ORSA Journal on Computing
Vol. 3, No. 2, Spring 1991

0899-1499/91/0302-0157 $01.25
© 1991 Operations Research Society of America

Locating Minimal Infeasible Constraint Sets in Linear Programs

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

ERIK W. DRAVNIEKS *Systems and Computer Engineering, Carleton University, Ottawa, Ontario K1S 5B6, Canada*

(Received. December 1989; final revision received. October 1990; accepted December 1990)

With ongoing advances in hardware and software, the bottleneck in linear programming is no longer a model solution, it is the correct formulation of large models in the first place. During initial formulation (or modification), a very large model may prove infeasible, but it is often difficult to determine how to correct it. We present a formulation aid which analyzes infeasible *LP*s and identifies minimal sets of inconsistent constraints from among the perhaps very large set of constraints defining the problem. This information helps to focus the search for a diagnosis of the problem, speeding the repair of the model. We present a series of *filtering* routines and a final integrated algorithm which guarantees the identification of at least one minimal set of inconsistent constraints. This guarantee is a significant advantage over previous methods. The algorithms are simple, relatively efficient, and easily incorporated into standard LP solvers. Preliminary computational results are reported.

As hardware and software have advanced in recent years, the solution of very large linear programs has become routine. The bottleneck is no longer a numerical solution, it is the correct initial formulation of the model. Inadvertent errors are difficult to prevent, especially when integrating several smaller models into a larger one, or when modifying a complex model.

When a large model is infeasible, it is very difficult to proceed. You need to know where the problem is in order to repair it, but it is hard to determine which constraints are in conflict by simple inspection guided by the nonzero artificial variables in the phase 1 solution. Where there are hundreds or thousands of constraints, automated assistance in localizing the infeasibility is a necessity.

This article develops an algorithm for automatically localizing an infeasibility to a minimal set of causative constraints. The algorithm can be terminated when the first such minimal set is found, or used repeatedly to find others.

This sort of *algorithmic engine* can be used alone, but is most effective as part of a larger toolkit for infeasibility analysis which may use several algorithmic engines and intelligent tools in arriving at a useful diagnosis of the problem. Additional algorithmic engines are described below, and by Greenberg and Murphy.[12]

Cognitive analysis imports and uses knowledge from the problem domain, and must generally be applied to the output of algorithmic engines to complete the diagnosis. For example, a minimal causative set of constraints consisting of a network flow structure and a single lower bound on an outflow arc suggests that the problem is a reversed arc. Cognitive analysis is the domain of humans and intelligent assistants such as expert systems, syntax analyzers, etc., as suggested by Greenberg and Murphy.[12] Algorithmic engines of the type developed in this article speed the diagnosis by focussing the cognitive tools.

In this paper we are concerned solely with algorithmic engines for locating minimal causative sets of constraints. Though various mathematical approaches to postinfeasibility analysis have been described previously, our algorithm and contemporary work by Gleeson and Ryan[6] are the first robust infeasibility localizers reported.

Charnes and Cooper[2] presented some of the earliest work on the problem, and gave an enlightening comparison of infeasibility analysis and goal programming. Their work is the antecedent of the elastic filtering algorithm described later.

An early paper by Roodman[16] describes how to eliminate an infeasibility. When the phase 1 *LP* terminates with some of the artificial variables having nonzero values, sensitivity analysis is used to find the minimum adjustment to the right hand sides of the corresponding constraints to achieve feasibility. This is not useful if the adjusted constraint is in fact correct while some other conflicting constraint is in error. Roodman does not give a method for finding the other members of the minimal infeasible set.

Subject classifications: Programming: linear: algorithms, theory Identifying minimal infeasible constraint sets.

Greenberg[7-11] describes a set of heuristics which rely on tracing back through a series of manipulations of the model, such as removal of redundant constraints, bound tightening, path and cycle generation, and matrix analysis. While not designed specifically for locating minimal infeasible sets of constraints, these heuristics often do so, but cannot guarantee such a result.

Murty[14] [p. 237] describes how to use the phase 1 *LP* solution to find a set of constraints which is preventing feasibility. The shadow prices of the phase 1 solution are used to implicate possible causative constraints as is done by Roodman, and in addition, the reduced costs for the original variables are used to implicate possible causative nonnegativity constraints. However, the indicated set may consist of a number of minimal infeasible sets, and no method is given for further localization. We use Murty's method in our sensitivity filtering algorithm, but add further localization routines.

Van Loon[17] presents a simplex variant and a set of necessary and sufficient conditions for the recognition of a minimal infeasible set. Unfortunately, the search for the elements of the set is undirected, leaving no option but a combinatorially explosive exhaustive search. His method also suffers from problem blow-up because equality constraints must be converted to a pair of inequalities and nonnegativity constraints must be explicitly added to the working constraint set. Greenberg and Murphy[12] point out that Van Loon's method could be extended to find minimal causative sets more efficiently by pivoting through alternative bases.

Gleeson and Ryan[6] describe a complete localization algorithm. They use a variant of Farkas Theorem of the Alternative to obtain a polytope in which each vertex indexes the members of a minimal infeasible set of constraints. Their method shares the drawbacks of van Loon's method, but has the advantage of a directed and efficient search. In the absence of degeneracy, each new pivot gives a new minimal infeasible set.

Very rough theoretical comparisons (see Section 7.3) indicate that our algorithm may be faster than that of Gleeson and Ryan. Full-scale computational testing awaits an implementation of Gleeson and Ryan's method.

Our approach to localizing an infeasibility is to gradually eliminate constraints from the original set defining the problem until those remaining constitute a minimal infeasible set. We call this *filtering* the constraint set. Three basic filtering routines are developed: deletion, elastic and sensitivity, which are then combined into a recommended integrated filtering algorithm. Deletion filtering is the cornerstone, providing a positive identification of a single minimal causative set, but it is relatively slow. Elastic and sensitivity filtering speed the process by eliminating large numbers of uninvolved constraints quickly.

This article first presents some background material and definitions, then develops the basic routines and the complete integrated algorithm. Examples are provided.

1. Irreducibly Inconsistent Systems of Constraints

Van Loon[17] introduced the term "irreducibly inconsistent system" to describe a minimal infeasible set of constraints.

Definition: An *irreducibly inconsistent system* (*IIS*) is a minimal set of inconsistent constraints.

An *IIS* may include some nonnegativity constraints on the variables, but it is not possible to construct an *IIS* entirely from nonnegativity constraints. On the other hand, as long as the *LP* solution algorithm enforces variable nonnegativity automatically (as the simplex method does), it is possible to find an *IIS* by identifying the participating functional constraints only, and then finding the participating nonnegativity constraints afterward.

Perhaps for this reason, van Loon, Roodman, and others concentrated on finding the functional constraints in an *IIS* without identifying the participating nonnegativity constraints. We identify participating nonnegativity constraints explicitly, so we use the following definition to make the difference clear.

Definition: An *irreducibly inconsistent set of functional constraints* (*IISF*) is the complete subset of functional constraints in an *IIS*.

Note that when an *IIS* does not include any nonnegativity constraints, then the *IISF* is the same as the *IIS*.

Figure 1 shows a system of constraints having two *IIS*s: $\{A, B, C\}$ and $\{B, C, x_1 \geqslant 0\}$; note that $\{A, B, C\}$ is also an *IISF*. As defined below, these two *IIS*s are overlapped.

Definition: An *IIS* is *overlapped* if it shares at least one constraint with another *IIS*.[5]

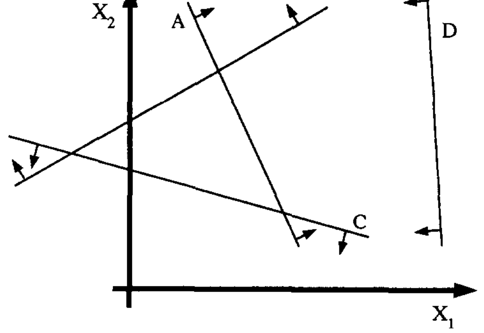


Figure 1. Overlapped *IIS*s forming a cluster.

DEFINITION: A *cluster* of *IIS*s is a maximal set of *IIS*s such that each *IIS* overlaps at least one other *IIS* of the cluster.

As would be expected, the *cardinality* of an *IIS* (*IISF*) means the number of constraints that makes up the *IIS* (*IISF*).

Many infeasible models will have only a few *IIS*s, or a small cluster of overlapped *IIS*s. One or more incorrect constraints can, however, have the effect of creating many *IIS*s, similar to the ripple effect in computer programming where a single incorrect FORTRAN statement may generate numerous errors. It is generally sufficient for the modeller to identify one *IIS* from each cluster in order to proceed.

We will also use the following theorem:

THEOREM 1. *If there are n variables in the original LP, the maximum cardinality of any IIS is* $n + 1$.

Proof. see Chvatal[4] [p. 24].

2. Deletion Filtering

Deletion filtering is a novel algorithm which guarantees the identification of all of the functional constraints in exactly one *IIS*. This property is of fundamental importance for *IIS* localization: no other algorithm provides this positive identification. The algorithm is computationally expensive and is best applied after a large set of constraints has been whittled down by other means.

Given an *LP* having one or more *IIS*s, deletion filtering operates by considering each functional constraint individually, as follows. Temporarily remove the constraint from the *LP*, then test the reduced *LP* for feasibility. If the reduced *LP* is infeasible, then remove the constraint permanently; if the reduced *LP* becomes feasible, then return the constraint to the *LP*. Continue in this fashion until all of the constraints have been tested.

The Deletion Filtering Algorithm

DEFINITION: Q is a set of functional constraints, q_i.

```
1. Set $Q$ equal to the infeasible set of functional constraints under consideration.
2. Test constraints for possible deletion: For each $q_i$ in $Q$ DO:
    2.1 Determine whether $Q \setminus q_i$ is feasible or infeasible using a phase 1 LP solution.
    2.2 IF $Q \setminus q_i$ is feasible THEN continue with next $q_i$
        ELSE ($Q \setminus q_i$ infeasible) SO
            $Q = Q \setminus q_i$, continue with next $q_i$.
OUTPUT: Upon termination, the set $Q$ will be exactly one IISF. One or more nonnegativity constraints may also be involved in the IIS, but these are not explicitly identified.
```

2.1. Theorems and Discussion

THEOREM 2. *If there is at least one IISF in the constraint set input to the deletion filtering algorithm, then the output set will contain exactly one IISF.*

Proof. As the initial constraint set is infeasible, and constraints are removed only when Q will remain infeasible, then Q contains at least one *IISF* at all times. As constraints remain in Q only if their removal would render the constraint set feasible, they must be members of an *IIS*, by definition. All constraints remaining in Q at termination meet this condition, so they must all be members of the same *IIS*: if there were two or more *IIS*s, then you would be able to remove at least one constraint and Q would remain infeasible. ■

Deletion filtering is easily extended to the nonnegativity constraints as follows. Remove the sign restrictions on the nonnegative variables in the *IISF* by introducing a pair of variables (see Winston[18], [p. 154]), then add explicit zeroing constraints on the "negative" variable of each pair, and deletion filter only these zeroing constraints in conjunction with the *IISF*.

Deletion filtering is unaffected by multiple or overlapped *IIS*s: *IIS*s or parts of *IIS*s are removed as long as at least one *IIS* remains. However, the order in which the constraints are tested determines which *IIS* is reported. When finished, there is no indication of whether other *IIS*s exist.

2.2. Time Complexity of Deletion Filtering

Deletion filtering is a brute force method. Each test for feasibility is an ordinary phase 1 *LP*. Where there are m functional constraints in the original infeasible *LP*, there will be m phase 1 *LP* solutions to carry out. At worst, all of the constraints in the original problem will constitute a single *IIS*, so all of the phase 1 *LP*s will be of size m. By Theorem 1, this can only happen when the number of constraints is $\leq n + 1$, where n is the number of variables. The phase 1 *LP*s will ordinarily be of smaller and smaller size as the number of retained constraints decreases.

3. Elastic Programming and Phase 1 *LP*s

Brown and Graves[1] introduced the term "elastic programming" to describe the addition of extra variables which allow constraints to "stretch" to increase the size of the feasible region. The basic idea is much older: it is the principle used in the phase 1 *LP* to obtain an initial feasible solution.

Constraints are converted to elastic form by introducing "elastic variables" v_i which permit the constraint to stretch (or "relax") in the direction which increases the feasible region. The elastic constraint is easily reconverted

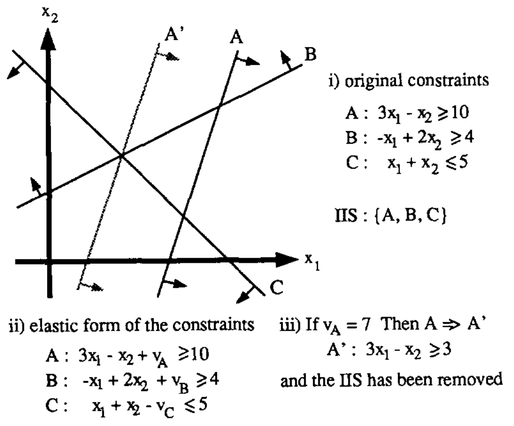


Figure 2. Elastic constraints and the removal of an *IIS*.

to nonelastic form by removing the elastic variables. The conversion rules are:

Constraint type	*Nonelastic*	*Elastic*
$\geq$	$\sum_i a_i x_i \geq b$	$\sum_i a_i x_i + v \geq b$
$\leq$	$\sum_i a_i x_i \leq b$	$\sum_i a_i x_i - v \leq b$
$=$	$\sum_i a_i x_i = b$	$\sum_i a_i x_i + v' - v'' = b$

One elastic variable must be added and one elastic variable must be subtracted from each equality constraint to allow it to stretch in either direction. All elastic variables are constrained to be nonnegative.

The resistance to stretching implied by the term "elastic" is supplied by creating a new objective: minimize the sum of the elastic variables.

The principal difference between elastic programming and the phase 1 *LP* is that any constraint can be stretched in elastic programming, while in the phase 1 *LP* only $\geq$ constraints are truly elastic, and = constraints are elastic in one direction only.

A nonzero elastic or artificial variable corresponds to an adjustment of the right hand side of a constraint when viewed from the original solution space (i.e., the constraint stretches). This effect is illustrated in Figure 2.

4. Elastic filtering

Elastic filtering speeds *IIS* localization by quickly eliminating non-*IIS* functional constraints from large models. It uses elastic programming and the fact that stretching constraints in an *IIS* sufficiently permits a feasible solution.

The algorithm generates a series of *LP*s in which some of the functional constraints are elastic and some are nonelastic. After each *LP* is solved, any stretched constraints are reconverted to nonelastic form by removing the elastic variables, i.e. they are *enforced*. Enforced constraints are members of some *IIS*, since only *IIS* constraints are stretched in the first place. Enforcing functional constraints forces another elastic member of the *IIS* to stretch when the next *LP* is solved. When all of the members of an *IISF* have been enforced, the next *LP* will be infeasible, halting the method.

The output of the elastic filter is the set of enforced constraints, which must contain at least one *IISF*. The advantage of elastic filtering is that only $s + 1$ *LP*s at most are solved before termination, where s is the cardinality of the smallest *IISF* in the model.

At any intermediate iteration, a regular phase 1 solution may be needed prior to the use of the elastic objective function. This initial phase 1 solution determines whether the current elastic *LP* is feasible or infeasible.

This full-blown use of elastic programming to identify a small set containing an *IISF* is novel, though related notions were advanced by Charnes and Cooper[2] in connection with goal programming, and by Roodman[16] in connection with finding a set of adjustments to the model in order to attain feasibility.

The Elastic Filtering Algorithm

DEFINITIONS.

i. Q_{input} is the initial set of input functional constraints.
ii. Q_{output} is the final set of output functional constraints.
iii. E is a set of elastic constraints, e_i.
iv. V is a set of elastic variables, v_i.
v. R is a set of stretched ("relaxed") elastic constraints resulting from the solution of an *LP*.
vi. The function elastic(Q) operates on a set of nonelastic constraints Q to convert them to elastic form. The inverse function enforce(E) operates on a set of elastic constraints E, converting them back to nonelastic form by removing the elastic variables.
vii. The operator evin(E) identifies the elastic variables in the set E of elastic constraints.

1. Initialize the sets of constraints and variables:
   a. $Q_{\text{output}} = R = \varnothing$.
   b. $E = \text{elastic}(Q_{\text{input}})$.
   c. $V = \text{evin(E)}$.
2. Solve this *LP* to determine if the original *LP* is feasible:
   minimize $Z = \sum_i v_i$, where $v_i \in V$,
   subject to: E, nonnegativity of variables.
   IF $Z = 0$ (original *LP* is feasible) THEN STOP.
   ELSE obtain the set R of stretched elastic constraints, set $Q_{\text{output}} = \text{enforce}(R)$.
3. Reduce the set of elastic constraints by removing the stretched constraints:
   a. $E = E \setminus R$.
   b. $R = \varnothing$.
   c. $V = \text{evin}(E)$.

```
4. Solve the $LP$:
      minimize $Z = \sum_i v_i$, where $v_i \in V$,
      subject to: $E$, $Q_{\text{output}}$, nonegativity of variables.
   IF infeasible, THEN STOP.
   ELSE this results in the set $R$,
      $Q_{\text{output}} = Q_{\text{output}} \cup \text{enforce}(R)$, go to Step 3.
OUTPUT: At termination $Q_{\text{output}}$ contains at least one
   $IISF$, and no non-$IIS$ constraints. The complete $IIS$(s)
   may also involve nonnegativity constraints which are
   not explicitly identified.
```

4.1. Theorems and Discussion

When enough constraints are elastic, there will be feasible solutions to the *LP*s in Steps 2 and 4. Strictly speaking then, there are no *IIS*s in these *LP*s, but for ease of reference we will refer to *IIS*s in these elastic sets, meaning *IIS* which exist in the original problem.

LEMMA 3. *Elastic filtering stretches only elastic constraints belonging to an IIS.*

Proof. This follows easily from (*i*) the fact that the cost of stretching a constraint is strictly positive, and (*ii*) only *IIS* constraints need to be stretched to achieve feasibility. ∎

LEMMA 4. *If the set Q_{input}* contains at least one IISF, and the current Q_{output} set does not contain an IISF, then Step 4 will stretch only previously unstretched elastic constraints, and it will stretch at least one such constraint from each IIS.

Proof. In Steps 2 and 4, the *LP* must stretch at least one elastic constraint from each *IIS* in order to achieve a feasible solution. As the set Q_{output} does not contain an *IISF*, at least one constraint from each *IISF* is still in *E*. Accordingly, a feasible solution to the elastic *LP* is attained.

Because all previously stretched constraints have been enforced, they can no longer be stretched. Therefore only constraints which have not been previously stretched are eligible for stretching. ∎

THEOREM 5. *The elastic filtering output set will contain at least one IISF if and only if an IIS exists in the original LP.*

Proof. If no *IIS* exists in the original *LP*, then no constraints will be stretched and the algorithm will terminate at Step 2 with the message that the problem is feasible.

If at least one *IIS* exists in the original *LP*, then Step 2 must stretch at least one constraint from each *IISF* to achieve feasibility, so *Z* will be nonzero. Accordingly, the algorithm will not terminate prematurely.

By Lemma 4, as long as Q_{output} does not contain an *IISF*, Step 4 will always stretch at least one previously unstretched *IISF* constraint, which will then be added to Q_{output}. Thus at each iteration of Step 4, Q_{output} is increased by at least one *IISF* constraint.

Each *IISF* is composed of a finite number of constraints, so the algorithm will terminate in a finite number of steps, as soon as at least one *IISF* is in Q_{output}, because then no constraints from the *IIS* will be elastic in the *LP* of Step 4, rendering it infeasible. ∎

4.2. Time Complexity of Elastic Filtering

Let s be the minimum of the cardinalities of all of the *IISF*s in the original *LP*. Then the time complexity of elastic filtering is at worst $(s + 1)^*$(time complexity of *LP* solutions in Steps 2 and 4). The proof uses the fact that at least one new constraint from each *IISF* is identified at each iteration of the algorithm (LEMMA 4). Thus a complete *IISF* is identified in at most s iterations, and the $(s + 1)$th *LP* will be infeasible, terminating the algorithm. Theorem 1 limits the size of s.

4.3. Efficiency Improvements

During each iteration of the algorithm, stretched constraints are enforced from elastic form back to nonelastic form. This is equivalent to changing the coefficients of a basic variable in the original constraint matrix and objective function (i.e., the original coefficients of the nonzero elastic variables are changed to zero).

Just as in ordinary sensitivity analysis, it may be possible to update the current solution and use it as an advanced start for finding the solution to this new problem, depending on the number of coefficients that are changed simultaneously. The updated solution will normally be infeasible, necessitating the use of the dual simplex method to arrive at a final solution. See Winston[18] for example.

5. Sensitivity Filtering

When an *LP* is infeasible, sensitivity analysis can be applied to the phase 1 or elastic solution. Sensitivity filtering uses the fact that stretching a constraint is equivalent to altering the right hand side (*rhs*) of the constraint. The phase 1 or elastic solution will show sensitivity to an infinitesimal adjustment of the *rhs*'s of some of the *IIS* constraints, but never to the rhs of an non-*IIS* constraint.

5.1. Theorems and Discussion

We refer below only to elastic *LP*s, but the results apply equally to phase 1 *LP*s.

The sensitivity of the elastic objective function to an infinitesimal change in the *rhs* of a functional constraint is signalled by a nonzero shadow price (“dual elevator”) in the final tableau of the elastic *LP*.

DEFINITION. F is the set of functional constraints having nonzero shadow prices in the optimal tableau of an elastic *LP* whose objective function value is nonzero.

THEOREM 6. *The set F contains at least one IISF.*

Proof. Murty[14] [pp. 237–238] shows that a linear combination of the constraints in F, along with the nonnegativity constraints, is infeasible. Therefore, F must contain at least one *IISF*. ■

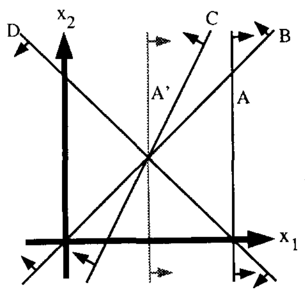
Roodman[16] [pp. 918–919] provides similar reasoning and is listed as a reference by Murty.

OBSERVATION 7. *The set F is not guaranteed to contain all of the functional constraints of all of the IISs in the original problem.*

Proof. This is easily shown by example, as given in Figures 3 and 4. ■

Figure 4 shows that the shadow prices correctly indicate the members of an *IISF* despite a degeneracy among the original constraints, though again not all of the *IISF*s are identified.

DEFINITION. N is the set of original variables having a nonzero reduced cost in the optimal tableau of an elastic *LP* whose objective function value is nonzero.



A : $x_1 \geq 2$
B : $x_1 - x_2 \leq 0$
C : $2x_1 - x_2 \leq 1$
D : $x_1 + x_2 \leq 2$

IISs : {A, B, D} {A, C, D}

Constraint A is made elastic by the addition of an appropiate elastic variable v_A . Solution of the elastic LP gives :

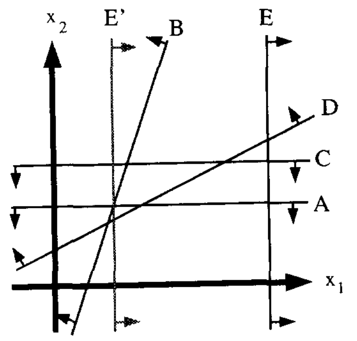
$x_1 = 1 \quad x_2 = 1 \quad v_A = 1$

Constraint A is adjusted to A' : $x_1 \geq 1$.

There is a tie for the leaving basic variable in the second tableau when x_2 enters the basis :

Case 1 : s_3 leaves the basis		Case 2 : s_4 leaves the basis	
constraint	final shadow price	constraint	final shadow price
A	1	A	1
B	0	B	-1/2
C	-1/3	C	0
D	-1/3	D	-1/2
Indicates IIS {A, C, D}		Indicates IIS {A, B, D}	

Figure 4. Shadow prices where there is a degeneracy among the original constraints.



A : $x_2 \leq 2$
B : $3x_1 - x_2 \leq 2$
C : $x_2 \leq 3$
D : $-0.5x_1 + x_2 \geq 1$
E : $x_1 \geq 5$

IISs : {A, B, E} {A, D, E} {C, B, E} {C, D, E}

Constraints D and E are made elastic by the addition of appropiate elastic variables v_D and v_E . Solution of the elastic LP gives :

$x_1 = 4/3 \quad x_2 = 2 \quad v_E = 11/3$

Constraint E is adjusted to E' : $x_1 \geq 4/3$

The adjusted original solution space is a point defined by the constraints A, B, E'.

Constraint	Shadow price
A	-1/3
B	-1/3
C	0
D	0
E	1

Shadow price analysis indicates IIS {A, B, E}, but not the other 3 IISs

Figure 3. The shadow prices for the elastic solution do not indicate all of the members of all of the *IIS*s.

THEOREM 8. *The set N indexes nonnegativity constraints involved in IISs.*

Proof. See Murty[14] [p. 238]. ■

Theorem 8 provides a means of identifying the nonnegativity constraints which are part of the *IIS*s.

CONJECTURE 9. *The set F* ∪ *N contains at least one IIS from each cluster if IISs.*

We have not been able to prove Conjecture 9, but because *IIS* clusters share no constraints, it is reasonable to assume that constraint adjustments to clear the *IIS*s in one cluster will not affect the constraints in another cluster. Accordingly, one would expect that at least one constraint from each cluster would appear in the set $F \cup N$.

When viewed from the original variable space, an elastic (or phase 1) *LP* achieves an artificial feasible solution by adjusting the *rhs*'s of some of the functional constraints, as illustrated in Figures 2–4. The *rhs* adjustments move the constraints just enough in the original space to achieve a feasible solution, therefore the constraints in the corresponding optimum cornerpoint must be those involved in the *IIS*. The nonzero shadow prices are associated with the functional constraints defining the cornerpoint.

This also explains why some of the *IIS*s are not identified by *rhs* sensitivity analysis: the movement of the constraints in the original space alters the problem. As an elastic constraint moves towards its final cornerpoint position, another *IIS* may be removed en route, so the shadow prices for the members of this *IISF* are zero.

Sensitivity filtering is very inexpensive, requiring only the inspection of the final tableau of the phase 1 *LP*, for example, and it has the important advantage of explicitly identifying nonnegativity constraints which are part of some *IIS*.

The Sensitivity Filtering Algorithm

DEFINITIONS.

i. E is a set of elastic functional constraints, e_i.
ii. Q is a set of nonelastic functional constraints, q_i.
iii. Q_{input} defines a partially elastic set of input functional constraints: $Q_{\text{input}} = E \cup Q$. Enforce$(E) \cup Q$ must be an infeasible set.
iv. Let $V = \text{evin}(E)$.
v. $\boldsymbol{p}$ is the row vector of shadow prices.
vi. $\boldsymbol{c}_{bv}$ is the row vector of basic variable objective function coefficients.
vii. $\boldsymbol{B}^{-1}$ is the basis inverse matrix.
viii. F is the output set of functional constraints in some *IIS*.
ix. N is the output set of nonnegativity constraints in some *IIS*.

```
1. Solve the elastic LP:
   minimize $Z = \sum_i v_i$, such that $v_i \in V$,
   subject to $E$, $Q$, nonnegativity of variables.
   Select the appropriate case:
   CASE 1. LP is infeasible.
   Elastic LP is incorrectly formulated. Convert to full elastic form and go to step 1.
   CASE 2. $Z = 0$.
   STOP. Original LP is feasible.
   CASE 3. $Z > 0$.
   Go to Step 2.
2. Calculate the shadow prices:
   $\boldsymbol{p} = \boldsymbol{c}_{bv}\boldsymbol{B}^{-1}$
3. $F = \emptyset$.
   FOR each constraint $i$ in $E \cup Q$ DO:
      IF $p_i \neq 0$ THEN $F = F \cup \{\text{constraint } i\}$.
   $N = \emptyset$.
   FOR each original variable $x_i$ DO:
      IF the reduced price of $x_i$ in the final tableau $\neq 0$
         THEN $N = N \cup \{x_i \geq 0\}$.
   OUTPUT: At termination, the set $F \cup N$ contains at least one IIS (and possibly at least one IIS from each cluster according to Conjecture 9).
```

5.2. Identifying More *IIS* Constraints

Observation 7 shows that sensitivity filtering may not identify all of the constraints that are involved in *IIS*s. Where the goal is to identify a maximum number of *IIS*s with a single iteration of sensitivity filtering, additional *IIS* constraints can be found by examining the basis inverse matrix, $\boldsymbol{B}^{-1}$. The nonzero elements in a row of $\boldsymbol{B}^{-1}$ corresponding to a basic elastic variable index the functional constraints whose *rhs*'s affect the final value of the elastic variable.

Some of the constraints so indexed may not have been identified by the shadow prices, and can be added to F, creating the new set F^+. This may happen when, for example, the constraint effect of increasing a certain elastic variable is exactly counterbalanced by its decreasing effect on another elastic variable, hence the net shadow price is zero. This is what happens in the example in Section 7.1.

Where there is a degeneracy in the solution of the elastic *LP*, as indicated by a basic variable with a value of zero, there may be other constraints that are tight at the degenerate point which form part of an *IIS*. Again, these other constraints are indexed by the nonzero elements in the row of $\boldsymbol{B}^{-1}$ belonging to the basic variable with value zero. These constraints can be added to F^+ if they are not already there, and will be filtered out later if they prove not to be part of an *IIS*.

5.3. Time Complexity

The time complexity of sensitivity filtering is negligible beyond the initial solution of the elastic *LP*. Where the original elastic *LP* is a phase 1 *LP* which must be solved anyway, then the only additional cost is the calculation of $\boldsymbol{p}$, which requires m^2 multiplications and $m(m-1)$ additions, the subsequent m comparisons to zero to identify elements of F, and n comparisons to zero to identify elements of N.

6. The Integrated Filtering Algorithm

The three filtering routines can be combined in various ways to create efficient integrated algorithms. Given that infeasibility is usually discovered by solving a phase 1 *LP*, and that sensitivity filtering of the phase 1 solution is very cheap, we can assume that this will be the first step in any integrated algorithm.

Subsequent filtering of F (or F^+) is needed to positively identify individual *IISF*s. Where there are multiple clusters of *IIS*s, there are probably multiple *IIS*s in $F \cup N$ (Conjecture 9).

Depending on whether the goal is to identify a single *IIS* as quickly as possible, or to identify as many *IIS*s at reasonable cost as possible, final filtering can proceed in a number of different ways.

6.1. Deletion / Sensitivity Filtering

In deletion filtering, each time a functional constraint is dropped and the remaining constraint set is infeasible, a new phase 1 solution is available. Sensitivity filtering this solution may eliminate several other constraints, reducing the number of *LP*s which must be solved, thereby increasing the speed of the *IIS* identification significantly.

Sensitivity filtering has the advantage of directly identifying nonnegativity constraints which are part of an *IIS*. Once the *IISF* has been found, if the appropriate phase 1 *LP* has not already been solved while deletion testing another constraint, then it is necessary to construct and solve an additional phase 1 *LP* in order to identify the nonnegativity constraints in the *IIS*.

The nonnegativity constraints identified in this manner, when combined with the *IISF* constraints, may form a cluster rather than a single *IIS*. In other words, where more than one nonnegativity constraint is implicated, the functional constraints in an *IISF* may be the overlapped part of a cluster. It is generally useful to know all of the nonnegativity constraints that are affected by the *IISF*, so the extra cost of identifying a single *IIS* is not usually justified. If necessary, the extension of the deletion filtering algorithm described in Section 2.1 can be applied.

Deletion/sensitivity filtering may be comparatively inefficient, depending on the number of *IIS*s and clusters in the problem. Let s be the minimum of all of the *IISF* cardinalities, and let u be the number of *IIS* clusters in the problem. If $u > s$ then deletion/sensitivity filtering may still be inefficient compared to elastic filtering because a relatively large number of *LP*s must be solved.

6.2. Elastic / Sensitivity Filtering

Sensitivity filtering can be applied to the solution obtained for each elastic *LP* solved during elastic filtering. Additional constraints may be eliminated in this manner, depending on how the nonzero elastic variables adjust the constraints in the original solution space. This idea also applies to the phase 1 *LP* solution which terminates the elastic filtering algorithm by signaling infeasibility.

6.3. Finding Other *IIS*s

Either of the final filtering methods described above will locate a single *IISF*, which we can label $IISF_1$, with the associated nonnegativity constraints. If $IISF_1$ has cardinality less than the cardinality of F, then there may be other *IISF*s in F. If we remove $IISF_1$ from F and continue filtering, then we may find another *IISF*, say $IISF_2$. We continue in this fashion until either F is empty, or the constraints which remain do not contain an *IISF*. This guarantees that we will find at least one *IIS* from each cluster represented in $F \cup N$.

6.4. Design of the Integrated Filtering Algorithm

The obvious first step in the integrated filtering algorithm is to sensitivity filter the original phase 1 solution which signals infeasibility. A final deletion filter is needed to positively identify a single *IIS*. The deletion/sensitivity and elastic/sensitivity filters can be used at intermediate steps.

Generally speaking, it is best to use elastic/sensitivity filtering first when the initial sensitivity filtering output set is relatively large, and to go directly to deletion/sensitivity filtering when the set is relatively small. Computational studies are needed to find heuristics to guide this decision. Possible heuristics include a decision based on the absolute number of constraints in the initial set F or F^+, or on the ratio of the number of constraints in F or F^+ to the total number of functional constraints. The integrated filtering algorithm assumes the existence of such a heuristic.

The integrated filtering algorithm incorporates the method of Sec. 6.3 for checking for other *IISF*s in the initial input set F or F^+. Where the goal is to identify only a single *IIS*, then Step 3 of the algorithm can be omitted.

The Integrated Filtering Algorithm

```
INPUT: a final phase 1 solution which signals infeasibility.

Definitions:

  i. $F$ is a set of functional constraints, initially equal to
     the set identified by the sensitivity filtering of a phase
     1 $LP$ (either $F$ or $F^+$ as described in Section 5.2).
 ii. $Q$ is an intermediate set of functional constraints.
iii. $Q_{iisf}$ is a set of constraints constituting an $IISF$,
     resulting from an application of the deletion/sensitivity
     filtering algorithm.

1. $Q = \emptyset$.
   Sensitivity filter the phase 1 result to obtain the set $Q$.
   IF indicated by heuristic decision rule THEN
      elastic/sensitivity filter $Q$ to produce a possibly
      reduced set $Q$.
2. $Q_{iisf} = \emptyset$.
   Deletion/sensitivity filter $Q$ to obtain $Q_{iisf}$. Report
   members.
   IF phase 1 solution of $Q_{iisf}$ not already available
   THEN
      Set up and solve the phase 1 $LP$ based on $Q_{iisf}$ and
      nonnegativity of all variables in $Q_{iisf}$.
   Sensitivity filter the phase 1 solution of $Q_{iisf}$; report
      the nonnegativity constraints that are implicated.
3. $F = F \setminus Q_{iisf}$.
   IF $F = \emptyset$, then STOP.
   Solve the phase 1 $LP$ based on $F$ and nonnegativity of
   all variables in $F$.
   IF $Z = 0$ ($F$ is feasible) THEN STOP.
   GO TO step 1.
OUTPUT: at least one $IIS$ (or $IIS$ cluster overlapped on
   the $IISF$) per cluster in $F \cup N$.
```

7. Examples and Computational Results

The examples presented in Sections 7.1 and 7.2 are small enough that the results can be verified by inspection. To appreciate the practical value of these methods, imagine that these small *IIS*s are embedded in very large original problems, on the order of thousands of constraints. As the examples are small, we do not use elastic/sensitivity filtering as an intermediate step.

The *LP* census of each step is expressed as $\sum_k k(g_k)$ where k is *LP* size in number of functional constraints, and g_k is the number of *LP*s at this size. The index k runs from 1 to m, where m is the total number of functional constraints in the original *LP* model. For example, 6(3) means 3*LP*s of size 6 constraints. Since the total cost of solving an *LP* is approximately proportional to the cube of the number of functional constraints, it is possible to calculate a very rough overall measure of effort as $\sum_k g_k k^3$.

It is informative to put the total effort into perspective by comparing the measure of effort with the measure of effort for the original phase 1 *LP* solution of the problem. We calculate the relative measure of effort as follows: $\left(\sum_k g_k k^3\right)/m^3$.

As these examples show, the algorithms are very efficient when evaluated using the rough measures of effort. Preliminary computational results are reported in Section 7.4.

7.1. Example: A Single Cluster of *IIS*s

This example is based on an economic analysis problem. The original functional constraints are:

1: $0.8x_3 + x_4 \leqslant 10{,}000$
2: $x_1 \leqslant 90{,}000$
3: $2x_6 - x_8 \leqslant 10{,}000$
4: $-x_2 + x_3 \geqslant 50{,}000$
5: $-x_2 + x_4 \geqslant 87{,}000$
6: $x_3 \leqslant 50{,}000$
7: $-3x_5 + x_7 \geqslant 10{,}000$
8: $0.5x_5 + 0.6x_6 \leqslant 300{,}000$
9: $x_2 - 0.05x_3 = 5{,}000$
10: $x_2 - 0.04x_3 - 0.05x_4 = 4{,}500$
11: $x_2 \geqslant 80{,}000$

There is a large cluster which contains at least these *IIS*s: $\{1,4,5,9\}$, $\{1,4,9, x_4 \geqslant 0\}$, $\{1,5,9, x_3 \geqslant 0\}$ $\{1,5,10, x_3 \geqslant 0\}$, $\{1,10,11\}$, $\{4,6,9\}$, $\{4,6,11\}$, $\{5,6,9,10\}$, $\{6, 9, 11\}$.

The filtering process outputs different *IIS*s depending on whether the rows of B^{-1} are examined in addition to the shadow prices. When only the shadow prices are considered, the input set is F and filtering proceeds as follows:

1. Sensitivity filtering the phase 1 *LP* solution yields $F = \{1,4,5,6,10,11\}$.
2. Deletion/sensitivity filtering F yields the *IIS* $\{4,6,11\}$. *LP* census is 6(1) + 3(4).
3. Setting $F = F \setminus Q_{\text{iisf}}$ yields $F = \{1,5,10\}$.
4. Phase 1 solution of F shows it is feasible. Stop. *LP* census is 3(1).

The single *IIS* $\{4,6,11\}$ is identified, and the total *LP* census is 6(1) + 3(5). The total overall measure of effort is 351. The relative measure of effort is 0.26, so the localization cost about one quarter as much as the original phase 1 solution.

When the relevant rows of B^{-1} are also analyzed, constraint 9 is added to give the augmented set $F^+ = \{1,4,5,6,10,11,9\}$. Constraint 9 is not identified by the shadow prices because its increasing effect on the artificial variables for constraints 4 and 5 is exactly conterbalanced by its decreasing effect on the artificial variables for constraints 10 and 11. In this case filtering proceeds as follows:

1. Sensitivity filtering the phase 1 solution and analysis of B^{-1} yields $F^+ = \{1,4,5,6,10,11,9\}$.
2. Deletion/sensitivity filtering yields the *IIS* $\{6,9,11\}$. Total *LP* census is 6(1) + 5(1) + 4(1) + 3(3) + 2(2).
3. Setting $F^+ = F^+ \setminus Q_{\text{iisf}}$ yields $F^+ = \{1,4,5,10\}$.
4. The phase 1 solution of F^+ shows it is infeasible. *LP* census is 4(1). Initial sensitivity filter gives $Q = \{1,4,5,10\}$.
5. Deletion/sensitivity filtering Q yields the *IIS* $\{1,4,5\}$. *LP* census is 3(4). In this case, an extra phase 1 *LP* to detect nonnegativity constraints in the *IIS* is not needed because the very last deletion/sensitivity phase 1 *LP* signalled infeasibility itself, so its reduced costs could be read.
6. Setting $F^+ = F^+ \setminus Q_{\text{iisf}}$ yields $F^+ = \{10\}$.
7. The phase 1 solution of F^+ shows it is feasible. Stop. *LP* census is 1(1).

Two *IIS*s are identified: $\{6,9,11\}$ and $\{1,4,5\}$. The total *LP* census is 6(1) + 5(1) + 4(2) + 3(7) + 2(2) + 1(1). The total overall measure of effort is 675, and the relative measure of effort is 0.51, so the localization cost about half as much as the initial phase 1 solution, and identified 2 *IIS*s.

7.2. Example: Two Clusters of *IIS*s

In this example, there are two distinct clusters of *IIS*s, each cluster having only one *IIS*. The two *IIS*s are $\{1,2,3\}$ and $\{4,5,6\}$. The original constraints are:

1: $-0.5x_1 + x_2 \geqslant 0.5$
2: $2x_1 - x_2 \geqslant 3$
3: $3x_1 + x_2 \leqslant 6$
4: $x_5 \leqslant 2$
5: $3x_4 - x_5 \leqslant 2$

6: $x_4 \geqslant 5$
7: $x_1 + x_5 \leqslant 10$
8: $x_1 + 2x_2 + x_4 \leqslant 14$
9: $x_2 + x_4 \geqslant 1$

Filtering proceeds as follows:

1. Sensitivity filtering the phase 1 solution yields $F = \{1,2,3,4,5,6\}$. No constraints are added by analyzing the rows of B^{-1}.
2. Deletion/sensitivity filtering yields the *IIS* $\{4,5,6\}$. *LP* census is 5(1) + 3(1) + 2(3).
3. Setting $F = F \setminus Q_{\text{iisf}}$ yields $F = \{1,2,3\}$.
4. The phase 1 solution of F shows it is infeasible. *LP* census is 3(1). Sensitivity filtering the result yields $Q = \{1,2,3\}$.
5. Deletion/sensitivity filtering Q yields the *IIS* $\{1,2,3\}$. *LP* census is 3(1) + 2(3).
6. Setting $F = F \setminus Q_{\text{iisf}}$ yields $F = \varnothing$. Stop.

Both *IIS*s are discovered. The total *LP* census is 5(1) + 3(3) + 2(6), for an overall measure of effort of 254. The relative measure of effort is 0.35, so the cost of localization is about one third the cost of solving the original phase 1 *LP*, and two *IIS*s are discovered.

7.3. A Comparison to the Method of Gleeson and Ryan

An expression for the computational complexity of the method proposed by Gleeson and Ryan[6] is readily derived from their paper, which permits a theoretical comparison of their method and ours. For Gleeson and Ryan's method to function correctly, each equality constraint in the *LP* must first be converted to a pair of inequalities, and the nonnegativity constraints must be explicitly added to the set of working constraints.

Let us define the following parameters for the original *LP*: m is the total number of constraints, e is the number of equality constraints, k is the number of nonnegative variables, n is the total number of variables. Now Gleeson and Ryan's converted *LP* will have n variables and $(m + e + k)$ constraints. When Farkas theorem is applied, the resulting system has $(m + e + k)$ variables and $(n + 1)$ constraints, and when the vertex enumeration algorithm is applied, the time to find the first *IIS* is $O[(n + 1)(m + e + k)^2]$.

To compare the two methods, we will use this measure of complexity as a rough measure of effort for Gleeson and Ryan's method. For the problem of Sec. 7.1, $n = 8$, $m = 11$, $e = 2$, $k = 8$, so the measure of effort is $(8 + 1)(11 + 2 + 8)^2 = 3969$, compared to 351 for the filtering algorithm. For the problem of Section 7.2, $n = 4$, $m = 9$, $e = 0$, $k = 4$, so the measure of effort is $(4 + 1)(9 + 0 + 4)^2 = 845$, compared to 176 for the filtering algorithm to find the first *IIS*.

In these small examples it appears that the filtering algorithm may be superior to Gleeson and Ryan's method, but true computational comparison awaits an implementation of their method.

The computational complexity expression for Gleeson and Ryan's method shows that it operates at a disadvantage in any system which has numerous nonnegativity constraints and equality constraints. This is true of many general *LP*s, but it is particularly true for the important special class of networks.

7.4. Preliminary Computational Results

Computational results were obtained by modifying the well-known MINOS *LP* solver[13] to perform both deletion filtering and deletion/sensitivity filtering. Some modifications to the algorithms as described were needed because of the manner in which MINOS implements the simplex algorithm.

MINOS converts the input *LP* into the form: Minimize $\boldsymbol{c}^T\boldsymbol{x}$ subject to: $\boldsymbol{Ax} + \boldsymbol{Is} = \boldsymbol{0}$, $\boldsymbol{1} \leqslant [\boldsymbol{x},\boldsymbol{s}]^T \leqslant \boldsymbol{u}$, where a slack variable s_i is added to each equation in the converted *LP*. Phase 1 testing uses the FORMC procedure[15] which allows both slack variables and original variables to violate their bounds as necessary to achieve a solution. The procedure minimizes the number of variables which violate their bounds; if the number is zero, then feasibility has been attained.

Filtering concentrates solely on the upper and lower bounds on the variables. Because the original and the slack variables can be treated identically in the MINOS formulation, an *IIS* can be positively identified directly, rather than by first finding the *IISF* and then checking the nonnegativity constraints.

During sensitivity filtering of phase 1 results, bounds are retained if they have been violated, or if the corresponding variable shows sensitivity at that bound. Otherwise they are discarded. For numerical reasons, when the upper and lower bounds on a variable are identical (as in equality constraints), either both bounds are kept or both are eliminated.

The deletion filter operates first on the bounds on the original variables and then on the bounds on the slack variables. The deletion/sensitivity filter reverses this order. This has the effect of concentrating on infeasibilities in the functional constraints in both cases.

Some preliminary computational results are presented in Table I. The five test problems span a range of sizes, where the size is the number of finite bounds on the variables in the converted *LP*. These results were obtained using Microsoft Fortran version 5.0 under DOS 3.3 on a 20 MHz 80386-based computer with numeric coprocessor.

The algorithms work well in practice: verifiable *IIS*s are found in modest amounts of time. In fact the theoretical relative measures of effort are on the same order of magnitude as the actual time ratios for the examples in Sections 7.1 and 7.2. As expected, the deletion/sensitivity

Table I
IIS Localization Using the Modified MINOS

	Problem Section 7.1	Problem Section 7.2	WOODINFE	FOREST6	PILOT4I
No. finite bounds	21	13	173	196	1886
Phase 1 solution (sec)	0.8	0.8	2.3	3.7	471.7
Deletion filter results					
No. bounds in *IIS*	4	3	3	125	44
Time Ratio, *IIS* : phase 1	1.07	1.00	2.90	8.18	2.65
Deletion/sensitivity filter results					
No. bounds in *IIS*	5	3	3	122	152
Time ratio, *IIS* : phase 1	0.86	0.78	0.30	3.61	0.31
Size ratio, *IIS* : original	0.24	0.23	0.02	0.62	0.08

filter is faster than the straight deletion filter, taking as little as about one tenth of the time in one case. However, the deletion filter is still reasonably quick, likely due to the fact that MINOS uses the last basis (in this case from the previous phase 1 result) as an advanced start for the next *LP*.

There appears to be some correlation between the relative size of the *IIS* (expressed as a ratio of the cardinality of the *IIS* to the cardinality of the original *LP*) and the time ratio for *IIS* localization using deletion/sensitivity filtering. This is probably related to the number of deletion filtering steps that are eliminated by the sensitivity filter.

Where there are several *IIS*s in the model, the different filtering methods are likely to locate different *IIS*s. For the problem of Section 7.1, the *IIS* $\{1, 5, 10, x_3 \geqslant 0\}$ was found by the deletion/sensitivity filter, while the *IIS* $\{6, 9, 11\}$ was found by the deletion filter. For the problem of Section 7.2, the *IIS* $\{1, 2, 3\}$ was found by the deletion/sensitivity filter, while the *IIS* $\{4, 5, 6\}$ was found by the deletion filter. Two *IIS*s of very different sizes are found by the two filters in the case of PILOT4I.

Important cognitive issues are raised by these observations. Are smaller *IIS*s more easily diagnosed? If this is so, should the *IIS*-finding algorithms be modified so that they search for smaller *IIS*s first? How useful are very large *IIS*s without further diagnostic tools? If further diagnostic tools are needed, what form should they take?

While beyond the scope of this paper, these cognitive issues are very promising areas for further research. An experimental link between the modified MINOS and the ANALYZE system[7, 10] is already operational.

8. Conclusions and Future Research

IIS-finding algorithmic engines have great practical value in focussing the diagnostic efforts when large *LP*s prove infeasible, thereby speeding model repair. The ideal algorithm should be robust and efficient.

The novel integrated filtering algorithm reported here is robust: it always identifies at least one *IIS* in an infeasible *LP*, and at least one from each cluster in $F \cup N$. This is a significant advantage over previous methods which cannot guarantee the identification of an *IIS*.

The integrated algorithm is relatively simple, involving nothing more than solution inspection and repeated solution of phase 1 or elastic *LP*s, and so is easily incorporated into standard *LP* solvers such as MINOS with very little extra code. It is normally quite efficient for two reasons: first, the *LP*s solved beyond the first phase 1 *LP* are frequently fairly small, and second, each phase 1 *LP* is usually substantially similar to the previous one, so that most of the basis can be reused. Preliminary computational results support these conclusions.

Improvements to the integrated algorithm will be the subject of future research. A computational study should be undertaken to find heuristic rules for deciding when to initiate the elastic/sensitivity filtering step. This will likely concentrate on the numbers of constraints and variables involved, and the pattern of entries in the coefficient matrix (e.g., subsets of constraints involving different subsets of variables).

Preliminary work on using the rows of B^{-1} as early indicators of *IIS*s has been done. Nonzero elements of a row corresponding to a nonzero artificial variable index constraints which affect the final value of that artificial variable. These constraints are likely to be involved in an *IIS* which causes the nonzero value of the artificial variable. This notion could form the basis of another heuristic, e.g. using such a row set as the initial set of enforced constraints during elastic/sensitivity filtering.

A concerted effort to prove or disprove Conjecture 9 must also be made. We see this as part of a larger project exploring the properties of infeasible models and elastic *LP*s.

Another interesting project is to compare the speeds of the integrated algorithm and the method described by Gleeson and Ryan.[6] The rough calculation in Section 7.3 indicate that the filtering algorithm may be superior, but full implementations and computational comparisons need to be carried out. Testing of the filtering algorithm implemented in MINOS is ongoing.

Related problems suitable for future research include the set covering problem of finding the minimum number of constraints which must be repaired in order to achieve feasibility,[6] and finding the minimum cost set of repairs.[16]

Modifications to the filtering algorithms permitting efficient operation on network problems, both pure and with side constraints, have been proposed,[3] and await computational testing.

Finally, the cognitive issues posed by the availability of *IIS*-finding algorithmic engines should be explored. How can these new tools best be used to assist in the diagnosis of infeasibilities?

ACKNOWLEDGMENTS

The support of this research by the Natural Sciences and Engineering Research Council of Canada through an operating grant to J.W. Chinneck is gratefully acknowledged. MINOS was made available by the Consortium for the development of an Intelligent Mathematical Programming System, co-ordinated by Harvey Greenberg, who also offered valuable suggestions during the preparation of this paper. John Stone of Ketron Management Sciences Inc. was an excellent guide to the inner workings of MINOS.

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. A. Charnes and W.W. Cooper, 1961. *Management Models and Industrial Applications of Linear Programming*, Vol. I, John Wiley & Sons, New York.
3. J.W. Chinneck, 1990. Localizing and Diagnosing Infeasibilities in Networks, Technical Report SCE-90-14, Systems and Computer Engineering, Carleton University, Ottawa.
4. V. Chvatal, 1983. *Linear Programming*, W.H. Freeman, New York.
5. E.W. Dravnieks, 1989. Identifying Minimal Sets of Inconsistent Constraints in Linear Programs: Deletion, Squeeze and Sensitivity Filtering, M.Sc. thesis, Systems and Computer Engineering, Carleton University, Ottawa, Canada.
6. J. Gleeson and J. Ryan, 1990. Identifying Minimally Infeasible Subsystems of Inequalities, *ORSA Journal on Computing 2:1*, 61-63.
7. H.J. Greenberg, 1983. A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models, *ACM Transactions on Mathematical Software 9:1, 18-56.*
8. H.J. Greenberg, 1987. Diagnosing Infeasibility for Min-Cost Network Flow Models, Part I: Dual Infeasibility, *IMA Journal of Mathematics in Management 1*, 99-109.
9. H.J. Greenberg, 1987. Computer-Assisted Analysis for Diagnosing Infeasible or Unbound Linear Programs, *Mathematical Programming Studies 31*, 79-97.
10. H.J. Greenberg, 1987. ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models, *Operations Research Letters 6:5*, 249-255.
11. H.J. Greenberg, 1988. Diagnosing Infeasibility for Min-Cost Network Flow Models, Part II: Primal Infeasibility, *IMA Journal of Mathematics Applied in Business and Industry 2*, 1-12.
12. H.J. Greenberg and F.H. Murphy, 1990. Approaches to Diagnosing Infeasible Linear Programs, Technical Report, Mathematics Department, University of Colorado at Denver.
13. 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.
14. K.G. Murty, 1983. *Linear Programming*, John Wiley & Sons, New York.
15. J.L. Nazareth, 1987. *Computer Solution of Linear Programs*, Oxford University Press, New York.
16. G.M. Roodman, 1979. Post-Infeasibility Analysis in Linear Programming, *Management Science 25:9*, 916-922.
17. J. van Loon, 1981. Irreducibly Inconsistent Systems of Linear Inequalities, *European Journal of Operational Research 8*, 283-288.
18. W.L. Winston, 1987. *Operations Research: Applications and Algorithms*, PWS Publishers, Boston.