Chapter 1

PREPROCESSING COMPLEMENTARITY PROBLEMS

Michael C. Ferris and Todd S. Munson
Computer Sciences Department
University of Wisconsin at Madison
Madison, WI 53706
ferris,tmunson@cs.wisc.edu

Abstract Preprocessing techniques are extensively used by the linear and integer programming communities as a means to improve model formulation by reducing size and complexity. Adaptations and extensions of these methods for use within the complementarity framework are detailed. The preprocessor developed is comprised of two phases. The first recasts a complementarity problem as a variational inequality over a polyhedral set and exploits the uncovered structure to fix variables and remove constraints. The second discovers information about the function and utilizes complementarity theory to eliminate variables. The methodology is successfully employed to preprocess several models.

Keywords: mixed complementarity, preprocessing

1. INTRODUCTION

General purpose codes for solving complementarity problems have previously lacked one significant feature: a powerful preprocessor. The benefits of preprocessing have long been known to the linear [1, 2] and integer [19] programming communities, yet have not been studied from a complementarity perspective. The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. In this paper, a comprehensive preprocessor is developed for the mixed complemen-

tarity problem and computational experience with an implementation is reported.

Complementarity problems arise in a variety of disciplines; a multitude of applications from engineering and economics are described in [10]. The AMPL and GAMS modeling languages aid practitioners in developing and solving these applications by providing a mixed complementarity problem (MCP) format, enabling their models to be communicated directly to available solvers [6, 9]. The problem generated by the modeling languages and accepted by most solvers is the (box-constrained) variational inequality:

$$0 \in F(x) + N_{[L,U]}(x), \tag{1.1}$$

where $F : \Re^n \to \Re^n$ is continuously differentiable and $[L, U]$ represents a Cartesian product of closed, not necessarily compact, intervals. In this definition, $N_X(\cdot)$ is the normal cone [18] of X defined by

$$N_X(x) := \begin{cases} \{y \mid \langle \bar{x} - x, y \rangle \leq 0, \ \forall \bar{x} \in X\} & \text{if } x \in X \\ \emptyset & \text{otherwise,} \end{cases}$$

under the assumption that X is a closed convex set. Note that (1.1) is just the standard variational inequality

$$x \in X \text{ and } \langle F(x), \bar{x} - x \rangle \geq 0 \ \forall \bar{x} \in X,$$

with X representing the set $[L, U]$.

The preprocessor for complementarity problems works upon two equivalent manifestations of the same model. To understand the basic methodology developed, consider a standard quadratic programming problem:

$$\begin{array}{ll} \min & \frac{1}{2} x^T Q x + c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0, \end{array} \tag{1.2}$$

where $Q \in \Re^{n \times n}$ is a symmetric matrix, $A \in \Re^{m \times n}$, $b \in \Re^m$, and $c \in \Re^n$. (1.2) can be posed as a variational inequality in one of two ways. First, when dual variables, λ, are introduced, the complementary slackness conditions for quadratic programs form the box constrained variational inequality:

$$0 \in \begin{bmatrix} Q & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} + \begin{bmatrix} c \\ -b \end{bmatrix} + \begin{bmatrix} N_{\Re^n_+}(x) \\ N_{\Re^m_+}(\lambda) \end{bmatrix}. \tag{1.3}$$

Alternatively, the first order conditions can be succinctly written as a polyhedrally constrained variational inequality:

$$0 \in Qx + c + N_C(x), \tag{1.4}$$

where $C = \{x \mid Ax \geq b, x \geq 0\}$. Since C is a geometric object, a computationally attractive algebraic representation can be chosen for C. Exploiting this fact is a key concept in the preprocessor developed.

As will become evident, the mixed complementarity problem can be written in forms analogous to (1.3) and (1.4). Each formulation is used by a distinct phase of the preprocessor. The majority of the preprocessor reductions documented involve exploiting the polyhedral set C in (1.4). As mentioned above, the MCP is communicated to a solver as a box-constrained variational inequality (more similar to (1.3)) that is not conducive to this analysis. From the problem description, the polyhedral structure in C will need to be recovered before it can be used. Once this is achieved, the general inequalities in the set C can be used to modify the bounds L_i and U_i on a variable x_i. Note in particular

$$N_{[L,U]}(x) = \prod_{i=1}^{n} N_{L_i,U_i}(x_i)$$

and that if $x_i = L_i = U_i$ then $N_{[L_i,U_i]}(x_i) = \Re$. Hence, fixing a variable x_i means that the corresponding constraint

$$0 \in F_i(x) + N_{[L_i,U_i]}(x_i) \equiv 0 \in F_i(x) + \Re$$

is trivially satisfiable. Thus, preprocessing in the complementarity case attempts to fix variables and thus remove constraints.

Section 2. begins by detailing the process used to uncover and exploit polyhedral structure in an MCP. The general idea is to reformulate (1.1) in a form similar to (1.4), with a general polyhedral set C replacing $[L, U]$. The representation of the set C can then be modified by either removing constraints or bounding variables. When converted back to a mixed complementarity problem a reduction in the number of variables is realized. Note that the process developed in this section recovers most checks done by traditional linear programming codes [1] when given the complementary slackness necessary and sufficient conditions for linear programs, but is applicable to a larger class of problem.

Further reductions to the MCP can be made by utilizing information about F and its Jacobian, ∇F, as developed in Section 3.. In particular, the range of F is used to eliminate variables from the model. Row and column duplicates are also removed. By detecting special block structure, a sequence of smaller problems can be solved to find an answer to the original problem.

Finally, both phases are incorporated into a complete preprocessor for mixed complementarity problems in Section 4.. Computational results for some test problems are presented indicating the success of the procedure outlined.

More information about the problem must be provided to the preprocessor than is necessary to solve it. The basic requirement is a listing of the linear and nonlinear elements in the Jacobian of F. This knowledge is sufficient to find and utilize special structures. The AMPL [11] and GAMS [3] environments already provide this information. Users of other interfaces, such as MATLAB and NEOS [8], will need to develop the appropriate routines. Some checks in Section 3. based on the nonlinear functions need to know the range of F over X. Routines to calculate these intervals are not currently provided by any of the interfaces.

2. POLYHEDRAL CONSTRAINTS

The first stage of the preprocessor detects polyhedral structure in a mixed complementarity problem. The structure is exploited by transforming the source problem into a model of lower dimension where C is the intersection of a closed product of intervals and a polyhedral set. The representation of C is then modified by removing constraints and changing bounds. The resultant MCP has fewer variables. After the preprocessed model has been solved, a solution to the original MCP is recovered with a postsolve step.

2.1 PRESOLVE

Polyhedral structure can be exploited when given a special type of complementarity problem. Suppose the variables can be split into (x, y) and (1.1) has the form:

$$0 \in \begin{bmatrix} F(x) - A^T y \\ Ax - b \end{bmatrix} + \begin{bmatrix} N_X(x) \\ N_Y(y) \end{bmatrix}, \tag{1.5}$$

where $F : \Re^n \to \Re^n$ is continuously differentiable, $A \in \Re^{m \times n}$, $b \in \Re^m$, $X \subseteq \Re^n$ is a Cartesian product of closed intervals, and $Y \subseteq \Re^m$ is a Cartesian product of $\Re$, $\Re_+$, or $\Re_-$. Note that if $Y_i = \{0\}$ then y_i can be fixed at zero and the corresponding $A_{i,}.x - b_i$ removed. Further, if $Y_i = [L_i, \infty)$ or $Y_i = (-\infty, U_i]$ for some finite L_i or U_i then an appropriate change of variables, possibly adding constant vectors to $F(x)$ and b, replaces Y_i with $\Re_+$ and R_- respectively.

A related problem to (1.5) is to find an $\bar{x}$ solving:

$$0 \in F(x) + N_{X \cap \{x | b - Ax \in Y^o\}}(x), \tag{1.6}$$

where Y^o denotes the polar cone of Y which is defined as

$$Y^o := \{y \mid \langle y, \bar{y} \rangle \leq 0, \ \forall \bar{y} \in Y\}.$$

Associated with such an $\bar{x}$ is a multiplier $\bar{y}$, constructed as a solution to the following linear optimization problem:

$$\begin{array}{ll} \min_{y \in Y} & \langle A\bar{x} - b, y \rangle \\ \text{s.t.} & 0 \in F(\bar{x}) - A^T y + N_X(\bar{x}). \end{array} \tag{1.7}$$

(1.5), (1.6), and (1.7) are formally related by the following theorem.

Theorem 1 *(Propositions 1 and 2 of [17]) Under the assumptions placed on X, Y, and the structure of the problem given above the following hold:*

1. *If $(\bar{x}, \bar{y})$ solves (1.5) then $\bar{x}$ is a solution to (1.6).*
2. *If $\bar{x}$ solves (1.6) then the optimization problem (1.7) has a nonempty solution set. Further, for any $\bar{y}$ solving (1.7), $(\bar{x}, \bar{y})$ solves (1.5).*

Theorem 1 provides the machinery used by the first stage of the preprocessor. The Jacobian matrix, ∇F is stored in both row- and column-oriented data structures. Utilizing the information provided about the types of the elements in the Jacobian, a row and column possessing the necessary skew symmetric structure of (1.5) can be quickly identified. Theorem 1 is then applied to this single row and column to create a problem of the form (1.6). The polyhedral set, $X \cap \{x \mid b - Ax \in Y^o\}$, is then checked for possible reductions, that is whether the general constraint, $b - Ax \in Y^o$, can be moved into the bound constraint, X. The new set $\tilde{X} \cap \{x \mid b - Ax \in \tilde{Y}^o\}$ is identical to $X \cap \{x \mid b - Ax \in Y^o\}$ but the MCP recovered using Theorem 1 on the reduced model is typically simpler. The identification and modification continues until no further simplifications are made. Note that Theorem 1 is only applied to a small number of constraints at a time during preprocessing while [17] uses the machinery to ready a problem for solution by a polyhedrally constrained variational inequality solver [20]. Finding a set of polyhedral constraints with maximum size from (1.1) is a harder problem and is not considered.

Information about any modifications performed are pushed onto a stack. The stack is a convenient data structure with two basic operations: pushing an element onto the top and popping an element from the top. Changes are pushed onto the stack in the order performed and are popped off the stack in the reverse order during the postsolve.

The complete algorithm for the first phase of the preprocessor is as follows:

A.1 Mark all rows and columns with the skew symmetric structure as eligible candidates, excluding any rows complementary to a variable with finite lower and upper bounds.

A.2 Using some ordering, pick one of the candidates and transform the problem into a polyhedral-constrained equation using Theorem 1.

A.3 Analyze the polyhedral set and modify the representation as detailed below. Push any changes on top of the stack.

A.4 Transform the modified problem back to box-constrained form.

A.5 Repeat steps A.2–A.4 until there are no reductions possible.

The implementation performs all simple reductions (Section 2.1.1) first. Once all of these are completed, forcing constraints (Section 2.1.2) and redundant rows (Section 2.1.3) are checked. In a nonlinear model, additional rows and columns can become linear when variables are fixed. Therefore, after all tests are completed on the current list of eligible candidates, another pass is made through the Jacobian to mark new eligible rows and columns which are checked using A.2–A.5. When no new eligible rows and columns are created the process stops.

2.1.1 Simple Reductions. The simplest reduction that can be made is when an eligible row contains zero elements. This corresponds to the case where the polyhedral set in the transformed problem is:

$$X \cap \{x \mid b \in Y^o\}. \tag{1.8}$$

If (1.8) is empty, then the original problem has no solution. Otherwise, the polyhedral component is irrelevant and (1.8) can be replaced with:

$$X \cap \{x \mid b \in \{0\}^o\}.$$

Note that $\{0\}^o = \Re$, rendering the constraint meaningless. When transformed back to the original space, the multiplier is fixed at 0 and removed from the problem, resulting in a reduction of one variable and the corresponding constraint.

Another simple reduction occurs when the eligible row contains a single element. The polyhedral set in this case is:

$$X \cap \{x \mid b - ax_i \in Y^o\}. \tag{1.9}$$

Since Y is $\Re_+$, $\Re_-$, or $\Re$, the constraint will be either $b - ax_i \leq 0$, $b - ax_i \geq 0$, or $b - ax_i = 0$ respectively. Each of these imply simple bounds on x_i, which can be explicitly placed in X. Therefore, (1.9) is replaced by the set:

$$\tilde{X} \cap \{x \mid b - ax_i \in \{0\}^o\},$$

where $\tilde{X}$ includes the tightened bounds on x_i. This modification results in a reduction of at least one variable.

The final case considered is when an eligible equality row, i, contains elements in exactly two columns, j and k, one of which is a column singleton. Assume variable k is the column singleton. If row k is also eligible, then X is modified by changing the bounds on x_j to make x_k free. Immediately following this change, the k constraint is preprocessed out of the model using the singleton check described above.

2.1.2 Forcing Constraints. Forcing constraints are constraints for which, given the bounds on the variables, there is exactly one feasible point. Once it is known that only one solution is possible, all variables appearing in the constraint can be fixed, potentially leading to a large reduction in problem size.

Let the polyhedral constraint be written in the form:

$$X \cap \{x \mid b - a^T x \in Y^o\}. \tag{1.10}$$

Without loss of generality, assume that $Y = \Re_+$ which means that $Y^o = \Re_-$. (1.10) can then be explicitly stated as:

$$X \cap \{x \mid b \leq a^T x\}.$$

Using X, bounds, $\underline{a}$ and $\bar{a}$, can be implied such that $\underline{a} \leq a^T x \leq \bar{a}$ for all $x \in X$. The ranges are determined as follows:

$$\begin{aligned} \underline{a} &= \sum_{\{i|a_i>0\}} a_i L_i + \sum_{\{i|a_i<0\}} a_i U_i \\ \bar{a} &= \sum_{\{i|a_i>0\}} a_i U_i + \sum_{\{i|a_i<0\}} a_i L_i, \end{aligned}$$

where L_i and U_i are the lower and upper bounds on variable i respectively. If $\bar{a} < b$ the problem is infeasible. Otherwise, if $\bar{a} = b$, there is only one feasible point and the values of the variables are fixed at U_i for all i with $a_i > 0$ and L_i for all i with $a_i < 0$. Set (1.10) is then replaced with:

$$\tilde{X} \cap \{x \mid b - a^T x \in \{0\}^o\}, \tag{1.11}$$

where $\tilde{X}$ contains the fixed variables. The net result is that the forcing constraint and a number of variables are removed from the original problem.

Case	Action
$Y_1 = \Re$ and $Y_2 = \Re$	If $b = c$ remove one of the constraints, otherwise the problem is infeasible.
$Y_1 = \Re$ and $Y_2 = \Re_+$	If $b \geq c$ remove the inequality constraint, otherwise the problem is infeasible.
$Y_1 = \Re_+$ and $Y_2 = \Re_+$	If $b \geq c$ remove the constraint associated with Y_2 otherwise remove the Y_1 constraint.
$Y_1 = \Re_-$ and $Y_2 = \Re_+$	If $b < c$, the problem is infeasible. Otherwise if $b = c$ make one of the constraints an equation and remove the other. Otherwise, it is a range constraint; nothing is done by the preprocessor.

Table 1.1 Redundant Rows Cases

2.1.3 Redundant Rows. Redundancy in the Jacobian matrix can cause difficulty for many algorithms. Therefore, it is advantageous to remove as much redundancy as possible. The algorithm given in [21] is used to identify duplicate rows. All eligible constraints are checked simultaneously with the algorithm. After finding two duplicate rows, any inconsistencies are uncovered (meaning that the model in unsolvable) and one of the constraints is removed wherever possible. Without loss of generality, let the constraint set be written as

$$X \cap \left\{ x \;\middle|\; \begin{array}{l} b - ax \in Y_1^o \\ c - ax \in Y_2^o \end{array} \right\}. \tag{1.12}$$

Several cases are presented in Table 1.1 along with the associated action taken. The other cases are symmetric to those given in the table.

2.1.4 Extensions. The requirements in Theorem 1 can be slightly weakened. Let $D \in \Re^{n \times n}$ be a positive diagonal matrix. Then the following form will suffice instead of (1.5):

$$0 \in \left[\begin{array}{c} F(x) - DA^T y \\ Ax - b \end{array} \right] + \left[\begin{array}{c} N_X(x) \\ N_Y(y) \end{array} \right]. \tag{1.13}$$

(1.13) can be reduced to (1.5) by applying a diagonal row scaling of $\left[\begin{array}{cc} D & 0 \\ 0 & 1 \end{array} \right]^{-1}$ and recalling that the normal cone does not change under multiplication by a positive diagonal matrix.

2.2 POSTSOLVE

Once the algorithm has solved the preprocessed model, all of the presolve steps must be undone in the reverse order to recover a solution to the original model. The stack of presolve records is used for this purpose. The following steps are performed:

B.1 Remove a presolve record from the top of the stack.

B.2 Transform the problem into the polyhedral-constrained setting.

B.3 Undo the changes made to the model.

B.4 Solve the optimization problem (1.7) using $\bar{x}$ to obtain $\bar{y}$. The generated $(\bar{x}, \bar{y})$ solves the model before the presolve step was performed.

B.5 Repeat until the presolve stack is empty.

The optimization problem (1.7) is typically only in one dimension and is trivial to solve. Care must be taken when calculating $N_X(\bar{x})$ because the algorithm may only find a solution to within a prespecified tolerance. Therefore, variables within some tolerance of their bounds should be treated as if they are exactly on their bounds when constructing the normal cone.

The only case where two variables are involved in the optimization problem is when two inequalities are replaced by one equation. The optimization problem in this case has an objective function equal to zero because at the solution $A\bar{x} - b = 0$. Therefore, a feasible point need only be generated. In the presolved model, $0 \in F(x) - a\hat{y} + N_X(\bar{x})$ for the solution $(\bar{x}, \hat{y})$ given. Without loss of generality, assume $Y_1 = \Re_+$ and $Y_2 = \Re_-$. Select $y_1 \in Y_1 = \Re_+$ and $y_2 \in Y_2 = \Re_-$ such that $y_1 + y_2 = \hat{y}$. These conditions can always be trivially met. Because the inclusion holds at $\hat{y}$, it also holds for the y_1 and y_2 selected, which is then a feasible point as required.

In the unfortunate case that the algorithm fails to solve the preprocessed model, the optimization problem may have no solution either because it is infeasible or unbounded. In this case, a value for the multiplier is chosen in Y such that the norm of the error in the box constrained representation is minimized given $\bar{x}$. This greedy heuristic will lead to the best possible value in terms of the residual at each stage in the unrolling of the preprocessing steps, but not necessarily the least residual solution overall.

3. STRUCTURAL IMPLICATIONS

The second phase of the preprocessor utilizes complementarity theory to eliminate variables from the MCP. The reductions documented are based on the rows and columns of the Jacobian, ∇F. The main ingredient for the row-based rules is uniqueness. If the value for a variable can be uniquely determined prior to solving the remainder of the problem, it is fixed and removed. The column-based rules rely upon an existence argument. Once a solution to the reduced model is known, a solution to the original problem always exists. Mechanisms developed include using interval evaluations, uncovering duplicate rows and columns, and exploiting special structure. Note that when a row with zero elements and corresponding zero column are present in a model, the variable can always be fixed at an appropriate value and removed.

3.1 INTERVALS

An interval evaluator determines the tightest possible $\underline{\mathrm{F}}$ and $\bar{F}$ such that for all $x \in X$, $\underline{\mathrm{F}} \leq F(x) \leq \bar{F}$. For example, with a linear constraint, $F_i(x) = a^T x - b$, the bounds

$$\underline{\mathrm{F}}_i = \sum_{\{j|a_j>0\}} a_j L_j + \sum_{\{j|a_j<0\}} a_j U_j - b$$
$$\bar{F}_i = \sum_{\{j|a_j>0\}} a_j U_j + \sum_{\{j|a_j<0\}} a_j L_j - b$$

can be used where L_j and U_j are the lower and upper bounds on variable j respectively. The range of a nonlinear function is dependent upon the model and the bounds must be computed by a user supplied routine.

Using the ranges, variables in the model can be fixed. If $\underline{\mathrm{F}}_i > 0$, then x_i must be fixed at its finite lower bound or the problem is infeasible. Furthermore, if $\bar{F}_i < 0$, then x_i must be fixed at its finite upper bound or the problem is infeasible. Some of the constraints in the model will imply tighter bounds on the variables; i.e. a linear constraint. These can then be used by the interval evaluator to strengthen the range of other constraints, leading to more variables being fixed.

3.2 DUPLICATES

Duplicate rows and columns can be very problematic for a solver. By applying the same algorithm used in the polyhedrally constrained case (Section 2.1.3), two such linear rows or columns can be identified. First consider the case of two duplicate rows in the problem. Without loss of

Case	Action
$Y = \Re$ and $Z = \Re$	If $b \neq c$ the problem is infeasible. Otherwise, nothing is done.
$Y = \Re$ and $Z = \Re_+$	If $b > c$ fix z at its lower bound. Otherwise, if $b < c$, the problem is infeasible.
$Y = \Re_+$ and $Z = \Re_+$	If $b > c$ fix z at its lower bound. If $b < c$ fix y at its lower bound.
$Y = \Re_-$ and $Z = \Re_+$	If $b < c$, the problem is infeasible. Otherwise, nothing is done.

Table 1.2 Duplicate Rows Cases

generality, the model can be written as:

$$0 \in \begin{bmatrix} F(x,y,z) \\ a^T(x,y,z) + b \\ a^T(x,y,z) + c \end{bmatrix} + \begin{bmatrix} N_X(x) \\ N_Y(y) \\ N_Z(z) \end{bmatrix}.$$

Table 1.2 discusses the reductions that can be made.

To remove column duplicates, one of the variables needs to be free and the other must have two finite bounds. The problem in this case is:

$$0 \in F(x) + ay + az + N_{X \times \Re \times [L,U]}(x,y,z),$$

where L and U are the finite lower and upper bounds on z. The reduction removes the z variable from the problem and solves the reduced system to obtain $(\bar{x}, \bar{y})$. If $F_z(\bar{x}) + a_z\bar{y} > 0$, then $\hat{z} = L$. Otherwise, $\hat{z} = U$. Set $\hat{y} = \bar{y} - \hat{z}$. Then $(\bar{x}, \hat{y}, \hat{z})$ solves the original problem as can be easily verified.

3.3 SPECIAL STRUCTURE

For a system of nonlinear equations, if the problem has the form:

$$\begin{bmatrix} F(x) \\ G(x,y) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

with $F : \Re^k \to \Re^k$, then $F(x) = 0$ can be solved giving $\bar{x}$ and then a $\bar{y}$ solving $G(\bar{x}, y) = 0$ can be found. If $F(x) = 0$ has multiple solutions, this procedure may fail by finding an $\bar{x}$ for which $G(\bar{x}, y) = 0$ has no solution. For example, consider $F(x) = x^2 - 1$ and $G(x,y) = x + y^2$. $F(x) = 0$ has two solutions $\bar{x} = 1$ and $\bar{x} = -1$. Choosing $\bar{x} = 1$ leads to the case where $G(\bar{x}, y)$ has no solution. If $F(x) = 0$ has at most one solution, this case

is precluded provided the original problem has a solution. Similarly, the difficulty is alleviated if $G(\bar{x}, y)$ has a solution $\bar{y}$ for all $\bar{x}$, since whatever $\bar{x}$ is found, the system $G(\bar{x}, y) = 0$ is solvable. This section applies similar block reduction schemes to the mixed complementarity problem.

Consider a problem of the form:

$$0 \in \begin{bmatrix} F(x) \\ G(x,y) \end{bmatrix} + \begin{bmatrix} N_X(x) \\ N_Y(y) \end{bmatrix},$$

where, as usual, X and Y are Cartesian products of intervals. There are two sets of reductions that can be made. If $0 \in F(x) + N_X(x)$ has a unique solution, $\bar{x}$, then x can be fixed and the algorithm will only work on the reduced problem $0 \in G(\bar{x}, y) + N_Y(y)$. If $F(x)$ is an affine function, i.e. $F(x) = Ax - b$, then it is known that $0 \in F(x) + N_X(x)$ has a unique solution if A_X, the normal map associated with this variational inequality, is coherently oriented [16]. For example, when $X = \Re^k_+$, this condition is equivalent to A being a P-matrix. For simple cases, coherent orientation can be checked; e.g. when $k = 1$ or 2. In particular, when $F(x) = ax - b$ is a row singleton with a linear element on the diagonal, then coherent orientation is $a \neq 0$ when $X = \Re$ and $a > 0$ in all other cases. Satisfaction of this condition guarantees uniqueness of the solution. When $k = 2$, the condition is again that A is a P-matrix, unless one or more of the intervals defining X is $\Re$. In these later cases, the conditions are weaker. Table 1.3 summarizes all of the checks for coherent orientation. The preprocessor identifies double blocks by finding a linear row with two elements, one of which is on the diagonal. A check of the row corresponding to the other variable is performed to see if there is a doubleton block.

The other reduction to consider is where $0 \in G(x, y) + N_Y(y)$ has a solution for all $x \in X$. In this case, $0 \in F(x) + N_X(x)$ is solved to find $\bar{x}$ and then a $\bar{y}$ satisfying $0 \in G(\bar{x}, y) + N_Y(y)$ is found. Assume that $G(x, y)$ is linear in y, i.e. $G(x, y) = H(x) + By$. The coherent orientation conditions outlined above applied to B suffice in this case as well, since they guarantee existence as well as uniqueness. However, to guarantee only existence, weaker conditions are sufficient. For $k = 1$ it is necessary and sufficient to have coherent orientation or Y compact. When $k = 2$, the conditions are outlined in Table 1.4 and are derived from Theorem 2 in [14] and [13]. The conditions given for the cases where there is at least one free variable are necessary and sufficient to guarantee existence for all $x \in X$.

In the nonlinear setting, intervals on the Jacobian elements can be used to verify conditions related to uniqueness and existence of a solution. For example in the single element case, if the value of the Jacobian

Case	Coherent Orientation Condition
$X = \Re$	$A_{1,1} \neq 0$
$X = [L, \infty)$	$A_{1,1} > 0$
$X = (-\infty, U]$	$A_{1,1} > 0$
$X = [L, U]$	$A_{1,1} > 0$
$X = \Re \times \Re$	$det(A) \neq 0$
$X = \Re \times [L, \infty)$	$det(A) \neq 0$ and $sign(det(A)) = sign(A_{1,1})$
$X = \Re \times (-\infty, U]$	$det(A) \neq 0$ and $sign(det(A)) = sign(A_{1,1})$
$X = \Re \times [L, U]$	$det(A) \neq 0$ and $sign(det(A)) = sign(A_{1,1})$
All other cases	$det(A) > 0$ and $sign(A_{1,1}) = sign(A_{2,2}) = 1$

Table 1.3 Coherent Orientation Conditions. X is assumed to be a Cartesian product of intervals with L and U being two finite numbers.

Case	Condition
$Y = \Re$	$B_{1,1} \neq 0$
$Y = [L, \infty)$	$B_{1,1} > 0$
$Y = (-\infty, U]$	$B_{1,1} > 0$
$Y = [L, U]$	nothing
$Y = \Re \times \Re$	$det(B) \neq 0$
$Y = \Re \times [L, \infty)$	$det(B) \neq 0$ and $sign(det(B)) = sign(B_{1,1})$
$Y = \Re \times (-\infty, U]$	$det(B) \neq 0$ and $sign(det(B)) = sign(B_{1,1})$
$Y = \Re \times [L, U]$	$B_{1,1} \neq 0$
$Y = [L, \infty) \times [\tilde{L}, \infty)$	$B > 0$ or $(det(B) > 0$ and $sign(B_{1,1}) = sign(B_{2,2}) = 1)$
$Y = [L, \infty) \times (-\infty, \tilde{U}]$	$B > 0$ or $(det(B) > 0$ and $sign(B_{1,1}) = sign(B_{2,2}) = 1)$
$Y = [L, \infty) \times [\tilde{L}, \tilde{U}]$	$B_{1,1} \neq 0$
$Y = (-\infty, U] \times (-\infty, \tilde{U}]$	$B > 0$ or $(det(B) > 0$ and $sign(B_{1,1}) = sign(B_{2,2}) = 1)$
$Y = (-\infty, U] \times [\tilde{L}, \tilde{U}]$	$B_{1,1} \neq 0$
$Y = [L, U] \times [\tilde{L}, \tilde{U}]$	nothing

Table 1.4 Existence Conditions. Y is assumed to be a Cartesian product of intervals with L, $\tilde{L}$, U, and $\tilde{U}$ being finite numbers.

element is always positive, i.e. it is a P-function, then the existence and uniqueness is always guaranteed and the same substitutions can be performed. Finding the solution becomes more difficult, as it involves solving a nonlinear problem.

4. RESULTS

The preprocessing algorithm that was implemented alternates between exploiting the polyhedral structure and the functions. Initially all possible reductions based on the polyhedral constraints are made. Then all reductions based on the functional implications are made. These two steps are repeated until no changes are made to the model.

The potential for preprocessing is mainly limited to finding and exploiting linear parts of the problem. Interval evaluators are not available at present in the modeling language environments. The majority of the reductions made come from exploiting polyhedral structure. However, the reductions from the second stage can also be significant to the success of the algorithm.

The preprocessor was tested on three different sets of problems. The first test compares the performance of the MCP preprocessor to the one used by the commercial CPLEX code [15]. Using the linear programs contained in NETLIB [12], the first order conditions from linear programming were constructed and given to the MCP preprocessor. CPLEX was given the original linear program. Reported in Tables 1.5 and 1.6 are the sizes of the preprocessed models. CPLEX is capable of performing aggregations, while the MCP preprocessor currently does not. Therefore, in the tables, the size of the model produced by CPLEX both with aggregations (With) and without aggregations (Without) are stated. As evidenced by the table, the MCP preprocessor is competitive with CPLEX on linear programs when aggregations are not allowed. One interesting point to note is that the problems `fit*p` and `fit*d` are primal-dual pairs - the MCP preprocessor generates an identical system in both cases. Exploiting dual information in the `fit*p` problems significantly reduces the size of the preprocessed models.

A second test was performed using quadratic programming problems reformulated using the complementary slackness conditions (1.3). Some artificial quadratic programs were created for testing purposes from the NETLIB collection. A term of $\frac{1}{2}x^Tx$ was added to the objective function and the resulting complementary slackness conditions were given to the preprocessor and the PATH algorithm [5]. Table 1.7 reports the size reductions and compares the solution times on the original and preprocessed models. On the problems successfully preprocessed, the

Model	Size	CPLEX With	CPLEX Without	MCP Preprocessor
adlittle	153	147	147	146
afiro	59	48	52	56
agg	651	271	275	433
agg2	818	530	538	743
agg3	818	531	541	743
bandm	777	398	483	467
beaconfd	435	55	220	205
blend	157	108	140	149
bnl1	1807	1443	1668	1670
bnl2	5769	3031	4226	4341
boeing1	909	711	713	720
boeing2	320	281	281	292
bore3d	547	105	182	191
brandy	431	265	311	311
capri	608	383	547	547
cycle	4743	2700	3416	3884
czprob	4221	2904	3349	3430
d2q06c	7338	6450	6871	6286
d6cube	6588	5844	5867	6423
degen2	978	855	977	974
degen3	3321	3125	3321	3310
dfl001	18301	13062	17091	16915
e226	505	397	411	414
etamacro	1006	754	850	821
fffff800	1378	933	983	1284
finnis	1066	739	786	808
fit1d	1050	1048	1048	1050
fit1p	2304	2054	2054	1050
fit2d	10525	10388	10388	10525
fit2p	16525	16525	16525	10525
forplan	554	466	476	483
ganges	2990	1202	2177	2466
gfrd-pnc	1708	1116	1656	1656
greenbea	7691	4055	5900	5763
greenbeb	7679	4044	5892	5738
grow15	945	945	945	945
grow22	1386	1386	1386	1386
grow7	441	441	441	441
israel	316	304	304	304

Table 1.5 Comparison of CPLEX Preprocessor to the one developed on NETLIB problems

		CPLEX		
Model	Size	With	Without	MCP Preprocessor
kb2	84	67	79	82
lotfi	461	399	399	408
nesm	3585	3325	3373	3440
perold	1937	1571	1769	1757
pilot4	1380	1111	1200	1210
sc105	207	117	207	207
sc205	407	231	405	405
sc50a	97	57	97	97
sc50b	96	56	96	96
scagr25	971	591	841	734
scagr7	269	159	229	194
scfxm1	787	612	694	698
scfxm2	1574	1228	1388	1396
scfxm3	2361	1844	2082	2094
scorpion	746	172	590	453
scrs8	1659	913	1429	1438
scsd1	837	837	837	817
scsd6	1497	1497	1497	1481
scsd8	3147	3147	3147	3135
sctap1	780	608	608	660
sctap2	2970	2303	2303	2500
sctap3	3960	3111	3111	3340
share1b	342	297	315	310
share2b	175	168	172	172
shell	2061	1427	1935	1935
ship04l	2478	2174	2182	2208
ship04s	1818	1426	1482	1500
ship08l	4995	3569	3569	3611
ship08s	3099	1760	1858	1890
ship12l	6469	4756	4756	4790
ship12s	3805	2114	2258	2288
stair	741	512	740	740
stocfor1	228	113	190	188
stocfor2	4188	2474	3822	3825
tuff	878	514	738	788
wood1p	2838	1898	1898	1971
Total	181745	137202	155734	150753

Table 1.6 Comparison of CPLEX Preprocessor to the one developed on NETLIB problems (cont.)

Model	Original Size	Original Solution Time (sec.)	Preprocessed Size	Preprocessed Solution Time (sec.)
agg	651	6.6	454	1.0
beaconfd	435	1.1	283	0.7
finnis	1066	9.1	918	1.5
lotfi	461	5.9	434	0.9
nesm	3585	57.6	3481	53.0
scorpion	746	1.1	617	0.8
ship08s	3099	6.3	1966	3.3
tuff	878	4.3	849	3.9

Table 1.7 Comparison of PATH solution times on QP models with and without preprocessing.

Model	Original Size	Original Solution Time (sec.)	Preprocessed Size	Preprocessed Solution Time (sec.)
electric	158	1.3	140	0.5
explcp	16	0.0	0	0.0
forcebsm	184	0.1	72	0.2
forcedsa	186	0.1	70	0.1
golanmcp	4321	80.9	4304	25.0
merge	9536	2254.6	8417	1954.2

Table 1.8 Comparison of PATH solution times on MCP models with and without preprocessing.

reductions in time are significant. Some other quadratic programs from other sources were also tested. On one of the models, hwayoung, over 70% of the variables were removed by the preprocessor reducing the size from 46123 variables to 13655.

Finally, the models in MCPLIB [4] were given to the preprocessor. The results on these models are less encouraging than the other two tests. This stems from a lack of linear problems in the test set and the inability to obtain interval evaluations for the nonlinear functions. Many of the models did not benefit from preprocessing. However, some successes are reported in Table 1.8. Note that the `explcp` model that is supposed to display exponential behaviour for Lemke's algorithm is completely solved in the preprocessor. Some of the preprocessing performed was detrimental. For example, the force* models became harder to solve after preprocessing even though they were significantly reduced in size.

Overall the results of the preprocessor are very encouraging. Unfortunately, many of the models that are currently in MCPLIB do not have large amounts of exploitable linear structure, so the benefits of preprocessing them are limited. Further work in exploiting range constraints and aggregations is warranted. An interval evaluator is planned for the GAMS modeling language which will enable the nonlinear models to be more effectively preprocessed.

Acknowledgments

This material is based on research supported by National Science Foundation Grant CCR-9972372 and Air Force Office of Scientific Research Grant F49620-98-1-0417.

References

[1] E. Andersen and K. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.

[2] A. Brearley, G. Mitra, and H. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 8:54–83, 1975.

[3] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[4] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[5] S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.

[6] M. C. Ferris, R. Fourer, and D. M. Gay. Expressing complementarity problems and communicating them to solvers. *SIAM Journal on Optimization, forthcoming*, 1999.

[7] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming, forthcoming*, 1999.

[8] M. C. Ferris, M. P. Mesnier, and J. Moré. NEOS and Condor: Solving nonlinear optimization problems over the Internet. *ACM Transactions on Mathematical Software, forthcoming*, 1999.

[9] M. C. Ferris and T. S. Munson. Complementarity problems in GAMS and the PATH solver. *Journal of Economic Dynamics and Control, forthcoming*, 1999.

[10] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39:669–713, 1997.

[11] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, 1993.

[12] D. M. Gay. Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–12, 1985.

[13] M. S. Gowda. Applications of degree theory to linear complementarity problems. *Mathematics of Operations Research*, 18:868–879, 1993.

[14] M. S. Gowda. An analysis of zero set and global error bound properties of a piecewise affine function via its recession function. *SIAM Journal on Matrix Analysis and Applications*, 17:594–609, 1996.

[15] ILOG CPLEX Division, Incline Village, Nevada. *CPLEX Version 6.5.* http://www.cplex.com/.

[16] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.

[17] S. M. Robinson. A reduction method for variational inequalities. *Mathematical Programming*, 80:161–169, 1998.

[18] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

[19] M. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[20] H. Sellami and S. M. Robinson. Implementation of a continutation method for normal maps. *Mathematical Programming*, pages 563–578, 1997.

[21] J. Tomlin and J. Welch. Finding duplicate rows in a linear programming model. *Operations Research Letters*, 5(1):7–11, 1986.