This article was downloaded by: [155.69.24.171] On: 15 October 2017, At: 22:15
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Deletion Presolve for Accelerating Infeasibility Diagnosis in Optimization Models

Yash Puranik, http://orcid.org/0000-0003-2087-9131Nikolaos V. Sahinidis

To cite this article:
Yash Puranik, http://orcid.org/0000-0003-2087-9131Nikolaos V. Sahinidis (2017) Deletion Presolve for Accelerating Infeasibility Diagnosis in Optimization Models. INFORMS Journal on Computing 29(4):754-766. https://doi.org/10.1287/ijoc.2017.0761

Full terms and conditions of use: http://pubsonline.informs.org/page/terms-and-conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2017, INFORMS

Please scroll down for article—it is on subsequent pages

INFORMS is the largest professional society in the world for professionals in the fields of operations research, management science, and analytics.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

http://pubsonline.informs.org/journal/ijoc/

INFORMS JOURNAL ON COMPUTING
Vol. 29, No. 4, Fall 2017, pp. 754–766
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Deletion Presolve for Accelerating Infeasibility Diagnosis in Optimization Models

Yash Puranik,[a] Nikolaos V. Sahinidis[a]

[a] Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

Contact: puranik@cmu.edu (YP); sahindis@cmu.edu, http://orcid.org/0000-0003-2087-9131 (NVS)

Received: June 17, 2015
Revised: January 5, 2016; May 2, 2016
Accepted: March 20, 2017
Published Online: September 28, 2017

https://doi.org/10.1287/ijoc.2017.0761

Copyright: © 2017 INFORMS

Abstract. Whereas much research in the area of optimization is directed toward developing algorithms for optimization of feasible models, the diagnosis of infeasible models has not received as much attention. Identification of *irreducible infeasible sets* (IISs) can facilitate the process of correcting infeasible models. Several filtering algorithms have been proposed for IIS identification but efficient implementations are available only for linear programs. We propose a novel approach for IIS identification that is applicable to linear programs (LPs), nonlinear programs (NLPs), mixed-integer linear programs (MIPs), and mixed-integer nonlinear programs (MINLPs). The approach makes use of a deletion presolve procedure that exploits bounds tightening techniques to reduce the model to an infeasible set (IS) in a computationally efficient manner. The IS is subsequently reduced to an IIS by applying one of the currently available exact filtering algorithms for IIS identification. We implement the proposed deletion presolve along with four filtering algorithms for IIS identification within the global solver BARON. The effectiveness and usefulness of the proposed approach is demonstrated through computational experiments on a test set of 790 infeasible LPs, NLPs, MIPs, and MINLPs. Deletion presolve rapidly eliminates a large fraction of the problem constraints and speeds up the filtering algorithms by over forty times on average. Speedups of as high as 1,000 times are observed for some problems, while, for 40% of the test problems, the deletion presolve itself reduces the original model to an IIS.

History: Accepted by Michela Milano, Area Editor for Constraint Programming and Hybrid Optimization.

Keywords: infeasibility analysis • irreducible infeasible sets • constraint programming • MINLP

1. Introduction

Research over the past few decades has resulted in tremendous developments in optimization theory, algorithms, and software, concurrent with significant advances in computing hardware. As a result, the complexity of the optimization problems of practical interest that can be solved efficiently is increasing. It is now possible to solve certain NLPs with thousands and even millions of variables to local optimality (Gill et al. 2002, Wächter and Biegler 2006, Biegler and Zavala 2009). LPs and in some cases MIPs with millions of variables and constraints are solved to optimality (Bixby 2002, 2012). As optimization models continue to grow larger, infeasibilities in these models become harder to diagnose and correct. The importance of the problem of infeasibility diagnosis is highlighted from the statistics of problem submissions to the global NLP/MINLP solver BARON (Tawarmalani and Sahinidis 2004) through the NEOS server (Czyzyk et al. 1998). Out of approximately 18,000 problems submitted to BARON in 2012 and 2013, almost 7% were infeasible. These statistics suggest that modelers are frequently faced with the problem of correcting infeasible models. Systematic approaches are necessary for analyzing such models.

It is difficult to substitute domain knowledge for the diagnosis of infeasible models. However, the effort of a domain expert can be reduced substantially by systematic techniques for infeasibility diagnosis. Relevant approaches include identification of a maximal feasible subsystem of constraints (Amaldi et al. 1999), identification of a minimal set of changes to achieve feasibility (Murty et al. 2000), and identification of an irreducible infeasible set (IIS). Also known as an irreducible inconsistent set or minimal infeasible subsystem, an IIS is defined as an infeasible set with every proper subset being feasible. Once an IIS is obtained, the domain expert can analyze it to determine if the model is indeed infeasible or if some components need to be corrected. These steps may have to be repeated in an iterative fashion if multiple sources of infeasibilities are present. The concept of an IIS is closely related to the concept of an explanation from the constraint programming literature, where an explanation of failure is defined as a minimal unsatisfiable subset of the user constraints (Hagg et al. 2006).

In general, there can be an exponential number of IISs in a model (Chakravarti 1994). Ideally, we would like an identified IIS to be of small cardinality, so that it becomes simple for the modeler to analyze the cause

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

of the infeasibility. However, the problem of finding a minimum cardinality IIS is NP-hard (Chakravarti 1994). Nonetheless, any IIS identified usually provides some useful information to the modeler. Greenberg (1992) performed an empirical study with diagnostic tests on infeasible blending models to identify an IIS. His conclusion was that isolating an IIS always gave useful information that helped in correcting the model. IIS isolation avoids the context dependence of minimum change approaches toward repairing infeasibilities. Approaches for IIS isolation for nonlinear programs are available, whereas identification of a maximal feasible subsystem of constraints for nonlinear programs is more difficult. For these reasons, we chose IIS isolation as the method of diagnosis in our work.

The idea of infeasibility isolation is old, with the first reference to an IIS being made by Carver (1921). van Loon (1981) provides ways of recognizing minimal infeasible subsets in linear programming problems. Systematic ways of identifying an IIS for linear programs were proposed by Chinneck and Dravnieks (1991). Their algorithms include the deletion filter, the elastic filter, the sensitivity filter, and an integrated filtering algorithm that utilizes all of these filtering algorithms. Several software implementations are available to obtain an IIS for linear problems. These include MINOS(IIS) (Chinneck and Saunders 1995), CPLEX (IBM 2016), XPRESS (FICO 2015), and the modelling system AIMMS (AIMMS 2015). Gurobi Optimization (2015) has algorithms implemented for isolation of an IIS even for mixed-integer linear programs.

Similar methods have been proposed for the analysis of infeasible nonlinear programs (Chinneck 1995). However, the applicability of these methods to NLPs is fairly restricted because these problems are more susceptible to numerical difficulties. Chinneck (1995) introduced the idea of an MIS (Minimal Intractable System) to denote a system that causes a particular local solver under fixed parameter values to return infeasibility, while every subset of this system is termed feasible by the same solver under the same parameter conditions. Ideally, we would like to obtain an infeasibility isolation based only on the properties of the problem under consideration, independent of the solution algorithm and its settings. Methods for isolating infeasibilities rely on the solution of a large number of very similar problems. For LPs, solutions to these problems can be obtained from solutions of a similar problem via basis updates. However, such "hot start" methods are relatively difficult to implement in practice for NLPs. For NLPs, identification of a feasible point is a definite proof of feasibility. An exhaustive (global) search is required to provide a conclusive proof of infeasibility for nonconvex problems (Horst and Tuy 1996).

The challenges for IIS identification in NLPs/MINLPs have been well recognized in the literature (Chinneck 1995, 2008). When filtering methods for IIS isolation were initially proposed in the early 90s, global optimization technology was in a nascent stage. However, the past two decades have seen a tremendous development in the area of global optimization, with important theoretical results and efficient implementations now available. With the maturity attained by global optimization solvers, it is now possible to use them in isolation of IISs for NLPs and MINLPs. The first and until recently only available general implementation for IIS isolation for LPs, MIPs, NLPs, and MINLPs was offered through the LINDO API (Atlihan and Schrage 2008). Its authors sought to minimize the effort involved in global optimization subproblems through a binary search for the set of constraints in an IIS.

In this work, we propose a new infeasibility diagnosis approach that is applicable to LPs, MIPs, NLPs, and MINLPs, convex as well as nonconvex. A key insight for the proposed approach is that feasibility-based bounds tightening techniques, also known as bounds propagation in the constraint programming literature, are often able to prove infeasibility with very little computational effort. Such algorithms are typically implemented in the form of highly efficient presolve steps in optimization software. We wish to exploit the capabilities of bounds tightening techniques to eliminate parts of the infeasible model not relevant to causing infeasibility in a very efficient manner. A reduction in the model also leads to a reduction in the number of the more computationally expensive feasibility problems that need to be solved to obtain an IIS. Furthermore, to have a definite proof that a model is indeed infeasible, we rely on the use of the exhaustive global solver BARON (Tawarmalani and Sahinidis 2004). BARON employs spatial branch and bound to solve problems to global optimality. Therefore, when BARON terminates with an infeasible status for a model, we have proof for infeasibility even when the model is nonconvex. Equally importantly, BARON has bounds tightening techniques available, making it an ideal solver to implement the proposed algorithm.

The primary contribution of this paper is the proposal for the use of a computationally efficient deletion presolve in the context of searching for an IIS. When used in isolation, this presolve is guaranteed to return an infeasible subset (IS) (Guieu and Chinneck 1999) of the original model. Existing filtering algorithms can then be applied to further reduce this IS to an IIS. The application of exact filtering algorithms reveals that the IS returned by the deletion presolve algorithm is a true IIS for 40% of a collection of 790 infeasible LPs, MIPs, NLPs, and MINLPs. When deletion presolve is used to complement four existing filtering algorithms, it speeds up these algorithms by 14 to 61 times.

RIGHTSLINK
Copyright Clearance Center

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Additional contributions of the paper include details of a related implementation in BARON and extensive computational results that compare various methodologies for IIS isolation. In comparison to the specialized LP/MIP solver Gurobi, we demonstrate that our general nonconvex NLP/MINLP implementation for IIS isolation often provides smaller cardinality infeasible subsets for LPs and MIPs. Finally, our work makes publicly available a collection of 790 infeasible optimization problems. When we began this work, one of the key challenges involved was the lack of an extensive library of infeasible problems for testing and development. While it is possible to create synthetic infeasible test instances, such instances may involve systematic biases. We compiled a library of infeasible test problems from user submissions to BARON through the NEOS server. These problems originate from applications of various users around the world. This library can be used to facilitate the testing and development of infeasibility diagnosis tools.

The remainder of the paper is organized as follows. The bounds tightening techniques that form the basis of the proposed algorithm are described in Section 2. Section 3 briefly reviews some of the algorithms previously proposed for the isolation of an IIS. Section 4 describes the proposed approach for identification of an IIS. The algorithmic aspects of our infeasibility isolation implementation in the solver BARON are discussed in Section 5. Computational results with the proposed algorithm are presented in Section 6, followed by conclusions in Section 7.

2. Bounds Tightening for Optimization Models

Optimization solvers routinely utilize some forms of presolve techniques before they attempt to solve an optimization model. The aim of presolve is to simplify a given problem by modifications such as removing redundant constraints, eliminating fixed variables, and inferring tighter bounds for variables from constraints. The simplified version of the problem is often significantly easier to solve than the original model. One of the earliest and most cited works on presolve techniques is by Brearley et al. (1975), who addressed LPs. Since then, these presolve techniques have been studied extensively. For example, the AMPL and AIMMS modelling systems have dedicated presolve algorithms that are applied to all model forms (Fourer and Gay 1994, Hunting 2011). The idea of reasoning and inference based on constraints forms the basis of constraint programming. While constraint programming is typically utilized for constraint satisfaction problems, it can also be utilized for optimization problems (Kumar 1992). Constraints are systematically propagated to eliminate inconsistent values in variable domains (Bessiere 2006) to achieve consistency.

In this section, we describe some of the important techniques utilized by BARON during its presolve. BARON performs range reduction steps at every node, employing different methods like feasibility-based bounds tightening (Ryoo and Sahinidis 1995, Shectman and Sahinidis 1998, Sahinidis 2003), optimality-based bounds tightening (Ryoo and Sahinidis 1995, 1996; Tawarmalani and Sahinidis 2004) and probing (Ryoo and Sahinidis 1995, 1996). Here, we describe the feasibility-based bounds tightening methods that we wish to exploit in the proposed deletion presolve procedure.

Feasibility-based bounds tightening methods involve utilizing problem constraints to infer tighter bounds on variables. Quite often, the bounds on variables specified by the modeler are weak, and it is possible to tighten them by drawing inferences from the problem constraints. Consider an optimization model of the following general form:

$$\left.\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(\mathbf{x}) \\ \text{s.t.} & g(\mathbf{x}) \le 0 \\ & h(\mathbf{x}) = 0 \\ & \mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u \end{array}\right\}. \tag{1}$$

The tightest possible bounds based on the constraints of the aforementioned model can be obtained by solving the following problems for each of the n variables ($k = 1, \ldots, n$):

$$\left.\begin{array}{ll} \min\limits_{\mathbf{x}\in\mathbb{R}^n} & \pm\mathbf{x}_k \\ \text{s.t.} & g(\mathbf{x}) \le 0 \\ & h(\mathbf{x}) = 0 \\ & \mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u \end{array}\right\}.$$

Since the constraints of these models are potentially nonconvex, these are expensive global optimization problems. A computationally inexpensive way of tightening bounds uses constraints one at a time to infer bounds for the variables. For example, consider a set of the following linear constraints:

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i, \quad i = 1, \ldots, k,$$
$$\mathbf{x}_l \le \mathbf{x} \le \mathbf{x}_u.$$

The following inequalities are implied by every linear constraint:

$$x_h \le \frac{1}{a_{ih}}\left(b_i - \sum_{j\ne h} \min(a_{ij}x_j^U, a_{ij}x_j^L)\right), \quad a_{ih} > 0,$$
$$x_h \ge \frac{1}{a_{ih}}\left(b_i - \sum_{j\ne h} \min(a_{ij}x_j^U, a_{ij}x_j^L)\right), \quad a_{ih} < 0.$$

If these inequalities imply tighter bounds on x_h than the ones specified by the model, the bounds can be updated.

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Similarly, bounds can be inferred for nonlinear constraints. Consider a bilinear term of the form $x_i = x_j x_k$. Then,

$$x_i \le \max\{x_j^L x_k^L, x_j^L x_k^U, x_j^U x_k^L, x_j^U x_k^U\}$$

and

$$x_i \ge \min\{x_j^L x_k^L, x_j^L x_k^U, x_j^U x_k^L, x_j^U x_k^U\}$$

represent valid bounds for the variable x_i. The bounds for x_i can similarly be updated, in case they are tighter than the ones supplied by the user. These methods, also termed *poor man's LPs* and *poor man's NLPs* (Shectman and Sahinidis 1998, Sahinidis 2003), can be iteratively utilized to obtain a significant reduction in variable domains.

BARON uses factorable reformulation techniques to construct its convex relaxations for a problem (Ryoo and Sahinidis 1996). Through factorable reformulation techniques, a problem is reformulated into atomic functional forms by introducing new variables and new equations. These atomic functional forms are simple in nature like the bilinear term for example. Therefore, under the context of factorable reformulation techniques, feasibility-based bounds tightening steps can be easily carried out even for nonlinear programs through recursive application on elementary operations. Consider the following example:

$$\begin{aligned} \min_{x,y}\ & \log(xy) \\ \text{s.t.}\ & x^2 + y^2 \le 4, \\ & xy \ge 5, \\ & 1 \le x \le 5, \\ & 1 \le y \le 5. \end{aligned}$$

A factorable reformulation will proceed by introducing a new variable for every nonlinearity occurring in the model. Thus, we replace z_1 for x^2, z_2 for y^2, z_3 for xy, and z_4 for $\log(z_3)$.

A factorable reformulation of the model is thus given by

$$\begin{aligned} \min_{x,y}\ & z_4 \\ \text{s.t.}\ & z_1 + z_2 \le 4, \\ & z_3 \ge 5, && (2) \\ & z_1 = x^2, && (3) \\ & z_2 = y^2, && (4) \\ & z_3 = xy, \\ & z_4 = \log(z_3), \\ & 1 \le x \le 5, \\ & 1 \le y \le 5. \end{aligned}$$

While a factorable reformulation for a given model is not necessarily unique and different reformulations may require different computational effort, any factorable reformulation can be used to prove infeasibility (Horst and Tuy 1996). With the aforementioned reformulation, we can now start iterating over constraints one at a time to infer tighter bounds on variables. From constraint (2), we infer that variable z_3 has bounds $[5, \infty)$. We infer that variable z_1 must be in $[1, 25]$ to satisfy constraint (3). Similarly, z_2 must be in $[1, 25]$ to satisfy constraint (4). The process of tightening bounds through constraints can be repeated iteratively to achieve further domain reduction.

If variable bounds cross during tightening, the algorithm provides an immediate proof of infeasibility. For the previous example, after iterating through all constraints repeatedly, the bounds of variable x become $[2.5, 2]$. We therefore conclude that this model is infeasible. We wish to exploit this utility of the bounds tightening steps for the proposed deletion presolve procedure for IIS detection. Before we describe the proposed method in Section 4, we present a brief review of existing IIS isolation algorithms in Section 3.

3. Review of Algorithms for IIS Isolation

Several algorithms have been proposed for the isolation of an IIS. Each algorithm proceeds differently and, on termination, provides exactly one IIS to the user. Since a given problem may have an exponential number of IISs, the choice of the algorithm and the ordering of problem constraints determine the nature of the IIS obtained. This section describes some of the important algorithms for IIS isolation. The following example will be used to aid the description of various algorithms in this section.

$$\begin{aligned} & 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 \\ & \quad - 0.0022053x_3x_5 \le 92, && (5) \\ & 80.51249 \times 0.0071317x_2x_5 + 0.0029955x_1x_2 \\ & \quad + 0.0021813x_3^2 \le 110, && (6) \\ & 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 \\ & \quad + 0.0019085x_3x_4 \le 25, && (7) \\ & 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 \\ & \quad - 0.0022053x_3x_5 \ge 0, && (8) \\ & 80.51249 \times 0.0071317x_2x_5 + 0.0029955x_1x_2 \\ & \quad + 0.0021813x_3^2 \ge 90, && (9) \\ & 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 \\ & \quad + 0.0019085x_3x_4 \ge 20, && (10) \\ & x_1 \ge 78, && (11) \\ & x_2 \ge 33, && (12) \\ & x_3 \ge 27, && (13) \\ & x_4 \ge 27, && (14) \\ & x_5 \ge 27, && (15) \\ & x_1 \le 102, && (16) \end{aligned}$$

$$x_2 \le 45, \tag{17}$$
$$x_3 \le 45, \tag{18}$$
$$x_4 \le 45, \tag{19}$$
$$x_5 \le 45. \tag{20}$$

The example is the famous Himmelblau 11 problem (Himmelblau 1972), which was adapted into its infeasible version by Chinneck (1995). Equations (6) and (9) in the previous example contain a typo. The product signs, denoted by ×, in Equations (6) and (9) were mistyped and should actually be plus signs. In this section, we will see how the isolation of an IIS can help detect this error.

3.1. Deletion Filter

The deletion filter was introduced by Chinneck and Dravnieks (1991) for the analysis of linear programs, and it was further extended by Chinneck (1995) for the analysis of nonlinear programs. It is described in Algorithm 1.

Algorithm 1 (Deletion filter)

```
Require: An infeasible set of constraints
  for All constraints in input set do
    Drop selected constraint temporarily from test set
    Test the reduced set for feasibility
    if feasible then
      return dropped constraint to test set
    else
      drop constraint permanently
    end if
  end for
Ensure: A set of constraints forming an IIS.
```

The deletion filter (Alg-DF) loops exactly once through all constraints. While looping through the constraints, each constraint is dropped one at a time. If the resulting model remains infeasible, then an IIS exists within the reduced test set of constraints. Thus, the constraint dropped temporarily can be dropped permanently from consideration. On the other hand, if the model becomes feasible once the constraint is dropped, it is important in causing the inconsistency in the model. Therefore, the constraint is reinstated and retained in the test set. Upon termination, this algorithm identifies one IIS.

The deletion filter when applied to the Himmelblau problem, leads to the following steps:

1. Remove (5): {(6)–(20)} infeasible. (5) removed permanently
2. Remove (6): {(7)–(20)} feasible. (6) retained
3. Remove (7): {(6), (8)–(20)} infeasible. (7) dropped permanently.
4. Remove (8): {(6), (9)–(20)} infeasible. (8) removed permanently

On continuing these steps by looping through all the constraints, we are left with the IIS {(6), (11), (12), (15)}.

3.2. Additive Filter

The additive method was first introduced by Tamiz et al. (1996). A simplified version of the technique is presented in Chinneck (2008). As shown in Algorithm 2, the additive filter (Alg-AF) works in the opposite way of the deletion filter. Constraints are sequentially added to a test set until the test set becomes infeasible. The last constraint added to this test set is a cause of inconsistency and a part of an IIS. This constraint is added to the member set I. The test set is then cleared of all other constraints but this last one, and constraints are sequentially added again until infeasibility is obtained again. The constraint that triggers this infeasibility is retained in the test set, and added to the member set I. The entire process is repeated until the member set I is infeasible. The member set I represents an IIS for the problem.

Algorithm 2 (Additive filter)

```
Require: An infeasible set of constraints
  Initialization:
  $C$: Ordered set of constraints
  $T$: Test set of constraints
  $I$: Members of IIS identified so far
  $T = \emptyset, I = \emptyset$
  Filter loop:
  $T = I$
  for constraint $c_i$ in C do
    $T = T \cup \{c_i\}$
    if $T$ infeasible then
      $I = I \cup \{c_i\}$
      if $I$ feasible then
        Repeat the filter loop from the beginning
      else
        Exit
      end if
    end if
  end for
Ensure: A set of constraints forming an IIS in $I$.
```

The additive filter when applied to the Himmelblau problem, proceeds as follows: {(5)}, {(5), (6)}, ..., {(5)–(14)} are all feasible. However, {(5)–(15)} is infeasible. Therefore, (15) is identified as a part of an IIS. Now, the tests are repeated as {(15)},{(15),(5)},{(15),(5)–(6)} until the following infeasible set {(15), (5)–(12)} is obtained. Thus, (12) is identified as a part of an IIS. The algorithm proceeds to identify the IIS {(6), (11), (12), (15)}.

3.3. Hybrid Methods

The methods presented so far can be adapted and combined in multiple ways to lead to more efficient and faster methods. For instance, both the additive and deletion filtering method can be improved by treating multiple constraints in groups (Guieu and Chinneck 1999). In particular, in deletion filtering, one can drop

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

k constraints at a time. If dropping the k constraints preserves infeasibility, it saves $k-1$ feasibility tests. Guieu and Chinneck (1999) consider several different schemes for grouping constraints. Similar ideas can be applied to additive filtering methods. Instead of adding one constraint at a time to the test set, one can add k constraints. If adding k constraints preserves feasibility, this saves $k-1$ feasibility tests.

Guieu and Chinneck (1999) also present an improved version of the additive filter called the dynamic reordering additive method. The method is based on the insight that, if a subproblem is feasible, we can evaluate all the remaining constraints at the current feasible point. Constraints that satisfy the current solution can be directly added to the set T. This avoids a significant number of feasibility tests.

The additive and deletion methods can be combined together (Guieu and Chinneck 1999) to provide a faster filtering technique called the additive-deletion filter (Alg-ADF). The additive method is employed by adding constraints one at a time in a test set until infeasibility in that test set is triggered. Deletion filtering can then be applied on this reduced test set. If the ordering of constraints is such that infeasibility is triggered with a very small number of constraints in the additive method, the deletion filter has to solve a smaller problem, thus requiring fewer feasibility tests and leading to a speedup in isolating an IIS.

A binary grouping strategy was employed in the QUICKXPLAIN algorithm for computing relaxations for arbitrary constraint programming and satisfiability problems (Junker 2004). This algorithm is based on the insight that, by formulating subproblems by splitting the original model in half, significant computational benefits can be derived. For example, if, in the deletion filter, half the constraints in an infeasible model are relaxed, and the model status still remains infeasible, then these constraints can be eliminated from consideration through the solution of a single feasibility problem. Atlihan and Schrage (2008) have described two algorithms, depth first binary search filter (Alg-DFBS) and a generalized binary search filter (Alg-GBS) that utilize a similar divide-and-conquer strategy for isolating an IIS for infeasible problems. They show with empirical studies that the grouping strategies used in Alg-DFBS and Alg-GBS can be beneficial for nonlinear programs. These grouping techniques reduce the overall number of feasibility tests employed.

4. Deletion Presolve Algorithm

We propose a deletion presolve algorithm that can speed up the IIS isolation algorithms of the previous section. Chinneck (2008) indicates that presolve can be used to detect infeasibility and describes special cases when presolve can diagnose the infeasibility as well. However, when presolve leads to a long sequence of reductions leading to infeasibility, analyzing this sequence and isolating an IIS is difficult. In our work, we wish to utilize presolve as an efficient infeasibility test. The proposed approach is based on the use of bounds tightening techniques to prove infeasibility. In our experience with practical industrial models that motivated this work, we observed that these techniques are often able to prove infeasibility very quickly. The techniques exemplified in Section 2 are computationally very inexpensive. We wish to leverage the utility of this efficient bounds tightening mechanism to rapidly eliminate a large number of constraints. We can then apply any filtering algorithm on the reduced constraint set to obtain an IIS.

Algorithm 3 describes the proposed method. We refer to the deletion presolve procedure as DP. This algorithm maintains an IS through its iterations and is therefore guaranteed to return an IS. Any of the filtering algorithms of Section 3 can be applied to this IS to further obtain an IIS. DP can efficiently eliminate constraints from the model. Thus, the exact filtering algorithms would have a smaller, and thus easier to analyze, infeasible model.

Algorithm 3 (Deletion presolve)

```
Require: An infeasible set of constraints
  for All constraints in input set do
    Drop the selected constraint temporarily
      from test set
    Apply bounds tightening techniques on the
      reduced set
    if bounds tightening proves infeasibility then
      drop constraint permanently
    else
      return dropped constraint to test set
    end if
    Reset variable bounds to original bounds
  end for
Ensure: A set of constraints forming an IS.
```

To illustrate the main idea, consider the Himmelblau 11 example as described in Section 3. We describe what happens when we apply the bounds tightening based deletion presolve algorithm to this problem.

1. Remove (5): {(6)–(20)} proven infeasible by bounds tightening. (5) removed permanently
2. Remove (6): {(7)–(20)} cannot be proven infeasible by bounds tightening. (6) retained
3. Remove (7): {(6), (8)–(20)} proven infeasible by bounds tightening. (7) removed permanently
4. Remove (8): {(6), (9)–(20)} proven infeasible by bounds tightening. (8) removed permanently

If we complete looping through all constraints, we are left with the set {(6), (11), (12), (15)}. While we still need to prove that this is an IIS by applying any of the filtering methods, the benefits of the proposed deletion presolve become apparent through this example.

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

We have already achieved a substantial reduction in the problem by eliminating twelve out of the sixteen constraints. This reduction was computationally inexpensive. The deletion filter, when applied to this system, is applied to a much smaller set of constraints, and returns quickly as well, proving that this constraint set is indeed an IIS. In the computational results section, we will demonstrate that, for a large number of problems, this deletion presolve step is sufficient to reduce a problem to an IIS.

The variable bounds are reset to the original bounds after every bounds tightening iteration step. This is necessary since bounds inferred at an iteration of the deletion presolve loop may no longer be valid after dropping some constraints. In general, a reduction in the set of constraints is not guaranteed by deletion presolve. There could exist models for which bounds tightening fails to prove infeasibility during any iteration of the algorithm. For such models, no elimination of constraints are achieved. However, bounds tightening is computationally inexpensive. Thus, in cases where there is no reduction in the problem's constraint set through deletion presolve, the deterioration in performance over the pure filtering method is expected to be only marginal.

5. Computational Implementation in BARON

We implemented the proposed deletion presolve in BARON, along with Alg-DF, Alg-AF, Alg-ADF, and Alg-DFBS. The user has the option to choose between a pure filtering algorithm and the filtering algorithm combined with deletion presolve. In general, there can be potentially exponentially many IISs in a given model, with the isolated IIS being a function of the choice of the algorithm employed and the ordering of the problem constraints. Having a basket of multiple algorithmic implementations provides the user with flexibility in isolating different IISs, where an IIS isolated from one algorithm could potentially be easier to analyze than the IISs obtained from other algorithms. This capability also allows us to test and compare the impact of deletion presolve on four different algorithms for IIS isolation. In our implementation, we also provide the user with the capability of randomizing the order of the problem constraints by specifying a seed for the random number generator. This facility provides additional flexibility to the user in utilizing multiple orderings for problem constraints to isolate different IISs for an infeasible model. It also makes it possible for us to address the question whether multiple IISs indeed appear for the problems in our test set.

Our implementation treats equality constraints ($h(x) = 0$) as a combination of a less-than or equal-to ($h(x) \leq 0$) and a greater-than or equal-to ($h(x) \geq 0$) constraint. This allows the implementation to specifically return whether the $\leq$ component or the $\geq$ component of an equality constraint is a part of the IIS. This provides the user additional information as to whether the right-hand side of an equality constraint must be increased or decreased to achieve feasibility.

In the case of multiple IISs in a model, the ordering of constraints determines which IIS is obtained. Since rows with fewer variables are easier to interpret, we implemented heuristics to result in IISs that contain as many simple rows as possible, preferably only with simple variable bounds. In addition to first searching for blatant bound infeasibilities, simple IISs are sought via ordering heuristics that are tailored to the filtering algorithm used (Chinneck 1997). The deletion filter returns that IIS for which the first member is tested last (Chinneck 2008). Therefore, we order constraints so that row constraints occur before column bounds, allowing for maximum elimination of row constraints. Further, the row constraints are ordered so as to have constraints with higher number of variables occurring first, allowing for elimination of the more "complicated" constraints at the initial stage of the algorithm. Similarly, for the additive filter, the IIS returned is one whose last member is tested first (Chinneck 2008). To apply a similar heuristic for the additive filter, all variable bounds are contained in the testing set at all times, and only the row constraints are added one by one to test for an IIS. The heuristics will return an infeasible set with row constraints in an IIS and all column bounds in the model. The excess column bounds are then eliminated via deletion filtering to isolate an IIS. Constraints are ordered so as to have constraints with fewer variables occurring first, allowing for isolation of an IIS with fewer variables. The additive-deletion filter proceeds by introducing all column bounds initially in the test set followed by the row constraints one at a time. This directs the search toward an IIS with fewer row constraints and more column bounds.

The application of the deletion presolve algorithm affects which IISs might be obtained for an infeasible model. If constraints that are part of some IISs are eliminated by deletion presolve, these IISs cannot be obtained by the subsequent application of any filtering algorithm. Thus, while we expect that deletion presolve will lead to faster IIS isolation when followed by filtering algorithms, we also provide the user with a choice of using the pure filtering algorithm without deletion presolve. This may allow for different IISs to be isolated for some models that would not be possible after employing deletion presolve.

For MINLP models, it is intuitively easier to check for the correctness of general integrality restrictions and integer bounds in a model from the understanding of the physical system. If the user is convinced of the correctness of the integer variables in the model, this information can be conveyed to the IIS detection module by

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

setting a corresponding user option. When this option is selected, our implementation assumes the correctness of general integer variable definitions, reducing the number of feasibility problems to be solved. For binary variables, we assume that errors in definitions of binary variables are rarer and are significantly easier to correct. Our implementation assumes the correctness of binary variable definitions irrespective of user-specified options for general integer variables. Therefore, for MINLP models, any IIS returned by the IIS detection module must be analyzed with respect to the corresponding binary variables present in the constraints identified as a part of the IIS.

If a time out occurs before an IIS is isolated, the implementation returns the infeasible subset obtained by DP. For the specific case of the additive filter and the additive-deletion filter, the IS that triggers infeasibility is stored. If a time out occurs before IIS isolation is complete, this infeasible subset is then returned to the user. While this IS may be reducible, it is still likely to contain fewer constraints than the original model. For large and complicated models, the first few iterations for the deletion filter would correspondingly solve large problems, making time outs more likely. In contrast, the additive filter and the additive-deletion filter algorithms will start with an empty set of constraints and sequentially add constraints until infeasibility is triggered. In case an IIS is contained within the first few constraints in the model, it is likely identified with relatively minor computational effort with the latter algorithms. Thus, for large models where time outs are expected, the additive filter and the additive-deletion filter, both with deletion presolve, should be favored.

A practical difficulty in global optimization is the issue of "missing bounds." For problems where some nonlinear terms are not bounded, all global optimization algorithms run into difficulties in trying to construct relaxations of unbounded terms. BARON, in particular, attempts to solve the problem despite the missing bounds and, if it is unable to make progress in the face of missing bounds, it automatically sets missing bounds to presumably sufficiently large values. In such cases, BARON does not make optimality or infeasibility claims. Missing bounds is an even bigger challenge for IIS isolation, since the subproblems created during all algorithms for IIS isolation systematically relax variable and constraint bounds, thus making it less likely for bounds tightening to infer bounds on nonlinear terms. In the spirit of "safety bounds" technique of Guieu and Chinneck (1999), in case bounds are automatically set for subproblems and feasibility cannot be proved, we make conservative decisions in our implementation. For instance, when missing bounds have been automatically set in a subproblem for the deletion filter, even when no feasible solution is returned by BARON, we assume the constraint dropped at this particular iteration of the deletion filter to be a part of the IIS. We make corresponding decisions for all the other filters from the point of view of robustness. In this way, we guarantee that the output returned by the algorithm for an infeasible model is indeed an infeasible subset (IS).

6. Computational Results

As part of our work, we collected a set of 790 infeasible problems. These problems were compiled from the infeasible models submitted over a two-year period to the BARON solver through the NEOS server. Over this period, users from 44 countries submitted to BARON a total of 25,136 problems, out of which the infeasible problems were mined. As a result, our test set contains a diverse collection of problems of different types and from different application areas. To test for feasibility in models submitted via the NEOS server, the constraint satisfaction tolerance and integrality tolerance were both set to 10^{-5}. This test library is available at http://minlp.com/nlp-and-minlp-test-problems. In preparing the library, the test problems were sanitized and all user-related information was eliminated by running the models through the GAMS Convert facility (GAMS Development Corporation 2015).

Statistics on the test set are provided in Table 1. As seen, over half of the problems are MINLPs, while almost one quarter of the problems are linear (LPs and MIPs). Regarding problem dimensions, we see that they average in the thousands of constraints and variables, with the largest problems containing more than 102 thousand constraints and variables. The MIPs

Table 1. Test Set Statistics

Problems		Constraints			Variables			Integer variables			Size		
Type	Number	Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.
LP	43	4	64,017	7,227	2	32,217	3,498	—	—	—	6	139,040	13,026
NLP	190	2	75,579	1,723	4	50,325	1,740	—	—	—	2	151,158	4,559
MIP	140	4	13,521	1,144	21	102,830	3,561	3	33,600	1,530	8	107,525	4,123
MINLP	417	2	102,330	1,688	2	60,000	1,575	1	40,000	436	2	51,101	2,562
Total	790	2	102,330	1,901	2	102,830	2,071	1	40,000	711*	2	151,158	3,888

*Averaged only over the set of MIPs and MINLPs.

RIGHTSLINK

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

contain more than a thousand integer variables on average, while the MINLPs average more than 400 integer variables. Finally, the column labeled "size" denotes the number of row and column bounds in the problem that must be analyzed for infeasibility. Size equals the number of inequalities plus twice the number of equalities plus all finite variable bound in the model. This quantity determines the number of subproblems that may have to be solved by filtering algorithms for IIS detection. This number averages close to 4,000, but there are only 20 problems for which this count exceeds 20,000. Clearly, many models contain a large number of unbounded variables that do not contribute to this statistic.

The computations were designed to investigate the following questions:

- Do multiple IISs exist for the given set of infeasible problems?
- What is the cardinality of the IISs encountered?
- Do the number and cardinality of IISs depend on problem size?
- Which filtering algorithm is best?
- Does the deletion presolve algorithm help?
- How does the proposed approach compare against established technology?

All the runs reported next were subjected to a time limit of 500 seconds on a 64-bit Intel Xeon X5650 2.66 GHz processor running CentOS release 7.

6.1. Existence of Multiple IISs for Infeasible Models

Theoretically, there can exist exponentially many IISs in an infeasible model (Chakravarti 1994). Multiple IISs in an infeasible model can indeed be seen in practice, as demonstrated by Table 2. The table summarizes the number of IISs found and the corresponding number of problems for which multiple IIS were isolated. For the tabulation of these statistics, we considered only those IISs for which we could verify infeasibility and irreducibility. We ignored any ISs for which irreducibility could not be guaranteed as a result of either "missing bounds" or timeouts for the IIS identification routine or the verification routine. We ran the four implemented algorithms with and without deletion presolve. Further, three different problem orderings were utilized and hence we can only find a maximum of 24 different IISs in a model. We could potentially isolate more IISs by using the filtering algorithms with different orderings. For 590 problems, i.e., for more than 73% of the test set, more than one verified IIS was obtained by the basket of algorithms that we have implemented. The possibility that even more IISs may exist for these problems cannot be ruled out.

Table 2. Multiplicity of IISs Across Problems

No. of IISs found	No. of test problems
1	118
2	95
3	105
4	115
5	107
6	68
7	44
8	25
9	19
10	10
11	2

Table 3. Number of IISs Found Uniquely by Each Algorithm

Algorithm	Unique IISs
DP + Alg-DF	222
Alg-DF	83
DP + Alg-AF	39
Alg-AF	76
DP + Alg-ADF	46
Alg-ADF	240
DP + Alg-DFBS	200
Alg-DFBS	243
Total	1,149

Table 3 describes the number of IISs identified uniquely by a single algorithmic setting of our implementation. Across the test set, a total of 2,836 unique and verified IISs were isolated. Over 40% of these verified IISs were identified only by a single algorithm. These IISs could not be identified by any other filtering algorithm over the three different problem orderings that were used for testing. These results demonstrate the importance of having a basket of algorithms implemented for obtaining different IISs of an infeasible model.

6.2. Cardinality of Identified IISs

The average cardinality of IISs isolated as a percentage of the problem size across various algorithms and problem types is described in Table 4. On average, the IISs identified contain less than a third of the row constraints and about a third of the column bounds present in the original model. There are only minor variations in IIS cardinalities across the different algorithms implemented. The cardinalities of the IISs obtained across various problem types are fairly consistent over all the algorithms. It is important to note that the average numbers of IIS cardinalities are biased based on problem size. For smaller problems, even a small IIS identified corresponds to a higher percentage based on problem size. Similarly, some of the larger problems time out within the prescribed time limit. Figure 1 describes the fraction of rows contained in an IIS and the corresponding number of problems in the test library. For 371 models in the library, this number is less than 5%. These results demonstrate that the IIS

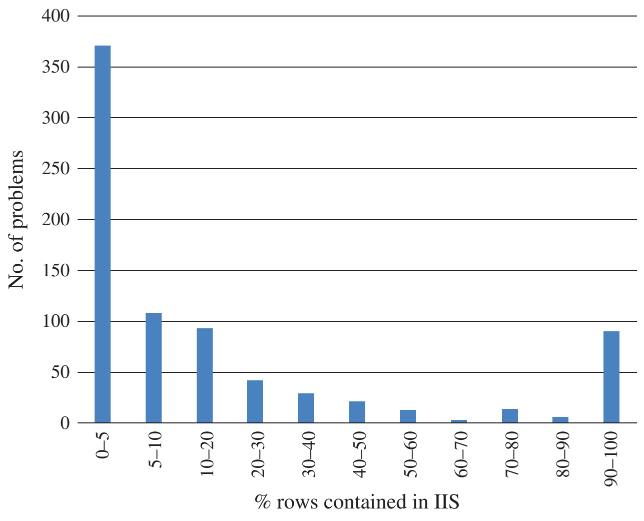
RIGHTSLINK
Copyright Clearance Center

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Table 4. Average Row and Column Percentages in an IIS for Each Algorithm

	DP + Alg-DF		DP + Alg-AF		DP + Alg-ADF		DP + Alg-DFBS	
Problem type	% rows	% cols	% rows	% cols	% rows	% cols	% rows	% cols
LP	32	32	31	32	34	32	24	23
NLP	22	57	31	57	35	55	34	53
MIP	25	19	25	21	26	19	22	35
MINLP	22	27	28	27	26	29	28	28
Overall average	23	33	28	34	28	34	28	34

Figure 1. (Color online) Percentage Rows Contained in an IIS Obtained by DP + Alg-DF



obtained is often only a tiny subset of the original problem, and therefore much easier to analyze. The number of models for which this fraction is between 90%–100% is somewhat high because this includes large problems which time out as well as some of the smaller problems for which the number of rows in an IIS is a high fraction of the original rows.

6.3. Comparisons Across Filtering Algorithms

Comparing the sizes of IISs returned by the different algorithms from Table 4, we see that the performance of the algorithms is very similar. Computational times with the algorithms are tabulated in Table 5. The performance of deletion filter, additive filter and the additive-deletion filter are again found to be very similar. The DFBS filter seems to outperform the other filters on LPs. However, the performance of DFBS deteriorates over NLPs and MINLPs. The DFBS algorithm also times out the most number of times. Only 29 problems timed out with the deletion filter, 39 problems with the additive filter, 29 problems with the additive-deletion filter, whereas 151 problems timed out with the depth-first binary search filter. The DFBS algorithm seems to outperform the other algorithms on LPs, however the other algorithms fare better on the more challenging NLPs and MINLPs.

6.4. Impact of Deletion Presolve

To characterize the impact of deletion presolve on IIS isolation, we define presolve efficiency as the ratio of the row constraints and column bounds eliminated by deletion presolve and the row constraints and column bounds eliminated by deletion presolve followed by filtering in obtaining an IIS. Thus, the presolve efficiency can be thought of as the gap that is closed by the deletion presolve algorithm in reducing the problem to an IIS. Table 6 presents the impact of deletion presolve across all the algorithms. Speedup is defined as the ratio of the computational time required to obtain an IIS with a pure filtering algorithm and the computational time required to obtain an IIS with deletion presolve followed by the filtering algorithm. For almost 40% of the problems in the test, the presolve efficiency is 1, indicating that deletion presolve has reduced the problem to an IIS, with no further reduction needed by the subsequent filtering algorithm. High presolve efficiency leads to a substantial speedup

Table 5. Average Computational Time to Obtain an IIS

Problem type	DP + Alg-DF Time(s)	DP + Alg-AF Time(s)	DP + Alg-ADF Time(s)	DP + Alg-DFBS Time(s)
LP	138	137	133	70
NLP	129	136	140	162
MIP	106	122	116	120
MINLP	102	111	114	170
Overall average	111	121	122	154

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

Table 6. Impact of Deletion Presolve Across All Algorithms

	Alg-DF	Alg-AF	Alg-ADF	Alg-DFBS
No. of problems with presolve efficiency = 1	317	311	319	287
Number of problems with presolve efficiency = 0	324	318	327	249
Average presolve efficiency for remaining problems	0.88	0.87	0.86	0.84
Average speedup	14	47	61	39

in computational time. These speedups are highest for the additive-deletion filter. Even for the deletion filter, where the speedup is at a minimum, the speedup is still an impressive fourteen fold. While the presolve efficiencies are zero for a large number of problems, since the deletion presolve is computationally highly efficient, it only leads to a marginal performance deterioration on these problems. Consequently, on average, impressive speedups are obtained over the entire test. These results demonstrate the value of the deletion presolve algorithm in obtaining an IIS.

6.5. Performance Comparisons with Gurobi

XPRESS has algorithms implemented for IIS isolation only for LPs, while Gurobi has implementations for IIS isolation for both LPs and MIPs. CPLEX's conflict refiner offers a more general tool for analyzing infeasible MIPs as well as quadratic programs. LINDO API offers an implementation utilizing the DFBS and the GBS algorithms for IIS isolation for LPs, MIPs and even nonconvex NLPs and MINLPs. We have implemented the DFBS in our own IIS isolation module. In our testing, the DFBS timed out most often in comparison to other filtering algorithms. We do not expect the GBS to be efficient because it requires the solution of multiple global optimization problems to optimality. We compare our implementation with Gurobi as a representative commercial tool. We performed this comparison on two data sets. First, we used the set of LPs and MIPs identified from the infeasible models submitted to BARON via the NEOS server as previously described. This test set has 183 LP and MIP instances. We refer to this set as Test Set A. We also compared results on a test set used by Guieu and Chinneck (1999) for their testing of problems from www.netlib.org/lp/infeas. This test set contains 50 LPs and MIPs and is referred to as Test Set B.

Gurobi has two algorithms implemented for IIS isolation that can be selected via a user-specified option. If an algorithm is not specified, Gurobi chooses a method automatically. However, the Gurobi manual does not reveal what algorithms are implemented in the software. For this reason, we chose the default Gurobi option that selects between the two algorithms automatically. For these tests, BARON was run with the deletion presolve only, and is therefore guaranteed to return an IS. As observed from results in Section 6.4, this output is indeed an IIS for many problems.

Gurobi is a specialized solver for LPs and MIPs. When dealing with infeasible LPs or MIPs, linearity can be exploited via numerous techniques, including sensitivity filter (Chinneck and Dravnieks 1991), elastic filter (Chinneck and Dravnieks 1991), pivoting methods (van Loon 1981), reciprocal filter (Chinneck 1997), and interior point methods (Greenberg 1996). These methods can be combined to form hybrid methods specific to LPs and MIPs to provide very fast and efficient approaches that exploit linearity. Since the implementation for IIS isolation in BARON is general and applicable to all problem types (LPs, NLPs, MIPs, and MINLPs), we do not utilize any of these fast methods applicable specifically for LPs/MIPs. We therefore expected Gurobi to perform better than BARON in terms of speed for this class of problems. This is indeed the case as seen in Figures 2 and 3. These two figures present results across problem instances in Test Sets A and B, respectively. Each of the two figures shows two ratios: the computational time ratio (ratio

Figure 2. (Color online) Comparison Between Gurobi and Deletion Presolve on Test Set A

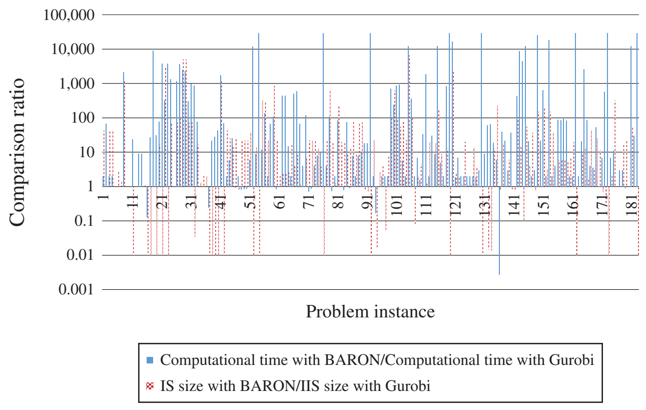
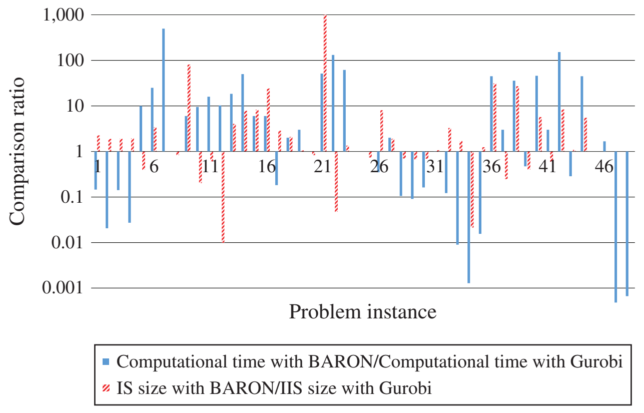


Figure 3. (Color online) Comparison Between Gurobi and Deletion Presolve on Test Set B



RIGHTSLINK Copyright Clearance Center

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.

of computational time with BARON to the computational time with Gurobi) and the size ratio (ratio of IS size with BARON to the IIS size with Gurobi). Gurobi clearly outperforms BARON on Test Set A in terms of computational speed. The results are more balanced on Test Set B, for many problems of which BARON is much faster than Gurobi. In terms of IS/IIS size, there is no clear winner between Gurobi and BARON. We can think of the size of an IS/IIS as an indicator of its quality because smaller infeasibility sets are easier to analyze and are thus of better quality than larger ones. There are many problems for which BARON finds better quality infeasibility sets than Gurobi and vice versa. Thus, we see that BARON's deletion presolve is competitive with Gurobi despite the fact that it does not utilize any specialized methods to exploit linearity.

7. Conclusions

We have developed a systematic approach for the treatment of infeasible linear, nonlinear, integer linear, and mixed-integer nonlinear programs. Our approach to infeasibility analysis is based on the isolation of irreducible infeasible sets. We have proposed a deletion presolve that allows for rapid elimination of problem constraints and bounds not present in an IIS. This algorithm utilizes bounds tightening techniques to efficiently eliminate candidate row and column bounds before a filtering algorithm is used to isolate an IIS. We implemented four filtering algorithms: deletion filter, additive filter, additive-deletion filter, and depth-first binary search filter. Computational tests on a test set of 790 infeasible models demonstrate that the deletion presolve algorithm results in several orders of magnitude speedups of exact filtering algorithms in the process of finding an IIS. Deletion presolve is beneficial for all four filtering algorithms. These speedups are consistent across problem sizes and across problem types (LPs, NLPs, MIPs, and MINLPs). Our implementation in BARON is competitive with Gurobi's IIS isolation implementation for LPs/MIPs, in the sense that BARON finds smaller cardinality ISs for many problems. The basket of algorithms implemented along with facilities for randomizing the order of problem constraints allows for great flexibility in isolating distinct IISs to aid the diagnosis of infeasible models.

Acknowledgments

The authors thank Air Liquide for providing partial financial support and an industrial model that motivated the work on this topic. They also thank three anonymous referees for their thorough comments that helped considerably streamline the presentation of this work.

References

AIMMS (2015) AIMMS modeling language. http://www.aimms.com/.

Amaldi E, Pfetsch ME, Trotter LE Jr (1999) Some structural and algorithmic properties of the maximum feasible subsystem problem. Cornuejols G, Burkard RE, Woeginger GJ, eds. *Integer Programming and Combinatorial Optimization* (Springer-Verlag, Berlin), 45–59.

Atlihan MK, Schrage L (2008) Generalized filtering algorithms for infeasibility analysis. *Comput. Oper. Res.* 35(5):1446–1464.

Bessiere C (2006) Constraint propagation. Rossi F, van Beek P, Walsh T, eds. *Handbook of Constraint Programming*, Chap. 2 (Elsevier, Amsterdam), 29–83.

Biegler LT, Zavala VM (2009) Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization. *Comput. Chem. Engrg.* 33(3):575–582.

Bixby RE (2002) Solving real-world linear programs: A decade and more of progress. *Oper. Res.* 50(1):3–15.

Bixby RE (2012) A brief history of linear and mixed-integer programming computation. Grotschel M, ed. *ISMP 2012, Berlin*, 107–121.

Brearley AL, Mitra G, Williams HP (1975) Analysis of mathematical programming problems prior to applying the simplex algorithm. *Math. Programming* 8(1):54–83.

Carver WB (1921) Systems of linear inequalities. *Ann. Math.* 23(2): 212–220.

Chakravarti N (1994) Some results concerning post-infeasibility analysis. *Eur. J. Oper. Res.* 73(1):139–143.

Chinneck JW (1995) Analyzing infeasible nonlinear programs. *Comput. Optim. Appl.* 4(2):167–179.

Chinneck JW (1997) Finding a useful subset of constraints for analysis in an infeasible linear program. *INFORMS J. Comput.* 9(2): 164–174.

Chinneck JW (2008) *Feasibility and Infeasibility in Optimization* (Springer, New York).

Chinneck JW, Dravnieks EW (1991) Locating minimal infeasible constraint sets in linear programs. *ORSA J. Comput.* 3(2):157–168.

Chinneck JW, Saunders MA (1995) MINOS(IIS) Version 4.2: Analyzing infeasibilities in linear programs. *Eur. J. Oper. Res.* 81(1): 217–218.

Czyzyk J, Mesnier M, Moré J (1998) The NEOS server. *IEEE Comput. Sci. Engrg.* 5(3):68–75.

FICO (2015) FICO® Xpress Optimization Suite. Accessed July 6, 2017, http://www.fico.com/en/products/fico-xpress-optimization-suite/.

Fourer R, Gay DM (1994) Experience with a primal presolve algorithm. Hager WW, Hearn DW, Pardalos PM, eds. *Large Scale Optimization: State of the Art* (Springer, Boston), 135–154.

GAMS Development Corporation (2015) GAMS/CONVERT Solver Documentation. Accessed July 6, 2017, http://www.gams.com/dd/docs/solvers/convert/index.html.

Gill PhE, Murray W, Saunders MA (2002) SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. Optim.* 12(4): 979–1006.

Greenberg HJ (1992) An empirical analysis of infeasiblity diagnosis for instances of linear programming blending models. *IMA J. Math. Bus. Indust.* 4(2):163–210.

Greenberg HJ (1996) Consistency, redundancy, and implied equalities in linear systems. *Ann. Math. Artificial Intelligence* 17(1): 37–83.

Guieu O, Chinneck JW (1999) Analyzing infeasible mixed-integer and integer linear programs. *INFORMS J. Comput.* 11(1):63–77.

Gurobi Optimization (2015) GUROBI Optimizer 6.5. Accessed July 6, 2017, http://www.gurobi.com/.

Hagg A, Junker U, O'Sullivan B (2006) A survey of explanation techniques for configurators. Sinz C, Haag A, eds. *Proc. ECAI-2006 Workshop on Configuration, Riva del Garda, Italy*, 44.

Himmelblau DM (1972) *Applied Nonlinear Programming* (McGraw-Hill, New York).

Horst R, Tuy H (1996) *Global Optimization: Deterministic Approaches*, Third ed. (Springer-Verlag, Berlin).

RIGHTSLINK

Hunting M (2011) A nonlinear presolve algorithm in AIMMS. An AIMMS White Paper, AIMMS, Haarlem, Netherlands.

IBM (2016) CPLEX Optimizer. Accessed July 6, 2017, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Junker U (2004) QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. *AAAI'04: Proc. 19th National Conf. Artificial Intelligence, San Jose, CA*, 167–172.

Kumar V (1992) Algorithms for constraint-satisfaction problems: A survey. *AI Magazine* 13(1):32–44.

Murty KG, Kabadi SN, Chandrasekaran R (2000) Infeasibility analysis for linear systems: A survey. *Arabian J. Sci. Engrg.* 25(1, Part C): 3–18.

Ryoo HS, Sahinidis NV (1995) Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Comput. Chem. Engrg.* 19(5):551–566.

Ryoo HS, Sahinidis NV (1996) A branch-and-reduce approach to global optimization. *J. Global Optim.* 8(2):107–139.

Sahinidis NV (2003) Global optimization and constraint satisfaction: The branch-and-reduce approach. Bliek C, Jermann C, Neumaier A, eds. *Global Optimization and Constraint Satisfaction*, Lecture Notes in Computer Science, Vol. 2861 (Springer, Berlin), 1–16.

Shectman JP, Sahinidis NV (1998) A finite algorithm for global minimization of separable concave programs. *J. Global Optim.* 12(1): 1–36.

Tamiz M, Mardle SJ, Jones DF (1996) Detecting IIS in infeasible linear programmes using techniques from goal programming. *Comput. Oper. Res.* 23(2):113–119.

Tawarmalani M, Sahinidis NV (2004) Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Math. Programming* 99(3):563–591.

van Loon J (1981) Irreducibly inconsistent systems of linear inequalities. *Eur. J. Oper. Res.* 8(3):283–288.

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

Downloaded from informs.org by [155.69.24.171] on 15 October 2017, at 22:15 . For personal use only, all rights reserved.