

# Faster Maximum Feasible Subsystem Solutions for Dense Constraint Matrices

Fereshteh Fakhra Firouzeh\*, John W. Chinneck, Sreeraman Rajan

*Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada*

---

## Abstract

Finding the largest cardinality feasible subset of an infeasible set of linear constraints is the *Maximum Feasible Subsystem* problem (MAX FS). Solving this problem is crucial in a wide range of applications such as machine learning and compressive sensing. Although MAX FS is NP-hard, useful heuristic algorithms exist, but these can be slow for large problems. We extend the existing heuristics for the case of dense constraint matrices to greatly increase their speed while preserving or improving solution quality. We test the extended algorithms on two applications that have dense constraint matrices: binary classification, and sparse recovery in compressive sensing. In both cases, speed is greatly increased with no loss of accuracy.

*Keywords:* Maximum Feasible Subsystem, Linear Programming, Classification, Compressive Sensing

---

## 1. Introduction

Finding the maximum cardinality feasible subsystem of an infeasible set of linear constraints is known as the *Maximum Feasible Subsystem* problem (MAX FS) [1]. Its complement is the *Minimum Unsatisfied Linear Relation* problem (MIN ULR) [2] of finding the minimum number of constraints to remove from an infeasible set such that the remaining constraints have a feasible solution. Moreover, all infeasible systems have at least one *Irreducible Infeasible Subsets* (IISs) in their original set of constraints. An IIS is a set

---

\*Corresponding author

*Email address:* behnazfakhrafirouzeh@cmail.carleton.ca (Fereshteh Fakhra Firouzeh)

of constraints that is infeasible, but is rendered feasible if any constraint is removed. If all of the IISs in the model are enumerated, then a MIN ULR solution is found as a minimum cover of the IISs; this is known as the *minimum IIS cover* problem [3]. In this paper, MAX FS, MIN ULR, and MIN IIS COVER are the same problem and the terms will be used interchangeably.

MAX FS is NP-hard [4, 5, 6] so exact solution algorithms are only available for relatively small instances, but solutions are still needed in a wide variety of applications including machine learning [7], misclassification minimization [8], training of neural networks [2], telecommunications [9], computational biology [10], and compressive sensing (CS) [11]. To be useful in practice, MAX FS heuristics should obtain high quality solutions (large feasible subsets, or equivalently, small MIN ULR subsets) quickly.

The MAX FS problem can be formulated for exact solution in various ways, including as a mixed-integer linear program [12], a linear program with equilibrium constraints (LPEC) [13, 14, 15, 16], or as a type of set-covering problem [17, 18, 19, 20]. As shown in chapter 7 of [12], there are practical difficulties in using any of these exact methods to solve this NP-hard problem (i.e. they can be used for a very small MAX FS problem). For this reason, heuristic solutions are required.

Chinneck [3, 21] developed a set of effective linear programming (LP)-based heuristics for solving the MAX FS problem, and showed experimentally that they give better results than methods such as MISMIN, a state-of-the-art LPEC solver at the time [21]. Variants of Chinneck’s methods have been developed more recently for finding sparse solutions to underdetermined systems of linear equations in unbounded variables [11] for sparse recovery in compressive sensing.

This paper extends Chinneck’s algorithms to provide major improvements in solution speed for the case of dense constraint matrices, based on the observation that dense constraint matrices provide multiple similar candidates for removal during the MAX FS solution process. The new extensions are evaluated on the following two applications that have dense constraint matrices: binary classification, and sparse recovery in compressive sensing.

Chinneck’s algorithms remove constraints from the original set one by one until what remains is the heuristic MAX FS solution; the removed constraints constitute the heuristic MIN ULR solution. An inner loop assesses candidate constraints, and an outer loop removes the single constraint chosen by the inner loop. Two extensions that greatly improve solution speed when a dense

constraint matrix provides multiple similar candidates are proposed in this paper. Extension 1 amalgamates the two loops into a single one that both assesses the candidates and removes *multiple* constraints at each iteration. Extension 2 provides an early exit under certain conditions. When tested on binary classification and compressive sensing sparse recovery, the extended heuristics greatly reduce solution time while preserving or improving solution quality.

The paper is structured as follows. Section 2 reviews the existing MAX FS solution heuristics. The new extensions are developed in Section 3. The experimental results for binary classification are given in Section 4 and for compressive sensing sparse recovery in Section 5. Section 6 concludes the paper and outlines future work.

## 2. LP-based MAX FS Solution Heuristics

Chinneck’s original polynomial time heuristics [3] are briefly summarized here. All variants first elasticize the infeasible set of linear row constraints (and possibly variable bounds) by adding nonnegative *elastic variables* as shown in Table 1. The elastic variables measure the extent to which each constraint is violated, so a linear program (LP) minimizing their sum finds the minimum total violation for an infeasible set of constraints and bounds. There are two types of elastic models: the *standard elastic LP* elasticizes only the row constraints, and the *full elastic LP* elasticizes both the row constraints and the column bounds. The objective for a full elastic LP is:  $\min Z = \sum_i (e_i^+ + e_i^-) + \sum_j (e_j^+ + e_j^-)$  for constraints  $i = 1 \dots m$  and bounded variables  $j = 1 \dots n$ ; the second term is omitted for a standard elastic LP.  $Z = 0$  indicates a feasible system.

Table 1: Elasticizing constraints by adding non-negative elastic variables.

Type	Nonelastic	Standard elastic	Full elastic
Row Cons.	$\sum_j a_{ij}x_j \geq b_i$	$\sum_j a_{ij}x_j + e_i \geq b_i$	$\sum_j a_{ij}x_j + e_i \geq b_i$
	$\sum_j a_{ij}x_j \leq b_i$	$\sum_j a_{ij}x_j - e_i \leq b_i$	$\sum_j a_{ij}x_j - e_i \leq b_i$
	$\sum_j a_{ij}x_j = b_i$	$a_{ij}x_j + e_i^+ - e_i^- = b_i$	$\sum_j a_{ij}x_j + e_i^+ - e_i^- = b_i$
Var. Bnds	$x_j \geq l_j$	$x_j \geq l_j$	$x_j + e_j^+ \geq l_j$
	$x_j \leq u_j$	$x_j \leq u_j$	$x_j - e_j^- \leq u_j$

As shown in Fig. 1, the General MAX FS algorithm of [3] has an inner loop (line 10) and an outer loop (line 5). The inner loop tests each member

of a set of *candidate constraints* in *CandidateSet* by examining their effect on  $Z$  when temporarily removed from the model. The candidate constraint that most reduces  $Z$  when removed is selected as the *Winner* candidate, and is subsequently removed from the model and added to *MINULR* in the outer loop. This process continues until  $Z = 0$  (feasibility is reached). The constraints remaining in the model constitute the heuristic MAX FS solution; the removed constraints in *MINULR* constitute the heuristic MIN ULR solution. Algorithm variants differ in how the list of candidate constraints is constructed.

The heuristic relies on *Observation 3* from [3], which notes that a constraint that appears in more than one IIS will reduce  $Z$  more than a constraint that appears in a single IIS because it will eliminate multiple IISs upon removal. This means that a large drop in  $Z$  when a constraint is removed is a useful sign that the constraint is part of the MINULR set. The larger the drop in  $Z$ , the more IISs the constraint likely covers.

There are efficiency enhancements in terms of algorithm processing time. If the *CandidateSet* has a single element (line 6), then that element is added to *MINULR* and the algorithm exits: there is no need to solve the LP. As the *Winner* constraint is updated, so is the list of candidates for the next round, *NextCandidateSet*. All versions of the algorithm solve multiple LPs, but each LP is identical to the last one except for two constraints, so simplex advanced starts to speed the process considerably.

Algorithm 1 to 3 of the base algorithm differ in how the list of candidate constraints is constructed, as described next.

### 2.1. Algorithm 1

Candidate constraints are those to which the elastic objective function is sensitive (nonzero dual price). Constraints not in this list do not affect  $Z$  when dropped, so they are not candidates (*Observation 4* from [3]). This is the longest possible candidate list, but its length can be limited as follows. Sort the constraints in descending order by their absolute dual cost and take only the first  $k$  elements of the sorted list. This is referred to as Algorithm 1( $k$ ). For clarity in the sequel, unlimited lists are indicated by  $k = \infty$ .

### 2.2. Algorithm 2

Algorithm 2 uses a different criterion for constructing the *CandidateSet* based on *Observation 5* in [21]: a good predictor of the magnitude of the drop in  $Z$  obtained by deleting a violated constraint  $j$  is given by Eqn. (1):

---

**Figure 1** General MAX FS Algorithm [3]

---

```
1:  $MINULR \leftarrow \emptyset$ 
2: Set up elastic LP.
3: Solve elastic LP.
4: Construct  $CandidateSet$ .
5: do [outer loop]:
6:   if  $|CandidateSet| = 1$  then
7:     Add the single candidate to  $MINULR$  and exit.
8:   end if
9:    $WinnerZ \leftarrow \infty$ .
10:  for each candidate  $j$  in  $CandidateSet$  do [inner loop]:
11:    Delete candidate  $j$ .
12:    Solve elastic LP.
13:    if  $Z = 0$  then
14:      Add candidate  $j$  to  $MINULR$  and exit.
15:    end if
16:    if  $Z < WinnerZ$  then
17:       $Winner \leftarrow$  candidate  $j$ .
18:       $WinnerZ \leftarrow Z$ .
19:      Construct  $NextCandidateSet$ .
20:    end if
21:    Reinststate candidate  $j$ .
22:  end for
23:  Add  $Winner$  to  $MINULR$ .
24:  Delete  $Winner$  permanently.
25:   $CandidateSet \leftarrow NextCandidateSet$ .
26: end do
OUTPUT:  $MINULR$  is a heuristic MIN ULR solution.
```

---

$$Product : e_j \times |constraint\ j\ dual\ price| \quad (1)$$

For equality constraints, the larger of the two elastic variables is used in Eqn. (1). Algorithm 2 can also limit the length of the candidate list. First sort the candidate constraints in descending order by product magnitude, and then add only the top  $k$  constraints to *CandidateSet*; this is Algorithm 2( $k$ ). List length  $k = 1$  is particularly useful in some applications, where it is referred to as the *maximum product* algorithm.

### 2.3. Algorithm 3

Algorithm 3 uses *Observation 6* [21]: for satisfied constraints, a good predictor of the *relative* magnitude of any drop in  $Z$  that may be obtained by deleting the constraint is given by  $|constraint\ dual\ price|$ . Algorithm 3 uses 2 lists: (i) for violated constraints, the list of product magnitudes (Eqn. (1)), and for satisfied constraints (ii) the list of absolute constraint sensitivities. Both lists are independently sorted in descending order, and the top  $k$  elements are taken from each list, resulting in  $2k$  candidates in total. This is referred to as Algorithm 3( $k$ ).

### 2.4. Finding Sparse Solutions to Linear Systems

A sparse solution is one having very few nonzeros. Algorithms 1-3 can be used to find sparse solutions to general systems of linear constraints as follows. Given the system  $\mathbf{Ax} \{ \leq, =, \geq \} \mathbf{b}$ , add the variable zeroing constraints  $\mathbf{x} = \mathbf{0}$ . Now apply any of Algorithms 1-3 with candidates chosen only from among the constraints  $\mathbf{x} = \mathbf{0}$ . The MAX FS solution provides the sparsest solution by allowing the smallest number of nonzero variables. Finding the sparsest solution to an underdetermined system of linear equalities in unbounded variables is of particular interest in a number of applications. We focus on this problem here.

Chinneck's algorithms have been adapted for underdetermined systems of linear equations in unbounded variables. Jokar and Pfetsch [22] modified Chinneck's algorithm as follows. Given the  $m \times n$  linear system  $\mathbf{Ax} = \mathbf{b}$  in unbounded variables, all variables  $x_j$  are first replaced by  $u_j - v_j$  where both  $u_j$  and  $v_j$  are nonnegative, and the following LP is constructed:

$$\min Z = \sum_j (u_j + v_j) \quad s.t. \quad \mathbf{A}(\mathbf{u} - \mathbf{v}) = \mathbf{b}, \quad \mathbf{u} \geq \mathbf{0}, \mathbf{v} \geq \mathbf{0} \quad (2)$$

This LP has  $m$  constraints in  $2n$  nonnegative variables. The objective function attempts to drive all variables to zero in a manner similar to driving elastic variables to zero in Algorithms 1-3.

The basic algorithm follows the logic in Fig. 1, except that candidates are variables and not constraints. Additional differences from Fig. 1:

- Candidate variables are those not in the MIN ULR set and having a nonzero value in the current LP solution. Candidates are sorted in decreasing order by the magnitude of  $|u_j - v_j|$ .
- Candidate variables are tested in the inner loop by temporarily removing them from the objective function by setting the objective coefficients of the associated  $u_j, v_j$  pair to zero and re-solving the LP.
- A candidate variable  $x_j$  is moved into the MIN ULR set by resetting the objective function coefficients of  $u_j$  and  $v_j$  to zero permanently so that  $x_j > 0$  no longer affects  $Z$ .
- At exit, the variables in the MIN ULR set are those that can take nonzero values in the sparse solution.
- Extraneous variables are sometimes included in the MIN ULR solution. A simple post-processing step may remove them. First construct  $\mathbf{A}_{mn}$  by eliminating all columns from  $\mathbf{A}$  that are not in the MIN ULR solution. For each variable in the MIN ULR set in turn, force it to zero and test the feasibility of  $\mathbf{A}_{mn}\mathbf{x} = \mathbf{b}$ ; if feasible then the tested variable is permanently removed from the MIN ULR set.

The well-known *BasisPursuit* (BP) algorithm [23] solves system (2) a single time. This works well if the solution is very sparse, but when this is not the case, Fig.1 with the modifications above is more effective.

Several other recent variants have been shown to be very effective for this problem [24, 11]:

- **Method B** uses system (2). The objective coefficients of the  $u_j$  and  $v_j$  associated with variable  $x_j$  moved into the MIN ULR set are permanently reset to 0.1 instead of zero [24]. Since  $Z$  never reaches 0, exit is instead triggered when  $|CandidateSet| = 0$ .

- **Method M** combines Method B with *Basis Pursuit* (BP) [23], based on the observation that if BP fails, it returns a solution that is typically of size  $m$  or nearly so. Thus, the BP result (the set of nonzeros in the first solution of (2)) is taken when the number of nonzeros is less than  $m - 3$ . Method B is applied when the number of nonzeros in that first solution is larger than  $m - 3$ .
- **Method C** has explicit elastic variable zeroing constraints, resulting in the following LP:

$$\min Z = \sum_j (e_j^+ + e_j^-) \text{ s.t. } \begin{bmatrix} \mathbf{A} & \mathbf{0}_{m \times n} & \mathbf{0}_{m \times n} \\ \mathbf{I} & \mathbf{I} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{e}^+ \\ \mathbf{e}^- \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{m \times 1} \\ \mathbf{0}_{n \times 1} \end{bmatrix} \quad (3)$$

where  $e_j^+$  and  $e_j^-$  are nonnegative,  $\mathbf{A}$  is  $m \times n$  and  $\mathbf{I}$  is  $n \times n$ . This LP has  $m + n$  constraints in  $3n$  variables. There are two lists of candidates: (1) a list based on the magnitude of nonzero variables not yet assigned to the MIN ULR set, and (2) a list based on the dual prices of the variable zeroing constraints  $x_j + e_j^+ - e_j^- = 0$ . The selected variable is added to the MIN ULR set by zeroing the objective coefficients of the corresponding elastic variables.

### 3. Extensions to Increase Solution Speed

The existing algorithms are effective, but there are opportunities for improving their speed. Limiting the length of the candidate list is helpful, but there are still two bottlenecks: evaluating multiple candidates in the inner loop, and removing just a single constraint in the outer loop. New Extension 1 eliminates the inner loop entirely and removes multiple candidate constraints simultaneously in the single remaining loop. The trick is in identifying which constraints to remove in each iteration. New Extension 2 provides an early exit under certain conditions.

Each of the extensions can be combined with any of the existing list construction algorithms to create new combinations, e.g. combining Algorithm 2( $\infty$ ) and Extension 1 creates Algorithm 2( $\infty$ )E1.

#### 3.1. Extension 1(E1)

There are three ways to assemble the candidate list in Algorithms 1-3. Regardless of the type of list, previous research shows that the order in the

sorted list of candidates is important: those earlier in the list are much more likely to be part of the MIN ULR than those later in the list. Thus, if we remove multiple constraints at each iteration of a single loop, it makes sense to select them from the start of the sorted list. This argument is especially potent when the constraint matrix is dense. In a dense constraint matrix, all constraints involve all of the variables, so all of the constraints are similar in structure. A consequence is that many of the constraints have similar impacts on  $Z$ . For example, in the classification application, where the constraints are derived from the data points, data points that are close together generate similar constraints. These similar constraints have similar ranking scores in the candidate list, and hence have similar effects on  $Z$  when removed from the LP, leading to this **Observation**: *When candidates have similar ranking measures as the top-ranked element in the candidate list, they are likely to appear in the final MIN ULR set.*

This observation allows both the elimination of the inner loop to test individual candidate constraints and the simultaneous removal of multiple candidates in the single remaining loop. The question is how to determine which ranking scores are “similar” to the top score. We sort the candidates by descending ranking score and apply standard methods for detecting abrupt changes in the mean of subsets of the set [25]. We select for removal all candidates up to the first identified abrupt change.

### 3.2. Extension 2(E2)

Extension 2 provides an early exit. If the number of candidates is less than or equal to  $\ell$ , then permanently remove all candidates and exit the algorithm. This is designated E2( $\ell$ ).

## 4. Application: Binary Classification

Amaldi [2], Parker [17], Chinneck [26, 27, 21], and Silva [28] have shown how the binary classification problem can be transformed into a MAX FS problem:

**Given** a training set of  $I$  data points ( $i = 1 \dots I$ ) in  $J$  dimensions ( $j = 1 \dots J$ ) where the class of each point is known (Type 0 or 1).

**Define** a constraint for each data point:

- For points of Type 0:  $\sum_j d_{ij}w_j \leq w_0 - \epsilon$

- For points of type 1:  $\sum_j d_{ij}w_j \geq w_0 + \epsilon$

where  $d_{ij}$  indicates the value of attribute  $j$  for point  $i$ . The  $d_{ij}$  are known constants,  $\epsilon$  is a small positive constant and the variables  $w_j$  are unrestricted.

Most datasets cannot be completely separated by a single hyperplane, so the initial system is infeasible. After the MAX FS solution, the final LP is feasible, and its solution returns the separating hyperplane  $\sum_j d_{ij}w_j = w_0$  which correctly classifies all of the data points in the MAX FS subset while those in the MIN ULR subset are incorrectly classified. This is heuristically the most accurate hyperplane.

#### 4.1. Binary Classification Algorithm

Prior work shows that Algorithm 2 provides a major speed-up over Algorithm 1 for binary classification [21], with a tiny loss in average accuracy. Violated constraints correspond to misclassified points, so the product ranking score (Eqn. 1) used in Algorithm 2 estimates the drop in  $Z$  relatively accurately. A standard elastic program is used because the variables are unrestricted. Since all attribute values are specified for all data points, the constraint matrix,  $\mathbf{D}$ , is dense; hence, Extension 1 can be applied. The revised algorithm is designated as Algorithm 2E1, and is summarized in Fig. 2.

To illustrate the workings of Extension 1, Fig. 3 shows the first sorted candidate list for the “Ozone Level Detection” dataset. The first sharp drop-off occurs after the fifth candidate, so the first five constraints/points are removed in this case.

#### 4.2. Comparators

We compare Algorithm 2E1 with two existing algorithms: (i) original Algorithm 2( $\infty$ ) and (ii) original Algorithm 2(1). Both have been shown to provide high accuracies in binary classification [21].

#### 4.3. Evaluation Metric

Accuracy is used to evaluate the quality of the solutions:

$$Accuracy = \frac{\text{Total \# correctly classified instances}}{\text{Total \# instances}} \quad (4)$$

For comparison with previous results [21], the training set is identical to the entire dataset.

---

**Figure 2** Algorithm 2E1

---

$MINULR \leftarrow \emptyset$   
Set up the standard elastic LP.  
**do:**  
  Solve LP.  
  **if**  $Z = 0$  **then**  
    Exit.  
  **end if**  
  Construct *CandidateSet* using Eq. 1.  
  **if**  $|CandidateSet| = 1$  **then**  
    Add the single candidate to *MINULR* and exit.  
  **end if**  
  Sort candidates from largest to smallest product.  
  Select top  $p$  candidates for removal based on abrupt change in product score.  
  Add the  $p$  selected candidates to *MINULR* and remove them from the LP.  
**end do**  
**OUTPUT:** *MINULR* is a heuristic MIN ULR solution.

---

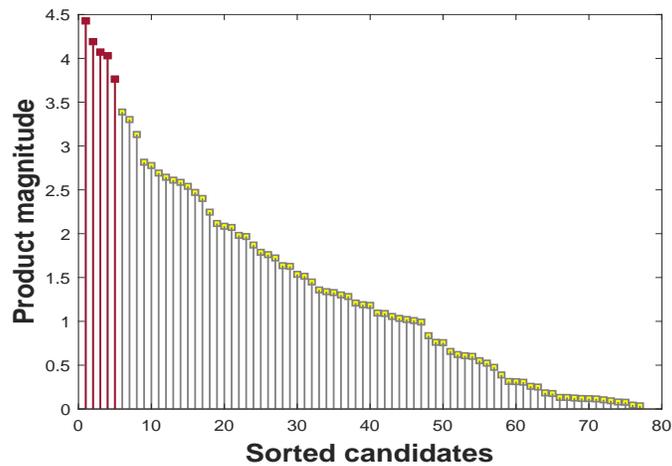


Figure 3: First Sorted Candidate List for “Ozone Level Detection” Dataset.

#### 4.4. Data Sets

Twenty binary classification problems are derived from datasets in the UCI Repository of Machine Learning Databases [29]. Table 2 summarizes their characteristics.

Table 2: Classification Datasets

<b>Dataset</b>	<b>Instances</b>	<b>Features</b>
Breast Cancer Wisconsin Original	683	9
Cardiotocography (Type 1 vs. others)	2126	34
Cardiotocography (Type 2 vs. others)	2126	34
Car Evaluation (Type 1 vs. others)	1728	6
Car Evaluation (Type 2 vs. others)	1728	6
Car Evaluation (Type 3 vs. others)	1728	6
Car Evaluation (Type 4 vs. others)	1728	6
Climate Model Simulation Crashes	540	18
Credit Approval	653	15
Diabetic Retinopathy Debrecen	1151	19
Glass Identification	214	9
Heart Disease	297	13
Hepatitis Domain	112	18
Ionosphere	351	33
BUPA Liver Disorders	341	6
Ozone Level Detection	1848	102
Parkinson's Data Set	195	22
Pima Indians Diabetes	768	8
Thyroid Gland Data	215	5
Vowel Recognition Data	990	11

#### 4.5. Software and Hardware

All algorithms are implemented in Matlab version 2020. The linear programming solver is MOSEK via the MOSEK Optimization Toolbox for Matlab version 8.1.0.56 [30]. Abrupt changes in the ranking scores in the candidate list are identified using the Matlab *ischange()* function. The computations are carried out on a 3.40 GHz Intel core *i7* machine with 16.0 GB RAM, running Windows 10.

#### 4.6. Results and Discussion

The results in this section show the higher accuracy and greater speed of Algorithm 2E1 vs. original Algorithms 2( $\infty$ ) and 2(1) for binary classification. Table 3 reports algorithm accuracies with the best results bolded.

Table 4 shows the difference from the best accuracy on each data set. The last three rows give additional metrics: the largest difference from the best accuracy, the average difference from the best accuracy, and number of best accuracies obtained by each algorithm.

Algorithm 2E1 provides better total accuracy. It has the smallest average difference from best accuracy, and provides the best accuracy for 12 of the 20 data sets while it was 8 for the two existing algorithms. The largest difference from the best accuracy for Algorithm 2E1 is  $-1.38$ , which is smaller than those for the other two ( $-4.28$  for both).

Table 3: Algorithm Accuracies for Classification Datasets.

Dataset	Accuracy		
	Algorithm 2( $\infty$ )	Algorithm 2(1)	Algorithm 2E1
Breast Cancer Wisconsin Original	98.10	98.10	<b>98.39</b>
Cardiotocography (Type 1 vs.others)	99.72	99.72	<b>99.86</b>
Cardiotocography (Type 2 vs.others)	99.67	99.58	<b>99.81</b>
Car Evaluation (Type 1 vs.others)	78.47	78.47	<b>82.75</b>
Car Evaluation (Type 2 vs.others)	<b>96.01</b>	<b>96.01</b>	<b>96.01</b>
Car Evaluation (Type 3 vs.others)	87.79	88.42	<b>89.01</b>
Car Evaluation (Type 4 vs.others)	<b>98.15</b>	98.09	98.09
Climate Model Simulation Crashes	<b>98.89</b>	98.70	98.52
Credit Approval	<b>88.05</b>	86.68	86.68
Diabetic Retinopathy Debrecen	79.06	<b>79.41</b>	78.62
Glass Identification	84.11	<b>84.56</b>	83.18
Heart Disease Cleveland	<b>89.56</b>	<b>89.56</b>	<b>89.56</b>
Hepatitis Domain	92.86	<b>96.43</b>	<b>96.43</b>
Ionosphere	98.01	<b>98.01</b>	97.73
BUPA Liver Disorders	75.37	<b>75.66</b>	74.78
Ozon Level Detection	98.86	99.13	<b>99.19</b>
Parkinsons Data Set	94.36	94.87	<b>96.41</b>
Pima Indians Diabetes	<b>80.47</b>	80.08	79.69
Thyroid Gland Data	<b>94.42</b>	<b>94.42</b>	<b>94.42</b>
Vowel Recognition Data	98.38	98.38	<b>98.48</b>

Table 5 compares the number of linear programs solved ( $LPs$ ) and the processing time in seconds ( $sec$ ) for each algorithm, with best results in bold face. Algorithm 2E1 requires the fewest LP solutions and the least processing time in all cases. Table 6 reports the percentage reduction in number of LPs and processing time for Algorithm 2E1 vs. the comparators for each dataset. For example, the processing times for Algorithms 2( $\infty$ ) and 2E1 are 3.1 and 0.3 seconds respectively for the first dataset in Table 5. Therefore, Arithm 2E1 provides a 90.32% reduction in processing time when compared to Algorithm 2( $\infty$ ) for the first dataset in Table 6. The last two rows of Table 6 show the maximum and average speed improvements obtained by

Table 4: Differences from Best Accuracy.

Dataset	Algorithm 2( $\infty$ )	Algorithm 2(1)	Algorithm 2E1
Breast Cancer Wisconsin Original	-0.29	-0.29	<b>0</b>
Cardiotocography (Type 1 vs.others)	-0.14	-0.14	<b>0</b>
Cardiotocography (Type 2 vs.others)	-0.14	-0.23	<b>0</b>
Car Evaluation (Type 1 vs.others)	-4.28	-4.28	<b>0</b>
Car Evaluation (Type 2 vs.others)	<b>0</b>	<b>0</b>	<b>0</b>
Car Evaluation (Type 3 vs.others)	-1.22	-0.59	<b>0</b>
Car Evaluation (Type 4 vs.others)	<b>0</b>	-0.06	-0.06
Climate Model Simulation Crashes	<b>0</b>	-0.19	-0.18
Credit Approval	<b>0</b>	-1.37	-1.37
Diabetic Retinopathy Debrecen	-0.35	<b>0</b>	-0.79
Glass Identification	-0.45	<b>0</b>	-1.38
Heart Disease Cleveland	<b>0</b>	<b>0</b>	<b>0</b>
Hepatitis Domain	-3.57	<b>0</b>	<b>0</b>
Ionosphere	<b>0</b>	<b>0</b>	-0.28
BUPA Liver Disorders	-0.29	<b>0</b>	-0.88
Ozon Level Detection	-0.33	-0.06	<b>0</b>
Parkinsons Data Set	-2.05	-1.54	<b>0</b>
Pima Indians Diabetes	<b>0</b>	-0.39	-0.39
Thyrod gland data	<b>0</b>	<b>0</b>	<b>0</b>
Vowel Recognition Data	-0.1	-0.1	<b>0</b>
<b>Largest Difference</b>	-4.28	-4.28	<b>-1.38</b>
<b>Average Difference</b>	-0.66	-0.46	<b>-0.27</b>
<b>No. of bests</b>	8	8	<b>12</b>

Algorithm 2E1 when compared with the other two algorithms. Algorithm 2E1 reduces the average number of LPs by about 96.56% and 67.90% when compared to Algorithms 2( $\infty$ ) and 2(1), respectively, and also reduces the solution time by 94.77% and 71.31% when compared to Algorithm 2( $\infty$ ) and Algorithm 2(1) respectively.

Algorithm 2E1 thus reduces solution time by an average 70-90% when compared to the existing algorithms, while also marginally improves the total accuracy on average.

## 5. Application: Sparse Recovery in Compressive Sensing

Finding a sparse solution to an underdetermined system of linear equations in unbounded variables is an essential step in compressive sensing (CS) [31]. A common way to compress an input signal vector  $\mathbf{x}$  in CS is to multiply it by a random matrix  $\mathbf{A}$  of size  $m \times n$  where  $m \ll n$  to yield a compressed vector  $\mathbf{b}$  of size  $m$ , which is transmitted or stored. The input vector is assumed to be sparse, so the decompression or recovery process tries to find a sparse solution given only  $\mathbf{A}$  and the compressed vector  $\mathbf{b}$ . The CS *sparse*

Table 5: Algorithm speeds (number of LPs, seconds) for binary classification.

Dataset	Algorithm 2( $\infty$ )		Algorithm 2(1)		Algorithm 2E1	
	LPs	Sec	LPs	Sec	LPs	Sec
Breast Cancer Wisconsin Original	100	3.1	12	0.7	<b>6</b>	<b>0.3</b>
Cardiotocography (Type 1 vs.others)	35	4.3	4	0.6	<b>2</b>	<b>0.3</b>
Cardiotocography (Type 2 vs.others)	67	5.1	4	0.6	<b>2</b>	<b>0.3</b>
Car Evaluation (Type 1 vs.others)	2583	350.6	372	28.1	<b>8</b>	<b>0.7</b>
Car Evaluation (Type 2 vs.others)	462	28.7	69	4.5	<b>1</b>	<b>0.1</b>
Car Evaluation (Type 3 vs.others)	1458	183.2	203	13.8	<b>10</b>	<b>0.9</b>
Car Evaluation (Type 4 vs.others)	205	16.7	33	2.3	<b>5</b>	<b>0.4</b>
Climate Model Simulation Crashes	77	3.2	7	0.3	<b>5</b>	<b>0.2</b>
Credit Approval	1237	37.1	87	7.3	<b>1</b>	<b>0.1</b>
Diabetic Retinopathy Debrecen	4723	625.2	238	25.6	<b>16</b>	<b>1.4</b>
Glass Identification	292	3.1	33	1.5	<b>7</b>	<b>0.3</b>
Heart Disease Cleveland	404	5.3	31	1.0	<b>6</b>	<b>0.2</b>
Hepatitis Domain	16	0.4	4	0.2	<b>2</b>	<b>0.1</b>
Ionosphere	104	4.2	7	<b>0.3</b>	<b>5</b>	<b>0.3</b>
BUPA Liver Disorders	581	14.8	83	2.7	<b>10</b>	<b>0.3</b>
Ozon Level Detection	542	159.7	14	4.1	<b>7</b>	<b>2.1</b>
Parkinsons Data Set	103	2.7	7	<b>0.4</b>	<b>4</b>	<b>0.4</b>
Pima Indians Diabetes	1325	41.7	153	6.7	<b>12</b>	<b>0.5</b>
Thyrod gland data	61	1.3	12	0.4	<b>6</b>	<b>0.2</b>
Vowel Recognition Data	146	4.0	14	0.5	<b>7</b>	<b>0.3</b>

*recovery problem* is to find a sparse solution  $\mathbf{y}$  to the underdetermined system  $\mathbf{A}\mathbf{y} = \mathbf{b}$ , where  $\mathbf{y}$  is of size  $n \times 1$  and all of its components are unbounded.

Numerous algorithms have been developed to solve the sparse recovery problem [32, 33, 34]. Recently developed MAX FS based algorithms (Methods B, C, and M) provide better quality solutions than the state-of-the-art algorithms in certain CS scenarios, though they can be comparatively slow [11]. Several choices from various families of random matrices can result in a dense  $\mathbf{A}$ , and thus the recovery process becomes more suitable for the extended algorithms introduced here. This section examines whether the new algorithm extensions can increase the speed of the MAX FS based algorithms for the sparse recovery problem without loss of solution quality.

### 5.1. MAX FS for Sparse Recovery

Methods B, C, and M (Section 2.4) have been applied to CS sparse recovery and are observed to provide improvements over the state of the art in certain scenarios. Previous work [11] showed Method M to be one of the better methods for CS sparse recovery. So we combine it with our two extensions to create Method ME1E2( $\ell$ ). The steps are summarized in Fig. 4.

Method ME1E2( $\ell$ ) assumes BP failure if the length of the first candidate list is greater than the defined threshold  $\ell$ . We set  $\ell$  to  $m - 3$ , as done for

Table 6: Percentage reductions in LPs solved and solution times by using Algorithm 2E1 instead of original Algorithms 2( $\infty$ ) and 2(1) for classification.

Dataset	Algorithm 2E1 vs. Algorithm 2( $\infty$ )		Algorithm 2E1 vs. Algorithm 2(1)	
	LPs	Sec	LPs	Sec
Breast Cancer Wisconsin Original	94	90.32	50	57.14
Cardiotocography (Type 1 vs.others)	94.29	93.02	50	50
Cardiotocography (Type 2 vs.others)	97.01	94.12	50	50
Car Evaluation (Type 1 vs.others)	99.69	99.80	97.85	97.51
Car Evaluation (Type 2 vs.others)	99.78	99.65	98.55	97.78
Car Evaluation (Type 3 vs.others)	99.31	99.51	95.07	93.48
Car Evaluation (Type 4 vs.others)	97.56	97.60	84.85	82.61
Climate Model Simulation Crashes	93.51	93.75	28.57	33.33
Credit Approval	99.92	99.73	98.85	98.63
Diabetic Retinopathy Debrecen	99.66	99.78	93.28	94.53
Glass Identification	97.60	90.32	78.79	80
Heart Disease Cleveland	98.51	96.23	80.64	80
Hepatitis Domain	87.5	75	50	50
Ionosphere	95.19	97.62	28.57	66.67
BUPA Liver Disorders	98.28	97.97	87.95	88.89
Ozon Level Detection	98.71	98.68	50	48.78
Parkinsons Data Set	96.12	96.30	42.86	74.36
Pima Indians Diabetes	99.09	98.80	92.16	92.54
Thyrod gland data	90.16	84.61	50	50
Vowel Recognition Data	95.20	92.5	50	40
<b>Maximum Percent Reduction</b>	99.92	99.80	98.85	98.63
<b>Average Percent Reduction</b>	96.56	94.77	67.90	71.31

Method M in [11].

### 5.2. Comparators

Method ME1E2( $\ell$ ) is compared with the original Methods B(2), C(2), and M(2). We choose list length  $k = 2$  to provide fast solutions with good solution quality ( $k$  is typically 1 – 7 [21]). Postprocessing is not used in any of these three methods.

### 5.3. Evaluation Metric

In sparse recovery, if the number of nonzeros in recovered signal  $y$  equals the number of nonzeros in the input signal  $x$ , then  $y$  and  $x$  are usually identical. For this reason our main metric relates to the *sparsity* (number of nonzeros) of  $y$  vs.  $x$ . The *critical sparsity* is the largest number of nonzeros in the input vector  $x$  for which the sparsity of  $y$  reliably equals the sparsity of  $x$ . Higher critical sparsity means that input vectors that have been more compressed, or that they have more nonzeros and therefore can be reliably

---

**Figure 4** Algorithm ME1E2( $\ell$ )

---

$MINULR \leftarrow \emptyset$   
Set up the sparse recovery LP.  
**do:**  
  Solve LP.  
  Construct *CandidateSet*: variables  $x_j$  not in *MINULR* that have nonzero  $|u_j - v_j|$ .  
  **if**  $|CandidateSet| = \phi$  **then** exit.  
  **if** first iteration and  $|CandidateSet| \leq \ell$  **then**  
    Add all candidates to *MINULR* and exit.  
  **end if**  
  Sort candidates from largest to smallest  $|u_j - v_j|$ .  
  Select top  $p$  candidates for removal based on first abrupt change in  $|u_j - v_j|$ .  
  Add the  $p$  selected candidates to *MINULR* and set the objective coefficients of the associated  $u_j, v_j$  pairs to 0.1.  
**end do**  
**OUTPUT:** *MINULR* is small number of nonzeros forming a support for the system of equations.

---

recovered. We use the critical sparsity as the evaluation metric: higher is better.

#### 5.4. Test Models

We construct examples for which the solution is known. The system is  $\mathbf{A}_{128 \times 256} \mathbf{x}_{256 \times 1} = \mathbf{b}_{128 \times 1}$ , providing a compression ratio ( $CR$ ) =  $(1 - \frac{m}{n}) \times 100 = 50\%$ .  $\mathbf{A}$  is a random matrix with values ranging from -10 to 10, and hence dense. The constructed input vector  $\mathbf{x}$  has various numbers of nonzeros ranging from 12 to 96, randomly distributed among its elements, and with values randomly drawn from a normal distribution with mean 0 and variance 1.

#### 5.5. Hardware and Software

See Section 4.5.

#### 5.6. Results and Discussion

Table 7 compares Algorithm ME1E2( $\ell$ ) with existing methods, namely, Methods B(2), C(2), and M(2). Results in the table are averages over 100 instances at each sparsity level  $S$  (actual number of nonzeros in the solution). *Estimated sparsity (T)* is the average number of nonzeros returned by each method at each value of  $S$  over 100 instances, with the number of correct

results shown in brackets. Results that are accurate in all 100 cases are shown in boldface. The critical sparsity is the largest number shown in boldface for each method. Extended Algorithm ME1E2( $\ell$ ) has the same critical sparsity as the other methods.

Table 8 shows the number of LPs solved and the processing time for Method ME1E2( $\ell$ ) and the comparators. The smallest number of LPs and time in seconds are bolded. Method ME1E2( $\ell$ ) requires the fewest LP solutions at all values of  $S$ . It is significantly faster than the other algorithms, as shown in Table 9 which reports the percentage reduction in LP solutions required and processing times for Method ME1E2( $\ell$ ) when compared to others. The last two rows of the table show the maximum and the average percent improvement. At  $S = 52$ , Method ME1E2( $\ell$ ) reduces the solution time by 96.02%, 92.49%, and 92.83% compared to Methods C(2), B(2), and M(2), respectively. Method ME1E2( $\ell$ ) does not run faster than Method M(2) when the input is very sparse (up to  $S = 36$ ), but for  $S > 36$  it reduces solution time when compared to M(2) by 62.32% on average. Method ME1E2( $\ell$ ) provides a 98.82%, 97.55%, and 66.61% reduction in LPs solved when compared to Methods C(2), B(2), and M(2), respectively. It reduces the average solution time by 95.95%, 92.86%, and 62.32% when compared to Methods C(2), B(2), and M(2), respectively.

Method ME1E2( $\ell$ ) reduces solution time significantly with no loss of solution quality (as measured by critical sparsity) for this CS sparse recovery scenario.

Table 7: Average and critical sparsities of Method ME1E2( $\ell$ ) vs. Methods C(2), B(2) and M(2) for random matrices of size  $m = 128$  and  $n = 256$  in CS sparse recovery.

S	Estimated sparsity (T)			
	Method C(2)	Method B(2)	Method M(2)	Method ME1E2( $\ell$ )
12	12 <sup>(100)</sup>	12 <sup>(100)</sup>	12 <sup>(100)</sup>	12 <sup>(100)</sup>
16	16 <sup>(100)</sup>	16 <sup>(100)</sup>	16 <sup>(100)</sup>	16 <sup>(100)</sup>
20	20 <sup>(100)</sup>	20 <sup>(100)</sup>	20 <sup>(100)</sup>	20 <sup>(100)</sup>
24	24 <sup>(100)</sup>	24 <sup>(100)</sup>	24 <sup>(100)</sup>	24 <sup>(100)</sup>
28	28 <sup>(100)</sup>	28 <sup>(100)</sup>	28 <sup>(100)</sup>	28 <sup>(100)</sup>
32	32 <sup>(100)</sup>	32 <sup>(100)</sup>	32 <sup>(100)</sup>	32 <sup>(100)</sup>
36	36 <sup>(100)</sup>	36 <sup>(100)</sup>	36 <sup>(100)</sup>	36 <sup>(100)</sup>
40	40 <sup>(100)</sup>	40 <sup>(100)</sup>	40 <sup>(100)</sup>	40 <sup>(100)</sup>
44	44 <sup>(100)</sup>	44 <sup>(100)</sup>	44 <sup>(100)</sup>	44 <sup>(100)</sup>
48	48 <sup>(100)</sup>	48 <sup>(100)</sup>	48 <sup>(100)</sup>	48 <sup>(100)</sup>
52	52 <sup>(100)</sup>	52 <sup>(100)</sup>	52 <sup>(100)</sup>	52 <sup>(100)</sup>
56	56 <sup>(99)</sup>	56.03 <sup>(98)</sup>	56.04 <sup>(98)</sup>	56.10 <sup>(98)</sup>
60	64.91 <sup>(90)</sup>	64.24 <sup>(86)</sup>	65.59 <sup>(86)</sup>	66.43 <sup>(86)</sup>
64	75.49 <sup>(77)</sup>	76.3 <sup>(67)</sup>	77.32 <sup>(67)</sup>	77.0 <sup>(73)</sup>
68	92.21 <sup>(50)</sup>	94.53 <sup>(45)</sup>	94.48 <sup>(45)</sup>	92.95 <sup>(39)</sup>
72	108.65 <sup>(25)</sup>	106.87 <sup>(25)</sup>	107.63 <sup>(25)</sup>	106.77 <sup>(19)</sup>
76	119.55 <sup>(9)</sup>	121.85 <sup>(5)</sup>	120.92 <sup>(5)</sup>	120.9 <sup>(6)</sup>
80	125.41 <sup>(3)</sup>	126.52 <sup>(4)</sup>	125.56 <sup>(4)</sup>	125.27 <sup>(3)</sup>
84	127.58 <sup>(1)</sup>	127.55 <sup>(0)</sup>	127.6 <sup>(0)</sup>	127.14 <sup>(0)</sup>
88	128 <sup>(0)</sup>	127.98 <sup>(0)</sup>	128 <sup>(0)</sup>	128 <sup>(0)</sup>
92	128 <sup>(0)</sup>	128 <sup>(0)</sup>	128 <sup>(0)</sup>	128 <sup>(0)</sup>
96	128 <sup>(0)</sup>	127.96 <sup>(0)</sup>	128 <sup>(0)</sup>	128 <sup>(0)</sup>

Table 8: Number of LPs and processing times for Method ME1E2( $\ell$ ) vs. Methods C(2), B(2) and M(2) for random matrices of size  $m = 128$  and  $n = 256$  in CS sparse recovery.

S	Method C(2)		Method B(2)		Method M(2)		Method ME1E2( $\ell$ )	
	LPs	Sec	LPs	Sec	LPs	Sec	LPs	Sec
12	48	1.44	23	0.84	1	0.06	1	0.06
16	64	2.02	31	1.16	1	0.05	1	0.05
20	80.06	3.07	39	1.72	1	0.06	1	0.06
24	96.06	4.25	47	2.3	1	0.06	1	0.06
28	112.24	5.34	55	2.98	1	0.06	1	0.06
32	128.82	5.49	63	3.02	1	0.07	1	0.07
36	146.14	6.53	71	3.66	1	0.07	1	0.07
40	165.84	7.93	79	4.27	81	5.01	1.03	0.08
44	186.56	7.97	87	4.27	99.01	5.23	1.2	0.12
48	209.46	8.75	95	4.6	98	5.42	1.49	0.23
52	227.1	10.06	103	5.33	120.22	5.58	2.35	0.40
56	246.87	11.33	111.12	5.99	132.01	6.02	3.01	0.53
60	286.09	13.41	127.68	7.02	133.45	7.26	3.82	0.73
64	328.65	15.72	152.26	8.56	165.78	9.01	4.63	0.90
68	395.37	18.98	189.39	10.72	192.08	11.54	5.46	1.12
72	456.75	21.87	214.48	12.15	253.66	12.01	5.95	1.31
76	493	23.7	245.06	13.91	256.08	14.05	6.36	1.42
80	505.55	24.02	254.84	14.37	259.06	14.53	6.18	1.41
84	509.31	24.25	257.09	14.5	259.24	14.54	6.85	1.58
88	511.12	24.28	258.2	14.55	260.02	14.68	6.75	1.57
92	511.15	24.9	258.18	14.83	261.01	15.02	6.87	1.67
96	511.1616	24.52	257.9394	14.64	261	15.11	7.3	1.79

## 6. Conclusions

We propose two new extensions to Chinneck’s original MAX FS solution algorithms [3] [21] for use with dense constraint matrices. These extensions increase solution speed greatly with no loss of solution quality when tested in two applications: classification, and sparse recovery in compressive sensing:

- *Classification*: The new Algorithm 2E1 reduces the mean solution time by about 94.77% and 71.31% when compared to Algorithms 2( $\infty$ ) and 2(1), respectively, while slightly improving accuracy on average. The better classification accuracies reflect larger feasible subsets found by the algorithm over its predecessors.
- *Sparse recovery in compressive sensing*: The new Method ME1E2( $\ell$ ) reduces processing time by 95.95%, 92.86% and 62.32% when compared with Methods C, B and M, on average, with no reduction in the critical sparsity.

We expect that these results will generalize to other applications that have dense constraint matrices. We plan to investigate the use of the extended

Table 9: Percent reduction in number of LPs and processing times for Method ME1E2( $\ell$ ) vs. Methods C(2), B(2) and M(2) for random matrices of size  $m = 128$  and  $n = 256$  in CS sparse recovery.

S	Method ME1E2( $\ell$ ) vs. Method C(2)		Method ME1E2( $\ell$ ) vs. Method B(2)		Method ME1E2( $\ell$ ) vs. Method M(2)	
	LPs	Sec	LPs	Sec	LPs	Sec
12	97.92	95.83333	95.65	92.86	0	0
16	98.44	97.53	96.77	95.69	0	0
20	98.75	98.05	97.44	96.51	0	0
24	98.96	98.59	97.87	97.39	0	0
28	99.11	98.88	98.18	97.99	0	0
32	99.22	98.73	98.41	97.68	0	0
36	99.32	98.93	98.59	98.09	0	0
40	99.38	98.99	98.70	98.13	98.73	98.403
44	99.36	98.49	98.62	97.19	98.79	97.70
48	99.29	97.37	98.43	95	98.48	95.76
52	98.96	96.02	97.72	92.49	98.04	92.83
56	98.78	95.32	97.29	91.15	97.72	91.20
60	98.66	94.56	97.01	89.60	97.14	89.94
64	98.59	94.28	96.96	89.49	97.21	90.01
68	98.62	94.10	97.12	89.55	97.16	90.29
72	98.69	94.01	97.23	89.22	97.65	89.09
76	98.71	94.01	97.40	89.79	97.52	89.89
80	98.78	94.13	97.57	90.19	97.61	90.29
84	98.65	93.48	97.34	89.10	97.36	89.13
88	98.68	93.53	97.39	89.21	97.40	89.30
92	98.66	93.41	97.34	88.94	97.37	89.08
96	98.577	92.670	97.17	87.77	97.20	88.15
Maximum	99.38	98.99	98.70	98.13	98.79	98.40
Average	98.82	95.95	97.55	92.86	66.61	62.32

algorithms in the recovery of compressively sensed biomedical signals and dictionary learning. We also intend to extend the basic ideas to less dense constraint matrices, including those found in general LPs.

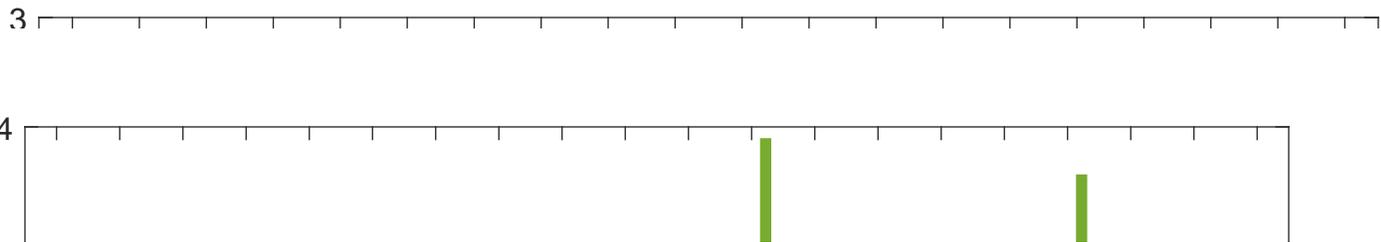
## References

- [1] E. Amaldi, M. E. Pfetsch, L. E. Trotter, Some structural and algorithmic properties of the maximum feasible subsystem problem, In International Conference on Integer Programming and Combinatorial Optimization (1999) 45–59.
- [2] E. Amaldi, From finding maximum feasible subsystems of linear systems to feedforward neural network design, PhD dissertation, Swiss Federal Institute of Technology at Lausanne (EPFL), 1994.

- [3] J. W. Chinneck, An effective polynomial-time heuristic for the minimum-cardinality iis set-covering problem, *Annals of Mathematics and Artificial Intelligence* 17 (1996) 127–144.
- [4] N. Chakravarti, Some results concerning post-infeasibility analysis, *Oper. Res.* 73 (1994) 139–143.
- [5] E. Amaldi, V. Kann, The complexity and approximability of finding maximum feasible subsystems of linear relations, *Oper. Res.* 147 (1994) 181–210.
- [6] J. K. Sankaran, A note on resolving infeasibility in linear programs by constraint relaxation, *Oper. Res.* 13 (1993) 19–20.
- [7] K. P. Bennett, E. J. Bredensteiner, A parametric optimization method for machine learning, *INFORMS Journal on Computing* 9 (1997) 311–318.
- [8] O. L. Mangasarian, Misclassification minimization, *Journal of Global Optimization* 5 (1994) 309–323.
- [9] F. Rossi, S. Smriglio, A. Sassano, Models and algorithms for terrestrial digital, *Annals of Operations Research* 107 (2001) 267–283.
- [10] M. Wagner, R. Elber, Large-scale linear programming techniques for the design, *Annals of Operations Research* 318 (2004) 301–318.
- [11] F. F. Firouzeh, J. W. Chinneck, S. Rajan, Maximum feasible subsystem algorithms for recovery of compressively sensed speech, *IEEE Access* 8 (2020) 82539–82550.
- [12] J. W. Chinneck, *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118, Springer Science & Business Media, 2007.
- [13] E. Amaldi, The maximum feasible subsystem problem and some applications, *Modelli e Algoritmi per l’Ottimizzazione di Sistemi Complessi*, A. Agnetis and G. D. Pillo, eds., Pitagora Editrice, Bologna, Italy (2003) 31–69.
- [14] O. L. Mangasarian, Misclassification minimization, *Journal of Global Optimization* 5 (1994) 309–323.

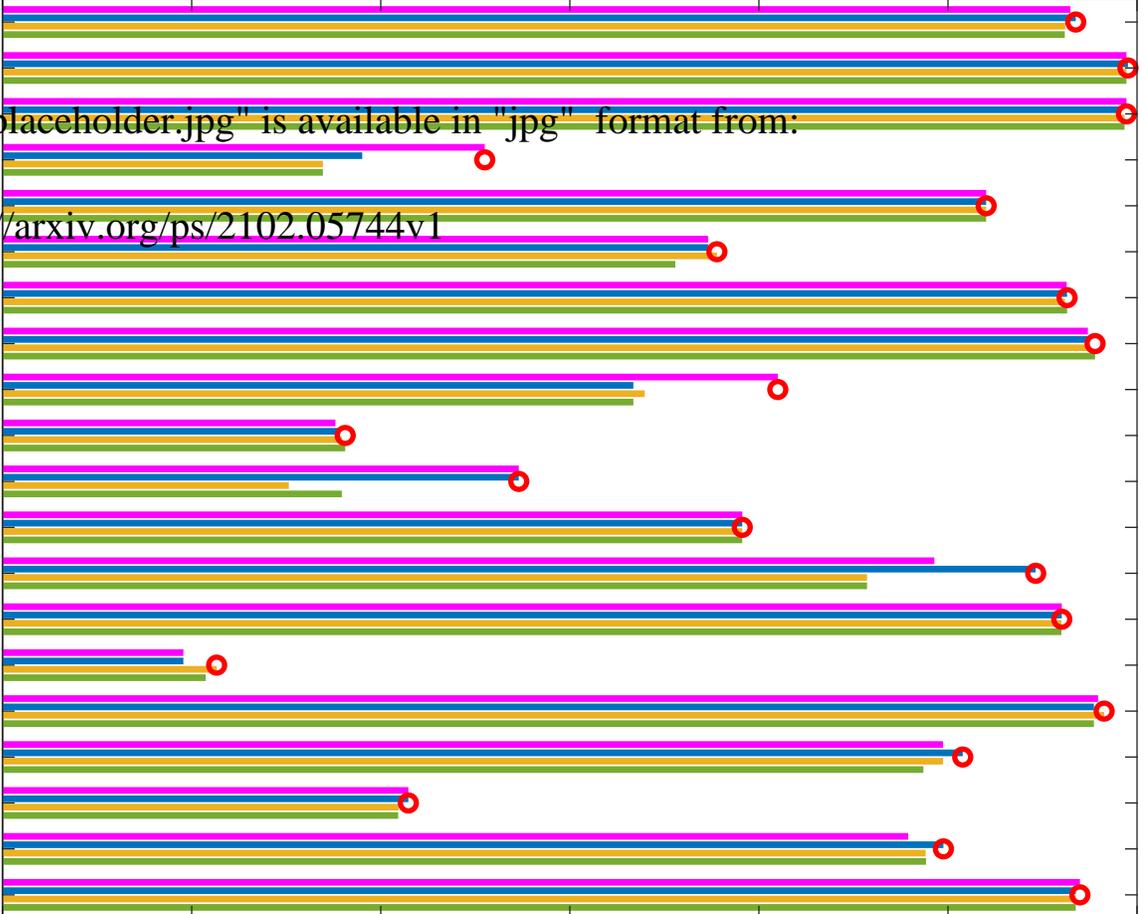
- [15] O. Mangasarian, Machine learning via polyhedral concave minimization, in: *Applied Mathematics and Parallel Computing*, Springer, 1996, pp. 175–188.
- [16] K. P. Bennett, E. J. Bredensteiner, A parametric optimization method for machine learning, *INFORMS Journal on Computing* 9 (1997) 311–318.
- [17] M. R. Parker, A set covering approach to infeasibility analysis of linear programming problems and related issues, Ph.D. thesis, University of Colorado at Denver Denver, Colorado, 1995.
- [18] M. Parker, J. Ryan, Finding the minimum weight iis cover of an infeasible system of linear inequalities, *Annals of Mathematics and Artificial Intelligence* 17 (1996) 107–126.
- [19] M. E. Pfetsch, The maximum feasible subsystem problem and vertex-facet incidences of polyhedra, Ph.D. thesis, TU Berlin, Berlin, 2003.
- [20] M. E. Pfetsch, Branch-and-cut for the maximum feasible subsystem problem, *SIAM Journal on Optimization* 19 (2008) 21–38.
- [21] J. W. Chinneck, Fast heuristics for the maximum feasible subsystem problem, *INFORMS Journal on Computing* 13 (2001) 210–223.
- [22] S. Jokar, M. E. Pfetsch, Exact and approximate sparse solutions of underdetermined linear equations, *SIAM Journal on Scientific Computing* 31 (2008) 23–44.
- [23] S. S. Chen, D. L. Donoho, M. A. Saunders, Atomic decomposition by basis pursuit, *SIAM review* 43 (2001) 129–159.
- [24] J. W. Chinneck, Sparse solutions of linear systems via maximum feasible subsets, *Les Cahiers du GERAD G-2018-104* (2018).
- [25] R. Killick, P. Fearnhead, I. A. Eckley, Optimal detection of changepoints with a linear computational cost, *Journal of the American Statistical Association* 107 (2012) 1590–1598.
- [26] J. W. Chinneck, Tailoring classifier hyperplanes to general metrics, in: *Operations research and cyber-infrastructure*, Springer, 2009, pp. 365–387.

- [27] J. W. Chinneck, Integrated classifier hyperplane placement and feature selection, *Expert Systems with Applications* 39 (2012) 8193–8203.
- [28] A. P. D. Silva, Optimization approaches to supervised classification, *European Journal of Operational Research* 261 (2017) 772–788.
- [29] D. Dua, C. Graff, UCI machine learning repository, 2017. URL: <http://archive.ics.uci.edu/ml>.
- [30] Mosek ApS, The mosek optimization software, 2018. URL: <https://www.mosek.com>, last accessed 2021.
- [31] E. J. Candès, M. B. Wakin, An introduction to compressive sampling, *IEEE signal processing magazine* 25 (2008) 21–30.
- [32] S. Qaisar, R. M. Bilal, W. Iqbal, M. Naureen, S. Lee, Compressive sensing: From theory to applications, a survey, *Journal of Communications and networks* 15 (2013) 443–456.
- [33] S. B. Meenakshi, A survey of compressive sensing based greedy pursuit reconstruction algorithms, *International Journal of Image, Graphics and Signal Processing* 7 (2015) 1–10.
- [34] F. Salahdine, N. Kaabouch, H. El Ghazi, A survey on compressive sensing techniques for cognitive radio networks, *Physical Communication* 20 (2016) 61–73.



Dataset

- Breast Cancer Wisconsin Original
- Cardiotocography (type 1 vs.others)
- Cardiotocography (type 2 vs.others)
- Car Evaluation (type 1 vs.others)
- Car Evaluation (type 2 vs.others)
- Car Evaluation (type 3 vs.others)
- Car Evaluation (type 4 vs.others)
- Climate Model Simulation Crashes
- Credit Approval
- Diabetic Retinopathy Debrecen
- Glass Identification
- Heart Disease Cleveland
- Hepatitis Domain
- Ionosphere
- BUPA Liver Disorders
- Ozon Level Detection
- Parkinsons Data Set
- Pima Indians Diabetes
- Thyrod gland data
- Vowel Recognition Data



This figure placeholder.jpg" is available in "jpg" format from:  
<http://arxiv.org/ps/2102.05744v1>

Accuracy