Fast Heuristics for the Maximum Feasible Subsystem Problem

John W. Chinneck
Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario K1S 5B6, Canada
chinneck@sce.carleton.ca

Given an infeasible set of linear constraints, finding the maximum cardinality feasible subsystem is known as the *maximum feasible subsystem problem*. This problem is known to be NP-hard, but has many practical applications. This paper presents improved heuristics for solving the maximum feasible subsystem problem that are significantly faster than the original, but still highly accurate.
(*Linear Programming; Artificial Intelligence; Heuristic; Analysis of Algorithms*)

1. Introduction

When a set of linear constraints is infeasible, it is often useful to find the maximum cardinality feasible subsystem. Finding this subsystem is known as the *maximum feasible subsystem problem (MAX FS)* (Amaldi et al. 1999). Finding the maximum feasible subsystem can also be viewed as finding the minimum number of linear constraints to remove such that the retained constraints constitute a feasible system, which is known as the *minimum unsatisfied linear relation problem (MIN ULR)* (Amaldi 1994). The two problems have complementary objective functions.

All infeasible systems have one or more *irreducible infeasible subsystems (IISs)* of constraints (Chinneck and Dravnieks 1991). An IIS has the property that it is itself infeasible, but any proper subsystem is feasible. Finding an IIS in a large and complex linear program is a very useful step in the diagnostic process, so algorithms for isolating IISs have been adopted in most major linear-programming solvers (Chinneck 1996a, 1997). Deleting at least one of its members eliminates an IIS, hence an infeasible set of constraints can be rendered feasible by deleting at least one member of every IIS it contains. Finding the smallest cardinality set of constraints to cover all IISs is known as the *minimum-cardinality IIS set-covering problem (MIN IIS COVER)* (Chinneck 1996b).

For the purposes of this paper, MAX FS, MIN ULR, and MIN IIS COVER are the same problem. We will use the terms interchangeably, usually choosing the term that is most appropriate for the particular version of the problem at hand.

As shown by Sankaran (1993) and Chakravarti (1994), MAX FS is NP-hard, as are the identical MIN ULR and MIN IIS COVER. Amaldi and Kann (1995) showed that the problem is also NP-hard for homogeneous systems of inequalities (both strict and nonstrict) and binary coefficients. There are many uses for solutions to these problems. Amaldi (1994) identifies applications including computational geometry, an application of the CLOSED HEMISPHERE problem in political science, linear numeric editing, discriminant analysis, and the training of neural networks for machine learning. Chinneck (1996b) points out applications in analyzing infeasible linear programs to provide assistance to model formulators. It may be convenient to transform other NP-hard problems into MAX FS, making the heuristic methods developed here available for their solution.

Amaldi and others (Amaldi 1994, Amaldi and Kann 1995, Amaldi et al. 1999) have extensively analyzed the approximability of MAX FS, but there has been relatively little development of algorithms for solving the problem. Parker and Ryan (1996) describe

INFORMS Journal on Computing © 2001 INFORMS
Vol. 13, No. 3, Summer 2001 pp. 210–223
0899-1499/01/1303/0210$5.00
1526-5528 electronic ISSN

an exact solution to MIN IIS COVER. Each iteration of the algorithm involves the integer-programming solution of a MIN IIS COVER subproblem over a subset of the IISs in the whole problem. If the cover solution at any iteration solves the entire problem, then the algorithm halts; else a new IIS is generated, added to the set, and a new iteration begins. The exact solution of each integer-programming subproblem has nonpolynomial time requirements; however it is straightforward to convert the method to a polynomial-time heuristic by substituting a heuristic in place of the exact integer-programming algorithm. A further difficulty is that the method requires the conversion of the original problem into a version that typically has more rows than the original.

Chinneck (1996b) describes an effective polynomial-time heuristic for MIN IIS COVER. Briefly, it operates by deleting one constraint at a time from the original model in a sequence of iterations that terminates when the reduced set of constraints becomes feasible. Each iteration has these steps: (i) assemble a list of candidates for testing, (ii) test each candidate by solving an LP, and (iii) permanently delete the constraint that gives the best value of the test measure.

This heuristic shows 100% accuracy over a suite of test examples (Chinneck 1996b). However, it can be slow on some models because of the necessity of solving one LP for every member of the candidate list. A speedier algorithm is desirable in many cases, for example in recalculating classifier separating surfaces as new data are added. This paper presents an improved version of the original heuristic that is significantly faster, with only a small trade-off in accuracy in isolated cases in the suite of test examples. The speed-up is due to new insights that permit a drastic reduction in the length of the list of candidate constraints for removal. This in turn reduces the number of LP solutions needed, resulting in a significant speed-up for the larger examples.

A standard problem in machine learning and data classification is the placement of a separating surface (normally a hyperplane) in such a way that it completely separates points in one set from points in another set. This is not normally possible with real data, so the objective is usually to place the separating surface so as to minimize the number of incorrectly classified points. This latter problem is easily transformed into a restricted version of the MIN IIS COVER problem (Amaldi 1994, Parker 1995, Chinneck 1998). The restriction arises in that the converted constraint set consists entirely of inequalities: there are no variable bounds or equality constraints. Because of this transformation, research results from the classification (e.g. Mangasarian 1994, Chen and Mangasarian 1996, Bennett and Bredensteiner 1997) and machine learning (e.g. Frean, 1992) communities are of interest. Methods from these fields can also be viewed as algorithms for solving a restricted version of MIN IIS COVER.

The remainder of this paper develops the details of the algorithm, and presents empirical results on two suites of test examples.

2. Algorithms

A brief review of the original algorithm from Chinneck (1996b) is needed. Recall that a linear program can be elasticized by adding appropriate nonnegative elastic variables, analogous to artificial variables in an ordinary phase 1 formulation (Brown and Graves 1975). In a *standard elastic program* only the row constraints are elasticized. In a *full elastic program*, all constraints are elasticized, including both rows and column bounds. As indicated below, rows are elasticized by adding nonnegative elastic variables e_i, and columns are elasticized by adding nonnegative elastic variables $\underline{e_j}$.

nonelastic row	elastic row
$\Sigma_j a_{ij} x_j \geq b_i$	$\Sigma_j a_{ij} x_j + e_i \geq b_i$
$\Sigma_j a_{ij} x_j \leq b_i$	$\Sigma_j a_{ij} x_j - e_i \leq b_i$
$\Sigma_j a_{ij} x_j = b_i$	$\Sigma_j a_{ij} x_j + e'_j - e''_j = b_i$
nonelastic variable bound	**elastic variable bound**
$x_j \geq l_j$	$x_j + \underline{e}_j \geq l_j$
$x_j \leq u_j$	$x_j - \underline{e}_j \leq u_j$

The associated elastic objective function seeks to minimize the sum of the elastic variables. An elasticized set of constraints cannot itself be infeasible, but it does provide information about any infeasibility in the original non-elastic model. The elastic objective-function value expresses the sum of the constraint

violations in the original model, called the *sum of the infeasibilities* (SINF). If SINF equals zero, then the original model is feasible. Each nonzero elastic variable indicates a violated constraint in the original model; this *number of infeasibilities* (NINF) is another measure of the infeasibility of the original model. The MAX FS problem (or MIN ULR or MIN IIS COVER problems) is then a problem of minimizing NINF, rather than minimizing SINF.

The original algorithm relies on a series of observations. The most important of these is **Observation 3** (Chinneck 1996b): *The elimination of a constraint that is a member of a minimum-cardinality IIS set cover should reduce SINF more than the elimination of a constraint that is not a member of a minimum-cardinality IIS set cover.* This is true because a member of a minimum-cardinality IIS set cover will normally remove more than one IIS when there are multiple interacting IISs in the model.

Due to Observation 3, the heart of the original algorithm is testing constraints by removing them temporarily to determine the amount of reduction in SINF that is obtained. **Observation 4** (Chinneck 1996b) helps to reduce the number of constraints that must be tested: *Constraints to which the elastic objective function is not sensitive do not reduce SINF when removed from the model.* Hence constraints to which the elastic objective function is not sensitive can be excluded from the list of candidates to test. (Operationally, a constraint is "sensitive" if the associated variable has a nonzero reduced cost.)

Observation 1 (Chinneck 1996b) shows that the original phase 1 procedure that signals infeasibility of the constraint set, regardless of the algorithm used, provides an upper bound on the cardinality of the MIN IIS COVER: *Upon termination of the solver phase 1, the number of violated constraints is an upper limit on the cardinality of the IIS set cover, and the set of violated constraints is an IIS set cover.* This information can be used to halt the more advanced algorithms when the cardinality of the cover that they are isolating exceeds that of the cover already found by the phase 1 procedure. Finally, **Observation 2** (Chinneck 1996b) provides a convenient early exit for the algorithm in simple cases: *If the phase 1 procedure reports a single violated constraint, then that violated constraint constitutes a minimum cardinality IIS set cover.*

See Chinneck (1996b) for detailed discussions of Observations 1–4.

A slightly revised version of the original algorithm, here designated as Algorithm 1, is given in Figure 1. Algorithm 1 can use any variant of an elastic program. Chinneck (1996b) identifies four such variants, including the standard and full elastic programs, the simple textbook phase 1, and the MINOS-style phase 1 (MINOS seeks to reduce NINF, and terminates when it recognizes that feasibility cannot be achieved; SINF is not necessarily minimized upon termination; see Wolfe 1965). The original implementation uses a standard elastic program. In the variants of Algorithm 1 developed later, it will be advantageous to use full elastic programs.

2.1. New Algorithms

The speed of Algorithm 1 is largely determined by two elements: (i) the time required for the initial determination of infeasibility via the solver phase 1 procedure, and (ii) the length of the list of candidates in `CandidateSet`, i.e. the list of constraints to which the elastic objective function is sensitive. There is little that can be done to shorten the time needed to determine infeasibility in the first instance. In this paper, we devise ways of increasing the speed of Algorithm 1 by shortening the list of candidates; objective-function sensitivity is no longer a sufficient criterion for admission to `CandidateSet`.

Because Algorithm 1 is extremely effective, we would like any modifications to it to delete the same constraints, and in the same order. Ideally, we would like to identify directly the correct constraint for removal at each iteration, and place only that single constraint on the list of candidates in `CandidateSet`. It turns out that we can use the evaluations developed in the next two observations to assess each potential candidate quickly without solving an LP. Then only a short list of the most promising candidates is added to `CandidateSet` for testing via LP solution. Very often, the first constraint on the list is indeed the correct constraint for removal.

Observation 5. *For constraints that are violated in the original model, a good predictor of the magnitude of the*

```
INPUT: Constraints defining the original infeasible model.

STEP 0: CoverSet = ϕ.
        Set up elastic LP.

STEP 1: Solve elastic LP.
        If NINF=1:
                Add the single violated constraint to CoverSet.
                Exit.
        HoldSet = {constraints to which the elastic objective function is sensitive}.

STEP 2: MINSINF = ∞.
        CandidateSet = HoldSet.
        For each constraint in CandidateSet:
                Delete the constraint.
                Solve elastic LP.
                If SINF=0 then:
                        Add constraint to CoverSet.
                        Exit.
                If SINF < MINSINF then:
                        Set winner = currently deleted constraint.
                        Set MINSINF = SINF.
                        HoldSet = {all constraints to which elastic objective function is
                        sensitive}.
                        If NINF = 1, nextwinner = single violated constraint.
                        Else nextwinner=ϕ.
                Reinstate the constraint.

STEP 3: Add winner to CoverSet.
        If nextwinner ≠ ϕ then:
                Add nextwinner to CoverSet.
                Exit.
        Delete the winner constraint permanently.
        Go to Step 2.

OUTPUT: CoverSet is a small set of constraints covering the IISs.
```

Figure 1 Original Algorithm 1 for MIN IIS COVER (Chinneck 1996b)

drop in SINF that will be obtained by deleting the constraint is given by the product (constraint violation) × |(constraint sensitivity)|.

When converted to a full elastic program, "constraint violation" in the original model is given by the value of the elastic variable associated with a constraint. If there are two elastic variables associated with a constraint, as for equality and range constraints, then the constraint violation is the maximum value of the two elastic variables. "Constraint sensitivity" refers to the reduced cost of the variable associated with the constraint. The absolute value of the constraint sensitivity is used because the sign, determined by the constraint sense ($\leq, \geq, =$), is irrelevant since all violations are relaxations of the constraint, regardless of constraint sense.

Having a nonzero elastic variable in the elasticized model is equivalent to changing the right-hand-side value of the constraint in the original model. Thus the product in Observation 5, obtained from the elastic version of the model, is the same as operating on the original model to estimate the change in the objective value caused by relaxing the right hand side by the amount given by the nonzero elastic variable. As shown in elementary texts on simple sensitivity analysis, this is a perfectly accurate estimator of the change in SINF, provided that the basis in the original model does not change. Of course, the basis in the original model *does* change when an active constraint is deleted, so Observation 5 provides an underestimate of the change in SINF. However, the estimator is frequently very accurate, as shown empirically in Table 3 in Section 4.1.

Observation 5 suggests a revision to Algorithm 1. In Steps 1 and 2 of Algorithm 1, instead of setting HoldSet = {constraints to which the elastic objective function is sensitive}, find HoldSet as follows:

1. Select the violated constraints, and arrange them in order from largest to smallest value of the product (constraint violation) × |(constraint sensitivity)|.

2. Fill HoldSet with the top k elements of the ordered list (or all of the elements of the list if there are fewer than k).

We will refer to this variant as *Algorithm 2(k)*, where k refers to the length of the candidate list. Empirical results using this algorithm are presented in the next section. Note that a list length of 1 is frequently successful.

Because we wish to estimate the effect of every constraint via the product in Observation 5, Algorithm 2 requires the use of a fully elastic version of the original model (i.e. variable bounds must be elasticized as well as rows). This is straightforward in solver implementations that already permit literal constraint violations (by violating the bounds on the variable associated with the constraint) during their Phase 1 procedure, but it may require the explicit addition of elastic variables in other solver implementations.

Concentrating solely on the violated constraints is often successful because the elastic objective function is itself trying to minimize SINF, hence it tends to violate the constraints that most reduce SINF. However, in some cases with numerous infeasibilities, it may be possible to obtain a larger drop in SINF by deleting a constraint that is not currently violated. Observation 6 describes an indicator for identifying unviolated constraints that are good candidates for deletion.

Observation 6. *For constraints that are not violated in the original model, a good predictor of the relative magnitude of the drop in SINF that will be obtained by deleting the constraint is given by |(constraint sensitivity)|.*

Observation 6 provides a way of ordering the set of unviolated constraints for use in a candidate list. Unlike the product described in Observation 5, Observation 6 does not provide a direct estimate of the size of the drop in SINF expected when the constraint is deleted, only the relative size (i.e. a constraint with a larger |sensitivity| is expected to provide a larger drop in SINF).

Observations 5 and 6 can be combined to provide another variant of Algorithm 1. In Steps 1 and 2 of Algorithm 1, instead of setting HoldSet = {constraints to which the elastic objective function is sensitive}, find HoldSet as follows:

1. Select the violated constraints, and arrange them in order from largest to smallest value of the product (constraint violation) × |(constraint sensitivity)|.

2. Fill HoldSet with the top k elements of the ordered list (or all of the elements of the list if there are fewer than k).

3. Select the unviolated constraints to which the elastic objective function is sensitive, and arrange them in order from largest to smallest |(constraint sensitivity)|.

4. Add the top k elements of this ordered list to the bottom of HoldSet (or all of the elements of the list if there are fewer than k).

We will refer to this variant as *Algorithm 3(k)*, where k refers to the length of each of the two lists. Note that a list length of k implies the solution of up to $2k$ LPs to identify the successful candidate.

We can also improve on Algorithm 1 by taking better advantage of Observation 1 to provide a safety exit when the more advanced algorithms perform poorly. Because the cardinality of the IIS cover provided by the phase 1 procedure is already known, any subsequently applied algorithm can be halted when its cover cardinality exceeds the cardinality of the IIS cover already provided by the phase 1 procedure.

Further, more than one phase 1 procedure would probably be applied in a practical implementation because the solver-native phase 1 procedure is unlikely to be a full elastic SINF minimization. The implementation would normally proceed as follows:

1. Native phase 1 method detects infeasibility and records the NINF and IIS cover.

2. Convert to full elastic version of model.

3. Minimize SINF in fully elastic model (using an advanced start provided by the native phase 1 solution) and record the NINF and IIS cover.

```
INPUT: Linear constraints defining the model.

STEP 0: CoverSize=0, CoverSet=ϕ, SafetySize=0, SafetySet=ϕ.
        If native phase 1 procedure detects feasibility, then exit.
        SafetySize = (cardinality of native phase 1 cover).
        SafetySet = {members of native phase 1 cover}.
        If SafetySize=1 then:
                CoverSize=1.
                CoverSet=SafetySet.
                Exit.
        Set up elastic LP.
        Solve elastic LP using advanced start provided by the native phase 1 procedure.
        If (elastic cover cardinality) < SafetySize then:
                SafetySize = (cardinality of elastic phase 1 cover).
                SafetySet = {members of elastic phase 1 cover}.
        If SafetySize=1 then:
                CoverSize=1.
                CoverSet=SafetySet.
                Exit.
        HoldSet = {constraints meeting selection criteria}.

STEP 1: MINSINF = ∞.
        CandidateSet = HoldSet.
        For each constraint in CandidateSet:
                Delete the constraint.
                Solve elastic LP.
                If SINF=0 then:
                        Add constraint to CoverSet.
                        CoverSize=CoverSize+1.
                        Exit.
                If SINF < MINSINF then:
                        Set winner = currently deleted constraint.
                        Set MINSINF = SINF.
                        HoldSet = {constraints meeting selection criteria}.
                        If NINF = 1, nextwinner = single violated constraint.
                        Else nextwinner=ϕ.
                Reinstate the constraint.

STEP 2: Add winner to CoverSet.
        CoverSize=CoverSize+1.
        If nextwinner ≠ ϕ then:
                Add nextwinner to CoverSet.
                CoverSize=CoverSize+1.
                Exit.
        Delete the winner constraint permanently.
        If CoverSize ≥ (SafetySize-1) then
                CoverSet=SafetySet.
                CoverSize=SafetySize.
                Exit.
        Go to Step 1.

OUTPUT: CoverSet is a set of constraints covering the IISs, with cardinality CoverSize.
```

Figure 2 Algorithm 4 for MIN IIS COVER

Between the two methods, the smaller NINF would act as a stopping condition for any more advanced algorithm.

Algorithm 4 (Figure 2) combines all of these observations into a generic framework. The possible selection criteria for inclusion in HoldSet include (i) phase 1 objective function sensitivity (as in Algorithm 1), (ii) high values of the product for violated constraints (as in Algorithm 2), or (iii) both high values of the product for violated con-

straints and high phase 1 constraint sensitivities (as in Algorithm 3).

2.2. Polynomial Time Behavior

Algorithm 1 has polynomial time complexity (Chinneck 1996b). Algorithms 2 and 3 improve this complexity by limiting the number of LPs that must be solved to find each removed constraint. Consider a set of linear constraints having p members, which could be rows or variable bounds. At worst, you may need to remove $p-1$ constraints to achieve feasibility. In this worst case, the number of LP solutions required would be upper bounded as follows:

- Algorithm 2(k) would require at most $k(p-1)$ LP solutions,
- Algorithm 3(k) would require at most $2k(p-1)$ LP solutions.

Given that linear programs can be solved in polynomial time (Karmarkar 1984), this upper bound is also polynomial. The algorithms achieve very good performance in practice, as shown in Section 4.

3. Implementation Notes

Several algorithms in five implementations are discussed in Section 4. Notes on the five implementations are provided here.

MINOS(IIS) is a version of MINOS 5.4 (Murtagh and Saunders 1993) modified for the analysis of infeasible linear programs (Chinneck 1994, 1996a, 1997), and is available at http://www.sce.carleton.ca/faculty/chinneck/minosiis.html. Among several methods of analyzing infeasible LPs, it includes an implementation of Algorithm 1. MINOS(IIS) first carries out a standard MINOS phase 1 procedure and reports the NINF found by this procedure. If necessary, the members of the cover found by the phase 1 procedure can be returned. If NINF is greater than 1, then a standard elastic program is set up and Algorithm 1 is carried out. Results in Section 4.2 are produced using a 400 MHz Pentium-III machine under Windows NT.

CLIIS is a program for data classification (Chinneck 1998). It also uses a modified version of MINOS 5.4 to implement Algorithm 1 as well as Algorithm 2(1). Both algorithms use a standard elastic program, but for the classification problem this is equivalent to a full elastic program because of the lack of variable bounds in the converted data sets. Results in Section 4.1 are produced using a 200 MHz Pentium Pro machine under Windows NT.

FEC, created for the purposes of this paper, implements both Algorithms 2 and 3 at user-controlled list lengths. It is again a modification of MINOS 5.4. Algorithms 2 and 3 require full elasticization of the model, and this is achieved in the FEC prototype by converting all variable bounds to elasticized row constraints, thereby freeing all of the variables. This is necessary because the built-in MINOS phase 1 procedure does not necessarily terminate at the minimum value of SINF. The larger number of rows in the converted LP slows the solution. For this reason, comparisons between FEC and the other programs are best made on the basis of the number of LPs solved, rather than by comparing clock times. The inefficient FEC prototype suffices for the purposes of this paper by permitting an analysis of the relative efficiency of Algorithms 2 and 3, but a full production implementation would instead use an approach that does not expand the number of rows in the LP. Results in Section 4 are produced using a 400 MHz Pentium-III machine under Windows NT.

The first full elastic solution completed by FEC is equivalent to a SINF-minimizing full elastic phase 1 procedure. The NINF at this stage is given by the number of constraints (both rows and column bounds) having nonzero elastic variables. If necessary, these cover members can be reported by FEC. In Section 4.2, the NINF results returned by the MINOS phase 1 procedure and the full elastic phase 1 are compared.

MISMIN is an implementation of a parametric programming algorithm for minimizing the number of misclassified points during classification (Bennett and Bredensteiner 1997). LP subproblems are solved by CPLEX (1994). Empirical results provided by Bennett and Bredensteiner (1997) show that MISMIN is among the best programs available for minimizing misclassification errors, hence it is a good standard for comparison. Results in Section 4.1, provided by the MISMIN authors, were produced using a 200 MHz Pentium machine under Windows 95.

Parker and Ryan (1996) provided results for the general LP test suite using their exact branch-and-

bound algorithm for solving MIN IIS COVER for general LPs. Their method is not comparable to the others on the basis of speed because of its inherent non-polynomial algorithm, but the exact results are valuable. Their implementation uses OSL to solve the integer-programming subproblems.

4. Empirical Results

We carry out experiments on two collections of examples, one collection arising from data-classification problems, and the other arising from the analysis of general infeasible LPs. Classification problems do not involve variable bounds or equality constraints, in contrast to general infeasible LP problems. However, some classification problems have very large MIN IIS COVER cardinalities, which make them interesting for analysis of the sort described here.

4.1. Classification Problems

The usual classification problem is to separate completely points of one type (e.g. type 0) from points of another type (e.g. type 1) by placing a separating surface (e.g. a hyperplane) in the n-dimensional space of attributes or features. Since complete separation is not usually possible, the minimum-misclassification cardinality problem is then to determine the smallest number of points to delete such that the remaining points can be completely separated by a single separating surface. As shown by Amaldi (1994) and also by Parker (1995), this problem is easily transformed into MIN IIS COVER. The conversion proceeds as follows:

Given: a training set of I data points ($i = 1 \cdots I$) in J dimensions ($j = 1 \cdots J$), in which the value of attribute j for point i is denoted by d_{ij}, and the class of each point is known (either Type 0 or Type 1).

Define a set of linear constraints as follows (one constraint for each data point):

- for points of Type 0: $\Sigma_j d_{ij} w_j \leq w_0 - \epsilon$
- for points of Type 1: $\Sigma_j d_{ij} w_j \geq w_0 + \epsilon$

where ϵ is a small positive constant. Note that the variables are the unrestricted w_j, where $j = 0 \cdots J$, while the d_{ij} are known constants.

If the data are completely separable by a single hyperplane, then any solution to the LP resulting from the conversion will yield a set of values for the w_j, which then defines the separating hyperplane $\Sigma_j d_{ij} w_j = w_0$. If the data cannot be completely separated by a single hyperplane, then the LP resulting from the conversion will necessarily be infeasible. Finding a solution to the MIN IIS COVER problem in this infeasible LP then also solves the classification problem of finding the smallest number of points to remove such that the remaining points are completely separable by a single hyperplane. Because the points removed will be incorrectly classified by the resulting hyperplane, this constitutes a method of finding a hyperplane that misclassifies the smallest number of points, an important goal in classification research.

Table 1 provides information about nine frequently analyzed binary classification problems taken from the publicly available UCI Repository of Machine Learning Databases (Blake and Merz 1998), a common source of classification test data. "Net points" is the number of data instances remaining after incomplete tuples are removed.

Table 2 compares the results obtained when three different algorithms are applied to these data sets:

- Algorithm 1 as implemented in CLIIS.
- Algorithm 2(1) (i.e. choosing only the violated constraint having the maximum product as the single candidate each time) as implemented in CLIIS.
- MISMIN (Bennett and Bredensteiner 1997).

For a direct comparison, all algorithms are applied to the entire data set (i.e. there is no separation into training and testing sets). The best results in terms of both accuracy (*%acc.*) and time (*secs*) are shown in boldface.

Table 1 Classification Data Sets

Data Set	Net Points	Number of Features
breast cancer	683	9
bupa	345	6
glass (type 2 vs. others)	214	9
ionosphere	351	34
iris (versicolor vs. others)	150	4
iris (virginica vs. others)	150	4
new thyroid (normal vs. others)	215	5
pima	768	8
wpbc	194	32

Table 2 Three Algorithms for Classification

	Original Alg. 1		Algorithm 2(1)		MISMIN	
Data set	% acc.	secs	% acc.	secs	% acc.	secs
breast cancer	**98.4**	17	**98.4**	4.3	98.2	**0.7**
bupa	75.1	159	**75.9**	1.3	73.9	**0.6**
glass (type 2 vs. others)	**81.8**	38	78.5	**0.6**	76.6	**0.6**
ionosphere	**98.3**	44	**98.3**	5.4	**98.3**	**2.6**
iris (versicolor vs. others)	**83.3**	5	**83.3**	**0.2**	82.0	0.3
iris (virginica vs. others)	**99.3**	0.4	**99.3**	**0.1**	**99.3**	0.3
new thyroid (normal vs. others)	**94.9**	3	**94.9**	**0.3**	93.5	**0.3**
pima	**80.6**	1434	80.2	7.2	80.5	**1.5**
wpbc	**96.9**	17	**96.9**	**1.2**	91.2	1.5
average:	89.8	216.2	89.5	2.3	88.2	0.9

Table 3 Accuracy of the Maximum-Product Predictor of ΔSINF

(Predicted ΔSINF)/ (Actual ΔSINF)	pima: No. of cases	bupa: No. of Cases	% of Total Cases
0.99 or above	82	23	45
0.98	27	18	19
0.97	14	9	10
0.96	10	11	9
0.95	4	6	4
0.94	2	3	2
0.93	1	2	1
0.92	3	1	2
0.91	2	1	1
0.90	0	2	1
less than 0.90	7	7	6
total misclassified	152	83	100%

Because MISMIN is among the best of the available programs for minimizing the number of classification errors in classification problems, it is a good standard for comparison. Bennett and Bredensteiner (1997) show that MISMIN performs favorably against such other well-known programs as OC1 (Murthy et al. 1994) and CSADT (Heath et al. 1993).

Table 2 shows that, for classification problems, Algorithm 1 is the most accurate, but also the slowest, while MISMIN is the fastest. Algorithm 2(1) provides a major speed-up over Algorithm 1 (several orders of magnitude in some cases), yielding times comparable to those for MISMIN (and sometimes faster). More significant, however, is that it does this with very little loss of accuracy.

An important difference between the approach taken in Algorithms 1 and 2 as opposed to many other methods is that Algorithms 1 and 2 remove points from the data sets one at a time instead of all at once as in other methods. This raises the possibility of "guiding" the removal process as it is underway. For example, if the classification accuracy of Type 0 points is lower than that of Type 1 points, then the removal process could be coerced to prevent the removal (and hence misclassification) of more Type 0 points until the classification accuracies are balanced. This idea is under active study at the moment.

Table 3 presents an in-depth assessment of the accuracy of the maximum-product heuristic estimator for the ΔSINF expected when a violated constraint is deleted. Results are presented for the two data sets that take significantly longer to solve using the original Algorithm 1: *pima* and *bupa*. For each constraint removed by Algorithm 2(1), the predicted ΔSINF is compared to the actual ΔSINF experienced when the constraint is actually deleted. Table 3 shows that the maximum-product heuristic is remarkably accurate in predicting ΔSINF: it is over 95% accurate in 87% of the cases examined, and over 90% accurate in 94% of the cases examined.

The high accuracies shown in Table 3 are partly due to the tightly constrained classification data sets. The algorithms delete constraints one by one, and as each constraint is deleted, the current approximate separating hyperplane (given by the phase 1 solution) shifts to a new position. When you need to delete 152 constraints to achieve feasibility as in the *pima* problem, then the deletion of a single individual constraint does not usually shift the position of the current approximate separating hyperplane very much. However, as you near the termination of the algorithm, there are fewer conflicting constraints, and the removal of a single constraint may permit a large movement of the approximate separating hyperplane. For this reason, the largest errors in prediction accuracy in Table 3 all occur very near the end of the list of constraint removals.

4.2. General Infeasible Linear Programs

A set of 29 infeasible LP models is available in the Netlib library at http://www.netlib.org/lp/infeas/. The model characteristics are summarized in Table 4.

The MINOS phase 1 procedure examines both SINF and NINF, and terminates when it decides that NINF cannot be further reduced (Wolfe 1965). This can be quite effective in finding small values of NINF: the MINOS phase 1 procedure reports a single violated constraint for 14 of the 29 models: *bgetam, box1, ceria3d, cplex2, ex72a, ex73a, forest6, galenet, gosh, klein1, pang, pilot4i, qual, vol1* (Chinneck 1996b). Step 1 of all algorithms then reports this single violated constraint as the IIS set cover, and of course this is the minimum-cardinality cover (Observation 2). This prevalence of single-member IIS set covers arises because a number of the models were feasible LPs into which a single infeasibility was manually introduced.

Table 4 Characteristics of the Netlib Infeasible LPs

Model	Rows	Cols	Nonzeroes
bgdbg1	349	407	1485
bgetam	401	688	2489
bgindy	2672	10116	75019
bgprtr	21	34	90
box1	232	261	912
ceria3d	3577	824	17604
chemcom	289	720	2190
cplex1	3006	3221	10664
cplex2	225	221	1059
ex72a	198	215	682
ex73a	194	211	668
forest6	67	95	270
galenet	9	8	16
gosh	3793	10733	97257
gran	2659	2520	20151
greenbea	2505	5405	35159
itest2	10	4	17
itest6	12	8	23
klein1	55	54	696
klein2	478	54	4585
klein3	995	88	12107
mondou2	313	604	1623
pang	362	460	2666
pilot4i	411	1000	5145
qual	324	464	1714
reactor	319	637	2995
refinery	324	464	1694
vol1	324	464	1714
woodinfe	36	89	209

We will concentrate on 14 of the remaining 15 more difficult models. Parker and Ryan (1996) have provided exact solutions for these 14 models using their integer-programming method, so there is a basis for measuring the quality of the solutions returned by the fast heuristics. The remaining model (*gran*) causes numerical difficulties for all of the implementations, including Parker and Ryan's, and is omitted. The minimum cover cardinalities are shown in boldface in Table 5.

We will look at several algorithms:

- the MINOS phase 1 procedure applied to the non-elastic model,
- the first solution of the full elastic version of the model (basically a full elastic phase 1),
- Algorithm 1 applied to the standard elastic version of the model,
- Algorithm 2(1) applied to the full elastic version of the model,
- Algorithm 2(7) applied to the full elastic version of the model,
- Algorithm 3(1) applied to the full elastic version of the model,
- Algorithm 3(7) applied to the full elastic version of the model.

Algorithm 4 is constructed by combining the phase 1 methods and the selection criteria from one of the latter 5 algorithms. Examining each of these elements individually will show how best to construct Algorithm 4.

The list length for Algorithms 2 and 3 can be set as desired. Shorter lists are faster, but longer lists are more accurate. With a sufficiently long list, Algorithm 3 is equivalent to Algorithm 1. Experimentation with different relatively short lists showed that a length of 7, particularly for Algorithm 3, is quite effective. Results with list lengths of 1 and 7 for Algorithms 2 and 3 are given in Table 5.

Table 5 separates the time required for the phase 1 method from the time required for the remainder of the algorithm. Hence the total computation time, from problem input through output of the IIS cover, is roughly the sum of the phase 1 time and the time listed with the algorithm in Table 5. For example, the

Table 5 Comparison of Algorithms on Difficult General LP Problems

	MINOS Phase 1		Algorithm 1			Full Elastic ph. 1	Algorithm 2(1)			Algorithm 2(7)			Algorithm 3(1)			Algorithm 3(7)		
Model	NINF	Secs	NINF	Secs	LPs	NINF	NINF	Secs	LPs	NINF	Secs	LPs	NINF	Secs	Lps	NINF	Secs	LPs
A																		
bgprtr	2	0.0	**1**	0.1	1	**1**	**1**	0.0	0	**1**	0.0	0	**1**	0.0	0	**1**	0.0	0
itest2	**2**	0.0	**2**	0.0	7	**2**	**2**	0.0	1	**2**	0.0	2	**2**	0.0	2	**2**	0.0	6
mondou2	**3**	0.0	**3**	1.5	384	8	7	0.0	6	5	0.4	25	6	0.1	11	5	0.5	53
reactor	3	0.1	**1**	0.3	25	**1**	**1**	0.0	0	**1**	0.0	0	**1**	0.0	0	**1**	0.0	0
woodinfe	**2**	0.0	**2**	0.0	47	**2**	**2**	0.0	1	**2**	0.0	2	**2**	0.0	2	**2**	0.0	4
B																		
bgdbg1	22	0.1	**12**	3.4	645	13	**12**	0.1	11	**12**	0.4	65	**12**	0.2	22	**12**	0.8	142
bgindy	14	12.1	**1**	2.3	1	2	**1**	0.1	1	**1**	0.1	1	**1**	0.1	1	**1**	0.1	1
chemcom	11	0.2	**1**	0.1	2	14	**1**	0.0	1	**1**	0.1	1	**1**	0.0	1	**1**	0.0	1
greenbea	3	34.8	**2**	50.8	404	6	**2**	7.2	1	**2**	25.3	6	**2**	12.2	2	**2**	25.4	13
C																		
itest6	3	0.0	**2**	0.0	10	5	4	0.0	4	**2**	0.0	7	4	0.0	7	**2**	0.0	8
klein3	6	3.1	**1**	17.2	53	23	9	1.6	9	**1**	4.0	7	4	1.4	8	**1**	4.0	7
D																		
cplex1	211	9.7	**1**	13.0	213	211	211	5.9	210	211	54.9	1455	4	5.5	8	**1**	5.1	9
klein2	3	0.7	**1**	1.3	17	5	3	0.1	2	3	0.8	7	2	0.4	4	**1**	1.1	11
E																		
refinery	2	0.2	**1**	1.1	36	7	3	0.0	2	3	0.1	9	3	0.0	4	2	0.1	18
# min NINF	3		14			**4**	8			10			8			12		
avg. NINF	20.5		2.2			21.4	18.5			17.6			3.2			2.4		
avg. secs.		4.4		6.5				1.1			6.2			1.4			2.7	
avg. LPs					131.8				17.8			113.4			5.1			19.5

Note: Times not directly comparable between Algorithm 1 vs. Algorithms 2 and 3.

total time for Algorithm 1 is approximately the sum of the time for the MINOS phase 1 and the time listed for Algorithm 1 (total time averages 10.9 seconds). Times are not given for the "full elastic phase 1 NINF" procedure, as this is expected to be provided by a two-stage process: the solvers native phase 1 followed by a full elastic phase 1 with an advanced start provided by the solver native phase 1 solution.

Computation times in Table 5 are not directly comparable between Algorithm 1 on the one hand and Algorithms 2 and 3 on the other hand due to implementation details, as explained in Section 3. However, Table 2 gives a sense of the magnitude of the speed-up when using Algorithm 2(1) versus Algorithm 1: Algorithm 2(1) requires about two orders of magnitude less time than is required for Algorithm 1.

Observations about expected speed can be made based on the number of LP solutions required by each method. As expected, Algorithm 1 requires the most LP solutions on average (131.8) while Algorithm 3(1) requires the fewest (5.1). These average results are somewhat skewed by *cplex1*. In 12 of the 14 models, Algorithm 2(1) requires the smallest number of LPs. Algorithms 2(1) and 3(1) are both very quick in comparison to Algorithm 1.

It is most instructive to concentrate on the models that require the most LPs for Algorithm 1 to solve. Let us look in detail at the 4 models that require more than 100 LPs for solution by Algorithm 1. Table 6 shows the ratio of the number of LPs needed by the new algorithms to the number of LPs needed by Algorithm 1 for these 4 models. Solutions that achieve a true MIN IIS COVER are shown in bold-

Table 6 Ratio of LPs Solved Compared to Algorithm 1

model	Alg. 2(1)	Alg. 2(7)	Alg. 3(1)	Alg. 3(7)
bgdbg1	**0.02**	**0.10**	**0.03**	**0.22**
cplex1	0.99	6.80	0.04	**0.04**
greenbea	**0.00**	**0.01**	**0.00**	**0.03**
mondou2	0.02	0.07	0.03	0.14
average	0.26	1.75	0.03	0.11

face. In 14 of the 16 solutions, the new algorithms require only a small fraction of the effort needed by Algorithm 1 (the exception is Algorithm 2 operating on *cplex1*), and are reasonably accurate. The selection criteria of Algorithm 3(7) used in the Algorithm 4 framework will achieve a MIN IIS COVER in all of these cases. Table 6 shows that the new algorithms achieve significant speed improvements for the hardest problems, in most cases.

Accuracy of the new algorithms is not perfect. Algorithm 2 does poorly on *cplex1*, in terms of both accuracy and speed. Ignoring *cplex1* gives Algorithm 2(1) an average of 3.0 LPs (instead of 17.8), and Algorithm 2(7) an average of 10.2 LPs (instead of 113.4). A corollary observation is that a change of algorithms can have a dramatic impact on accuracy for a particular model. Algorithms 1 and 3(7) are the only ones able to achieve the true MIN IIS COVER for *cplex1*.

The results in Table 5 are broken down into five groups. The five models in Group A are those whose MIN IIS COVER is found by one or both of the two phase 1 procedures applied. In fact, the MINOS phase 1 is the only procedure to find a MIN IIS COVER for *mondou2*. Because of this, in the framework of Algorithm 4, a true MIN IIS COVER will be found for all five models in Group A in conjunction with any of the versions of Algorithms 2 and 3. This argues for the inclusion of the MINOS-style phase 1 procedure in Algorithm 4, as does the excellent performance of the MINOS phase 1 on the other 14 models for which it found a single-member IIS cover.

The four models in Group B of Table 5 are those whose MIN IIS COVER is found by all of the new algorithms, including the fast short-list versions. The reduction in the number of LPs solved by the new algorithms versus Algorithm 1 is very dramatic for *bgdbg1* and *greenbea*. This underlines the effectiveness of the new algorithms.

The two models in Group C of Table 5 are those that require a longer list length to find a MIN IIS COVER. Each of the four new algorithms requires almost the same small number of LPs to arrive at a solution, but the quality of the results returned by Algorithms 2(7) and 3(7) is much better. This argues for the longer list lengths in Algorithms 2 and 3.

The 2 models in Group D of Table 5 benefit from the application of Algorithm 3. Even with a list length of 1, the *cplex1* model shows a cover cardinality of 4 using Algorithm 3 versus a cover cardinality of 211 using Algorithm 2. Algorithm 3(7) finds the true MIN IIS COVER in both cases. This argues for the use of the selection criteria of Algorithm 3 in the framework of Algorithm 4.

Finally, a MIN IIS COVER is not found using any heuristic method for the single model in Group E of Table 5. However, the best result, provided by both the MINOS phase 1 and by Algorithm 3(7), is very close to the optimum at only 1 greater than the true minimum cardinality.

The results obtainable via Algorithm 4 can be estimated from Table 5. Algorithm 4 coupled with any of the selection criteria in Algorithms 2(1), 2(7), 3(1), or 3(7) will find a MIN IIS COVER for all models in groups A and B. The safety exit provided by the phase 1 solutions will trigger in the following cases, saving the solution of a few LPs:

- *mondou2* with selection criteria 2(1), 2(7), 3(1), 3(7),
- *itest2* with selection criteria 2(1) and 3(1),
- *klein3* with selection criteria 2(1),
- *refinery* with selection criteria 2(1), 2(7), and 3(1).

This argues for the use of the Algorithm 4 framework.

These empirical results indicate that an effective version of Algorithm 4 would incorporate a MINOS-style phase 1 procedure, and would use the selection criteria of Algorithm 3 at list length 7. This provides a significant speedup for general LP problems with very little loss in accuracy: it fails to find a MIN IIS COVER only for the *refinery* model, and the error in that

case is just 1. This algorithm is about 7 times faster than Algorithm 1 on average.

For maximum speed at reasonable accuracy, use Algorithm 4 with selection criteria from Algorithm 3(1). This does not give a poor result on any of the test models. On the five models for which this combination does not achieve a MIN IIS COVER, the maximum distance from optimality is 3, and the average is 1.8. This algorithm is about 25 times faster than Algorithm 1 on average, and dramatically faster on many models.

5. Conclusions

Algorithm 4 with a MINOS-style native phase 1 procedure and the selection criteria of Algorithm 3(7) is recommended for maximum accuracy with a good improvement in speed. Algorithm 4 with a MINOS-style native phase 1 and the selection criteria of Algorithm 3(1) is recommended for maximum speed with a slightly larger degradation in accuracy.

The recommended algorithms provide faster heuristics for the MAX FS, MIN ULR, and MIN IIS COVER problems, with significant speed-ups as compared to previous heuristics, at little loss of accuracy. The main elements of these heuristics are two methods for identifying constraints that are promising candidates for deletion from the model: violated constraints having a large (constraint violation)$\times$|(constraint sensitivity)| product, and nonviolated constraints to which the elastic objective function is highly sensitive. In fact, the maximum-product measure is remarkably accurate at directly choosing the correct candidate for deletion in classification problems.

As described in the Introduction, various NP-hard problems can be transformed into MAX FS, MIN ULR, or MIN IIS COVER. Further research may reveal other transformations as simple and efficient as that for the data classification problem. All such transformed problems can then be attacked via the fast polynomial-time heuristics described here.

While the recommended algorithms can be used to analyze LP infeasibility after it has been discovered, they can also be incorporated directly in the solver as part of an improved phase 1 procedure whose goal is to minimize NINF.

An interesting feature of these algorithms is that constraints are deleted one at a time as the algorithm proceeds, in contrast to the all-at-once nature of some algorithms for classification (e.g. Bennett and Bredensteiner 1997). This feature can be used to influence the solution as it proceeds. For example, if the accuracy of one population type is less than the other in a classification problem, then the solution process can be influenced to favor improved accuracy of that population as the algorithm proceeds.

Finally, these algorithms are easily adapted to the related case of minimum-weight IIS set covering (Parker and Ryan 1996). In this version of the problem, different weights are attached to the various constraints in the model, and the idea is to find the minimum-weight set of constraints that renders the model feasible when removed. This is accomplished by attaching weights to the elastic variables in the elastic objective function. These objective function weights should be inversely related to the weights of the associated constraints.

Acknowledgments

Thanks to Erin Bredensteiner for providing the MISMIN results in Table 2. This research is partly sponsored by the Natural Sciences and Engineering Research Council of Canada via a research grant to the author.

References

Amaldi, E. 1994. From finding maximum feasible subsystems of linear systems to feedforward neural network design. Ph.D. thesis no. 1282, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland.

Amaldi, E., V. Kann. 1995. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science* **147** 181–210.

Amaldi, E., M. Pfetsch, L. Trotter Jr. 1999. Structural and algorithmic properties of the maximum feasible subsystem problem, *Proceedings of the Integer Programming and Combinatorial Optimization conference (IPCO'99), Lecture Notes in Computer Science* **1610**, Springer–Verlag, New York, NY. 45–59.

Bennett, K. P., E. Bredensteiner. 1997. A parametric optimization method for machine learning. *INFORMS J. on Computing* **9** 311–318.

Blake, C. L., C. J. Merz. 1998. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Brown, G., G. Graves. 1975. Elastic programming: a new approach to large-scale mixed integer optimization. ORSA/TIMS conference, Las Vegas, NV.

Chakravarti, N. 1994. Some Results Concerning Post-Infeasibility Analysis. *Eur. J. Oper. Res* **73** 139–143.

Chen, C., O. L. Mangasarian. 1996. Hybrid misclassification minimization. *Advances in Computational Mathematics* **5** 127–136.

Chinneck, J. W. 1994. MINOS(IIS): infeasibility analysis using MINOS. *Computers and Operations Research* **21** 1–9.

Chinneck, J. W. 1996a. Computer codes for the analysis of infeasible linear programs. *Journal of the Operational Research Society* **47** 61–72.

Chinneck, J. W. 1996b. An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem. *Annals of Mathematics and Artificial Intelligence* **17** 127–144.

Chinneck, J. W. 1997. Feasibility and viability. *Advances in Sensitivity Analysis and Parametric Programming*, T. Gal and H. J. Greenberg, eds., *International Series in Operations Research and Management Science* **6** 14-1–14-41, Kluwer Academic Publishers, Boston, Mass.

Chinneck, J. W. 1998. Improved linear classification via LP infeasibility analysis. Technical Report SCE-98-09, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

Chinneck, J. W., E. W. Dravnieks. 1991. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing* **3** 157–168.

CPLEX Optimization, Inc. 1994. Using the CPLEX callable library and CPLEX mixed integer library, CPLEX Optimization Inc., Incline Village, Nevada.

Frean, M. 1992. A. "thermal" perceptron learning rule. *Neural Computation* **4** 946–957.

Heath, D., S. Kasif, S. Salzburg. 1993. Learning oblique decision trees. *Proceedings of the 13th International Conference on Artificial Intelligence*, Chambery, France, Morgan Kaufmann, San Mateo, CA. 1002–1007.

Karmarkar, N. 1984. A new polynomial time algorithm for linear programming. *Combinatorica* **4** 373–395.

Mangasarian, O. 1994. Misclassification minimization. *Journal of Global Optimization* **5** 309–323.

Murtagh, B. A., M. A. Saunders. 1993. MINOS 5.4 user's guide (preliminary). Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA.

Murthy, S., S. Kasif, S. Salzberg. 1994. A System for induction of oblique decision trees. *J. Artificial Intelligence Research* **2** 1–32.

Parker, M. R. 1995. A set covering approach to infeasibility analysis of linear programming problems and related issues. Ph.D. thesis, Dept. of Mathematics, University of Colorado at Denver, Denver, Colorado.

Parker, M. R., J. Ryan. 1996. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence* **17** 107–126.

Sankaran, J. K. 1993. A note on resolving infeasibility in linear programs by constraint relaxation. *Operations Research Letters* **13** 19–20.

Wolfe, P. 1965. The composite simplex algorithm. SIAM Review **7** 42–54.

Accepted by W. David Kelton; received March 2000; revised November 2000; accepted January 2001.