPOPT	fSQP	Instance	IPOPT	fSQP
Syn05H	0	0	Syn30H	6	7
Syn05M	1	1	Syn30M	23	23
Syn05M02H	0	0	Syn30M02H	4	4
Syn05M02M	1	1	Syn30M02M	48	46
Syn05M03H	0	0	Syn30M03H	12	12
Syn05M03M	4	4	Syn30M03M	68	64
Syn05M04H	0	0	Syn30M04H	18	15
Syn05M04M	5	5	Syn30M04M	93	88
Syn10H	0	0	Syn40H	7	7
Syn10M	1	1	Syn40M	27	27
Syn10M02H	0	0	Syn40M02H	1	8
Syn10M02M	21	21	Syn40M02M	35	36
Syn10M03H	0	0	Syn40M03H	19	18
Syn10M03M	54	55	Syn40M03M	67	70
Syn10M04H	0	0	Syn40M04H	11	11
Syn10M04M	58	57	Syn40M04M	-1	79
Syn15H	0	0	Water0202	1	0
Syn15M	9	9	Water0202R	6	8
Syn15M02H	0	0	Water0303	0	0
Syn15M02M	13	13	Water0303R	14	28
Syn15M03H	0	0	fo7	-1	-1
Syn15M03M	18	18	fo7_2	72	73
Syn15M04H	0	0	fo8	-1	98
Syn15M04M	51	37	fo9	80	91
Syn20H	0	6	o7	57	-1
Syn20M	11	12	o7_2	71	76
Syn20M02H	0	0	trimloss12	2318	2998
Syn20M02M	42	42	trimloss2	32	32
Syn20M03H	0	0	trimloss4	1011	979
Syn20M03M	68	66	trimloss5	2434	2421
Syn20M04H	0	0	trimloss6	2926	3071
Syn20M04M	88	88	trimloss7	2954	3048