CRASH TECHNIQUES FOR LARGE-SCALE COMPLEMENTARITY PROBLEMS*

STEVEN P. DIRKSE† AND MICHAEL C. FERRIS‡

Abstract. Most Newton-based solvers for complementarity problems converge rapidly to a solution once they are close to the solution point and the correct active set has been found. We discuss the design and implementation of crash techniques that compute a good active set quickly based on projected gradient and projected Newton directions. Computational results obtained using these crash techniques with PATH and SMOOTH, state-of-the-art complementarity solvers, are given, demonstrating in particular the value of the projected Newton technique in this context.

Key words. complementarity problems, crash techniques, projected Newton, crossover

AMS subject classifications. 90C33, 49M15, 49M07

1. Introduction.

The mixed complementarity problem (MCP) can be viewed as a generalization of a system of nonlinear equations. It is completely determined by a nonlinear function $F : \mathbf{R}^n \to \mathbf{R}^n$ and upper and lower bounds on the problem variables. The variables z must lie between the given bounds ℓ and u, while constraints on the nonlinear function are determined by the bounds on the variables in the following manner:

$$\begin{array}{rcl} \ell_i < z_i < u_i & \Rightarrow & F_i(z) = 0 \\ z_i = \ell_i & \Rightarrow & F_i(z) \geq 0 \\ z_i = u_i & \Rightarrow & F_i(z) \leq 0. \end{array} \tag{1}$$

Note that a square system of nonlinear equations

$$F(x) = 0 \tag{2}$$

and the nonlinear complementarity problem (NCP) are both special cases of the MCP, each obtained by a suitable choice of the bounds ℓ and u. In order that Newton type methods can be applied, we will assume throughout this paper that F is continuously differentiable.

We will use the notation $\mathbf{B}$ throughout this paper to represent the set $\{z \mid \ell \leq z \leq u\}$, while the projection operator onto the set $\mathbf{B}$ is denoted by $\pi_{\mathbf{B}}(\cdot)$. We recall that the projection of x onto a closed convex set $\mathbf{B}$ is the closest point of $\mathbf{B}$ to x measured in the Euclidean norm, so that the ith component of $\pi_{\mathbf{B}}(x)$ is just the median value of x_i, ℓ_i and u_i. The set $\mathbf{B}$ can be replaced by any closed convex set for the majority of this paper. However, we restrict our attention to this set throughout the paper due to the fact that our implementations exploit the particular structure of $\mathbf{B}$ when performing the requisite projections.

While simplicial labeling techniques [37] and fixed point schemes [24, 38] possess very powerful theoretical properties, Newton based techniques have proven very effective for solving large-scale MCP's [7, 8, 12, 15, 17, 21, 23, 26, 31, 32, 33, 36, 39]. All of

* This material is based on research supported by National Science Foundation Grant CCR-9157632.

† GAMS Development Corporation, 1217 Potomac Street NW, Washington, D.C. 20007 (steve@gams.com).

‡ Computer Sciences Department, University of Wisconsin – Madison, 1210 West Dayton St., Madison, Wisconsin 53706 (ferris@cs.wisc.edu).

these algorithms maintain the excellent convergence properties of Newton's method near a solution to the MCP. However, the region on which this convergence is guaranteed for a full Newton step may be quite small. Much effort has gone into proving global convergence properties for algorithms when started from points far away from the solution. The main purpose of this paper is to describe and test heuristic techniques that are used to quickly identify the active set. Once an approximate active set has been identified, we revert to a standard Newton based code to solve the problem.

We first describe the reformulation of the MCP as a nonsmooth system of equations using the normal map [19, 34, 35] defined below. This will enable us to describe our Newton method based on similar techniques for smooth systems of equations.

DEFINITION 1.1 (NORMAL MAP). *Given the closed convex set* $\mathbf{B} \subset \mathbf{R}^n$ *and a function* $F : \mathbf{B} \to \mathbf{R}^n$*, the normal map* $F_{\mathbf{B}}(\cdot)$ *induced on* F *by* $\mathbf{B}$ *is defined as*

$$F_{\mathbf{B}}(x) := F(\pi_{\mathbf{B}}(x)) + (x - \pi_{\mathbf{B}}(x)).$$

The problem we consider is to find a zero of the normal map, namely an $x \in \mathbf{R}^n$ satisfying

$$0 = F_{\mathbf{B}}(x) = F(\pi_{\mathbf{B}}(x)) + (x - \pi_{\mathbf{B}}(x)). \tag{3}$$

The following theorem is easily established using properties of the projection operator and indicates the connection between zeros of the normal map and solutions of the MCP.

THEOREM 1.2. *Given a rectangular set* $\mathbf{B} := \{z \mid \ell \leq z \leq u\}$ *and function* $F : \mathbf{B} \to \mathbf{R}^n$*, then*

1. $x \in \mathbf{R}^n$ *is a zero of the normal map implies that* $z := \pi_{\mathbf{B}}(x)$ *solves MCP*
2. z *solves MCP implies that* $x := z - F(z)$ *is a zero of the normal map.*

The normal map is intimately related to the normal cone to the set $\mathbf{B}$.

DEFINITION 1.3 (NORMAL CONE). *For the convex set* $\mathbf{B}$ *and a point* $z \in \mathbf{B}$*, the normal cone to* $\mathbf{B}$ *at* z *is the set*

$$N_{\mathbf{B}}(z) := \{y \mid y^\top(c - z) \leq 0, \forall c \in \mathbf{B}\}.$$

When $z \notin \mathbf{B}$ *we define* $N_{\mathbf{B}}(z) = \emptyset$*.* Simple algebraic manipulations show that

$$N_{\mathbf{B}}(z) = \{y \mid y = -w + v, w, v \geq 0, w^\top(z - \ell) = 0, v^\top(u - z) = 0\}. \tag{4}$$

Definition 1.3 makes it clear that z solves (1) if and only if z solves the generalized equation

$$0 \in F(z) + N_{\mathbf{B}}(z). \tag{5}$$

We can therefore approach MCP as either a zero-finding problem (3) of the normal map over the set of variables $x \in \mathbf{R}^n$ or as the original problem of finding a $z \in \mathbf{B}$ satisfying (5). We will always use x to indicate the normal map variable and z to denote the variable in MCP.

Note that the normal map is not in general differentiable everywhere. Due to the polyhedral nature of the set $\mathbf{B}$, it is a piecewise smooth map. The pieces of the map can be associated in a very natural way with the active set at the projected point $z = \pi_{\mathbf{B}}(x)$. The active set at z is the set of indices for which $z_i = \ell_i$ or $z_i = u_i$; each active set therefore defines a face of the set $\mathbf{B}$. It is well known that the collection of these faces precisely determine the pieces of smoothness of the normal map. For example, it was shown in [35] that these pieces are the (full dimensional) polyhedral sets $\mathcal{F} + N_{\mathcal{F}}$ that are indexed by the faces $\mathcal{F}$; here $N_{\mathcal{F}}$ represents the normal cone to the face $\mathcal{F}$ at any point in its relative interior.

In the context of nonlinear equations, Newton's method proceeds by linearizing the smooth function F. Since $F_{\mathbf{B}}$ is nondifferentiable, the standard version of Newton's method for MCP approximates $F_{\mathbf{B}}$ at $x^k \in \mathbf{R}^n$ with the piecewise affine map

$$L_k := M\pi_{\mathbf{B}}(x) + q + x - \pi_{\mathbf{B}}(x) \tag{6}$$

where

$$M := \nabla F(\pi_{\mathbf{B}}(x^k)) \qquad \text{and} \qquad q := F(\pi_{\mathbf{B}}(x^k)) - M\pi_{\mathbf{B}}(x^k).$$

Thus, the piecewise smooth map $F_{\mathbf{B}}$ has been approximated by a piecewise affine map. The Newton point x_N^k (a zero of the approximation L_k) is found by generating a path p^k, parametrized by a variable t which starts at 0 and increases to 1 so that $p^k(0) = x^k$ and $p^k(1) = x_N^k$. The values of $p^k(t)$ at intermediate points in the path are generated to satisfy

$$L_k(p^k(t)) = (1-t)r^k, \tag{7}$$

where $r^k := F_{\mathbf{B}}(x^k)$. The path is known to exist locally under fairly standard assumptions and can be generated using standard pivotal techniques to move from one piece of the piecewise affine map to another. Further details can be found in [33].

A Newton method for MCP would accept x_N^k as a new approximation to the solution and re-linearize at this point. However, as is well known even in the literature on smooth systems, this process is unlikely to converge for starting points that are not close to a solution. In a damped Newton method for smooth systems of nonlinear equations (2), the merit function $F(x)^\top F(x)$ is typically used to restrict the step size and enlarge the domain of convergence. In our implementation of the above Newton method, which we call PATH [17], the piecewise linear path p^k is computed and searched using a non-monotone watchdog path-search. The watchdog technique allows path-searches to be performed infrequently, while the non-monotone technique allows the Newton point to be accepted more frequently. The combination of these techniques helps avoid convergence to minimizers of the merit function $\|F_{\mathbf{B}}(x)\|^2$ that are not zeros of $F_{\mathbf{B}}$, without detracting from the local convergence rates [17].

There are two difficulties associated with such globalization. The first problem (that the linearization is only a good approximation locally) is dealt with effectively by the line/path search techniques outlined above and is not the concern of this paper.

The second problem is that initially we may have a very poor estimate of which piece of the normal map contains the zero. Essentially, this amounts to the fact that the initial point gives a poor indication of the active set at a solution. In order to reduce the number of steps required to solve the problem, the starting point given to a solver can be adjusted using non-pivotal techniques so that the active set at the adjusted point more closely matches that at the solution to the MCP. Techniques for accomplishing this cheaply are the subject of this paper and will be referred to as "crash techniques". The underlying idea is closely related to the crash procedures that are now commonplace in production style linear programming codes [14, 30], although we largely ignore the problem of modifying the active set to generate a nonsingular basis. Our crash techniques are iterative in nature and employ a merit function to deal with the nonlinear nature of the problem. Once the initial point has been adjusted by such a technique (the crash phase of the algorithm), the original Newton type solver is initialized from the adjusted point.

Practical experience has already shown PATH [17] to be a very robust technique for solving nonlinear MCP's, and to be a reasonably fast one as well [7]. One reason PATH represents a significant increase in speed over other pivotal algorithms is that it begins pivoting for each subproblem (6) using the basis determined by the active set at the current point x^k, and not the all-slack basis used by the Josephy-Newton algorithm [27], for example. Close to a solution, the basis may change very little from subproblem to subproblem, if at all, so PATH does not need to do any pivoting, and behaves like Newton's method applied to smooth systems of equations.

Even so, pivotal codes for MCP such as MILES [36] and PATH [17] may be forced to perform a very large number of pivots in order to solve the initial subproblem. Since these solvers use an active set approach, each pivot corresponds to a variable entering or leaving the set of active constraints. Given such a technique, the number of pivots required to solve a subproblem is bounded below by the size of the difference between the initial and final active sets. This number can be expected to grow with problem size. Even if an efficient factorization updating scheme is used to perform each pivot step, the cost of this pivoting can become the dominant factor in solving these problems. The goal of the crash phase is to quickly adjust the initial point so that it corresponds more closely to such an active set so that very few pivots are required to solve the problem.

Section 2 begins with a description of a unified approach for our crash techniques using a given search direction, projections onto the feasible set, and a metric for comparing points $z \in \mathbf{B}$. A number of different choices for the search direction exist within this framework. Directions based on projected gradient algorithms from nonlinear programming are considered first in Section 2.1. Section 2.2 describes a Newton direction based on a system of reduced size determined by the active constraints, while Section 2.4 describes a modified Newton direction based on a smoothing [12] of the original problem. The computational testing of the crash procedures with both PATH and SMOOTH as the Newton method, along with some results and conclusions based on these tests, are described in Section 3. More extensive numerical results can be found in Appendix A.

2. Crash Techniqes. In nonlinear programming, a technique that has been fruitfully applied to identify a good active set is the projected gradient method [2]. Given the problem

$$\min_{z \in \mathbf{B}} f(z)$$

the projected gradient method determines z^{k+1} from z^k by choosing the largest α from $\{\alpha_{\max}, \alpha_{\max}/2, \alpha_{\max}/4, \ldots\}$ such that

$$z^{k+1}(\alpha) = \pi_{\mathbf{B}}(z^k - \alpha \nabla f(z^k))$$

satisfies

$$f(z^k) - f(z^{k+1}(\alpha)) \geq \sigma \nabla f(z^k)(z^k - z^{k+1}(\alpha)) \tag{8}$$

The scalars $\alpha_{\max} > 0$, and $\sigma \in (0, 0.5)$ are fixed. Conditions under which this method identifies the active set after a finite number of iterations are given in [10, 11]. Since there is potential for a large change in the current active set as a result of a single step in this method, these methods are used in large scale codes to determine an active set quickly, after which a Newton method is applied on the identified face [13]. Note that when $\mathbf{B}$ is polyhedral, $z^{k+1}(\alpha)$ defines a piecewise linear path for $\alpha \in (0, \infty)$. The fact that a large change in the active set can occur in the path search (8) should be contrasted with techniques that line search between z^k and $z^{k+1}(1)$ for which a small change in the active set is more typical.

Motivated by this work, all of our crash implementations are based on the following sequence of steps. Given a point $z^k \in \mathbf{B}$, let $z^{k+1}(\alpha) := z(\alpha)$ where

$$z(\alpha) = \pi_{\mathbf{B}}(z^k - \alpha d^k). \tag{9}$$

If the direction $d^k = F(z^k)$, and F is the gradient of a scalar function f, the direction is identical to that given above. Previous work on projected gradient algorithms for solving MCP and NCP includes [4], and sufficient conditions for the convergence of such a method to a solution of an MCP (which include the Lipschitz continuity and strong monotonicity of F) are given in [5]. These results are based on a contraction argument and require a fixed stepsize α chosen sufficiently small. Alternative arguments for the affine case based on the notion of forward-backward splitting can be found in [20] and its cited references. Results using an Armijo type line search similar to (8) are scarce due to the lack of a good merit function. Some convergence results for a number of projection methods based on a gap function derived from the corresponding variational inequality are given in [22, 28].

In order to perform an Armijo search similar to (8), we require a merit function to minimize, a function of the iterates $z \in \mathbf{B}$. The PATH solver uses $\|F_{\mathbf{B}}(\cdot)\|$, so to maintain consistency between the crash phase and the PATH algorithm, we use the relationship between z and x to define

$$x(\alpha) := \arg\min_x \{\|F_{\mathbf{B}}(x)\| \mid z(\alpha) = \pi_{\mathbf{B}}(x)\}. \tag{10}$$

This has the effect of choosing to evaluate $\|F_{\mathbf{B}}(\cdot)\|$ at the best $x \in \mathbf{R}^n$ *whose projection onto* $\mathbf{B}$ *is* $z(\alpha)$. The resulting path search uses the merit function $\|F_{\mathbf{B}}(x(\alpha))\|$ and chooses the least $m \in \{0, 1, 2, \ldots\}$ such that for $\alpha = (\frac{1}{2})^m$,

$$\|F_{\mathbf{B}}(x(\alpha))\| \leq (1 - \alpha\sigma)\, \|F_{\mathbf{B}}(x(0))\|\,. \tag{11}$$

Note that for $z \in \mathbf{B}$, the representation of the normal cone (4) allows us to define

$$x := z - w + v$$

so that if $-w + v \in N_{\mathbf{B}}(z)$

$$F_{\mathbf{B}}(x) = F(z) - w + v. \tag{12}$$

The computation of $x(\alpha)$ in (10) is therefore trivial and can be accomplished by assigning values to w_i and v_i in the same simple loop that is used to calculate $\|F_{\mathbf{B}}(x(\alpha))\|$. Essentially, values for w and v are determined by projecting $-F(z)$ onto $N_{\mathbf{B}}(z)$.

Since there is currently no strong theoretical justification that the directions we consider will be descent directions for $\|F_{\mathbf{B}}(\cdot)\|$, we need to terminate the crash procedure if we fail to make improvement (or progress becomes slow) and revert to using the Newton technique. The crash technique should also be stopped when the active set ceases to change. The following tolerances are used to determine when to terminate the crash phase.

$\alpha_{\min}$	minimum acceptable path length α
σ	decrease required in (11)
$k_{\max}$	maximum number of steps in crash phase
$\delta_{\max}$	maximum number of crash steps without a change in the active set
$\rho_{\min}$	minimum ratio of current decrease of $\|F_{\mathbf{B}}(\cdot)\|$ to maximum observed decrease in any previous crash step

If any of the above settings are not satisfied, the crash phase terminates. In addition, we require the problem to have at least $n_{\min}$ variables in order to attempt a crash phase. We spent some time tuning the various crash procedures that we now outline by adjusting the parameters mentioned above. The choice of parameters made for each technique is noted in the following sections describing how the particular directions are chosen.

2.1. Projected Gradient Directions. It is well known that if we define the (nonsmooth) mapping

$$H(z) := z - \pi_{\mathbf{B}}(z - F(z)), \tag{13}$$

then z solves MCP if and only if z is a zero of H. If we apply the standard forward-backward splitting scheme to H, we generate our first crash technique, which is just an implementation of the "projected gradient" technique (9) with

$$d^k = F(z^k).$$

Note that no factorization is performed, and only function evaluations (not Jacobian evaluations) are required, so that these directions are very cheap to calculate. We refer to this algorithm as PG. The parameter settings used in our computational tests were $\alpha_{\min} = 2^{-12}$, $\sigma = 0$, $n_{\min} = 1$, $k_{\max} = 50$, $\delta_{\max} = 1$ and $\rho_{\min} = 0.05$.

Unfortunately, the above direction frequently does not give descent for our chosen merit function. A more theoretically justifiable algorithm can be generated by considering the following problem motivated by the decomposition in (12)

$$\min_{z \in \mathbf{B}, -w+v \in N_{\mathbf{B}}(z)} (F(z) - w + v)^\top (F(z) - w + v).$$

We apply a splitting method (fixing w and v and solving for z, and conversely) to solve this problem. In fact, we generate z^{k+1} by considering

$$\min_{z \in \mathbf{B}} (F(z) - w^k + v^k)^\top (F(z) - w^k + v^k)$$

and update w^k and v^k using

$$\min_{-w+v \in N_{\mathbf{B}}(z^k)} (F(z^k) - w + v)^\top (F(z^k) - w + v).$$

The second problem is solved in the simple loop described in the previous section, whereas an iteration of the standard projected gradient method is used for the first problem. In effect, we just choose

$$d^k = \nabla F(z^k)^\top (F(z^k) - w^k + v^k)$$

in the general scheme (9). The parenthetical expression above is precisely the residual term that is used as the covering vector in the PATH algorithm. We term this algorithm PGT. The parameter settings used to test this algorithm were identical to those used in PG.

Note that unlike PG, the PGT direction requires a Jacobian evaluation and a matrix-vector product per iteration, while neither requires a factorization.

2.2. Projected Newton Directions. The chief drawback of the algorithms of Section 2.1 is the quality of the search directions obtained. While they were simple and inexpensive to compute, they did not in all cases provide us with a sufficient decrease in $\|F_{\mathbf{B}}(\cdot)\|$ or in an adequate change in the current set of active constraints. Motivated by the work of [3], in which similar results are reported, we now describe a technique in which a Newton direction for a reduced system is computed and used in a path search similar to that of (8). The Newton direction for the reduced system is essentially the same direction used in the initial pivot step of the path construction phase of the PATH solver. While the path construction algorithm stops at a boundary and recomputes the direction (i.e. performs a pivot step), the projected Newton technique takes a full or damped step and projects back onto the feasible set. By ignoring the bounds in the computation of the search direction and using them only when projecting, we can make large changes in the active set, while the reduced system makes the factorization involved in the computation of the search direction cheaper to perform.

In [31], Pang gives a global convergence result for Newton's method applied to B-differentiable functions, and applies this result to the NCP formulated using the mapping H defined in (13). He notes that the corresponding direction-finding problem has reduced size, although the active set used and the reduced system are different from ours.

We will use the notation $\mathcal{A}$ and $\mathcal{I}$ to indicate the following index sets:

$$\mathcal{A}(z) := \{i \mid \ell_i = z_i,\ F_i(z) \geq 0\} \bigcup \{i \mid u_i = z_i,\ F_i(z) \leq 0\} \tag{14a}$$

$$\mathcal{I} := \{i \mid i \notin \mathcal{A}\} \tag{14b}$$

Typically, the dependence of these sets on z will be implicit. The set $\mathcal{A}$ represents the box constraints that are active (and correct) at $z = \pi_{\mathbf{B}}(x)$, that is, for which F has the right sign. The set $\mathcal{I}$ is an approximation of the indices which are not active at the solution.

The reduced system

$$\nabla F_{\mathcal{I}\mathcal{I}}(z^k) d_{\mathcal{I}} = F_{\mathcal{I}}(z^k) \tag{15}$$

computes the nonzero part of the search direction d. Assuming a reordering of the variables, the new iterate is $z^{k+1} := z(\alpha)$, where

$$z(\alpha) := \pi_{\mathbf{B}}\left(\begin{bmatrix} z^k_{\mathcal{I}} \\ z^k_{\mathcal{A}} \end{bmatrix} - \alpha \begin{bmatrix} d_{\mathcal{I}} \\ 0 \end{bmatrix} \right). \tag{16}$$

Again, α is chosen via a path search (11) to reduce $\|F_{\mathbf{B}}(x(\alpha))\|$, where $x(\alpha)$ is defined using (10). The new iterate z^{k+1} leads to a new choice of index sets $\mathcal{A}$ and $\mathcal{I}$.

We refer to the algorithm described above as PN. The parameter settings used in our computational tests were $\alpha_{\min} = 2^{-12}$, $\sigma = 0.05$, $n_{\min} = 10$, $k_{\max} = 50$, $\delta_{\max} = 1$ and $\rho_{\min} = 0$. In addition, we required that the active set change in at least 10 index positions in the PN technique, as opposed to only one position in PG and PGT.

In solving the reduced system (15) above, the submatrix $\nabla F_{\mathcal{I}\mathcal{I}}(z^k)$ may be rank deficient. For example, if x^k is obtained from z^k as in (10), then this is the case when x^k is a local minimizer of $\|F_{\mathbf{B}}(\cdot)\|$, but $\left\|F_{\mathbf{B}}(x^k)\right\| \neq 0$. In [6], Billups shows it may be possible to solve a sequence of linearized problems with a perturbed function

F_λ derived from F to obtain a new iterate $\hat{x}$ such that $\|F_{\mathbf{B}}(\hat{x})\| < \|F_{\mathbf{B}}(x^k)\|$. The perturbed function F_λ takes the simple form $F_\lambda := F(x) + \lambda(x - \bar{x})$, where $\bar{x}$ is the current point. If an algorithm can guarantee monotonic descent in $\|F_{\mathbf{B}}(\cdot)\|$, this technique can be used to escape a region of convergence to a non-global minimizer of $\|F_{\mathbf{B}}(\cdot)\|$. In addition, λ can be chosen as large as is necessary for $\nabla F_{\mathcal{I}\mathcal{I}}(z^k) + \lambda I$ to be invertible.

In our projected Newton crash technique, we have implemented a procedure for perturbing the Jacobian of F when $\nabla F_{\mathcal{I}\mathcal{I}}(z^k)$ is singular for which multiples of the identity are added in until a numerically nonsingular matrix results. The algorithm chooses the minimum $\lambda \in \{0, 10, 100, \ldots\}$ such that $\nabla F_{\mathcal{I}\mathcal{I}}(z^k) + \lambda I$ is nonsingular. The perturbations are reduced at each iteration using

$$\lambda := \max(.9\,\lambda,\ \|F_{\mathbf{B}}(x)\|/100).$$

Unlike the PATH algorithm, the PN algorithm belongs entirely to the family of complementarity algorithms that determine their search directions by solving a single linear system based on a "derivative approximation" at the current point (e.g. SMOOTH[12], B-DIFF[25], SEMISMOOTH[26]). In spite of this, there are some differences between PN and these methods. While SMOOTH solves a system of order n at each iteration, PN solves a reduced system, as does B-DIFF. More importantly, PN computes a search direction for the original z variables only, and not for the slack variables w and v, as is the case with SMOOTH and B-DIFF. PN computes the correct update for the slacks only after the function F is evaluated at an updated point z. This can be important in nonlinear problems, as illustrated in the following example.

Consider the MCP defined by the function

$$F\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} := \begin{bmatrix} z_1 - 5 \\ z_1^2 + z_2^2 \end{bmatrix} \tag{17}$$

and the nonnegative orthant, where the initial iterate $x^0 = (1, -1)^\top$ is given, $z^0 = \pi_{\mathbf{B}}(x^0) = (1, 0)^\top$, $F_{\mathbf{B}}(x^0) = (-4, 0)^\top$, and $\|F_{\mathbf{B}}(x^0)\| = 4$. The linearization of F at z^0 is $Mx + q$, where $M := \begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix}$ and $q := \begin{bmatrix} -5 \\ -1 \end{bmatrix}$. Since $x_2^0 < 0$, the Newton direction for the current piece of the linear map is determined via the system

$$\begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} -4 \\ 0 \end{bmatrix}. \tag{18}$$

The computed direction $d = (-4, 8)^\top$ is the direction taken by B-DIFF, while SMOOTH takes a similar but somewhat different direction, depending on the smoothing parameter being used. If we consider the Newton point $x_N^1 := x^0 - d = (5, -9)^\top$, we see that $F_{\mathbf{B}}(x_N^1) = (0, 16)^\top$ and $\|F_{\mathbf{B}}(x_N^1)\| = 16$, even though $\pi_{\mathbf{B}}(x_N^1)$ is the solution to MCP($F, \mathbf{B}$). This is because the second row of (18) does a poor job of computing the correct update for x_2, the slack variable corresponding to $F_2(x)$. A line search results in the choice of $\alpha = .25$ and $x^1 = (2, -3)$. Three more steps are required before the solution is obtained.

In contrast, the PN algorithm computes an update to the inactive set of variables (i.e. z_1) based on the system $1 \cdot d = -4$. The updates to z take the form

$$z(\alpha) := \pi_{\mathbf{B}}\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \alpha \begin{bmatrix} -4 \\ 0 \end{bmatrix}\right). \tag{19}$$

When $\alpha = 1$, the PN algorithm determines that $z(\alpha) = (5,0)^\top$, $x(\alpha) = (5,-25)^\top$, and $\|F_{\mathbf{B}}(x(\alpha))\| = 0$, so that $z(\alpha)$ solves the MCP.

2.3. Smooth. We first describe the method that is implemented in the code SMOOTH [12] in the context of nonlinear complementarity problems and then show in the next section how we modify this to generate a new crash technique for MCP.

The algorithm is based on the mapping H defined in (13). Since this mapping is nonsmooth, Newton's method is not directly applicable. Instead of applying Newton's method on the pieces of smoothness of the projection operator, the nonsmooth function is smoothed, replacing the projection operator $\pi_{\mathbf{B}}(\cdot)$ by a function $P^\beta(\cdot)$. The functions P^β converge pointwise to $\pi_{\mathbf{B}}$ as $\beta \to \infty$. Effectively, the nonsmooth map H is replaced by

$$H^\beta(z) := z - P^\beta(z - F(z))$$

where $P^\beta : \mathbf{R}^n \to \mathbf{R}^n$ is defined by

$$P_i^\beta(z) := z_i - \frac{1}{\beta}\log(1 + \exp(-\beta z_i)).$$

The function P^β is chosen (componentwise) for the nonlinear complementarity problem as the integral of the sigmoid function. In the case of finite lower and upper bounds on a variable, a reformulation using (12) is used. No smoothing at all is performed when the corresponding variable is free. Let $S^\beta(z)$ be a diagonal matrix whose diagonal entries are given by

$$(S^\beta(z))_{ii} = \frac{\partial P_i^\beta}{\partial z_i}(z_i - F_i(z)).$$

To apply Newton's method, H^β is linearized about the point z^k, and d^k is defined as the difference between the solution of the linearization and z^k. It is easy to see that d^k satisfies

$$z^k - P^\beta(z^k - F(z^k)) + (I - S^\beta(z^k) + S^\beta(z^k)\nabla F(z^k))d^k = 0 \tag{20}$$

The code SMOOTH solves this system and applies a standard Armijo line search with merit function $H^\beta(z)^\top H^\beta(z)$. The parameter β is iteratively updated as

$$\beta = \max\{\beta, \sqrt{n}/\left\|H(z^k)\right\|\}$$

and replaced by $\sqrt{\beta}$ if $\beta < 1$.

The default version of the code SMOOTH uses the PN crash technique outlined in the previous section (without the proximal perturbation). The results that are reported under the heading PN(2) are precisely those used in [12] and are given below: $\alpha_{\min} = 2^{-12}$, $\sigma = 0.05$, $n_{\min} = 10$, $k_{\max} = 50$, $\delta_{\max} = \infty$ and $\rho_{\min} = 0$. Note that this "crash technique" does not terminate when the active set settles down but continues to try to solve the problem even in this case.

2.4. Projected Smooth Directions. We observed that the projected Newton crash technique considerably improved the SMOOTH algorithm, but that there were several instances in which the smoothing enabled a hard problem to be solved. We outline here a hybrid technique that modifies the search direction using smoothing, but maintains a smaller system to be solved at each step of the crash phase.

The main advantage of the projected Newton crash technique was the reduced sized system that is solved at every iteration. In order to generate a similar system, with smoothing, we examine the equation (20) in more detail. The first observation is that the constant term $P^\beta(z^k - F(z^k))$ is an approximation to $\pi_{\mathbf{B}}(z^k - F(z^k))$ so we replace it with the exact term. Since our projected path search (11) is used with the merit function $\|F_{\mathbf{B}}(x(\alpha)\|$, the fact that this approximation may stop d^k being a descent direction for $\left\|H^\beta\right\|$ is irrelevant. We use a slightly different form of smoothing to that given in Section 2.3, namely one that results in

$$(S^\beta(z))_{ii} = \frac{1 - \exp\left[\beta(\ell_i - u_i)\right]}{(1 - \exp\left[\beta(\ell_i - z_i + F_i(z))\right])(1 - \exp\left[\beta(z_i - F_i(z) - u_i)\right])}$$

The advantage of this formulation is that no special cases need to be considered; the instances where ℓ_i or u_i are infinite are taken care of automatically. We now consider the indices $i \in \mathcal{A}$. For such indices, the modified constant term in (20) is now exactly 0, and when $\beta \to \infty$ the corresponding value of $(S^\beta(z))_{ii}$ tends to 0, resulting in $d_i^k = 0$. The projected Newton technique takes such a step automatically, and for the projected smooth crash technique this is a very good approximation of the step that would be calculated by (20). Thus, we set $d_i^k = 0$ for all $i \in \mathcal{A}$. The remaining components of d_i^k are calculated from the smoothing equation:

$$z_{\mathcal{I}}^k - \pi_{\mathbf{B}}(z^k - F(z^k))_{\mathcal{I}} + (I - S_{\mathcal{I}\mathcal{I}}^\beta(z^k) + S_{\mathcal{I}\mathcal{I}}^\beta(z^k)\nabla F_{\mathcal{I}\mathcal{I}}(z^k))d_{\mathcal{I}}^k = 0.$$

The crash technique that uses this direction is called PS. The parameter settings used in our computational tests for PS were identical to those in PN. The smoothing parameter β is iteratively updated as

$$\beta = \max\{\beta, \sqrt{n}/\left\|F_{\mathbf{B}}(x(\alpha))\right\|\}$$

and replaced by $\sqrt{\beta}$ if $\beta < 1$. The proximal perturbation strategy used was also identical to that described in Section 2.2.

3. Computational Results. Computational tests of the various crash techniques discussed in this paper have been performed using the GAMS [9] modeling system. The models used were taken from the model library distributed with GAMS and from MCPLIB [16], a growing collection of models available via anonymous ftp from `ftp.cs.wisc.edu://math-prog/mcplib/`. All the codes tested are written in C and linked to GAMS using the CPLIB [18] interface library. Furthermore, the same linear algebra subroutines are used throughout. All tests were done using a SUN SPARC 10/51 with 96 Mb of RAM, allowing a meaningful comparison of solution times between algorithms.

In our tests, the same runs (a solve of a model from a particular starting point) were carried out using the PATH solver without benefit of a crash phase (heading NONE in the tables) and each of the following crash techniques; PG (Section 2.1), PGT (Section 2.1), PN (Section 2.2), and PS (Section 2.4). The same runs were also performed using SMOOTH without a crash phase and using different parameters for PN(2) (see Section 2.3). Tables 1 and 2 have been constructed in order to compress the results of all these runs, and allow us to draw some conclusions from our computational study. Detailed results for the individual runs can be found in Appendix A. The abbreviations used for the column headings in all these tables are precisely those given above.

TABLE 1
Costs and Benefits of Crash Techniques.

	Crash Technique				
metric	PG	PGT	PN	PS	PN(2)
very beneficial	7%	8%	40%	37%	33%
beneficial	13%	10%	46%	40%	47%
not costly	60%	66%	81%	67%	95%
not very costly	78%	85%	85%	76%	97%

In order to gauge the effectiveness (i.e., the costs and benefits) of using a particular crash technique, we compare the performance of a base algorithm with and without a crash phase, using the following metrics:

beneficial: We say a crash technique P that requires time T_P to complete a run is "beneficial" compared to completing a run without a crash phase in time T_N if $T_P \leq \frac{3}{4} T_N$, and "very beneficial" if $T_P \leq \frac{1}{2} T_N$.

not costly: We say a crash technique P is "not costly" compared to having no crash phase if $T_P \leq \frac{4}{3} T_N$, and "not very costly" if $T_P \leq 2\, T_N$.

Since this method of reporting is somewhat dependent on the base algorithm chosen, we give results for crash techniques applied to both PATH and SMOOTH.

Table 1 shows that the projected Newton based crash procedures are beneficial nearly half of the time, and in most of these cases they improved performance substantially. However, one of the design goals for a crash technique should be to improve performance on some or most of the problems without degrading performance on any of them. While this goal was not entirely reached, the PN technique was "not very costly" compared to the base PATH algorithm 85% of the time. It can be seen in the Appendix that most of the models on which it performed poorly are those for which the model size and run times are quite small. Compared to the base SMOOTH algorithm, using PN(2) was "not very costly" 97% of the time, and also gave substantial gains in a large portion of the runs.

While Table 1 shows comparisons between crash techniques, Table 2 compares the different combinations of crash techniques and base algorithms used. The more general column headings indicate the base algorithm, while the individual column headings indicate what crash technique was applied. In computing the cumulative time required for all the runs with a given algorithm, one must decide how to assign times to algorithms for failed runs. We chose the maximum times required for any successful algorithm on the run in question. In addition to cumulative time, the algorithms are compared on the following basis:

success: Success is achieved if a solution is computed.

competitive: We say the time T_P for crash procedure P is "competitive" with the time T_B taken by the *best algorithm* on that run if $T_P \leq 2\, T_B$, and "very competitive" if $T_P \leq \frac{4}{3} T_B$.

The data in Table 2 clearly shows that the projected Newton crash procedures are superior to the projected gradient ones in the overall success rate, the improvement over the base algorithm, and in the number of runs for which the algorithm's solution time is competitive with the best time obtained. The cumulative times required to complete all the runs also indicate that the Newton techniques did well. What the tables do not show is the dramatic improvement in time obtained on some of the larger models, where speedups on the order of 100 were achieved by each of the projected

TABLE 2
Algorithm Comparison

	PATH +					SMOOTH +	
metric	NONE	PG	PGT	PN	PS	NONE	PN(2)
very comp.	46%	15%	18%	66%	43%	18%	52%
competitive	51%	37%	43%	76%	62%	35%	61%
success	88%	77%	88%	96%	91%	78%	86%
total time	11221	10386	11296	692	819	9235	985

Newton crash techniques.

In comparing the PG and PN techniques, we see that the higher cost of each Newton iteration was fully justified by the quality of the search direction computed. The gradient directions, despite their low cost, were of limited utility in speeding up the solution process. One of the reasons for this is the sparsity of the Jacobian matrices involved. For a sparse matrix, doing a function-only evaluation in GAMS may take as much as 30 – 40% of the time required to evaluate both the function and its Jacobian. If the computation of the Newton direction from the reduced system of equations is similarly inexpensive, the difference in cost between the two types of iteration will be small enough to make the Newton direction most attractive.

Close inspection of the Appendix reveals cases where the combination of crash phase with the base algorithm fails but the base algorithm alone succeeds. This is due to the fact that every crash technique forces monotonic decrease in $\|F_{\mathbf{B}}(\cdot)\|$ and hence can converge to a local minimum of this merit function. In contrast, the base algorithm may escape such local minima using a combination of the watchdog and nonmonotone path search schemes. It is clear from our computational results that this phenomenon is unlikely to occur with the PN approach. In fact, it is much more likely that this approach can modify the initial point and move it into an appropriate domain of convergence for the base algorithm.

Given the success of the projected Newton crash technique in improving the basic PATH algorithm, one is led to ask whether the PN procedure could not be used as a solver in its own right. In order to test this, we ran all the models with PN procedure only. While this technique was quite fast (339 seconds cumulative, not including failed runs), it was not very robust, as it solved only 60% of the models. Thus, we see that while the basic PATH algorithm lacks some of the speed of the PN technique, it more than compensates for this in increased robustness.

4. Conclusions. We feel that the computational results presented here demonstrate the potential value of crash techniques applied to both pivotal and non-pivotal complementarity codes. Of the techniques considered, a projected Newton crash technique (PN) is the most effective. This crash technique results in a more robust solution process and significant overall gains in solution times. For this reason, it is the current algorithm default for GAMS/PATH, the commercial release of the PATH solver, and the default choice for SMOOTH as well.

The increase in robustness obtained using the PN technique is one of the chief advantages of that scheme, as practitioners value robustness more than speed. This gain in robustness is brought about largely by the perturbation techniques used in the PN technique in the rank deficient case. Similar techniques could also be incorporated into the standard PATH algorithm. This is a subject of current research.

Interior point and smoothing approaches circumvent the active set problem by replacing the active set identification problem by some topological difficulty, typically a penalty function or smoothing approximation. This has proven successful in many cases, although some skill is required in choosing appropriate parameters and dealing with ill-conditioning. However, such methods are typically outperformed by active set methods when a good approximation to the active set exists, for example with restarts, perturbation of previous problems or good starting points. Furthermore, some applications exploit the nature of a basic solution. Thus, there is a growing literature in linear programming [1, 29] related to "crossover". This is the technique of taking an approximation generated by an interior point method and identifying a "near-optimal" active set for constructing an advanced basis in a pivotal based technique. It is certainly possible that interior point methods or smoothing methods may be used as alternative crash procedures for mixed complementarity codes in the future. However, the initial active set is currently determined just using the point returned by the crash technique. Appropriate modifications that identify "almost active" constraints would be required if an interior point crossover were to be used. This is also the subject of current research.

REFERENCES

[1] A. B. Berkelaar, B. Jansen, K. Roos, and T. Terlaky, *Basis and tripartition identification for quadratic programming and linear complementarity problems*, tech. rep., Department of Econometrics and Operations Research, Erasmus University, Rotterdam, The Netherlands, 1996.

[2] D. P. Bertsekas, *On the Goldstein-Levitin-Poljak gradient projection algorithm*, IEEE Transactions on Automatic Control, AC-21 (1976), pp. 174–184.

[3] D. P. Bertsekas, *Projected Newton methods for optimization problems with simple constraints*, SIAM Journal on Control and Optimization, 20 (1982), pp. 221–246.

[4] D. P. Bertsekas and E. M. Gafni, *Projection methods for variational inequalities with application to the traffic assignment problem*, Mathematical Programming Study, 17 (1982), pp. 139–159.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[6] S. C. Billups, *Algorithms for Complementarity Problems and Generalized Equations*, PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, Aug. 1995.

[7] S. C. Billups, S. P. Dirkse, and M. C. Ferris, *A comparison of large scale mixed complementarity problem solvers*, Computational Optimization and Applications, forthcoming, (1996).

[8] S. C. Billups and M. C. Ferris, *QPCOMP: A quadratic program based solver for mixed complementarity problems*, Mathematical Programming, forthcoming, (1996).

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

[10] J. V. Burke and J. J. Moré, *On the identification of active constraints*, SIAM Journal on Numerical Analysis, 25 (1988), pp. 1197–1211.

[11] P. H. Calamai and J. J. Moré, *Projected gradient methods for linearly constrained problems*, Mathematical Programming, 39 (1987), pp. 93–116.

[12] C. Chen and O. L. Mangasarian, *A class of smoothing functions for nonlinear and mixed complementarity problems*, Computational Optimization and Applications, 5 (1996), pp. 97–138.

[13] A. R. Conn, N. I. M. Gould, and P. L. Toint, *LANCELOT: A Fortran package for Large–Scale Nonlinear Optimization (Release A)*, no. 17 in Springer Series in Computational Mathematics, Springer Verlag, Heidelberg, Berlin, 1992.

[14] CPLEX Optimization Inc., *Using the CPLEX(TM) Linear Optimizer and CPLEX(TM) Mixed Integer Optimizer (Version 2.0)*, Incline Village, Nevada, 1992.

[15] T. De Luca, F. Facchinei, and C. Kanzow, *A semismooth equation approach to the solution of nonlinear complementarity problems*, Preprint 93, Institute of Applied Mathematics, University of Hamburg, Hamburg, Germany, 1995.

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

[17] ———, *The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems*, Optimization Methods and Software, 5 (1995), pp. 123–156.

[18] S. P. Dirkse, M. C. Ferris, P. V. Preckel, and T. Rutherford, *The GAMS callable program library for variational and complementarity solvers*, Mathematical Programming Technical Report 94-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.

[19] B. C. Eaves, *On the basic theorem of complementarity*, Mathematical Programming, 1 (1971), pp. 68–87.

[20] J. Eckstein and M. C. Ferris, *Operator splitting methods for monotone linear complementarity problems*, TMC 239, Thinking Machines Corporation, Cambridge, MA 02142, 1992.

[21] A. Fischer and C. Kanzow, *On finite termination of an iterative method for linear complementarity problems*, Preprint MATH–NM–10–1994, Institute for Numerical Mathematics, Technical University of Dresden, Dresden, Germany, 1994.

[22] M. Fukushima, *Equivalent differentiable optimization problems and descent methods for asymmetric variational inequality problems*, Mathematical Programming, 53 (1992), pp. 99–110.

[23] S. A. Gabriel and J. J. Moré, *Smoothing of mixed complementarity problems*, Preprint MCS-P541-0995, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1995.

[24] C. B. Garcia and W. I. Zangwill, *Pathways to Solutions, Fixed Points, and Equilibria*, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1981.

[25] P. T. Harker and B. Xiao, *Newton's method for the nonlinear complementarity problem: A B–differentiable equation approach*, Mathematical Programming, 48 (1990), pp. 339–358.

[26] H. Jiang and L. Qi, *A new nonsmooth equations approach to nonlinear complementarity problems*, SIAM Journal on Control and Optimization, forthcoming, (1996).

[27] N. H. Josephy, *Newton's method for generalized equations*, Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.

[28] T. Larsson and M. Patriksson, *A class of gap functions for variational inequalities*, Mathematical Programming, 64 (1994), pp. 53–79.

[29] N. Megiddo, *On finding primal– and dual–optimal bases*, ORSA Journal on Computing, 3 (1991), pp. 63–65.

[30] B. A. Murtagh and M. A. Saunders, *MINOS 5.0 user's guide*, Technical Report SOL 83.20, Stanford University, Stanford, California, 1983.

[31] J. S. Pang, *Newton's method for B–differentiable equations*, Mathematics of Operations Research, 15 (1990), pp. 311–341.

[32] J. S. Pang and S. A. Gabriel, *NE/SQP: A robust algorithm for the nonlinear complementarity problem*, Mathematical Programming, 60 (1993), pp. 295–338.

[33] D. Ralph, *Global convergence of damped Newton's method for nonsmooth equations, via the path search*, Mathematics of Operations Research, 19 (1994), pp. 352–389.

[34] S. M. Robinson, *Mathematical foundations of nonsmooth embedding methods*, Mathematical Programming, 48 (1990), pp. 221–229.

[35] ———, *Normal maps induced by linear transformations*, Mathematics of Operations Research, 17 (1992), pp. 691–714.

[36] T. F. Rutherford, *MILES: A mixed inequality and nonlinear equation solver*. Working Paper, Department of Economics, University of Colorado, Boulder, 1993.

[37] H. E. Scarf, *The approximation of fixed points of a continuous mapping*, SIAM Journal on Applied Mathematics, 15 (1967), pp. 1328–1343.

[38] M. J. Todd, *Computation of Fixed Points and Applications*, vol. 124 of Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Heidelberg, 1976.

[39] B. Xiao and P. T. Harker, *A nonsmooth Newton method for variational inequalities: I: Theory*, Mathematical Programming, 65 (1994), pp. 151–194.

Table 3
Solution Times - PATH crash technique

model	NONE	PG	PGT	PN	PS
asean9a 1	18.82	24.46	25.36	18.74	16.25
bert_oc 1	41.16	41.21	39.73	3.06	5.84
bert_oc 2	55.04	61.54	85.92	3.21	5.00
bert_oc 3	43.83	47.76	46.08	2.51	3.15
bert_oc 4	53.14	56.56	86.94	2.60	3.06
bratu 1	436.40	442.45	448.86	148.09	122.42
cafemge 1	2.08	3.41	2.25	0.38	1.26
cafemge 2	0.33	0.45	0.39	0.33	0.51
cammcp 1	0.31	0.42	fail	0.39	0.81
co2mge 2	1.18	1.54	2.68	0.58	1.40
co2mge 3	0.34	0.55	0.40	0.30	0.37
co2mge 4	0.43	0.68	0.39	0.42	0.35
co2mge 5	0.43	0.78	0.54	0.39	0.37
co2mge 6	0.49	0.79	0.74	0.55	1.15
co2mge 7	0.66	0.90	1.49	0.59	0.64
dmcmge 1	fail	fail	fail	3.90	2.78
dmcmge 2	fail	fail	fail	0.57	2.41
ehl_kost 1	4.58	fail	9.35	5.01	19.37
ehl_kost 2	7.55	fail	11.60	17.05	30.32
ehl_kost 3	141.74	fail	151.03	9.76	46.48
ers82mcp 1	0.47	0.59	fail	0.47	0.38
eta2100 1	6.38	18.50	7.26	4.74	17.59
finmge 2	2.41	2.51	2.73	2.02	3.49
finmge 3	0.87	1.26	1.33	0.97	1.36
finmge 4	1.19	2.64	1.62	2.08	4.97
finmge 5	1.16	1.52	1.76	1.24	1.69
gemmcp 1	0.26	0.70	0.56	0.27	0.22
gemmcp 2	0.05	0.40	0.37	0.05	0.04
gemmcp 3	0.92	0.98	1.30	0.91	1.04
gemmcp 4	0.86	1.23	0.93	0.90	1.08
gemmcp 5	0.68	1.04	1.07	0.86	0.86
gemmge 2	4.87	4.87	7.75	3.00	5.67
gemmge 3	2.62	2.69	2.87	2.34	fail
gemmge 4	2.60	3.10	3.15	2.23	4.83
gemmge 5	9.49	12.66	9.74	8.77	3.30
hansmcp 1	0.38	0.32	0.32	2.14	fail
hansmge 1	0.18	0.22	0.31	0.28	0.40
hydroc20 1	7.13	7.06	12.09	0.54	0.49
jmu 1	fail	fail	fail	130.93	111.05
mr5mcp 1	0.66	fail	1.08	0.83	1.76

Table 4
Solution Times - PATH crash technique

model	NONE	PG	PGT	PN	PS
obstacle 1	20.15	10.05	15.12	2.97	3.14
obstacle 2	160.88	165.50	158.67	6.96	13.26
obstacle 3	80.07	66.06	76.97	6.09	8.83
obstacle 4	47.16	29.64	44.80	6.13	7.36
obstacle 5	71.45	14.04	54.14	8.02	10.95
obstacle 6	114.87	93.98	118.32	10.79	12.91
obstacle 7	92.10	64.22	93.59	7.61	10.00
obstacle 8	113.03	65.61	115.51	13.80	12.33
opt_cont 1	1.05	0.20	0.21	0.44	0.25
opt_cont127 1	246.48	209.45	253.35	8.36	8.79
opt_cont255 1	1303.73	1089.88	1322.91	18.76	19.21
opt_cont31 1	12.43	1.71	4.52	1.73	1.17
opt_cont511 1	7569.17	6540.97	7531.91	40.52	53.87
pgvon105 1	1.68	1.27	2.14	4.48	1.46
pgvon105 2	0.84	1.00	1.08	0.60	1.15
pgvon105 3	0.87	1.22	0.82	1.29	0.84
pgvon106 1	4.13	1.80	2.44	13.24	5.14
pgvon106 2	6.11	3.37	2.93	3.97	3.37
pgvon106 3	4.04	2.81	3.42	1.00	6.00
scarfbnum 1	0.07	fail	fail	0.37	0.74
scarfbnum 2	0.10	fail	0.31	0.51	0.68
scarfbsum 1	0.12	fail	0.12	6.00	0.49
scarfbsum 2	0.15	fail	0.17	2.37	fail
srtest1 1	0.05	0.25	0.28	0.04	0.04
srtest1 2	fail	fail	fail	38.75	16.10
taxmge 2	2.60	2.71	3.11	2.57	fail
taxmge 3	0.39	1.57	1.34	0.40	fail
taxmge 4	1.83	fail	9.02	13.68	13.07
taxmge 5	fail	fail	fail	fail	fail
taxmge 6	fail	fail	fail	fail	fail
taxmge 7	fail	fail	2.32	1.46	1.63
taxmge 8	fail	fail	1.56	1.35	1.58
taxmge 9	fail	fail	0.61	0.33	0.11
tobin 1	0.10	0.14	0.14	0.09	0.11
tobin 2	1.21	0.14	0.31	0.08	0.09
trade12 1	12.39	12.94	11.70	26.96	75.19
trade12 2	35.88	328.92	35.38	7.67	13.17
traffic 1	59.28	299.83	54.97	41.71	77.46
vonthmcp 1	0.62	0.90	0.56	fail	0.47
vonthmge 1	0.48	0.62	0.59	1.44	1.07

TABLE 5
Pre-processing Iterations - PATH crash technique

model	NONE	PG	PGT	PN	PS
asean9a 1	0	0	0	1	3
bert_oc 1	0	0	2	3	5
bert_oc 2	0	0	3	3	4
bert_oc 3	0	0	2	3	4
bert_oc 4	0	0	9	3	4
bratu 1	0	1	2	25	13
cafemge 1	0	2	0	1	23
cafemge 2	0	0	2	1	7
cammcp 1	0	2	fail	2	10
co2mge 2	0	0	2	1	7
co2mge 3	0	0	2	1	4
co2mge 4	0	0	2	1	4
co2mge 5	0	0	2	1	4
co2mge 6	0	0	3	1	4
co2mge 7	0	0	4	1	7
dmcmge 1	fail	fail	fail	1	0
dmcmge 2	fail	fail	fail	1	13
ehl_kost 1	0	fail	0	1	21
ehl_kost 2	0	fail	0	1	7
ehl_kost 3	0	fail	0	1	33
ers82mcp 1	0	0	fail	1	5
eta2100 1	0	1	2	1	50
finmge 2	0	1	0	1	6
finmge 3	0	0	2	1	4
finmge 4	0	2	2	1	3
finmge 5	0	1	2	1	5
gemmcp 1	0	0	2	1	1
gemmcp 2	0	0	2	0	0
gemmcp 3	0	1	2	1	5
gemmcp 4	0	1	2	1	5
gemmcp 5	0	0	2	1	4
gemmge 2	0	1	2	1	10
gemmge 3	0	1	2	1	fail
gemmge 4	0	1	2	1	2
gemmge 5	0	1	2	1	8
hansmcp 1	0	1	4	9	fail
hansmge 1	0	1	2	2	11
hydroc20 1	0	0	2	1	8
jmu 1	fail	fail	fail	0	0
mr5mcp 1	0	fail	0	1	16

Table 6
Pre-processing Iterations - PATH crash technique

model	NONE	PG	PGT	PN	PS
obstacle 1	0	6	3	10	11
obstacle 2	0	3	2	11	12
obstacle 3	0	3	5	10	12
obstacle 4	0	2	21	9	13
obstacle 5	0	4	6	6	5
obstacle 6	0	11	8	9	9
obstacle 7	0	11	5	9	10
obstacle 8	0	11	2	10	8
opt_cont 1	0	3	5	4	3
opt_cont127 1	0	2	3	4	4
opt_cont255 1	0	2	3	4	4
opt_cont31 1	0	4	10	4	3
opt_cont511 1	0	2	3	4	5
pgvon105 1	0	1	6	1	4
pgvon105 2	0	2	0	59	1
pgvon105 3	0	2	0	59	15
pgvon106 1	0	1	4	0	4
pgvon106 2	0	3	0	1	96
pgvon106 3	0	3	0	3	96
scarfbnum 1	0	fail	fail	1	50
scarfbnum 2	0	fail	4	1	49
scarfbsum 1	0	fail	0	1	16
scarfbsum 2	0	fail	0	1	fail
srtest1 1	0	1	0	0	0
srtest1 2	fail	fail	fail	1	0
taxmge 2	0	1	2	1	fail
taxmge 3	0	1	2	1	fail
taxmge 4	0	fail	2	1	3
taxmge 5	fail	fail	fail	fail	fail
taxmge 6	fail	fail	fail	fail	fail
taxmge 7	fail	fail	2	42	34
taxmge 8	fail	fail	2	42	34
taxmge 9	fail	fail	2	42	34
tobin 1	0	4	5	1	10
tobin 2	0	4	2	2	8
trade12 1	0	1	0	7	9
trade12 2	0	1	2	1	7
traffic 1	0	1	3	21	21
vonthmcp 1	0	2	0	fail	2
vonthmge 1	0	1	0	2	6

TABLE 7
Jacobian Evaluations - PATH crash technique

model	NONE	PG	PGT	PN	PS
asean9a 1	4	4	4	4	4
bert_oc 1	2	2	3	4	6
bert_oc 2	2	2	4	4	5
bert_oc 3	2	2	3	4	5
bert_oc 4	2	2	10	4	5
bratu 1	4	5	5	26	14
cafemge 1	33	43	31	7	24
cafemge 2	6	6	7	6	8
cammcp 1	5	5	fail	5	11
co2mge 2	14	14	20	7	12
co2mge 3	4	4	5	4	5
co2mge 4	5	5	6	5	5
co2mge 5	6	6	8	5	5
co2mge 6	7	7	9	7	10
co2mge 7	7	7	15	8	8
dmcmge 1	fail	fail	fail	18	13
dmcmge 2	fail	fail	fail	6	14
ehl_kost 1	6	fail	7	6	22
ehl_kost 2	8	fail	9	19	18
ehl_kost 3	68	fail	70	11	34
ers82mcp 1	6	6	fail	6	6
eta2100 1	16	32	16	13	56
finmge 2	5	6	5	7	12
finmge 3	4	4	5	4	5
finmge 4	5	6	6	8	8
finmge 5	5	6	6	5	6
gemmcp 1	2	2	3	2	2
gemmcp 2	1	1	2	1	1
gemmcp 3	6	7	7	6	6
gemmcp 4	6	7	6	6	6
gemmcp 5	5	5	6	5	5
gemmge 2	12	12	20	7	11
gemmge 3	6	7	7	6	fail
gemmge 4	7	8	8	6	9
gemmge 5	25	26	26	21	9
hansmcp 1	28	29	26	38	fail
hansmge 1	4	5	5	8	12
hydroc20 1	22	24	30	9	9
jmu 1	fail	fail	fail	25	25
mr5mcp 1	7	fail	7	7	17

Table 8
Jacobian Evaluations - PATH crash technique

model	NONE	PG	PGT	PN	PS
obstacle 1	3	3	5	11	12
obstacle 2	3	4	4	12	13
obstacle 3	2	3	8	11	13
obstacle 4	2	3	23	11	14
obstacle 5	2	3	9	7	7
obstacle 6	2	3	10	10	10
obstacle 7	2	3	6	10	11
obstacle 8	2	3	3	11	9
opt_cont 1	2	3	7	6	4
opt_cont127 1	2	3	4	6	5
opt_cont255 1	2	3	4	6	5
opt_cont31 1	2	3	12	6	4
opt_cont511 1	2	3	4	6	6
pgvon105 1	26	19	34	24	22
pgvon105 2	14	15	14	9	20
pgvon105 3	14	22	14	14	16
pgvon106 1	47	36	38	fail	69
pgvon106 2	60	44	42	fail	44
pgvon106 3	32	40	44	20	63
scarfbnum 1	5	fail	fail	14	55
scarfbnum 2	5	fail	14	15	54
scarfbsum 1	4	fail	5	39	20
scarfbsum 2	4	fail	5	18	fail
srtest1 1	1	2	1	1	1
srtest1 2	fail	fail	fail	29	12
taxmge 2	11	12	12	11	fail
taxmge 3	2	7	6	2	fail
taxmge 4	7	fail	11	11	14
taxmge 5	fail	fail	fail	fail	fail
taxmge 6	fail	fail	fail	fail	fail
taxmge 7	fail	fail	9	7	7
taxmge 8	fail	fail	8	6	7
taxmge 9	fail	fail	2	2	1
tobin 1	10	12	13	9	11
tobin 2	19	14	13	9	9
trade12 1	7	8	7	17	32
trade12 2	18	126	19	6	8
traffic 1	7	24	11	28	28
vonthmcp 1	12	12	12	fail	9
vonthmge 1	14	16	14	22	17

Table 9
Solution data - SMOOTH

	Time		Jac		PP_Its	
model	NONE	PN(2)	NONE	PN(2)	NONE	PN(2)
asean9a 1	23.22	18.42	4	4	0	3
bert_oc 1	4.96	3.15	4	4	0	3
bert_oc 2	61.03	3.36	41	4	0	3
bert_oc 3	6.86	2.88	5	4	0	3
bert_oc 4	13.78	2.52	11	4	0	3
bratu 1	79.36	151.62	7	26	0	25
cafemge 1	0.96	0.48	15	8	0	7
cafemge 2	0.58	0.33	8	6	0	5
cammcp 1	fail	0.38	fail	5	fail	4
co2mge 2	fail	0.44	fail	6	fail	5
co2mge 3	2.99	0.27	19	4	0	3
co2mge 4	0.95	0.36	7	5	0	4
co2mge 5	0.97	0.33	7	5	0	4
co2mge 6	fail	2.18	fail	13	fail	9
co2mge 7	0.73	0.56	8	8	0	7
dmcmge 1	23.04	5.61	84	23	0	10
dmcmge 2	206.18	0.87	447	6	0	5
ehl_kost 1	8.93	4.42	11	6	0	5
ehl_kost 2	21.31	11.80	18	12	0	11
ehl_kost 3	55.62	96.21	36	55	0	19
ers82mcp 1	0.46	0.43	5	6	0	5
eta2100 1	fail	48.73	fail	160	fail	50
finmge 2	5.83	6.81	13	13	0	0
finmge 3	1.38	0.85	4	4	0	3
finmge 4	11.37	10.56	20	20	0	0
finmge 5	4.19	1.53	12	6	0	5
gemmcp 1	0.06	0.27	1	2	0	1
gemmcp 2	0.05	0.06	1	1	0	0
gemmcp 3	1.00	0.85	5	6	0	5
gemmcp 4	0.82	0.83	4	6	0	5
gemmcp 5	0.81	0.69	4	5	0	4
gemmge 2	3.40	4.21	6	6	0	0
gemmge 3	4.04	2.23	7	6	0	5
gemmge 4	2.57	2.24	5	6	0	5
gemmge 5	3.25	3.11	6	7	0	6
hansmcp 1	0.12	0.12	8	8	0	0
hansmge 1	0.65	0.66	13	13	0	0
hydroc20 1	fail	0.40	fail	9	fail	8
jmu 1	127.77	130.81	181	181	0	0
mr5mcp 1	fail	0.63	fail	7	fail	6

TABLE 10
Solution data - SMOOTH

	Time		Jac		PP_Its	
model	NONE	PN(2)	NONE	PN(2)	NONE	PN(2)
obstacle 1	13.36	3.19	6	11	0	10
obstacle 2	26.13	6.89	10	12	0	11
obstacle 3	20.08	6.42	7	11	0	10
obstacle 4	18.64	5.73	7	11	0	10
obstacle 5	13.68	6.69	6	7	0	6
obstacle 6	19.59	10.78	7	10	0	9
obstacle 7	22.08	8.20	7	10	0	9
obstacle 8	18.18	13.02	6	11	0	10
opt_cont 1	3.92	0.42	10	6	0	5
opt_cont127 1	152.20	8.26	17	6	0	5
opt_cont255 1	253.77	17.77	14	6	0	5
opt_cont31 1	18.71	1.73	10	6	0	5
opt_cont511 1	fail	42.80	fail	6	fail	5
pgvon105 1	fail	fail	fail	fail	fail	fail
pgvon105 2	fail	fail	fail	fail	fail	fail
pgvon105 3	fail	fail	fail	fail	fail	fail
pgvon106 1	132.92	135.56	482	482	0	0
pgvon106 2	4.64	5.12	36	37	0	1
pgvon106 3	10.34	10.38	49	49	0	0
scarfbnum 1	0.32	0.31	20	20	0	0
scarfbnum 2	0.33	0.42	24	24	0	0
scarfbsum 1	0.23	0.26	11	11	0	0
scarfbsum 2	0.64	0.52	24	24	0	0
srtest1 1	0.04	0.05	1	1	0	0
srtest1 2	1.47	1.56	6	6	0	0
taxmge 2	fail	fail	fail	fail	fail	fail
taxmge 3	fail	fail	fail	fail	fail	fail
taxmge 4	6.56	fail	21	fail	0	fail
taxmge 5	fail	fail	fail	fail	fail	fail
taxmge 6	fail	fail	fail	fail	fail	fail
taxmge 7	fail	fail	fail	fail	fail	fail
taxmge 8	fail	fail	fail	fail	fail	fail
taxmge 9	fail	fail	fail	fail	fail	fail
tobin 1	0.15	0.20	11	12	0	1
tobin 2	0.15	0.10	10	12	0	11
trade12 1	47.65	31.72	24	18	0	1
trade12 2	14.05	8.28	7	6	0	5
traffic 1	79.62	83.76	37	37	0	0
vonthmcp 1	4.74	5.58	80	80	0	0
vonthmge 1	17.97	17.34	278	278	0	0