INFORMS Journal on Computing
Vol. 11, No. 1, Winter 1999

0899-1499/99/1101-0063 $05.00

Analyzing Infeasible Mixed-Integer and Integer Linear Programs

OLIVIER GUIEU AND JOHN W. CHINNECK / *Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario K1S 5B6, Canada, Email: chinneck@sce.carleton.ca*

(Received: August 1996; revised: November 1997, August 1998; accepted September 1998)

Algorithms and computer-based tools for analyzing infeasible linear and nonlinear programs have been developed in recent years, but few such tools exist for infeasible mixed-integer or integer linear programs. One approach that has proven especially useful for infeasible linear programs is the isolation of an *Irreducible Infeasible Set of constraints* (IIS), a subset of the constraints defining the overall linear program that is itself infeasible, but for which any proper subset is feasible. Isolating an IIS from the larger model speeds the diagnosis and repair of the model by focussing the analytic effort. This paper describes and tests algorithms for finding small infeasible sets in infeasible mixed-integer and integer linear programs; where possible these small sets are IISs.

Mixed-integer and integer linear programs (here collectively referred to as MILPs) are much harder to solve than ordinary linear programs (LPs) because of the inherent combinatorial nature of the solution approaches necessitated by the integer variables. Infeasible MILPs are even more difficult to analyze because they usually require numerous solutions of variations of the original model. In addition, when a branch-and-bound solution procedure finds the original model infeasible, little useful information (such as constraint sensitivity in linear programs) is initially available to guide the analysis of the infeasibility. Some form of automated assistance in analyzing infeasible MILPs is needed, especially as models grow in size in step with increases in computing power.

Few useful tools are currently available. Savelsbergh[17] describes a bound-tightening presolve procedure for MILPs (implemented in the MINTO solver[15]) that may detect infeasibility as a side effect of the reformulation. Backtracking the complete set of reformulation operations may then isolate a set of constraints and integer restrictions that cause the infeasibility. However, there is no guarantee that the presolver will detect infeasibility, or that the backtrack of the reformulation operations will provide any useful information. Greenberg also uses related bound-tightening methods for dealing with binary variables in the *reduce* command of his ANALYZE software.[10]

On the other hand, effective methods for assisting in the analysis of infeasible linear and nonlinear programs have been developed in recent years (e.g., [2–7]). These new methods concentrate on isolating an *Irreducible Infeasible Subset of constraints* (IIS) from among the larger set of constraints (both rows and column bounds) defining the model. The constraints in an IIS define an infeasible set, but have the property that any proper subset of the IIS constraints is feasible; the IIS is minimal in that sense. The isolation of an IIS accelerates the analysis and repair of the model infeasibility by focussing the analytic effort to a small portion of the entire model. In this article (based on work by Guieu[11]), we extend the general ideas used in isolating IISs in LPs and NLPs to the case of MILPs.

The LP IIS isolation methods can be applied directly only when the initial LP relaxation of the MILP is LP-infeasible. In that case, the IIS isolated by the LP methods applied to the initial LP relaxation is also a valid IIS for the MILP. We concentrate here on the more difficult case of analyzing infeasibility in MILPs for which the initial LP relaxation is feasible.

Direct porting of the LP IIS isolation methods to MILPs is difficult for several reasons. First, some of the LP IIS isolation methods rely on properties specific to LPs such as sensitivity analysis, various pivoting methods, and theorems of the alternative. Second, most of the LP methods rely on repeated solutions of slight variations of the model, with the important output being the feasibility status of the model variant. However, determining that a MILP is infeasible involves the full expansion of a branch-and-bound tree, with infeasibility recognized only when all of the leaf nodes prove infeasible. In contrast, feasibility is easily recognized as soon as any leaf node proves feasible. Third, as discussed in Section 1, MILP solution methods may fail to terminate, which means that the feasibility status of a model variant cannot be determined.

To avoid nontermination, an upper limit can be imposed on the computational resources expended on a particular model variant (e.g., an upper limit on the number of branch-and-bound nodes developed), which limits the algorithms to the identification of an *Infeasible Subset* (IS) rather than an IIS. In the remainder of this article, we develop methods for isolating small ISs in MILPs while hoping to identify IISs as often as possible. The methods are primarily based on algorithms originally developed for LPs (reviewed in Section 2), but take into account the difficulties mentioned above. Empirical results are presented in Section 6.

Subject classifications: Optimization
Other key words: Mixed-integer and integer linear programming, infeasibility

1. MILP Solution Methods and Properties

A general MILP can be thought of as an ordinary LP with a set of added integer restrictions on some or all of the variables. The constraints can be divided into three distinct subsets:

- *LC*: the set of linear constraints (or rows),
- *BD*: the set of variable bounds (upper and lower bounds, if any), and
- *IR*: the set of integer restrictions; variables in *IR* are restricted to taking on integer values while variables not in *IR* are real-valued. Some integer variables may be further restricted to be binary, having a solution restricted to the set {0, 1}.

We denote the presence of an integer restriction on a variable x_i by $[x_i]$. Binary variables are treated as integer variables with a lower bound of 0 and an upper bound of 1.

The entire MILP consists of a linear objective function plus the complete set of constraints {*LC*, *BD*, *IR*}. In an ordinary linear program, the set *IR* is empty. In an integer linear program, all of the variables are in *IR*. In a mixed integer program, at least one variable is in *IR* and at least one variable is not in *IR*.

The *LP-relaxation* of a MILP is created by considering only the objective function plus the subset of constraints {*LC*, *BD*}. Because the LP-relaxation has fewer restrictions, its feasible region is larger.

1.1 A Review of MILP Solution Methods

There are two main methods of solving MILPs in practice, cutting-plane methods and branch-and-bound, plus a hybrid of the two, branch-and-cut. Cutting-plane methods (e.g., [9, 12, 22]) work by iteratively adding constraints ("cuts") to the set *LC* (or *BD*), which reduce the size of the feasible region of the LP-relaxation such that the optimum solution of the LP-relaxation gradually approaches the optimum solution of the original MILP. In the cutting-plane method, infeasibility of the original MILP is detected when an added cut renders the current LP-relaxation infeasible.

The well-known branch-and-bound method (e.g., [21]) operates by creating a tree of nodes, each of which is an LP based on the original LP-relaxation with altered variable bounds. A node is *expanded* (child nodes are derived from it) when it has an LP-relaxation that is feasible, but for which the LP-relaxation optimum point has at least one integer variable that does not have an integral value; one such variable is chosen as the *branching variable.* Two child nodes are created by copying all of the constraints in the parent node and then altering the lower bound on the branching variable to create one child node and altering the upper bound on the branching variable to create the other child node.

The bounding function value of intermediate nodes is given by the objective function value of the LP-relaxation. There are various rules for choosing which unexpanded node to choose next for expansion. The *best-bound* rule chooses the unexpanded node having the best value of the bounding function anywhere on the branch-and-bound tree.

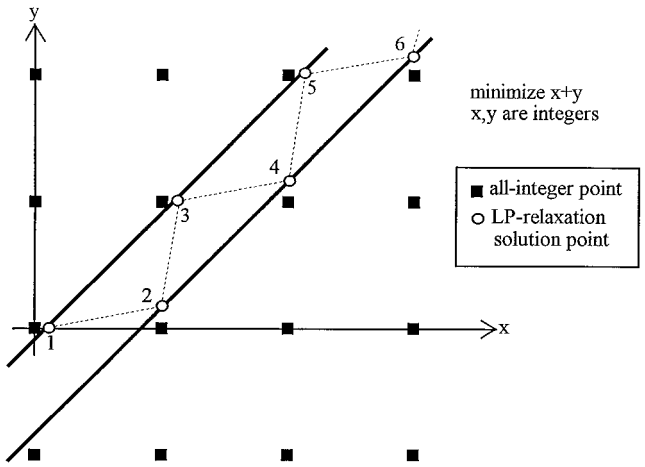

Figure 1. The branch-and-bound solution of this MILP fails to terminate.

The *depth-first* rule chooses the unexpanded node having the best value of the bounding function from among the group of child nodes just created. Leaves of the branch-and-bound tree are reached when either 1) the optimum solution of the LP-relaxation of a node also satisfies all of the integer restrictions (in which case it is also a feasible solution for the original MILP), 2) the LP-relaxation of the node is infeasible, or 3) the optimum value attained by the LP-relaxation of a node is worse than the best known MILP-feasible solution (does not occur in infeasible MILPs).

For our purposes, it is important to note that feasibility of a MILP is detected by the branch-and-bound method as soon as the first MILP-feasible node is created. However, infeasibility is only decided when the tree has been fully expanded and the LP-relaxation of every leaf node proves infeasible. Thus, it is generally much more computationally expensive to recognize infeasibility than feasibility of a MILP using branch-and-bound.

In certain cases, the size of the branch-and-bound tree can become very large, possibly exceeding available memory, and in special cases growing infinitely. For example, consider the MILP in Figure 1 in which both x and y are nonnegative integer variables. The nonnegativity constraint on y and the two parallel diagonal constraints form a pipe-shaped feasible region. Because the objective is to minimize $x + y$, the first LP-relaxation optimum solution is at the point marked 1. Because x is not integral at that point, the branch and bound procedure creates two child nodes; one is infeasible, and the LP-relaxation of the other has an optimum solution at point 2 in Figure 1. Now x has an integral value, but y does not, leading to the creation of two more child nodes on the branch-and-bound tree. The solution process alternates between x and y as the branching variable, causing the sequence of solutions to climb the pipe, as shown in Figure 1.

A branch-and-bound solution of the MILP in Figure 1 will never terminate. It is also easy to construct examples that will terminate, but that require an excessive number of

iterations to do so. For example, imagine that the two diagonal constraints in Figure 1 are angled very slightly towards one another so that they eventually cross at a great distance from the origin. It may take a great number of iterations before infeasibility can be determined. In the same manner, if the diagonal constraints are very slightly angled away from each other, it may require a great number of iterations before the first MILP-feasible point is reached.

The important point is that a branch-and-bound-based MILP solver may not be able to decide the feasibility status of a MILP in an acceptable number of iterations or within an acceptable upper limit on available computer memory.

The branch-and-cut method incorporates cutting-plane methods into the branch-and-bound framework (e.g., [13, 16, 22]). The main idea is to incorporate a cut into a branch-and-bound subproblem under certain conditions. This can speed the recognition of the feasibility status of difficult MILPs such as shown in Figure 1 when the cut either moves the solution into the vicinity of a feasible solution or renders the LP-relaxation infeasible.

2. Infeasibility Analysis for Linear Programs

Methods of isolating IISs in linear programs have been developed in recent years and are now available in commercial LP solvers such as LINDO[18] and CPLEX.[8] There are two basic algorithms that guarantee the isolation of a single IIS: the deletion filter and the additive method. Other algorithms can be used with these basic methods to speed the isolation or in an attempt to find an IIS having desirable characteristics such as few rows. See [5] for a complete review of IIS isolation methods for both linear and nonlinear programs.

Chinneck and Dravnieks[7] introduced the *deletion filter* for LPs, described in Algorithm 1. The deletion filter operates by testing the feasibility of the model when constraints are dropped in turn. At the end of a single pass through the constraints, the identification of a single IIS is guaranteed (see [7] for a proof).

Algorithm 1. The deletion filter.
Input: an infeasible set of constraints.
FOR each constraint in the set:
  Temporarily drop the constraint from the set.
  Test the feasibility of the reduced set:
    IF feasible THEN return dropped constraint to the set.
    ELSE (infeasible) drop the constraint permanently.
Output: constraints constituting a single IIS.

Tamiz et al[19, 20] introduced a method sometimes referred to as the *additive method.*[5] The main feature of the method is the *adding in* of constraints as the algorithm proceeds, until infeasibility is achieved, exactly the opposite of the approach taken in the deletion filter (see Algorithm 2). See [5] for a proof that the method returns exactly one IIS.

Algorithm 2. The additive algorithm.
C: ordered set of constraints in the infeasible model.
T: the current test set of constraints.
I: the set of IIS members identified so far.
Input: an infeasible set of constraints *C*.
Step 0: Set $T = I = \phi$.
Step 1: Set $T = I$.
  FOR each constraint c_i in *C*:
    Set $T = T \cup c_i$.
    IF *T* infeasible THEN
      Set $I = I \cup c_i$.
      Go to Step 2.
  END FOR.
Step 2: IF *I* feasible THEN go to Step 1.
  Exit.
Output: *I* is an IIS.

The additive method has the interesting property that once infeasibility is attained, constraints listed after the constraint that triggers infeasibility (say c_j) are never tested. This happens because c_j is added to *I*, which is then always part of *T*. Thus as other constraints are added to *T*, infeasibility will at least be attained by the time constraint c_{j-1} is added because this duplicates the last infeasible set. Of course, infeasibility may be attained before c_{j-1} is added, which cuts off even more constraints from consideration.

The *sensitivity filter* is a way of eliminating many uninvolved constraints quickly.[7] The sensitivity filter applies two theorems by Murty [14, pp. 237–238] to the phase 1 solution of an infeasible LP: if a row constraint or a column bound has a nonzero shadow price or reduced cost, then it must be part of some IIS. Constraints or bounds having shadow prices or reduced costs of zero can then be discarded, and the remaining constraints must contain at least one IIS. The deletion filter or the additive method must be applied to the output of the sensitivity filter to guarantee the isolation of a single IIS.

3. Properties of Infeasible MILP Branch and Bound Solutions

The branch-and-bound tree developed during the initial solution of an infeasible MILP contains valuable information that can be used during the subsequent infeasibility isolation. We develop three theorems in this regard.

Some initial definitions are in order. A *leaf node* of a branch-and-bound tree is either a node in which all of the IRs are satisfied or one in which the LP-relaxation is infeasible. An *intermediate node* is a node that is not a leaf node. For an intermediate node *K*, IR_K is the set of all IRs satisfied by the LP-relaxation at that node. $BBBD_K$ is the set of BDs added by the branch-and-bound procedure at some node *K* (intermediate or final).

THEOREM 1. An infeasible MILP does not have any IISs whose integer part is identical to the IR_K at any intermediate node.

PROOF. At an intermediate node *K*, the current set of constraints is $LC \cup BD \cup IR \cup BBBD_K$. Because the node is intermediate, the LP-relaxation is feasible, or, equivalently, $LC \cup BD \cup IR_K \cup BBBD_K$ is MILP feasible. An IIS having IR_K as its complete integer part must have as its linear part either $LC \cup BD \cup BBBD_K$ or some subset of it, but no such IIS can exist because it is already known that $LC \cup BD \cup IR_K \cup BBBD_K$ is MILP feasible. ■

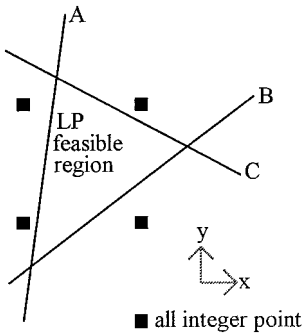

Figure 2. An infeasible MILP.

THEOREM 2. If a sensitivity filter is applied to every leaf node, and all original LCs and BDs having nonzero reduced costs are marked, then the set *IR* ∪ {marked LCs} ∪ {marked BDs} is infeasible.

PROOF. The unmarked LCs and BDs are not marked because they are not tight in any of the leaf nodes. Hence those unmarked LCs and BDs could have been relaxed in the original MILP and the same branch-and-bound tree would still have proven infeasibility of the modified MILP. ■

Some further definitions are needed. A *path* in a branch-and-bound tree is a set of branches leading from the root to a leaf in which each branch is labeled with the name of the integer variable that was branched on. The set of *active IRs* (A_T) is the union of all of the IRs for the variables in any of the paths in a branch-and-bound tree.

THEOREM 3. For an infeasible MILP, the set $LC \cup BD \cup A_T$ is infeasible.

PROOF. Given the MILP $LC \cup BD \cup A_T$, a branch-and-bound tree identical to the original branch-and-bound tree can be generated, arriving at the conclusion that $LC \cup BD \cup A_T$ is infeasible. ■

Notice also that each path provides an interesting candidate for an IS: the constraint set *LC* ∪ *BD* ∪ {IRs on variables in the path}. This candidate for an IS is more likely to prove infeasible because the set of branches in the path terminates at an infeasible node. There is no guarantee that the candidate IS is actually infeasible, however, because the path may consist partly or entirely of one-sided branches (i.e., a particular variable is branched upon only in the higher-valued direction or only in the lower-valued direction).

Note that where the MILP has multiple IISs, it may be possible to develop a different branch-and-bound tree for the same model (perhaps by varying parameters such as the bounding rule or branching variable selection rule) in which different sets of LCs, BDs, and IRs can be eliminated using Theorems 1–3. This happens when a different IIS drives the development of the branch-and-bound tree.

4. Basic Algorithms for Isolating Infeasibility in MILPs

The analysis of infeasible MILPs is complicated by the presence of the integer restrictions. Consider Figure 2 for example, in which both variables are integers. While the LP-relaxation is feasible, it is impossible to find a point in which both of the variables are integral simultaneously. The IIS in Figure 2 is {A, B, C, [x], [y]}. Note that we are interested only in MILPs in which the initial LP-relaxation is feasible. If the initial LP-relaxation is infeasible, then it is trivial to return an IIS isolated from the initial LP-relaxation by the well-developed methods for LPs.

To provide the most assistance in the analytic effort, any IIS isolated should have as few members as possible. Further, the IIS is easier to analyze if the number of IRs is small. Previous work[6] also shows that IISs in LPs are easier to understand if the number of LCs is small. Hence, we wish to isolate IISs in MILPs that have an overall small cardinality, and which tend to have few IRs and LCs.

The deletion filter and the additive method are very effective for finding IISs in LPs because LP solvers (issues of tolerance aside) are able to decide the feasibility status of a model with perfect accuracy. However, because of the failure of MILP solvers to meet acceptable limits on time and memory use under certain conditions (see Section 1), MILP solvers cannot provide a guarantee of accuracy in deciding model feasibility status. This necessitates certain modifications to the basic deletion filter and additive method, as described in the following subsections.

Both the deletion filter and the additive algorithm operate by examining the feasibility of various subsets of the constraints in the original model. It is possible that some test subproblems will fail to decide feasibility status within practical computation limits (time or memory). This is handled by imposing an upper limit on the number of branch-and-bound tree nodes generated by any subproblem (usually 10,000). If the test subproblem has not terminated within this limit, then the subproblem solution is abandoned. In the case of both algorithms, the conservative assumption is that the subproblem is feasible, causing the addition of the tested constraint to the output set.

The retention of the tested constraint in case of practical nontermination of a subproblem destroys the guarantee of identifying an IIS. The output set is instead only an *Infeasible Subset* (IS), as opposed to an IIS. However, the IS is still very useful in that it usually limits analytic effort to a much smaller portion of the entire model, thereby speeding the analytic effort.

The following elements are standard in all of the following algorithms and hence are omitted from the algorithm statements: 1) if the initial LP-relaxation is infeasible, then apply the existing LP infeasibility analysis methods to isolate an IIS, and 2) prescreen the bounds on integer-restricted variables for simple errors such as $0.5 \leqslant x \leqslant 0.8$, or $y \geqslant 2$ where y is binary.

4.1 Basic Deletion Filtering for MILPs

The basic deletion filter (Algorithm 1) can be applied directly to MILPs. This necessitates the solution of $|LC| + |BD| + |IR|$ MILPs, which can be quite time consuming, but is effective in identifying an IIS provided that no subproblem exceeds the computation limits. When the removal of a

particular constraint during deletion filtering generates a subproblem that exceeds the computation limit, that constraint is labeled *dubious* and is retained in the output set to guarantee that the output set is infeasible. If there is at least one dubious constraint in the output set, then the output set is an IS; whether it is also an IIS is not known. Any dubious constraints in the output set are candidates for elimination from the IS to possibly convert it to an IIS. Post-processing schemes to determine whether dubious constraints may be eliminated have been left to future research.

The probability of exceeding the computation limits for a subproblem is reduced if all variables are both upper and lower bounded. Failing this, it is preferable to deletion test the BDs as late in the process as possible, so that they remain in place to limit the number of branch-and-bound nodes needed before feasibility can be decided.

The speed of the deletion filter for MILPs is affected by whether the IRs are tested before the LCs or vice versa. Since branch-and-bound search trees tend to grow with larger numbers of IRs, it may be preferable to test IRs before LCs in the hope that some of the IRs will be eliminated early, so that subsequent MILP test problems have relatively smaller branch-and-bound trees. Accordingly, two versions of the deletion filter are proposed, the (*IR-LC-BD*) and the (*LC-IR-BD*) versions, in which the constraints are tested in the order indicated, as illustrated in Algorithms 3 and 4.

Algorithm 3. The (*IR-LC-BD*) deletion filter for MILPs.
LC_0, BD_0, IR_0 are the original sets of constraints.
Input: an infeasible MILP.
Step 0: Set *status* = "*IIS*".
  Set $T = LC_0 \cup BD_0$.
  IF T infeasible, go to Step 2.
  Set $T = T \cup IR_0$.
Step 1: FOR each $ir_k \in IR_0$:
    IF $T\backslash\{ir_k\}$ infeasible, set $T = T\backslash\{ir_k\}$.
    ELSE IF $T\backslash\{ir_k\}$ exceeds computation limit, set *status* = "*IS*", label ir_k *dubious.*
Step 2: FOR each $lc_k \in LC_0$:
    IF $T\backslash\{lc_k\}$ infeasible, set $T = T\backslash\{lc_k\}$.
    ELSE IF $T\backslash\{lc_k\}$ exceeds computation limit, set *status* = "*IS*", label lc_k *dubious.*
Step 3: Set $BD_1 = BD_0\backslash\{$BDs on variables not in $lc \in T\}$.
  Set $T = (T\backslash BD_0) \cup BD_1$.
  FOR each $bd_k \in BD_1$:
    IF $T\backslash\{bd_k\}$ infeasible, set $T = T\backslash\{bd_k\}$.
    ELSE IF $T\backslash\{bd_k\}$ exceeds computation limit, set *status* = "*IS*", label bd_k *dubious.*
Output: If *status* = "*IIS*", T is an IIS, else T is an IS.

Note that Step 3 of Algorithm 3 avoids testing of BDs on variables that are not represented in the remaining set of LCs. This time-saving step assumes that the variable BDs have been prescreened to eliminate the possibility of a simple reversal of the bounds on a variable. Algorithm 4 uses a similar idea to avoid testing IRs and BDs on variables that are no longer represented in the remaining set of LCs. Empirical tests of the relative efficiency of Algorithms 3 and 4 are reported in Section 6.

Algorithm 4. The (*LC-IR-BD*) deletion filter for MILPs.
LC_0, BD_0, IR_0 are the original sets of constraints.
Input: an infeasible MILP.
Step 0: Set *status* = "*IIS*".
  Set $T = LC_0 \cup BD_0 \cup IR_0$.
Step 1: FOR each $lc_k \in LC_0$:
    IF $T\backslash\{lc_k\}$ infeasible, set $T = T\backslash\{lc_k\}$.
    ELSE IF $T\backslash\{lc_k\}$ exceeds computation limit, set *status* = "*IS*", label lc_k *dubious.*
Step 2: Set $IR_1 = IR_0\backslash\{$IRs on variables not in $lc \in T\}$.
  Set $T = (T\backslash IR_0) \cup IR_1$.
  FOR each $ir_k \in IR_1$:
    IF $T\backslash\{ir_k\}$ infeasible, set $T = T\backslash\{ir_k\}$.
    ELSE IF $T\backslash\{ir_k\}$ exceeds computation limit, set *status* = "*IS*", label ir_k *dubious.*
Step 3: Set $BD_1 = BD_0\backslash\{$BDs on variables not in $lc \in T\}$.
  Set $T = (T\backslash BD_0) \cup BD_1$.
  FOR each $bd_k \in BD_1$:
    IF $T\backslash\{bd_k\}$ infeasible, set $T = T\backslash\{bd_k\}$.
    ELSE IF $T\backslash\{bd_k\}$ exceeds computation limit, set *status* = "*IS*", label bd_k *dubious.*
Output: If *status* = "*IIS*", T is an IIS, else T is an IS.

4.2 The Basic Additive Method for MILPs

In adapting the additive method for use with MILPs, there is again a choice of the order in which the classes of constraints are added. However, given our assumption that the initial LP relaxation is feasible, it makes sense to proceed as though the sets *LC* and *BD* have already been added without causing infeasibility. This leaves only the members of *IR* to be tested. Hence, the additive method for MILPs (Algorithm 5) begins by testing the addition of members of *IR* to $LC \cup BD$.

Unlike the deletion filter, the additive method is not able to directly identify dubious constraints. This is because the test set is maintained in a feasible or indeterminate state until sufficient constraints are added to render the test set infeasible. However, if no indeterminate subproblems are encountered in the course of the isolation, then it is known that the output set is an IIS. If at least one subproblem exceeds the computation limit, then the output set is an IS (it may also be an IIS, but this is not known).

The worst-case time complexity of the additive method occurs when the entire original problem is an IIS. In this case, during the first iteration, the method solves $n + 1$ MILPs, one for every constraint in the model, plus one MILP for the test of I. During the next iteration, n MILPs are solved, etc. The overall worst-case time complexity is then $\frac{1}{2}(|IR| + |LC| + |BD|)^2$ MILP solutions. However, by considering the model in stages as in Algorithm 5, the worst-case time complexity is reduced to $\frac{1}{2}(|IR|^2 + |LC|^2 + |BD|^2)$ MILP solutions.

4.2.1 Dynamic Reordering Additive Method

The additive method gradually builds up an infeasible model by adding constraints one at a time until infeasibility is attained. All of the intermediate test subproblems are feasible until the final constraint triggering infeasibility is added. However, a dynamic reordering of the constraints

can eliminate the need to solve some of the initial feasible subproblems. The main idea is as follows. If an intermediate test subproblem is feasible, then scan all of the constraints past the current constraint just added, and add to T all constraints that are satisfied by the current solution point. See Algorithm 6.

Algorithm 5. The basic additive method for MILPs.
C: ordered set of constraints in the original infeasible MILP ($IR_0 \cup LC_0 \cup BD_0$).
T: the current test set of constraints.
I: the set of IS members identified so far.
Input: an infeasible MILP.
Step 0: Set *status* = "*IIS*". Set $I = \phi$.
IF $LC_0 \cup BD_0$ infeasible, go to Step 2b.
Step 1: Set $T = I \cup LC_0 \cup BD_0$.
FOR each $ir_k \in IR_0$:
Set $T = T \cup \{ir_k\}$.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{ir_k\}$.
IF $I \cup LC_0 \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup LC_0 \cup BD_0$ infeasible, go to Step 2.
Go to Step 1.
Step 2: a. IF $I \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup BD_0$ infeasible, go to Step 3.
b. Set $T = I \cup BD_0$.
c. FOR each $lc_k \in LC_0$:
Set $T = T \cup \{lc_k\}$.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{lc_k\}$.
IF $I \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup BD_0$ infeasible, go to Step 3.
Go to Step 2b.
Step 3: a. IF I exceeds computation limit, set *status* = "*IS*".
ELSE IF I inconsistent, exit.
b. Set $BD_1 = BD_0 \backslash \{$BDs on variables not in $lc \in I\}$.
c. Set $T = I$.
d. FOR each $bd_k \in BD_1$:
Set $T = T \cup \{bd_k\}$.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{bd_k\}$.
IF I exceeds computation limit THEN set *status* = "*IS*".
ELSE IF I infeasible, exit.
Go to Step 3c.
Output: If *status* = "*IIS*", I is an IIS, else I is an IS.

Algorithm 6. Dynamic reordering additive method for MILPs.
C: ordered set of constraints in the original infeasible MILP ($IR_0 \cup LC_0 \cup BD_0$).
T: the current test set of constraints. I: the set of IS members identified so far.
Input: an infeasible MILP.
Step 0: Set *status* = "*IIS*". Set $I = \phi$.
IF $LC_0 \cup BD_0$ infeasible, go to Step 2b.
Step 1: Set $T = I \cup LC_0 \cup BD_0$.
FOR each $ir_k \in C$:
IF ir_k unmarked, set $T = T \cup \{ir_k\}$, ELSE skip to next iteration.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{ir_k\}$; set $C = C \backslash \{ir_j | j > k\}$.
IF $I \cup LC_0 \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup LC_0 \cup BD_0$ infeasible, go to Step 2.
Go to Step 1.
ELSE set $temp = \{ir_j | j > k, ir_j \in C, ir_j$ satisfied$\}$.
set $T = T \cup temp$; mark all members of *temp*.
Step 2: a. IF $I \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup BD_0$ infeasible, go to Step 3.
b. Set $T = I \cup BD_0$.
c. FOR each $lc_k \in C$:
IF lc_k unmarked, set $T = T \cup \{lc_k\}$, ELSE skip to next iteration.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{lc_k\}$; set $C = C \backslash \{lc_j | j > k\}$.
IF $I \cup BD_0$ exceeds computation limit THEN set *status* = "*IS*".
ELSE IF $I \cup BD_0$ infeasible, go to Step 3.
Go to Step 2b.
ELSE set $temp = \{lc_j | j > k, lc_j \in C, lc_j$ satisfied$\}$.
set $T = T \cup temp$; mark all members of *temp*.
Step 3: a. IF I exceeds computation limit, set *status* = "*IS*".
ELSE IF I inconsistent, exit.
b. Set $BD_1 = BD_0 \backslash \{$BDs on variables not in $lc \in I\}$.
c. Set $T = I$.
d. FOR each $bd_k \in BD_1$:
IF bd_k unmarked, set $T = T \cup \{bd_k\}$.
IF T exceeds computation limit THEN set *status* = "*IS*".
ELSE IF T infeasible THEN:
Set $I = I \cup \{bd_k\}$; set $BD_1 = BD_1 \backslash \{bd_j | j > k\}$.
IF I exceeds computation limit THEN set *status* = "*IS*".
ELSE IF I infeasible, exit.
Go to Step 3c.
ELSE set $temp = \{bd_j | j > k, bd_j \in BD_1, bd_j$ satisfied$\}$.
set $T = T \cup temp$; mark all members of *temp*.
Output: If *status* = "*IIS*", I is an IIS, else I is an IS.

Note that Algorithm 6 truncates the list of constraints scanned for feasibility past the constraint that most recently triggered test subproblem infeasibility. As explained in Section 2, these constraints are no longer relevant to the infeasibility isolation and their omission yields smaller intermediate test MILPs. While there is some cost associated with checking whether some constraints are satisfied by the current solution point, this is negligible compared to the cost of solving another MILP.

Algorithm 7. Basic additive/deletion method for MILPs.
T: the current test set of constraints.
I: the set of IS members identified so far.
Input: an infeasible MILP.
Step 0: Set *status* = "*IIS*". Set $I = \phi$.
IF $LC_0 \cup BD_0$ infeasible, go to Step 2a.
Step 1: Set $T = I \cup LC_0 \cup BD_0$.
FOR each $ir_k \in IR_0$:
Set $T = T \cup \{ir_k\}$.
IF T infeasible THEN:
Set $I = I \cup \{ir_k\}$.
IF $I \cup LC_0 \cup BD_0$ infeasible, go to Step 2.
Go to Step 1.
Step 2: a. Set $T = I \cup LC_0 \cup BD_0$.
b. FOR each $lc_k \in LC_0$:
IF $T\backslash\{lc_k\}$ infeasible, set $T = T\backslash\{lc_k\}$.
ELSE IF $T\backslash\{lc_k\}$ exceeds computation limit, set *status* = "*IS*", label lc_k *dubious.*
Step 3: Set $BD_1 = BD_0\backslash\{$BDs on variables not in $lc \in T\}$.
Set $T = (T\backslash BD_0) \cup BD_1$.
FOR each $bd_k \in BD_1$:
IF $T\backslash\{bd_k\}$ infeasible, set $T = T\backslash\{bd_k\}$.
ELSE IF $T\backslash\{bd_k\}$ exceeds computation limit, set *status* = "*IS*", label bd_k *dubious.*
Output: If *status* = "*IIS*", T is an IIS, else T is an IS.

4.3 Basic Additive/Deletion Method for MILPs

The additive method and the deletion filter can be combined in a straightforward way. As shown in Algorithm 7, the basic additive/deletion method proceeds by adding IRs to $LC_0 \cup BD_0$ until infeasibility is triggered, and then switches to the deletion filter to complete the isolation of the infeasibility. Note that the status of the output set as an IS or IIS is determined only during the deletion filtering portion of the algorithm, which also identifies individual dubious constraints. During the additive portion of the algorithm, indeterminate subproblems are treated in the same manner as feasible subproblems. Algorithm 7 is easily modified to incorporate the dynamic reordering version of the additive method in Step 1.

The time complexity of the additive/deletion method derives partly from the time complexity of the additive method as applied to the IRs and to the time complexity of the deletion filter as applied to the LCs and BDs. The worst-case time complexity is $O(|IR|^2 + |LC| + |BD|)$ MILP solutions.

5. Speed Improvements

Because all of the algorithms for isolating infeasibility in MILPs depend on the repeated solution of test subproblem MILPs derived from the original, the isolation can be slow. However, various tactics can improve the speed of the algorithms, as described below.

5.1 Grouping Constraints

To reduce the overall number of MILPs that must be solved during the isolation, constraints can be handled in groups. During deletion filtering, constraints can be temporarily dropped in groups (e.g., 10 at a time). If the reduced MILP is infeasible, the group of 10 constraints would be dropped permanently, saving the cost of solving 9 MILPs. If the reduced MILP proves feasible, then the group is reinstated and the algorithm backs up and tests each constraint individually, costing an additional 1 MILP solution. The overall net savings depends on how often the reduced model proves infeasible vs. feasible.

For the additive method, constraints can be added in groups (e.g., 10 at a time) until infeasibility is triggered. If the partial model proves feasible after the group has been added, there is a savings of 9 MILP solutions. If the partial model proves infeasible after the group is added, the group is removed and the algorithm backs up and adds the constraints one at a time, costing an additional 1 MILP solution. As for the deletion filter, the overall net savings depends on how often the partial model proves infeasible vs. feasible.

Numerous schemes for grouping constraints are possible. Five methods are summarized in Algorithm 8 for use with the deletion filter; note that these are easily modified for use with the additive method by substituting "feasible" for "infeasible" where the test MILPs are evaluated. Four of the five are adaptive methods in which the group size changes depending on the feasibility status of recent test MILPs.

Algorithm 8. Constraint grouping methods for deletion filtering.
GroupSize is the number of constraints in a group.
Grouping method 1: Fixed Group Size.
GroupSize fixed by user.
Grouping method 2: Additive Adaptive Grouping A.
Set *GroupSize* = 2.
IF test MILP infeasible THEN *GroupSize* = *GroupSize* + 2.
ELSE *GroupSize* = maximum[*GroupSize* − 2, 1]
Grouping method 3: Additive Adaptive Grouping B.
Set *GroupSize* = 2.
IF test MILP infeasible THEN *GroupSize* = *GroupSize* + 2.
ELSE *GroupSize* = 2.
Grouping method 4: Multiplicative Adaptive Grouping A.
Set *GroupSize* = 1.
IF test MILP infeasible THEN *GroupSize* = *GroupSize**2.
ELSE *GroupSize* = maximum[integer(*GroupSize*/2), 1].
Grouping method 5: Multiplicative Adaptive Grouping B.
Set *GroupSize* = 1.
IF test MILP infeasible THEN *GroupSize* = *GroupSize**2.
ELSE *GroupSize* = 1.

5.2 Safety Bounds

The infeasibility isolation process is slowed considerably when some of the test subproblems exceed preset computation limits, i.e., an individual MILP exceeds the limit on the maximum number of nodes in a branch-and-bound tree. The isolation process can be speeded up if the number of subproblems exceeding the limit is reduced (possibly to zero). Because the problem of practical nontermination is often associated with a lack of variable bounds in a subproblem, one approach is to add *safety bounds* to the model: extra BDs that limit the maximum and minimum value of every variable.

One method of selecting safety bounds is to ask the user to add them manually. These are then present in the model at all times and are never removed; they are not tested by any of the algorithms. The effect is to limit the solution space to a multidimensional box. There are then five possible outcomes:

- *No safety bounds are active at subproblem termination.* Interpretation: all algorithms are proceeding as intended.
- *One or more safety bounds are active at a feasible subproblem termination.* Interpretation: all algorithms are proceeding as intended.
- *One or more safety bounds are active at an infeasible subproblem termination and the unaltered subproblem is actually infeasible.* Interpretation: the safety-bounded version of the subproblem will also be found to be infeasible, so all algorithms will proceed as intended.
- *One or more safety bounds are active at an infeasible subproblem termination and the unaltered subproblem is actually feasible.* Interpretation: the feasible point for the unaltered subproblem is outside of the multidimensional box defined by the safety bounds. Both the deletion filter and the additive method will eliminate constraints from the output set that should have been retained; the output set will then not be infeasible (though it will define a MILP in which feasibility is difficult to achieve within practical computation limits). However, this potential difficulty is well flagged when at least one safety bound is active at an infeasible subproblem termination. The infeasibility analysis may need to be restarted after the active safety bounds are reset to provide a larger multidimensional box.
- *Subproblem exceeds computation limit.* Interpretation: safety bounds were insufficient insurance against subproblem practical nontermination. Algorithms will proceed to isolate an IS.

Of course, the essential difficulty is in identifying whether the third or the fourth outcome has occurred when a safety bound is active at an infeasible subproblem termination. There is no way to decide among the two outcomes without carrying out additional MILP solutions in which the safety bounds have been removed. These additional MILP solutions then run the risk of nontermination or excess computation time; exactly the reason that the safety bounds were introduced in the first place. For this reason, whenever a safety bound is active at an infeasible subproblem termination, the output set is labeled an IS rather than an IIS.

Several variations on the basic idea of safety bounds are possible. One is to introduce safety bounds for variables only when the original bounds are removed in the course of operation of one of the isolation algorithms. Another is to generate the appropriate safety bound automatically when an original variable bound is removed; e.g., by adjusting the original upper bound upward by a fixed or proportional amount or by adjusting the original lower bound downward in an analogous manner.

5.3 Using the Initial Branch-and-Bound Tree

As shown in Section 3, a great deal of information is contained in the original branch-and-bound tree that first signaled infeasibility, and this information can be used in the subsequent infeasibility isolation. Theorems 1–3 allow a certain amount of preprocessing of the MILP after it has been found infeasible and before the infeasibility isolation algorithms are applied. Theorem 2 allows the initial elimination of any unmarked LCs or BDs. Theorem 3 allows the initial elimination of any IRs that do not appear in A_T.

Also, as pointed out in Section 3, each path in the original branch-and-bound tree provides a candidate for the set of IRs in an IS. This set can be pruned by comparing the sets of IRs associated with the paths with the sets of IRs associated with the nodes. Any IR set associated with a node (and any subset of such a set) cannot be the entire IR set in an IS in conjunction with $LC \cup BD$ by Theorem 1.

These ideas can be combined as shown in Algorithm 9. For efficiency, as new IRP_i are discovered during the initial branch-and-bound solution, they can be checked against the current IRN^*. Similarly, as new IRN_i are discovered, the current members of IRP can be checked against it and its subsets. This would, however, slow the solution in the case of a feasible MILP.

Algorithm 9. Using information from the original solution to speed the infeasibility isolation.

IRP: IRP_i is the set of IRs defined by the variables in path i. $IRP = \{IRP_i | i = 1 \text{ to (number of paths)}\}$.

IRN: IRN_i is the set of IRs defined by the satisfied IRs at an intermediate node i. $IRN = \{IRN_i | i = 1 \text{ to (number of intermediate nodes)}\}$. $IRN^* = IRN \cup \{\text{all proper subsets of members of } IRN\}$.

LCM: the set of marked LCs.

BDM: the set of marked BDs.

Input: a MILP, feasibility status unknown.

Step 1: Solve the MILP. Compile the sets A_T, IRP, IRN, LCM, BDM while solving.

If feasible, exit.

Step 2: Set $IRP = IRP \backslash (IRP \cap IRN^*)$.

Order IRP from smallest to largest cardinality.

Step 3: For each $IRP_i \in IRP$:

IF $LCM \cup BDM \cup IRP_i$ infeasible THEN

Set $IRP' = IRP_i$.

Go to Step 4.

Set $IRP' = A_T$.

Step 4: Isolate an IIS or IS in $LCM \cup BDM \cup IRP'$ using any algorithm.

Output: An IIS or IS.

Table I. Characteristics of the Test Data Sets

Model	LCs	Variables	Nonzeros	BDs	General IRs	Binary IRs
bell3a	124	133	441	204	32	39
bell3b	124	133	441	204	32	39
bell4	106	117	385	181	30	34
bell5	92	104	340	162	28	35
dell	500	626	4580	825	6	99
flugpl	19	18	64	29	11	0
g503	41	48	144	72	0	24
misc01	55	83	746	164	0	82
misc02	40	59	414	116	0	58
misc03	97	160	2054	318	0	159
misc04	1726	4897	17253	4691	0	30
misc05	301	136	2946	204	0	74
misc06	821	1808	5860	1894	0	112
mod008	7	319	1562	638	0	319
mod013	63	96	288	144	0	48
p0201i	134	201	1925	402	0	201
stein9	14	9	54	18	0	9
stein15	37	15	135	30	0	15
stein27	119	27	405	54	0	27
stein45	332	45	1079	9	0	45
average	237.6	451.7	2055.8	518.0	7.0	72.5

5.4 Other Possibilities

The speed improvements suggested in Sections 5.1–5.3 are empirically tested in Section 6. However, there are a number of other possibilities that may be worth investigation. Two interesting ideas are outlined below.

5.4.1 Replacing the Original Objective Function

The original objective function does not play any useful role during infeasibility analysis. It can, in fact, slow the infeasibility isolation by the way in which it guides the development of the branch-and-bound tree. Speed improvement may be possible by replacing the original MILP objective by one that tends to decide feasibility status more rapidly.

When a subproblem proves MILP-infeasible (but LP-relaxation feasible), two child nodes are generated, each having a new constraint added based on the branching variable x_k, whose noninteger value in the parent node is α. The typical form of the added constraint (with nonnegative slack variable s_k included) is: $x_k + s_k = \lfloor \alpha \rfloor$ or $x_k - s_k = \lceil \alpha \rceil$. A new objective function can then be introduced: *minimize* Σs_k over all of the slack variables introduced during branching. The effect is to drive the MILP towards feasibility in a manner analogous to an ordinary LP phase 1, which should speed the decision of feasibility status in the test subproblems.

5.4.2 Choosing MILP Solver Settings

The settings of the MILP solver can have a great influence on the speed of a MILP solution. Because the algorithms for infeasibility isolation involve the solution of numerous test MILPs, it is worthwhile exploring whether the MILP solver

Table II. Original Determination of Infeasibility

Model	B&B Tree Nodes	LP Iterations	Time (sec.)
bell3a	12	144	0.23
bell3b	2	103	0.17
bell4	0	105	0.15
bell5	2	108	0.13
dell	6	767	3.48
flugpl	5012	3624	11.17
g503	0	45	0.05
misc01	28	410	0.95
misc02	54	501	0.55
misc03	8	299	0.80
misc04	2	2094	12.17
misc05	52	1118	3.42
misc06	38	3429	20.47
mod008	956	3949	15.42
mod013	124	490	0.79
p0201i	presolve	presolve	0.03
stein9	12	32	0.07
stein15	92	340	0.40
stein27	1218	5517	10.98
stein45	1112	11298	39.05
average	436.5	1718.7	6.02

Table III. Results for Simple IR-LC-BD Deletion Filtering (Algorithm 3)

Model	Dubious	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a	26	7	68	51	7284	44847	1:55
bell3b	48	8	75	77	1325018	2307950	2:18:33
bell4	14	15	65	43	1349744	1455399	2:01:55
bell5	25	6	53	78	1010942	2105135	1:39:32
dell	none	5	381	764	7512	1133866	1:35:03
flugpl	6	6	6	6	414413	241286	11:45
g503	4	1	33	15	213	6180	6
misc01	1|25	12	28	65	4929	41813	59
misc02	1|7	14	30	43	6610	45758	55
misc03	1|40	3	31	110	2982	53854	2:01
misc04	294|2477	3	1186	3731	20385	8199496	19:41:38
misc05	4|32	7	62	55	9486	188309	7:57
misc06	26|684	11	613	1017	44277	3201026	6:28:52
mod008	292	73	7	313	583635	2056161	1:56:59
mod013	21	15	50	50	16889	74834	1:44
p0201i	5	1	2	6	911440	6338045	5:04:14
stein9	none	6	13	0	1109	2217	3
stein15	none	6	36	0	3937	16185	19
stein27	none	14	109	0	139602	625873	19:43
stein45	none	16	227	4	1035530	10126733	7:40:27
average	16.4|185.3	11.5	153.8	321.4	344796.8	1913248.4	2:27:44

Table IV. Results for Simple LC-IR-BD Deletion Filter (Algorithm 4)

Model	Dubious	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a	65	13	46	65	746510	1089361	57:31
bell3b	61	51	55	62	733668	1212488	1:17:42
bell4	38	13	51	55	982327	1431763	1:19:53
bell5	59	45	24	63	692561	959428	51:21
dell	none	4	33	114	14747	122082	6:04
flugpl	8	6	11	8	355378	237514	10:48
g503	18	11	21	25	3672	12195	17
misc01	1|48	14	25	64	7826	38424	51
misc02	1|9	13	29	43	7784	48736	58
misc03	1|32	1	36	117	25929	166925	4:40
misc04	305|2284	3	1186	3731	1786804	23012352	101:47:54
misc05	4|23	7	62	55	776164	3996470	2:31:49
misc06	26|652	11	613	1017	622445	19420458	61:05:35
mod008	313	73	7	313	1125475	2588166	2:57:52
mod013	22	15	50	50	37383	144965	3:18
p0201i	5	1	2	6	616176	3689674	3:20:05
stein9	none	6	13	0	1074	2139	3
stein15	none	6	36	0	3943	16192	19
stein27	none	14	109	0	162993	704387	21:59
stein45	none	16	227	4	1280217	9144909	7:16:18
average	17|181.9	16.1	131.8	289.6	499153.8	3401931.4	9:12:46

Table V. Results for the Basic Additive Method (Algorithm 5)

Model	IIS?	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a		7	68	53	61689	601275	19:27
bell3b		1	65	55	5084	471787	14:46
bell4		13	65	43	228604	703262	39:26
bell5		1	49	48	3115	249427	7:32
dell	y	5	184	825	93906	9004289	14:50:57
flugpl		8	13	10	1988436	1239675	1:00:47
g503		1	33	15	327	20247	45
misc01		12	28	165	5587	74450	2:41
misc02		14	30	115	38268	312357	12:18
misc03		3	31	319	1806	59880	5:30
misc04	kil						
misc05		7	62	71	24683	1275287	1:13:37
misc06	kil						
mod008		37	7	638	191653	917646	45:14
mod013		15	50	51	41148	388142	10:33
p0201i		1	2	6	407	29604	1:50
stein9	y	6	13	0	1042	2688	7
stein15	y	6	36	0	5114	25212	51
stein27	y	14	109	0	244824	1323113	41:03
stein45	kil						
average		8.9	49.7	142.0	172687.8	982255.4	1:12:12

settings can be chosen so as to provide quick solutions for the intermediate test MILPs. Two solver settings in particular have a great influence on the speed of the MILP solution: the method of node selection and the method of branching variable selection.

The two common methods of node selection are *best-bound* and *depth-first*. Because determining that a MILP is infeasible requires a complete expansion of the branch-and-bound tree, neither method is likely to be faster for infeasible MILPs. However, when the MILP is feasible, it is likely that a depth-first node selection will reach feasibility faster. In addition, depth-first node selection allows reuse of the final LP basis from the parent node, which will be near-feasible for the child nodes. Because we need only to determine feasibility status when examining the test MILPs, depth-first node selection may be preferred.

A number of branching variable selection schemes are possible, including use of estimates of the value of the objective function, a simple list ordering, or a user-defined priority weighted ordering. A scheme that chooses the variable that is most infeasible[1] is also available.

6. Empirical Tests

6.1 The Set of Test Models

Finding suitable test models proved difficult. While infeasible MILP problems were collected from a number of well-known researchers, most of these proved to have infeasible initial LP-relaxations and so were easily analyzed by the existing LP-infeasibility analysis methods. Only two of the test models provided to us met our requirements: *dell*, provided by Robert Dell of the Naval Postgraduate School, and *p0201i*, provided by Ed Klotz of CPLEX Optimization Inc. The source of the infeasibility is not known for these two problems.

To provide a larger test set, feasible models covering a range of sizes and difficulties were taken from a set of MILP problems maintained by Robert Bixby and Andrew Boyd (available via anonymous ftp at "softlib.cs.rice.edu"). A single LC was added to each problem that caused MILP infeasibility while preserving the feasibility of the initial LP-relaxation. The added LC is a conversion of the objective function to a constraint, with a right hand side value between the original MILP solution value and the initial LP-relaxation value, and with an appropriate constraint sense ($\leqslant$ for a minimization, $\geqslant$ for a maximization).

Table I summarizes the characteristics of the MILPs in the test set. Table II gives the time, the number of branch-and-bound tree nodes, and the total number of LP pivots required by the CPLEX 3.0 MILP solver to determine that the models were infeasible. *p0201i* was found to be infeasible by the CPLEX presolver. Default settings were used for CPLEX, except in the case of *dell*, which CPLEX found feasible unless the variables were rescaled to cause infeasibility. Times are measured on a Sun 10/30c computer equipped with a 36 MHz SPARC Sun 4 CPU and 33 Mb of memory.

6.2 Software Prototype

The various infeasibility isolation algorithms were implemented in the C language and make use of the CPLEX 3.0 callable library[8] to solve the intermediate test MILPs. CPLEX uses a branch-and-bound method to solve the MILPs. Algorithm 9, which makes use of information col-

Table VI. Results for the Dynamic Reordering Additive Method (Algorithm 6)

Model	IIS?	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a		7	76	51	18350	275009	9:54
bell3b		1	65	58	2062	129240	5:35
bell4		18	65	43	313958	691259	38:35
bell5		1	49	48	1062	65040	2:34
dell	y	1	33	825	952	137833	55:43
flugpl		8	13	9	1491902	931704	46:18
g503		1	18	17	320	7139	23
misc01		20	27	165	8512	98611	2:36
misc02		1	31	117	6809	65374	1:38
misc03		10	30	319	5561	124957	7:00
misc04	kil						
misc05		16	68	118	46986	2365221	1:47:25
misc06	kil						
mod008		2	7	638	4034	19558	1:05
mod013		16	50	54	25245	341871	8:31
p0201i		1	2	6	334	10658	57
stein9	y	6	13	0	1069	2435	7
stein15	y	6	36	0	5015	22464	52
stein27	y	14	106	0	278979	1446623	45:27
stein45	kil						
average		7.6	40.5	145.2	130067.6	396176.2	19:41

lected during the initial solution of the MILP, uses a modified version of MINTO 2.0[15] to extract the needed information from CPLEX. In all cases, the number of branch-and-bound tree nodes generated for any intermediate test MILP is limited to a maximum of 10,000.

The various algorithms are usually interested only in the feasibility status of a particular subproblem. For this reason, the software prototype returns when either 1) the first feasible solution is found (i.e., it does not continue to optimality in this case) or 2) the model is proven infeasible by a full expansion of the branch-and-bound search tree.

6.3 Deletion Filtering

Tables III and IV present the results for deletion filtering. The MILP solver uses a depth-first search with the default branching scheme in both cases. A column labeled *dubious* appears in all tables of results in which individual dubious constraints can be identified. The column entry has the form $a|b|c$ in which a denotes the number of dubious IRs, b denotes the number of dubious LCs, and c denotes the number of dubious BDs ($b|c$ is used when $a = 0$ and c is used when $a = b = 0$).

Tables III and IV show that the deletion filter operates very slowly on an infeasible MILP, principally because of the need to run repeated test MILP solutions. The average isolation times for both versions of the deletion filter are multiple hours as opposed to a few seconds for the initial detection of infeasibility. However, the tables also support our conjecture that eliminating IRs before the other constraints (Table III) will speed the isolation, reducing the average isolation time by 73% as compared to eliminating the LCs before the other constraints (Table IV). Eliminating IRs before LCs also reduces the average number of branch-and-bound tree nodes and the average number of LP iterations.

Tables III and IV also show that the identification of an IIS is not often guaranteed by the basic deletion filter when an upper limit is placed on the number of branch-and-bound nodes in any particular MILP test problem. In both cases, IISs can be proven for only 5 of the 20 models, and in some models the fraction of dubious constraints (especially BDs) is quite high. Note, however, that there is some worthwhile isolation of the infeasibility: Table III shows that Algorithm 3 eliminates 85% of the IRs, 35% of the LCs, and 38% of the BDs.

Experiments with safety bounding showed reductions in the isolation size for most models. However, the feasibility status of the output sets was not checked, so it is not known whether the output isolations were in fact infeasible.

6.4 The Additive Method

For the tests in this section, the original objective function was replaced by the objective of minimizing the sum of all variables that are integer-restricted.

Table V presents results for the basic additive method. Note that the operation of the algorithm was killed in three cases (*misc04*, *misc06*, *stein45*) due to excessive time (200, 65, and 30 hours, respectively). When it does find an isolation, the additive method is comparable to the (IR-LC-BD) deletion filter, requiring on average about as much time for the isolation (1:12:12 vs. 1:18:38 for the deletion filter over the same 17 models). The inability to terminate in a reasonable

Table VII. Results for the Dynamic Reordering Additive/Deletion Method

Model	Dubious	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a	3	8	76	51	27094	113103	3:59
bell3b	7	1	65	56	562	27994	26
bell4	1	15	65	43	785144	1795039	1:24:10
bell5	2	1	49	49	444	15882	14
dell	1	1	33	114	5718	4248	3:26
flugpl	5|4	8	13	6	429312	298729	13:28
g503	1	1	18	18	180	2806	3
misc01	7	23	25	77	12708	115698	2:20
misc02	4	11	30	55	4472	46460	46
misc03	20	8	29	112	16021	92117	3:13
misc04	3	3	1186	3732	22187	11112753	30:37:05
misc05	88	12	73	100	49412	1216934	48:35
misc06	6	4	605	1046	16178	3331787	6:42:38
mod008	none	2	7	638	5062	38946	1:39
mod013	6	15	50	56	18169	144123	2:47
p0201i	1	1	2	7	961	8953	29
stein9	none	6	13	0	1238	2721	4
stein15	none	7	36	0	4464	17154	21
stein27	2	14	109	2	132148	742034	21:04
stein45	5	16	225	9	958773	10632330	8:00:03
average	0.3|8.1	7.9	135.4	308.6	124512.4	1487990.6	2:25:21

amount of time is not unexpected for the additive method. As shown in Algorithm 5, Step 3 of the additive method begins without any BDs on the test MILP. This tends to lengthen the process of deciding feasibility status.

The major advantage of the additive method is the average reduction in the number of IRs in the isolation (8.9 vs. 11.8 for the deletion filter over the same 17 models). Table VI shows that the dynamic reordering version of the additive method gives even better results in terms of the number of IRs in the isolation, and is significantly faster than the basic additive method when it manages to terminate in a reasonable amount of time.

6.5 The Additive/Deletion Method

The results for the dynamic reordering version of the additive / deletion method (summarized in Table VII) are slightly better than those for the basic additive / deletion method (Algorithm 7; results not given here). The advantage of the additive / deletion method over the additive method is that it is able to provide an isolation for all of the test models within a reasonable time. There are two advantages of the additive / deletion method over the deletion filter: 1) it provides isolations having fewer IRs, which should be easier to interpret, and 2) it provides isolations having fewer dubious constraints on average.

6.6 Grouping Constraints

A number of constraint grouping methods were implemented and assessed (see Algorithm 8) for the deletion filter, the additive method, and the additive / deletion method. The additive method with grouping was not able to complete in a reasonable time for several of the models. The best method was the IR-LC-BD deletion filter with a fixed group size of four constraints; results are given in Table VIII.

6.7 Using the Initial Branch-and-Bound Tree

As described in Algorithm 9, information collected during the initial branch-and-bound solution of the MILP can be used to speed the subsequent infeasibility isolation. Results are summarized in Table IX. MINTO[15] is used as an interface to CPLEX for these trials because it allows the collection of the data needed by Algorithm 9. The *dell* model is omitted because CPLEX finds it to be MILP feasible without rescaling, but MINTO does not permit the necessary rescaling. The default MINTO settings are used. Times reported include the development of the initial branch-and-bound tree plus the time for the isolation of an IS.

Note also that 9 of the test models require numerous branch-and-bound nodes for the initial recognition of infeasibility under the assumed solver settings. Because a fair amount of information is collected about each node and path in the branch-and-bound tree, an arbitrary upper limit of 3000 initial branch-and-bound nodes is imposed to avoid exhausting the available memory. Models needing more nodes are omitted, leaving 10 test models. In practice, it is simple to abandon the collection of branch-and-bound tree data when a preset upper limit on nodes is reached, and to then revert to one of the methods that does not use data collected during the initial detection of infeasibility.

Table IX shows that using the information in the initial

Table VIII. Results for the IR-LC-BD Deletion Filter with Fixed Group Size of 4 Constraints

Model	Dubious	IRs	LCs	BDs	B&B Nodes	LP Iterns	Time (h:m:s)
bell3a	26	7	68	51	4223	29575	48
bell3b	49	8	73	65	818253	1401317	1:22:00
bell4	14	15	65	43	665881	724382	58:24
bell5	32	6	53	48	460922	956234	45:35
dell	none	5	377	596	8154	1205454	1:40:05
flugpl	6	6	6	6	334498	192472	9:33
g503	4	1	33	15	159	4726	5
misc01	1|25	12	28	65	3763	31519	44
misc02	1|12	14	30	54	5416	36774	44
misc03	1|40	3	31	110	1836	27447	56
misc04	294|2477	3	1186	3731	15274	7263993	19:16:31
misc05	4|32	7	62	55	3867	88612	3:18
misc06	26|686	11	613	1019	25901	2534418	4:37:11
mod008	293	73	7	313	377935	1305983	1:15:39
mod013	21	15	50	50	13639	63361	1:28
p0201i	5	1	2	6	288547	1986777	1:38:22
stein9	none	6	13	0	605	1315	2
stein15	none	6	36	0	2436	10401	13
stein27	none	14	109	0	76689	347969	10:59
stein45	none	16	227	4	683225	6708833	5:07:51
average	16.4|186.1	11.5	153.4	311.6	189561.1	1246078.1	1:51:31

Table IX. Results for Using Information from the Initial Branch-and-Bound Tree (Algorithm 9)

		After Initial B&B Solution			After Deletion Filter				
Model	Dubious	IRs (paths)	LCs	BDs	IR (s)	LCs	BDs	B&B nodes	Time (h:m:s)
bell3a	38|46	19 (6)	103	113	7	70	41	3838	4:07
g503	9|11	1 (1)	41	43	1	30	18	100	5
misc01	1|48	9 (1)	55	112	5	28	56	530	23
misc02	1|22	15 (1)	40	97	9	32	33	1428	29
misc03	1|88	3 (1)	96	171	3	22	105	570597	42:44
misc04	1104|3599	2 (1)	1486	4009	1	1169	3699	1569	7:25:54
misc05	5|40	14 (3)	287	153	10	74	58	20920	9:22
misc06	516|872	13 (2)	774	1561	3	602	1141	3017	1:22:35
mod013	2|26	20 (0)	63	120	14	49	49	10232	1:25
stein9	none	13 (4)	14	18	6	13	0	1073	6
average	167.7|475.2	10.9 (2.0)	295.9	639.7	5.9	208.9	520.0	61330.4	58:43

branch-and-bound tree can greatly speed the isolation of an infeasibility. This is especially evident for the difficult *misc04* model in which the isolation time is cut from a previous low of 19:16:31 (Table VIII) to 7:25:54, i.e., a cut of 61%. However, the added processing may increase the time required for other models (e.g., *bell3a* requires only 48 seconds in Table VIII, but needs 4 minutes and 7 seconds in Table IX.

Assuming that the collection of branch-and-bound information is abandoned when an upper limit on the number of nodes in the initial branch-and-bound tree is exceeded, we can derive results for the entire test set (including *dell*) by using the times from Table VIII where they are not provided in Table IX. The average results over the 20 models are then: 10.5 IRs, 152.2 LCs, 298.4 BDs, 216492.2 branch-and-bound nodes, time 1:08:39. This seems an acceptable amount of time in practice.

7. Conclusions

The experiments show that it is possible to find a useful infeasibility isolation in an infeasible MILP. There are two main drawbacks: the process may be fairly slow, especially for larger MILPs having many IRs, and the isolation returned is often an IS instead of an IIS if an upper limit is

imposed on the maximum number of nodes generated in the solution of any test MILP.

While the provision of an isolation having specific properties (i.e., one having few IRs and few LCs if possible) is certainly desirable, at the current state of the art it is better to simply concentrate on finding true IISs and on doing so reasonably quickly. On the test set used here, the dynamic reordering additive/deletion method isolates ISs having the fewest dubious constraints (Table VII) and the smallest ISs on average. This method is recommended when it is important to find isolations that are as close to a true IIS as possible.

Because all of the methods are fairly slow due to the inherent problems of deciding feasibility in a MILP, the speed of the isolation may be more important in some cases. The fastest method tested gives an average isolation time of 1:08:39 compared to an average time for the initial detection of infeasibility of about 6 seconds for the relatively small models in the test set. Much larger times can be expected for large industrial-scale MILPs. For the models tested here, the fastest algorithm is a combination of 1) using information from the original branch-and-bound tree (Algorithm 9), followed by 2) the *IR-LC-BD* deletion filter, with 3) constraint grouping with fixed group size of four constraints. This method is recommended when it is important to find an isolation quickly.

There are many opportunities for the extension of the algorithmic building blocks presented here. Example questions include: Can Theorem 2 can be extended to IRs by looking at the BDs added by the branch-and-bound method that are sensitive in the leafs? Is it useful to extract information from the branch-and-bound tree developed for each intermediate test MILP? A more thorough examination of various ideas is also in order: the effect of using safety bounds, the value of replacing the original objective function, and the choice of MILP solver settings.

Numerous other interesting algorithmic ideas deserve further investigation, including:

- Will a branching variable selection scheme that chooses the variable that is most infeasible[1] speed the analysis?
- The extension of ideas from elastic filtering[7] may prove useful. For example it may be possible to add weighted binary variables to each constraint such that activating the binary variable releases the constraint. Minimizing the sum of these added binary variables determines a maximum feasible subset of constraints, from which infeasible subsets can be created for further analysis by adding back one of the eliminated constraints.
- Is it possible to create improved grouping schemes using ideas from binary search?

Acknowledgments

Thanks are due to Ed Klotz (CPLEX Optimization Inc.) for the provision of a test problem and for helpful discussions in the course of the research. Thanks also to Robert Dell (Naval Postgraduate School) for the provision of a test problem. Suggestions made by the anonymous referees also strengthened the paper. This research is partially supported by a Research Grant to J. W. Chinneck provided by the Natural Sciences and Engineering Research Council of Canada.

References

1. R. BREU and C.A. BURDET, 1974. Branch and Bound Experiments in Zero-One Programming, *Mathematical Programming Study 2*, 1–50.
2. J.W. CHINNECK, 1994. MINOS(IIS): Infeasibility Analysis Using MINOS, *Computers and Operations Research 21*, 1–9.
3. J.W. CHINNECK, 1995. Analyzing Infeasible Nonlinear Programs, *Computational Optimization and Applications 4*, 167–179.
4. J.W. CHINNECK, 1996. Computer Codes for the Analysis of Infeasible Linear Programs, *Journal of the Operational Research Society 47*, 61–72.
5. J.W. CHINNECK, 1997. Feasibility and Viability, in *Advances in Sensitivity Analysis and Parametric Programming*, T. Gal and H.J. Greenberg (eds.), International Series in Operations Research and Management Science 6, 14-1 to 14-41, Kluwer Academic Publishers, Boston.
6. J.W. CHINNECK, 1997. Finding a Useful Subset of Constraints for Analysis in an Infeasible Linear Program, *INFORMS Journal on Computing 9*, 164–174.
7. J.W. CHINNECK and E.W. DRAVNIEKS, 1991. Locating Minimal Infeasible Constraint Sets in Linear Programs, *ORSA Journal on Computing 3*, 157–168.
8. CPLEX OPTIMIZATION INC., 1994. *Using the CPLEX Callable Library*.
9. R.E. GOMORY, 1963. An Algorithm for Integer Solutions to Linear Programs, in *Recent Advances in Mathematical Programming*, McGraw-Hill Book Company, New York.
10. H.J. GREENBERG, 1993. *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*, Kluwer Academic Publishers, Boston.
11. O. GUIEU, 1995. *Analyzing Infeasible Mixed-Integer and Integer Linear Programs*, M.Sc. thesis, Systems and Computer Engineering, Carleton University, Ottawa, Canada.
12. A.J. HOFFMAN and M. PADBERG, 1985. LP-Based Combinatorial Problem Solving, *Annals of Operations Research 4*, 145–194.
13. E.L. LAWLER, J.K. LENSTRA, A.H.G. RINOOY KAN, and D.B. SHMOYS, 1985. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons, New York.
14. K.G. MURTY, 1983. *Linear Programming*, John Wiley & Sons, New York.
15. G.L. NEMHAUSER, M.W.P. SAVELSBERGH, and G.C. SIGISMONDI, 1994. MINTO: A Mixed INTeger Optimizer, *Operations Research Letters 15*, 47–58.
16. R.G. PARKER and R.L. RARDIN, 1988. Heuristic Aspects of Branch and Bound, in *Discrete Optimization*, Academic Press, Boston.
17. M.W.P. SAVELSBERGH, 1994. Preprocessing and Probing Techniques for Mixed Integer Programming Problems, *ORSA Journal on Computing 6*, 445–454.
18. L. SCHRAGE, 1991. *LINDO: An Optimization and Modeling System, 4th edition*, The Scientific Press, San Francisco.
19. M. TAMIZ, S.J. MARDLE, and D.F. JONES, 1996. Detecting IIS in Infeasible Linear Programmes using Techniques from Goal Programming, *Computers and Operations Research 23*, 113–119.
20. M. TAMIZ, S.J. MARDLE, and D.F. JONES, 1995. Resolving Inconsistency in Infeasible Linear Programmes, Technical Report, School of Mathematical Studies, University of Portsmouth, U.K.
21. W.L. WINSTON, 1995. *Introduction to Mathematical Programming: Applications and Algorithms*, Duxbury Press, Belmont, CA.
22. L.A. WOLSEY, 1989. Strong Formulations for Mixed Integer Programming: A Survey, *Mathematical Programming 45*, 173–191.