INFORMS Journal on Computing
Vol. 9, No. 2, Spring 1997

0899-1499/97/0902-0164 $05.00
© 1997 INFORMS

Finding a Useful Subset of Constraints for Analysis in an Infeasible Linear Program

JOHN W. CHINNECK / *Systems and Computer Engineering, Carleton University, Ottawa, Ontario K1S 5B6, Canada, Email: chinneck@sce.carleton.ca*

(Received: March 1993; revised: May 1996; accepted: September 1996)

Infeasibility is often encountered during the process of initial model formulation or reformulation, and it can be extremely difficult to diagnose the cause, especially in large linear programs. While explanation of the error is the domain of humans or artificially intelligent assistants, algorithmic assistance is available to isolate the infeasibility to a subset of the constraints, which helps speed the diagnosis. The isolation should be infeasible, of course, and should not contain any constraints which do not contribute to the infeasibility. Algorithms for finding such irreducible inconsistent systems (IISs) of constraints have been proposed, implemented, and tested in recent years. Experience with IISs shows that a further property of the isolation is highly desirable for easing diagnosis: the isolation should contain as few model rows as possible. This article addresses the problem of finding IISs having few rows in infeasible linear programs. Theory is developed, then implemented and tested on a range of problems using a modified version of MINOS 5.4 called MINOS(IIS).

Infeasibility is frequently encountered during the initial formulation or reformulation of linear programs (LPs), and it is often extremely difficult to explain the cause. This has become more of a problem in recent years as computing technology has advanced to the point that very large LPs are routinely formulated and solved. Automated assistance in analyzing infeasibility is needed. Although the diagnosis of the error (i.e., the explanation in human terms) is the domain of humans or artificially intelligent assistants such as expert systems, algorithmic methods can help by isolating the infeasibility to a subset of the constraints (both row and column bounds) defining the model. This speeds the diagnosis.

This article addresses the problem of isolating a subset of constraints in an infeasible linear program that is useful for further analysis leading to a diagnosis of the cause. "Useful" in this context means an isolation which accelerates the process of arriving at a diagnosis of the infeasibility. A useful isolation should be infeasible, of course, and should not contain any constraints which do not contribute to the infeasibility. Algorithms for finding such minimal sets, known as Irreducible Infeasible Systems (IISs), have been proposed, implemented, and tested in recent years (see [7] and the review in Section 1). Greenberg[18] performed an empirical comparison of three methods of LP infeasibility analysis and concluded that the isolation of minimal infeasible sets "performed consistently above midrange, and it never failed to provide useful information. It frequently gave an immediate diagnostic." See also Greenberg[20] for further analysis of the value of isolating IISs during the diagnostic process. The ability to isolate IISs is now available in at least three commercial LP solvers, including LINDO,[26] CPLEX,[10] and IBM's OSL.[21]

Experience with IIS methods on a range of infeasible LPs has shown that numerous different IISs can usually be isolated for a single modeling error. Further, which of the several IISs is reported to the user often has a major impact on the speed of diagnosis. Experiments with users show clearly that the IIS having the smallest number of rows is the most useful. For example, one LP showed two IISs: one involving 12 rows (of 2393 bounded rows) and 68 columns, and another involving one row and 93 columns. Although the first IIS is smaller in terms of the total number of bounds involved, the second is much easier to interpret and to diagnose. In another example, one IIS involved all 323 of the bounded rows, whereas another involved only 76, effectively confining further analysis to about one quarter of the original model.

It is not surprising that analysts prefer IISs having few rows. Column bounds are easy to understand, but rows tie together both variables and other rows in complicated ways. Limiting the number of rows reduces the complexity of the subsequent analysis. A small number of rows helps to pinpoint part of the model or a class of constraints for further analysis, e.g., the blending units, or perhaps the crude oil supply limits. The IIS identifies a set of constraints that all contribute mutually to the infeasibility; the analyst must decide which of the constraints is in error, or if none is in error, must conclude that the model really is infeasible for physical reasons. Fewer rows to examine makes this task much easier. Variable bounds are rapidly verified and are therefore of less consequence to the analysis process. The analyst would rather accept more variable bounds in the IIS in return for fewer rows. In general, minimizing the number of rows in an IIS also tends to reduce the number of variable bounds involved because the smaller number of rows interacts with fewer variables. A secondary effect of minimizing the number of rows is thereby to minimize the total size of the IIS.

The issue of finding IISs having a small row-cardinality

Subject classifications: Linear programing.
Other key words: infeasible.

was first raised by the author in a 1993 technical report.[4] Delayed in publication, this paper updates that 1993 report, taking into account further developments in the intervening years, notably the introduction of two new methods of isolating IISs which show promise in finding IISs having few rows. In addition, a recent article by the author partially addresses the issue of finding IISs having few rows in the context of an empirical comparison of computer codes for infeasibility analysis.[8] Before the 1993 work, IIS isolation algorithms were undirected: the first IIS found was reported, regardless of its row cardinality. Worse, the initial emphasis in the research was on finding IISs quickly, but as this article shows, the quickest algorithms for finding IISs most often return IISs having many rows.

One approach to finding the minimum row-cardinality IIS is to enumerate all of the IISs in the model, then choose the one having the fewest rows. Unfortunately, Chakravarti[3] shows that the number of IISs in an infeasible LP could be exponential in the worst case. This means that, in general, the minimum row-cardinality IIS cannot be found in polynomial time by enumeration methods. This article takes a different approach, abandoning any attempt to guarantee the isolation of the minimum row-cardinality IIS in favor of heuristic methods which try to directly isolate a single IIS having few rows.

The design of the methods, and their implementation in MINOS(IIS) version 4.0, is described in Section 2. A range of methods is applied to a set of test problems in Section 3, and their relative effectiveness in finding IISs having few rows is evaluated. Both the number of rows in the IISs and the solution times are considered in arriving at recommendations.

Some basic definitions follow.

Constraint is a generic term referring to any limitation placed on the model, including both *rows* (also known as *functional constraints*) and *column bounds*. The term *Irreducibly Inconsistent System (IIS)* of constraints, introduced by Carver[2] and subsequently used by van Loon,[28] means a set of constraints which is infeasible, but which becomes feasible if any one constraint is removed. It is minimal in that sense. Chinneck and Dravnieks[9] extended the theory of IISs in a series of further definitions. An *Irreducibly Inconsistent System of Functional constraints (IISF)* is the complete subset of functional constraints (rows) in an IIS. In other words, delete the listing of column bounds in the IIS to find the IISF. If the complete IIS contains column bounds, then the IISF derived from it is not infeasible. An IIS is *overlapped* if it shares at least one constraint with another IIS. A *cluster* of IISs is a maximal set of IISs constructed from a single IIS by iteratively adding all other IISs that overlap at least one other IIS in the set.

1. Isolating IISs

For a complete review of the theory and history of isolation of IISs in all types of mathematical programs, see Chinneck.[7] We review here only the methods that have been implemented to date, and thus are available for empirical evaluation.

Algorithm 1 The Deletion Filter

```
INPUT: an infeasible set of linear constraints.
FOR each constraint in the set:
  1. temporarily drop the constraint from the set.
  2. test the feasibility of the reduced set:
       IF feasible THEN
         return dropped constraint to the set.
       ELSE (infeasible)
         drop the constraint permanently.
END FOR.
OUTPUT: constraints constituting a single IIS.
```

1.1 The Filtering Algorithms for Isolating IISs

The filtering algorithms for isolating IISs are briefly reviewed below; see Chinneck and Dravnieks[9] and Chinneck[5] for full details. Characteristics of the algorithms relevant to finding IISs having few rows are elucidated.

The *deletion filter* presented in Algorithm 1 guarantees that only the members of a single IIS are isolated. It requires the solution of a phase 1 LP for each finite bound in the model, including variable bounds. Because the final basis of the previous deletion iteration is used as the initial basis for the next iteration, this is not as slow in practice as might be expected. Table II gives some results.

For our purposes, it is important to note that the order in which the constraints are tested affects which IIS is found. Consider a set of constraints denoted by {A, B, C, D, E, F} in which there are two IISs: {A, B} and {C, E, F}. If the constraints are tested in the order A to F, then the IIS {C, E, F} will be found. If the constraints are tested in the order F to A, then the IIS {A, B} will be found. The rule is: the IIS whose *first* constraint is tested *last* will be identified. This is because the other IISs can be removed en route and the LP will remain infeasible.

The *sensitivity filter* is applied to the final basis of an infeasible phase 1 LP and helps accelerate the isolation. Any nonbasic variable having a reduced cost or shadow price of zero identifies a constraint which does not contribute to the phase 1 objective function, i.e., does not contribute to the infeasibility. These constraints can then be eliminated because they are not part of the IIS being isolated. The sensitivity filter is highly effective in rapidly eliminating numerous constraints, but the deletion filter must be applied to the output to guarantee the identification of a single IIS.

The sensitivity filter selects from among the IISs in the model, regardless of the constraint ordering. Which IIS is selected depends on the details of the phase 1 method in the LP solver, but the underlying dynamics are the same. All phase 1 LPs operate by allowing extra degrees of freedom to the constraints, either by the addition of artificial variables (as in a textbook phase 1) or by allowing literal bound violations (as in MINOS). This can be viewed as a constraint shift (or *stretch*) in the original variable space. In minimizing a phase 1 objective function, the minimum sum of constraint stretches is sought. If the sum is zero, then the LP is feasible. However, if the sum is *not* zero, then the LP is infeasible,

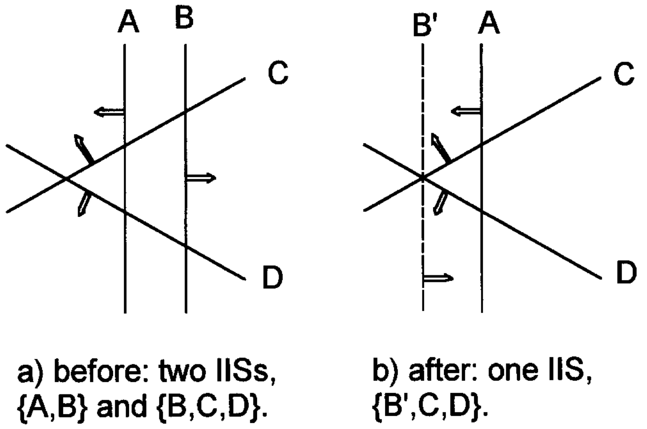
A
B
C
D
B'
A
C
D
a) before: two IISs, {A,B} and {B,C,D}.
b) after: one IIS, {B',C,D}.

Figure 1. The sensitivity filter may bypass smaller IISs.

and the final phase 1 basis has some stretched constraints. A sensitivity filter applied to this final basis identifies any stretched constraints and the constraints which are forcing them to stretch, because these are the constraints to which the phase 1 objective is sensitive. Because some constraints have stretched out of their original positions, other IISs, perhaps with smaller row cardinality, may be bypassed as illustrated in Figure 1. Fig. 1*b* shows how constraint B has stretched to minimize the phase 1 objective. The subsequent sensitivity filter applied to the final basis illustrated in Fig. 1*b* removes constraint A entirely because the phase 1 objective is not sensitive to it. The IIS {B, C, D} will be found instead of the smaller IIS {A, B}.

As shown empirically in Section 3, where there is more than one IIS, the sensitivity filter regularly isolates the IISs with more rather than fewer rows. This is probably because the overlapped constraints in a cluster are the cheapest to stretch because they eliminate more than one IIS at a time, and the larger IISs require a larger stretch on average, so the smaller IISs are bypassed.

A good acceleration of the IIS-isolation process is achieved by the *deletion/sensitivity filter* which combines the two by applying a sensitivity filter to the infeasible deletion filter iterations, as shown in Algorithm 2. This is the default algorithm used in MINOS(IIS) version 3.2.

The *reciprocal filter* applies where rows or variables have distinct upper and lower bounds and relies on the following simple theorem: In the absence of simple upper and lower bound reversal, if a row or column has distinct upper and lower bounds and one of the bounds is involved in an IIS, then the other bound cannot be involved in the same IIS. Thus as soon as one of the bounds is identified as belonging to an IIS, then the other bound can be discarded. MINOS(IIS) version 3.2 (and earlier) counted this as a special case of the elastic filter, but version 4.0 enumerates reciprocal filter eliminations explicitly.

The *elastic filter* uses the constraint-stretching concept of "elastic programming"[1] which is closely related to phase 1 ideas. Nonnegative "elastic variables," e_i are added to the constraints to provide elasticity,

Algorithm 2 The Deletion/Sensitivity Filter

```
INPUT: an infeasible set of linear constraints.
FOR each constraint in the set:
  1. temporarily drop the constraint from the set.
  2. test the feasibility of the reduced set:
     IF feasible THEN
       return dropped constraint to the set.
     ELSE (infeasible)
       i. drop the constraint permanently.
       ii. apply the sensitivity filter.
END FOR.
OUTPUT: constraints constituting a single IIS.
```

Algorithm 3 The Elastic Filter

```
INPUT: an infeasible set of linear constraints.
1. Make all constraints elastic by adding nonnegative elastic
   variables.
2. Solve LP using elastic objective function.
3. IF feasible THEN
     Enforce the constraints in which any $e_i>0$ by
       permanently removing their elastic variable(s).
     Go to step 2.
   ELSE (infeasible)
     Exit.
OUTPUT: the set of enforced constraints contains at least one
  IIS.
```

nonelastic constraint	elastic constraint
$\sum_j a_{ij}x_j \geq b_i$	$\sum_j a_{ij}x_j + e_i \geq b_i$
$\sum_j a_{ij}x_j \leq b_i$	$\sum_j a_{ij}x_j - e_i \leq b_i$ (1)
$\sum_j a_{ij}x_j = b_i$	$\sum_j a_{ij}x_j + e_i' - e_i'' = b_i$

Constraints stretch against the resistance provided by the elastic objective of minimizing the sum of the nonnegative elastic variables, which replaces the original objective function. As shown in Algorithm 3, the elastic filter solves the elastic LP (which will of course be feasible), and then removes the elastic variables from any constraint in which the elastic variables are positive (i.e., *enforces* the constraint) in a process of gradually "tightening" the LP. The cycle of solution and enforcement continues until the LP becomes infeasible, and the algorithm terminates. The output set of enforced constraints contains at least one IIS.

Because of the elastic objective function, constraints will stretch only if they must (i.e., they are part of an infeasibility) in order to achieve the minimum elastic objective function value. For this reason, the maximum number of iterations of the elastic filter is limited by the cardinality of the smallest IIS because at least one constraint from the smallest IIS must stretch in each elastic iteration. This property is important to finding small row-cardinality IISs.

If exactly one constraint in each IIS stretched in each iteration of the elastic filter, then the number of elastic iterations would exactly equal the cardinality of the smallest IIS. The elastic filter output set would then contain this minimum-cardinality IIS plus parts of larger IISs which would subsequently be eliminated by deletion and/or sensitivity filtering, leaving the minimum-cardinality IIS as the final result.

In practice, the attainment of the minimum value of the elastic objective function in an elastic filter iteration can stretch more than one constraint per IIS. However, experimentation (see Section 3) shows that the number of constraints stretched per elastic iteration is relatively small, and that the elastic filter is a remarkably effective heuristic for finding small-cardinality IISs.

1.1.1 The MINOS(IIS) Software

MINOS(IIS) is a version of MINOS 5.4[23] modified at Carleton University to incorporate the filtering algorithms for isolating IISs. See [5] for details. MINOS(IIS) operates exactly like MINOS 5.4, unless infeasibility is detected, in which case the IIS-isolating routines are called. MINOS converts the input LP to a bounded variable form, so constraint removal is easily accomplished by removing the bounds on the variables. The repeated phase 1 solutions needed by the deletion filter are considerably speeded by built-in facilities for basis reuse.

The phase 1 in MINOS is nonstandard. Two progress variables are tracked: the sum of the infeasibilities (i.e., the usual phase 1 objective function), and the number of infeasibilities (i.e., the number of violated constraints). Infeasibility is determined when MINOS recognizes that the number of infeasibilities cannot be made zero, even though another basis with a lower value of the sum of infeasibilities may exist. Nazareth[24] shows that this kind of phase 1 (known as "FORMC") has reduced costs which reflect the rate of change of the sum of infeasibilities when a nonbasic variable is introduced into the infeasible basis. In other words, the sensitivity filter operates correctly.

Version 3.2 of MINOS(IIS) incorporated the deletion, sensitivity, deletion/sensitivity, and reciprocal filters. The method could be selected under user control, with deletion/sensitivity as default. The user could also select the direction of the deletion filter, either forward (variable bounds followed by row bounds) or reverse (vice-versa), with reverse as default. Version 3.2 also included user-specifiable codes for guiding the search by placing constraints in categories, notably "encouraged to drop" and "discouraged from dropping." These codes were used internally to, in effect, reorder the deletion filter, placing the "encouraged to drop" constraints early in the list and placing the "discouraged from dropping" later in the list and protecting them from sensitivity filtering.

MINOS(IIS) version 4.0 was developed specifically for the research in this article and included various algorithms for isolating IISs having few rows. This included the first implementation of the elastic filter, and various column protection options, as described in Section 2. MINOS(IIS) is currently at version 5.0, which adds the ability to identify small-cardinality IIS set covers.[6]

Algorithm 4 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.
```

1.2 The Additive Algorithm

Tamiz et al.[27] introduced a method whose main feature is the *adding in* of constraints as the algorithm proceeds, exactly the opposite of the approach taken in the deletion filter. For this reason, it is referred to here as the *additive algorithm*, though Tamiz et al. refer to it as GPIIS because the original statement of the algorithm features the use of ideas from goal programming. However, the algorithm is easily simplified, as shown in Algorithm 4. See Chinneck[7] for a proof that the additive algorithm returns a single IIS.

As described by Tamiz et al.,[27] the additive algorithm is applied solely to the rows in the LP (column bounds are never removed), so the output of the procedure is an IISF rather than an IIS. In theory, however, the method could easily be extended to find complete IISs (see [7] for further details).

The additive algorithm isolates the IIS whose *last* member is tested *first*. This is because the loop in Step 1 of Algorithm 4 is exited the first time that T becomes infeasible, or equivalently, the first time that a complete IIS is in T. Thus, although parts of various IISs may be added to T as it builds up, the process exits only when the last member of any IIS is added to T. Thus, as for the deletion filter, which IIS is isolated by the additive method is affected by the ordering of the constraints.

Compared to the deletion filter, the additive method may require fewer tests of feasibility, and tests of smaller sets of constraints, especially when the IIS that is isolated is small relative to the cardinality of C. Tamiz et al.[27] report good results in finding IISs having few rows; their results are included in Table II for comparison.

1.3 Simplex Pivoting to Isolate IISs

Gleeson and Ryan[12] describe an algorithm for isolating IISs that uses a transformation of the problem to obtain a polytope in which, in the absence of degeneracy, each new pivot gives a new IIS. Their method is based on an efficient algorithm by Dyer[11] for enumerating all of the bases of a polytope. Parker and Ryan[25] develop modifications of

Gleeson and Ryan's method by showing that IISs can be identified based on the extreme rays of a conical construction. Parker and Ryan also implement both methods. These methods could be used to find the minimum row-cardinality IIS by enumerating all IISs and choosing the one having the fewest rows.

Parker and Ryan developed and implemented their method for the purpose of finding the minimum-cardinality IIS set cover (the minimum cardinality set of constraints which, when removed from the model, renders it feasible). Their method requires the identification of numerous IISs en route to finding the minimum cardinality IIS set cover, and they report the number of rows in the smallest row-cardinality IIS found during this process. These results are included in Table II for comparison. Better results could likely be obtained were their method tuned to the problem of finding the minimum row-cardinality IIS, but this has not yet been done.

1.4 Greenberg's Diagnostic Heuristics

Greenberg[13–17] described a set of manipulations of the model (such as bound tightening, path and cycle tracing, removal of redundancies, etc.) which can then be traced if infeasibility is found. The goal of the heuristic procedures is to provide a diagnostic explanation of the cause of the infeasibility. The process sometimes isolates an IIS, but cannot guarantee such a result. The method has been incorporated in Greenberg's ANALYZE software.[16, 19] ANALYZE is an ideal tool for use in concert with LP solvers which isolate IISs, since the effectiveness of the diagnostic procedures in ANALYZE is boosted by applying them to an IIS rather than to the entire model.

2. Isolating IISs Having Few Rows

Given that the enumeration of IISs to find the minimum row-cardinality IIS is combinatorially explosive, we concentrate below on heuristic methods for directly isolating a single IIS having few rows. The filtering algorithms are modified and extended for this purpose.

2.1 Simple Constraint Ordering in Deletion Filtering

Because the order of the constraints in the deletion filter list affects which IIS is found, simple constraint ordering can be used to influence the isolation. Because we wish to find IISs having few rows, we should try to eliminate rows first. For this reason, reverse-deletion filtering is preferred in MINOS(IIS) because this tests the row constraints first. Section 3 shows that reverse-deletion filtering is much more successful than forward-deletion filtering in identifying IISs having few rows.

2.2 Column Bound Protection in Deletion/Sensitivity Filtering

The simple constraint ordering discussed above works by making it easier to eliminate rows than column bounds. An extension of this idea is to provide specific protection to the column bounds, regardless of what other filtering methods are in operation. Two versions of column protection were first implemented in MINOS(IIS) version 4.0:

- *CP1:* all of the column bounds are protected from sensitivity filtering at all times, and the column bounds are automatically inserted at the end of the deletion filtering list, regardless of the order in which the row bounds are deletion tested.
- *CP2:* identical to CP1 except that the sensitivity filter is enabled for the column bounds after all of the rows have been deletion tested. The sensitivity filter is then applied to the column bounds regardless of whether it was applied to the rows.

CP1 is a very conservative strategy. CP2 recognizes that the number of rows in the IIS has been established by the time the column bounds are to be deletion tested, so the sensitivity filter can be used to speed the isolation without affecting the number of rows in the IIS. CP2 is used exclusively in the tests in Section 3.

2.3 Elastic Filtering

The tendency of elastic filtering to find small-cardinality IISs can be used to find IISs having few rows by restricting the elasticity to the row constraints. Only row constraints are elasticized and subsequently enforced; all column bounds are in force at all times, a form of column protection. The elastic filter is applied to the model first, before any other method, particularly the sensitivity filter which tends to bypass IISs having few rows.

The implementation of this strategy in MINOS(IIS) version 4.0 required the user to request space for the additional columns and elements needed by the elastic variables through the PHANTOM COLUMNS and PHANTOM ELEMENTS facility in MINOS. One elastic column is needed for each inequality row, and two for each row with both an upper and a lower bound, including equality rows. The number of new elements needed is twice the number of elastic columns (i.e., one element for the row, one element for the elastic objective function). The elastic objective function overwrites the original objective.

Deelasticization (enforcement) is achieved by setting both the upper and lower bounds on the appropriate elastic variables to zero. On exit from the elastic filter, all rows that are still elastic are eliminated from the model, and the columns associated solely with those elastic rows are removed from the remainder. The remaining rows and columns are then simply a portion of the original model and can be processed in the usual way.

The elastic filter is most effective if a minimum number of constraints stretch in each iteration. Because the elastic objective function weights each constraint equally, there is no penalty for stretching more than the minimum number where all contribute equally to the elastic objective function value. A heuristic for minimizing the number of stretched constraints is outlined below.

The idea is to use a different elastic objective function parameter for the elastic variable(s) for each row. This makes it better to stretch some constraints than others. Where there is a tie, this should encourage only one constraint in each IIS to stretch in a single elastic filtering iteration, i.e., the constraint having the smallest objective function parameter.

One simple way to assign distinct elastic objective function parameters is to increment each parameter by a small step, such as 0.1, giving values of 1.0, 1.1, 1.2, etc. Tests in Section 3 compare this heuristic with the baseline method in which all elastic objective function parameters are 1.0 (i.e., an increment of 0.0). Random assignment of distinct objective function parameters is probably preferable, but was not implemented. Preliminary tests with a variation in which the parameters were incremented by 0.1 but reset to 1.0 after every hundred increments were not promising.

The elastic filter as implemented in MINOS(IIS) version 4.0 is likely to be relatively slow because a conservative strategy is used in which the availability of an advanced basis to begin each iteration of the elastic filter is not assumed due to the increased "tightness" of the bounds as each iteration begins.

2.4 Complete Methods

To guarantee the positive identification of a single IIS, all complete methods must include the deletion filter, however various combinations of deletion-list ordering, column-protection options, and inclusion or exclusion of the sensitivity and elastic filters are possible. On theoretical grounds, the reverse-deletion list ordering, the column-protection options, the inclusion of the elastic filter, and the exclusion of the sensitivity filter are most likely to help isolate IISs with few rows. These theories are tested below.

3. Experiments

Ten IIS-isolation methods were applied to 14 test problems to determine their relative effectiveness in isolating IISs having few rows. In addition, results obtained by Tamiz et al.[27] using the additive algorithm and Parker and Ryan[25] using their simplex pivoting method are included for comparison. The number of rows in the IISs and the time taken to find them are evaluated in the tests.

Where the column protection options are tested, CP2 is used exclusively because it is faster than CP1 and the number of column bounds included in the IIS is irrelevant after the number of rows is set.

3.1 IIS-Isolation Methods Tested

A brief rationale for the inclusion of each method in the test set is given below.

- *Forward-Deletion Filter:* Expected to give the worst performance, this method is included as a baseline.
- *Reverse-Deletion Filter:* Expected to show the value of simple constraint ordering by giving better results than the forward-deletion filter.
- *Forward-Deletion Filter with CP2:* Because the column bounds are protected, this should give results comparable to the reverse-deletion filter, even though a forward-deletion direction is used.
- *Reverse-Deletion Filter with CP2:* Included for comparison with forward-deletion filter with CP2. Results expected to be similar. Results should be identical to reverse-deletion filtering because the CP2 option adds no extra column protection in this case.
- *Reverse-Deletion/Sensitivity Filter:* The default method used in MINOS(IIS) version 3.2.
- *Reverse-Deletion/Sensitivity Filter with CP2:* To provide column protection within a very quick method. Should improve the reverse-deletion/sensitivity filter results.
- *Elastic Filter (0.1) followed by Reverse-Deletion/Sensitivity Filter:* Test of the elastic filter including the 0.1 objective function parameter increment to discourage constraint stretches during elastic filtering. The elastic filter is coupled with the standard method used in MINOS(IIS) version 3.2 to complete the positive identification of a single IIS.
- *Elastic Filter (0.1) followed by Reverse-Deletion/Sensitivity Filter with CP2:* As above, plus a test of the effect of including the second-column protection option.
- *Elastic Filter (0.0) followed by Reverse-Deletion/Sensitivity Filter:* Here, the elastic filter has no incrementing of the objective function parameters (i.e., an increment of 0.0). This method is included for comparison to the 0.1-increment heuristic for minimizing constraint stretches during elastic filtering.
- *Elastic Filter (0.0) followed by Reverse-Deletion/Sensitivity Filter with CP2:* As above, for comparison with the heuristic for minimizing constraint stretches during elastic filtering.

3.2 Test Problems

The test problems cover a range of sizes and complexities, as summarized in Table I, which is arranged in order of increasing total number of MINOS bounds. The total number of rows (columns), bounded rows (columns), and MINOS row (column) bounds are listed, as are the total number of MINOS bounds, the total number of elements, and the number of seconds required to first determine infeasibility. The last column in Table I gives a lower bound on the total number of different IISs in the model; the exact number can be found only by a complete enumeration of all of the IISs in the model. The bound is found by choosing the largest from among the following numbers: (i) the maximum number of IISs generated by Parker and Ryan[25] using any of their three methods, and (ii) the number of distinct IIS row cardinalities given in Table II.

These test problems are a subset of the current netlib set of infeasible test problems (in the lp/infeas directory). The netlib set was originally established by the author using the models in Table I collected for this research, and later augmented by other models. As shown in Table II, with the exception of `cplex1`, all of the models have IISs of various row-cardinalities, so the set should provide a reasonable test of the ability of the methods to distinguish among IISs of different row-cardinalities.

Table I. Test Problem Characteristics

Problem	Number of Rows	Number of Bound Rows	Number of Row Bounds	Number of Columns	Number of Bound Columns	Number of Column Bounds	Total Bounds	Number of Elements	Phase 1 (secs)	Lower Bound on Number of IISs
itest6	12	11	13	8	8	8	21	23	0.4	6
galenet	9	8	10	8	8	16	26	16	0.4	3
woodinfe	36	35	70	89	89	103	173	209	1.0	2
forest6	67	66	96	95	95	100	196	270	1.4	5
refinery	324	323	646	464	464	714	1360	1694	35.4	38
qual	324	323	646	464	463	714	1360	1714	49.7	8
vol1	324	323	646	464	464	714	1360	1714	42.1	13
chemcom	289	288	552	720	720	864	1416	2190	42.1	3
reactor	319	318	465	637	637	1187	1652	2995	25.3	2
mondou2	313	312	624	604	604	1208	1832	1623	5.3	14
pilot4i	411	410	697	1000	912	1189	1886	5145	215.6	2
cplex1	3006	3005	4007	3221	3221	3439	7446	10664	565.1	1
greenbea	2505	2393	4595	5405	5401	5807	10402	35159	1183.8	4

3.3 Results

The results obtained on the test problems using the various IIS-isolation methods are summarized in Table II. The top number in each cell is the number of rows in the IIS (smallest number in each row is in boldface). The bottom number in the cells for the 10 filter-based methods is the ratio of the number of seconds to find the IIS to the number of seconds to complete the phase 1 solution which originally signaled infeasibility (smallest time ratio in each row is in boldface). Use of the time ratio assists in comparisons between implementations of the algorithms in different solvers and on different platforms. All time measurements are approximate only, due to disk-access and screen-write times. Note also that the absolute times for the solutions of a number of the smaller models are very small (tenths of a second), so the timing measures and ratios are not very accurate in these cases. The filtering methods were tested using a 25 MHz 80386/80387 PC clone (80486 PC clone for cplex1 and greenbea problems). MINOS(IIS) was compiled using Lahey Fortran version 3.01.

The results for the additive algorithm are as reported by Tamiz et al.[27] using an implementation of their algorithm in the FortLP solver[22] and solved on a 486/33 MHz PC. A forward ordering of the set of constraints is used. Tamiz et al. report on three variations of their method; the results for the best of the methods (called the *implicit method*) are given in Table II. Recall that Tamiz et al. find only IISFs in their experiments. For this reason, their time-ratio results are reported in Table II, but are not compared with the time ratios for the other methods, which reflect the time needed to isolate complete IISs.

The results for the method of Parker and Ryan are as reported in [25] using an implementation in the OSL solver,[21] though insufficient information to calculate the time ratio is provided. Recall that the intent of Parker and Ryan's method is to find IIS covers, rather than to find IISs having few rows; however their method enumerates many IISs, and so may find IISs having few rows. Parker and Ryan examine three variations of their method; the number of rows in the IIS having the fewest rows for each method is reported.

The next to last row in Table II reports, for each method, the average deviation from the smallest number of IIS rows found for each model. This is a measure of the effectiveness of the method in finding IISs having few rows. For example, forward-deletion filtering finds IISs having, on average over this test set, 37.1 more rows than the number of rows found by the best method. The last row in Table II reports, for each of the filtering methods, the average deviation from the smallest time ratio reported for each model. This is a measure of the speed of the method.

Using the next to last row in Table II, the methods can be rank-ordered on their ability to find IISs having few rows over the test set, as shown in Table III, which is subordered on the average deviation from the least-time ratio for each model (i.e., subordered on speed). Table III has four clear divisions:

- Class A: methods ranked 1–5 having average deviations from the least rows of only 0.4–0.6.
- Class B: methods ranked 6 and 7 having average deviations from the least rows of 1.1 and 1.3.
- Class C: methods ranked 8 through 12 having average deviations from the least rows of 4.2 to 5.7.
- Class D: methods ranked 13 and 14 having average deviations from the least rows of 26.3 and 37.1, clearly far worse than the other methods.

Table IV shows a ranking of the filtering methods by average deviation from the least-time ratio (i.e., ranked by speed, fastest to slowest). The implicit additive method and Parker and Ryan's methods are not included in Table IV because time-ratio data for these procedures is unavailable.

Table II. Number of Rows in IIS and Time Ratio

Model	Testing Methods* (1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	Implicit Additive Method	Parker and Ryan
itest6	3	**2**	3	**2**	4	**2**	3	3	**2**	**2**	**2**	3/3/3
	0.3	0.5	0.3	0.3	**0.1**	0.3	1.5	1.1	1.3	1.1	1.0	
galenet	7	**2**	**2**	**2**	3	**2**	3	**2**	**2**	**2**	**2**	3/†/3
	0.4	0.4	0.3	0.3	0.3	**0.2**	1.0	0.8	1.0	0.7	1.0	
woodinfe	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1/1/1**
	3.2	3.4	0.9	1.0	**0.2**	**0.2**	1.0	1.1	1.1	1.0	0.2	
forest6	66	**55**	59	**55**	66	**55**	64	58	**55**	**55**	**55**	59/59/64
	8.5	7.8	5.3	5.8	**3.5**	5.7	12.5	13.4	13.1	12.6	11.1	
refinery	158	**47**	55	**47**	67	57	78	50	51	52	**47**	61/63/62
	8.0	6.3	3.0	2.6	**0.9**	1.5	5.5	5.6	5.3	5.4	1.4	
qual	323	78	**76**	78	125	94	96	82	78	79	78	92/82/90
	4.1	5.3	2.1	2.3	**1.1**	1.9	4.9	4.9	4.8	5.0	2.7	
vol1	122	83	84	83	134	106	84	81	**80**	**80**	83	91/91/104
	6.2	5.1	2.1	2.2	1.8	**1.7**	5.0	5.1	5.3	5.5	2.5	
chemcom	45	**7**	**7**	**7**	153	**7**	**7**	**7**	**7**	**7**	**7**	15/13/15
	5.9	6.7	2.3	2.0	1.4	**0.7**	1.6	1.6	1.3	1.5	1.0	
reactor	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1/1/1**
	11.1	11.4	2.5	2.6	**0.2**	0.3	0.7	0.8	0.7	0.7	0.0	
mondou2	31	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	17/17/17
	41.6	44.0	10.6	10.3	**1.1**	1.4	7.5	7.7	6.4	8.8	1.9	
pilot4i	**1**	**1**	**1**	**1**	37	**1**	**1**	**1**	**1**	**1**	**1**	**1/1/1**
	2.5	3.4	0.8	1.0	0.3	**0.2**	0.3	0.3	0.4	0.4	0.9	
cplex1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5/†/5**
	3.2	3.2	1.4	1.6	**0.2**	0.3	1.1	1.1	1.0	1.2	1.5	
greenbea	12	**1**	**1**	**1**	24	**1**	**1**	**1**	**1**	**1**	**1**	**2/2/1**
	2.5	3.2	0.8	0.8	**0.1**	**0.1**	0.2	0.2	0.2	0.2	0.1	
avg. dev. from least rows	37.1	0.4	1.3	0.4	26.3	4.2	5.1	1.1	0.5	0.6	0.4	4.5/4.3/5.7
avg. dev. from least time ratio	6.7	7.0	1.7	1.8	0.1	0.3	2.5	2.5	2.5	2.6		

*(1) Forward-deletion filter; (2) Reverse-deletion filter; (3) Forward-deletion filter with CP2; (4) Reverse-deletion filter with CP2; (5) Reverse-deletion/sensitivity filter; (6) Reverse-deletion/sensitivity filter with CP2; (7) Elastic filter (0.1) followed by Reverse-deletion/sensitivity filter; (8) Elastic filter (0.1) followed by reverse-deletion/sensitivity filter with CP2; (9) Elastic filter (0.0) followed by reverse-deletion/sensitivity filter; (10) Elastic filter (0.0) followed by reverse-deletion/sensitivity filter with CP2.

†Method not applied.

3.4 Discussion

Table II shows that the best result for each model provides a very good localization of the infeasibility. On average, the best result includes only 15% of the rows in the entire model, an elimination of 85% of the model from further consideration. However, it is not known whether the best result found for each model has the minimum row-cardinality, so it is possible that these results can be improved even further.

Table III shows that several methods, those in Class A, give excellent results over the test set. The methods ranked 1–3 have identical deviations from the least number of rows, so the ordering shown in Table III is based on the speed subordering.

In particular, Table II shows that the top three methods in Table III give identical numbers of IIS rows over all of the models in the test set. Because the implicit additive method

Table III. Methods Ordered by Average Deviation from Least Rows

Method	Average deviation from least rows	Average deviation from least time ratio
1. reverse-deletion CP2	0.4	1.8
2. reverse-deletion	0.4	7.0
3. implicit additive method	0.4	n.r.
4. elas (0.0) rev-de/se CP2	0.5	2.5
5. elas (0.0) rev-del CP2	0.6	2.6
6. elas (0.1) rev-de/se CP2	1.1	2.5
7. forward-deletion CP2	1.3	1.7
8. reverse-de/se CP2	4.2	0.3
9. Parker and Ryan 2	4.3	n.r.
10. Parker and Ryan 1	4.5	n.r.
11. elastic (0.1) rev-de/se	5.1	2.5
12. Parker and Ryan 3	5.7	n.r.
13. reverse-de/se	26.3	0.1
14. forward-deletion	37.1	6.7

n.r.: not reported

Table IV. Methods Ranked by Average Deviation from Least Time Ratio

Method	Average deviation from least time ratio	Average deviation from least rows
1. reverse-de/se	0.1	26.3
2. reverse-de/se CP2	0.3	4.2
3. forward-deletion CP2	1.7	1.3
4. reverse-deletion CP2	1.8	0.4
5. elas (0.0) rev-de/se CP2	2.5	0.5
6. elas (0.1) rev-de/se CP2	2.5	1.1
7. elas (0.1) rev-de/se	2.5	5.1
8. elas (0.0) rev-del CP2	2.6	0.6
9. forward-deletion	6.7	37.1
10. reverse-deletion	7.0	0.4

proceeds with the rows from 1 to m (where there are m rows in the model) while leaving the column bounds intact, this is easily explained. Because the reverse-deletion filter proceeds with the rows from m to 1, followed by the column bounds, the *first* complete IIS seen by the additive method will be the same as the *last* complete IIS seen by the reverse-deletion filter. In addition, the reverse-deletion and the reverse-deletion CP2 method both operate on the rows in the same manner, so both will return IISs having the same number of rows.

All of the top three methods in Table III are affected by the ordering of the constraint rows. The question then remains as to whether some other ordering of the row constraints would permit these methods to find an IIS having fewer rows. In contrast, the elastic filtering variations ranked 4 and 5 in Table III are unaffected by the ordering of the constraint rows and give results that are almost identical to those returned by the top three methods. Their lower ranking may be an artifact of the test set which consists mainly of small models likely having relatively few IISs. In fact, the elastic filter gives better results as compared with the reverse-deletion filter with CP2 when applied to a different selection of models taken from the netlib set.[8] The elastic filtering variations will probably give better results when applied to very large models having many IISs. In such a case, the fixed ordering of the testing of the row constraints may prove disadvantageous for the additive algorithm and the deletion filter.

Table IV shows that the fastest method returns IISs having among the most rows. The speed is due to the use of sensitivity filtering, both directly after the initial detection of infeasibility, and thereafter in the deletion/sensitivity filter. Because of the initial emphasis on algorithm speed in IIS isolation research, the reverse-deletion/sensitivity filter was the default method in MINOS(IIS) version 3.2.

Some conclusions relative to the initial hypotheses can also be drawn:

- Simple constraint ordering is indeed effective, as shown by the much better results given by the reverse-deletion filter (best results over test set) as compared with the forward-deletion filter (worst results over test set).
- The column protection option CP2 is effective, always reducing the average number of rows in the IISs returned when compared with the same method without column protection. As predicted, CP2 makes no difference in the quality of the solution returned by the reverse-deletion filter, because it provides no extra column-bound protection in this case. However CP2 makes both the reverse- and the forward-deletion filters much quicker because the sensitivity filter is applied to the column bounds after the rows have been examined.
- The elastic filter (with objective parameter increment of 0.0) is very effective in finding IISs with few rows. The 0.1 objective parameter increment heuristic for improving the performance of the elastic filter is not effective, and is dominated in all cases by the elastic filter without the heuristic. Some ideas for speeding the elastic filter are given in Section 4.
- The sensitivity filter generally increases the number of rows in the IIS found by a method, unless the elastic filter has first been applied to ameliorate this effect by eliminating any large IISs. If the elastic filter is applied first, then the sensitivity filter can safely be used to speed the final isolation.

4. Conclusions

The experimental results presented here show that the heuristic methods for isolating IISs having few rows are extremely effective relative to the previous state of the art. Five methods give almost uniformly excellent results over the test set: (i) the reverse-deletion filter with CP2, (ii) the reverse-deletion filter, (iii) the implicit additive method, (iv) the elastic reverse-deletion/sensitivity filter with CP2 and no increment, and (v) the elastic reverse-deletion filter with CP2 and no increment.

Of the best methods, the reverse-deletion CP2 method is recommended for small to medium LPs due to the combination of effectiveness in finding IISs having few rows and speed. Another possibility is the implicit additive method since it gives excellent results and finds IISFs quickly, showing promise for quick results when extended to find full IISs. For larger LPs, the elastic filter followed by a reverse-deletion/sensitivity filter with CP2 and no increment is suggested because this is likely to prove much faster and possibly more effective on very large LPs. Though this hypothesis remains to be tested on a set of very large infeasible models, there is some empirical evidence supporting this idea.[8]

The five best methods all use a similar idea: discourage the elimination of column bounds while encouraging the elimination of rows. Simple constraint ordering during deletion filtering and in the additive algorithm ensures that rows are eliminated in preference to column bounds. The column protection options CP1 and CP2 act similarly in conjunction with other filtering methods. Elasticization of only the rows during elastic filtering has a similar purpose. In addition, all of the five best methods avoid the use of the sensitivity filter, which tends to isolate IISs having many rows, except when the elastic filter is applied first. The elastic filter tends to remove the IISs having many rows, so the sensitivity filter can be safely applied thereafter to speed the isolation.

The elastic filter is slowed by the conservative strategy for basis reuse. One promising avenue for speed improvement is the incorporation of the elastic filter directly into the MINOS phase 1 procedure. At present, once a variable bound has been satisfied, MINOS never again allows the bound to be violated during phase 1. Some alteration of these rules to incorporate the elastic filter would speed the process and also eliminate the extra memory needed by the current MINOS(IIS) implementation. The reverse-deletion filter with the CP2 column-protection option is the default method used in MINOS(IIS) version 5.0 because it does not require the user to request additional space for columns and elements.

There are several avenues for further research:

- How much speed-up of the elastic filter can be obtained by integrating it with the MINOS phase 1?
- How do the top five methods compare in effectiveness when applied to very large infeasible LPs?
- What is the time performance of the implicit additive algorithm when extended to find complete IISs?
- How fast and effective are the methods of Parker and Ryan[25] when tuned to find IISs having few rows?

The recommended methods for finding IISs having few rows are easily incorporated into LP solvers, requiring only added LP solutions and, for the elastic filter, extra columns and elements and a rewritten objective function. Facilities for the isolation of IISs having few rows should be a standard part of any commercial LP code, and a standard first step in the analysis of LP infeasibility. Narrowing the problem in this way greatly speeds the diagnosis.

Acknowledgments

Harvey Greenberg (University of Colorado at Denver), Tom Baker (Chesapeake Decision Sciences, Inc.), Jean-Francois Mondou (Hydro-Quebec), John Stone (Ketron Management Sciences, Inc.), Ed Klotz (CPLEX Optimization Inc.), and Bob Fourer (Northwestern University) provided test problems used in this paper. John Stone and Michael Saunders (Stanford University) provided helpful advice while I modified MINOS into MINOS(IIS). Harvey Greenberg and Tom Baker in particular challenged me to improve my early IIS algorithms. This research was partially supported by a research grant to the author provided by the Natural Sciences and Engineering Research Council of Canada.

References

1. G. Brown and G. Graves, 1975. Elastic Programming: A New Approach to Large-Scale Mixed Integer Optimization, presented at ORSA/TIMS conference, Las Vegas.
2. W.B. Carver, 1921. Systems of Linear Inequalities, *Annals of Mathematics 23*, 212–220.
3. N. Chakravarti, 1994. Some Results Concerning Post-Infeasibility Analysis, *European Journal of Operational Research 73*, 139–143.
4. J.W. Chinneck, 1993. Finding the Most Useful Subset of Constraints for Analysis in an Infeasible Linear Program, technical report SCE-93-07, Systems and Computer Engineering, Carleton University, Ottawa, Canada.
5. J.W. Chinneck, 1994. MINOS(IIS): Infeasibility Analysis Using MINOS, *Computers and Operations Research 21*, 1–9.
6. J.W. Chinneck, 1996. An Effective Polynomial-Time Heuristic for the Minimum-Cardinality IIS Set-Covering Problem, *Annals of Mathematics and Artificial Intelligence 17*, 127–144.
7. J.W. Chinneck, 1996. Feasibility and Viability, in *Recent Advances in Sensitivity Analysis and Parametric Programming*, T. Gal and H.J. Greenberg (eds.), to appear.
8. J.W. Chinneck, 1996. Computer Codes for the Analysis of Infeasible Linear Programs, *Journal of the Operational Research Society 47*, 61–72.
9. J.W. Chinneck and E.W. Dravnieks, 1991. Locating Minimal Infeasible Constraint Sets in Linear Programs, *ORSA Journal on Computing 3*, 157–168.
10. CPLEX Optimization Inc, 1994. Using the CPLEX Callable Library, CPLEX Optimization Inc, Incline Village, NV.
11. M.E. Dyer, 1983. The Complexity of Vertex Enumeration Methods, *Mathematics of Operations Research 8*, 381–402.
12. J. Gleeson and J. Ryan, 1990. Identifying Minimally Infeasible Subsystems of Equations, *ORSA Journal on Computing 2*, 61–63.
13. H.J. Greenberg, 1983. A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models, *ACM Transactions on Mathematical Software 9*, 18–56.
14. H.J. Greenberg, 1987. Diagnosing Infeasibility for Min-Cost Network Flow Models. I. Dual Infeasibility, *IMA Journal of Mathematics in Management 1*, 99–109.
15. H.J. Greenberg, 1987. Computer-Assisted Analysis for Diagnosing Infeasible or Unbound Linear Programs, *Mathematical Programming Studies 31*, 79–97.
16. H.J. Greenberg, 1987. ANALYZE: A Computer-Assisted Anal-

ysis System for Linear Programming Models, *Operations Research Letters 6,* 249–255.
17. H.J. GREENBERG, 1988. Diagnosing Infeasibility for Min-Cost Network Flow Models. II. Primal Infeasibility, *IMA Journal of Mathematics Applied in Business and Industry 2,* 1–12.
18. H.J. GREENBERG, 1992. An Empirical Analysis of Infeasibility Diagnosis for Instances of Linear Programming Blending Models, *IMA Journal of Mathematics in Business & Industry 4,* 163–210.
19. H.J. GREENBERG, 1993. *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE,* Kluwer, Boston.
20. H.J. GREENBERG, 1993. How to Analyze the Results of Linear Programs. 3. Infeasibility Diagnosis, *Interfaces 23,* 120–139.
21. IBM Corporation, 1991. *Optimization Subroutine Library Guide and Reference, Release 2,* IBM Corporation, Kingston, NY.
22. G. MITRA and M. TAMIZ, 1988. *FortLP Reference Manual,* NAG Ltd.
23. B.A. MURTAGH and M.A. SAUNDERS, 1987. *MINOS 5.1 User's Guide,* technical report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University.
24. J.L. NAZARETH, 1987. *Computer Solution of Linear Programs,* Oxford University Press, New York.
25. M. PARKER and J. RYAN, 1995. Finding the Minimum Weight IIS Cover of an Infeasible System of Linear Inequalities, *Annals of Mathematics and Artificial Intelligence,* to appear.
26. L. SCHRAGE, 1991. *LINDO: An Optimization and Modeling System,* 4th ed., The Scientific Press, San Francisco.
27. M. TAMIZ, S.J. MARDLE, and D.F. JONES, 1994. Detecting IIS in Infeasible Linear Programmes using Techniques from Goal Programming, *Computers and Operations Research,* to appear.
28. J. VAN LOON, 1981. Irreducibly Inconsistent Systems of Linear Inequalities, *European Journal of Operational Research 8,* 283–288.