, , 1–19 ()

A Comparison of Large Scale Mixed Complementarity Problem Solvers*

STEPHEN C. BILLUPS sbillups@carbon.cudenver.edu
Mathematics Department, University of Colorado, Denver, Colorado 80217

STEVEN P. DIRKSE steve@gams.com
GAMS Development Corporation, Washington, DC 20007

MICHAEL C. FERRIS ferris@cs.wisc.edu
Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

Received Oct 1, 1995; Revised March 22, 1996

Editor:

Abstract. This paper provides a means for comparing various computer codes for solving large scale mixed complementarity problems. We discuss inadequacies in how solvers are currently compared, and present a testing environment that addresses these inadequacies. This testing environment consists of a library of test problems, along with GAMS and MATLAB interfaces that allow these problems to be easily accessed. The environment is intended for use as a tool by other researchers to better understand both their algorithms and their implementations, and to direct research toward problem classes that are currently the most challenging. As an initial benchmark, eight different algorithm implementations for large scale mixed complementarity problems are briefly described and tested with default parameter settings using the new testing environment.

Keywords: complementarity problems, variational inequalities, computation, algorithms

1. Introduction

In recent years, a considerable number of new algorithms have been developed for solving large scale mixed complementarity problems. Many of these algorithms appear very promising theoretically, but it is difficult to understand how well they will work in practice. Indeed, many of the papers describing these algorithms are primarily theoretical papers and include only very minimal computational results. Even with extensive testing, there are inadequacies in the way the results are reported, which makes it difficult to compare one approach against another.

The purpose of this paper is to describe a testing environment for evaluating the strengths and weaknesses of various codes for solving large scale mixed complementarity problems. We believe that the environment is ideally suited for the computational study, development, and comparison of algorithm implementations. The careful description and documentation of the environment given here should

* This material is based on research supported by National Science Foundation Grant CCR-9157632 and the Air Force Office of Scientific Research Grant F49620-94-1-0036.

help algorithm designers focus their developmental efforts toward practical and useful codes. To exhibit its intended usage, we benchmark eight different algorithm implementations for large scale mixed complementarity problems with the new testing environment. At the same time, we intend to provide a convenient mechanism for modelers to provide new and challenging problems for use in solver comparison. As an added benefit, we believe the environment will help modelers determine which code best fits their needs.

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

$$\begin{aligned} \ell_i < z_i < u_i \;&\Rightarrow\; F_i(z) = 0 \\ z_i = \ell_i \;&\Rightarrow\; F_i(z) \geq 0 \\ z_i = u_i \;&\Rightarrow\; F_i(z) \leq 0. \end{aligned}$$

We will use the notation $\mathbf{B}$ to represent the set $[\ell, u]$.

Several special cases of this formulation are immediately obvious. For example, if $\ell \equiv -\infty$ and $u \equiv +\infty$ then the last two implications are vacuous and MCP is the problem of determining $z \in \mathbf{R}^n$ such that $F(z) = 0$.

As another example, the Karush-Kuhn-Tucker conditions for nonlinear programs of the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & g(x) \leq 0 \end{array}$$

are given by

$$\begin{array}{l} \nabla f(x) + \lambda \nabla g(x) = 0, \\ g(x) \leq 0,\ \lambda \geq 0,\ \lambda^T g(x) = 0. \end{array}$$

These are easily recast as an MCP by setting

$$z = \begin{bmatrix} x \\ \lambda \end{bmatrix},\ F(z) = \begin{bmatrix} \nabla f(x) + \lambda \nabla g(x) \\ -g(x) \end{bmatrix},\ \mathbf{B} = \mathbf{R}^n \times \mathbf{R}^m_+.$$

Here $\mathbf{R}^m_+$ represents the nonnegative orthant of $\mathbf{R}^m$. Many problems in economic equilibrium theory can be cast as MCPs and an overview of how this is accomplished is given in [31]. Other application areas are detailed in [7], [12]. There has been much recent interest in less traditional applications of the complementarity framework. Some of these are based on the generalized equation literature [28] that reformulates the MCP as $0 \in F(z) + N_{\mathbf{B}}(z)$. Here $N_{\mathbf{B}}(z)$ is the classical normal cone to the set $\mathbf{B}$ at the point z defined by

$$N_{\mathbf{B}}(z) := \left\{ y \mid y^T(x - z) \leq 0 \ \forall x \in \mathbf{B} \right\},$$

if $z \in \mathbf{B}$ and is empty otherwise.

Nonlinear complementarity problems appeared in the literature in [5]. The first algorithms for these problems were based on simplicial labeling techniques originally due to Scarf [32]. Extensions of these algorithms led to fixed point schemes [18], [33]. Newton techniques [8], [22], [30] that are based on successive linearization of the nonlinear problem have proven very useful for solving these problems, although their convergence analysis is less satisfactory than the fixed point theory. Recent extensions have looked at reformulating the nonlinear complementarity problem as a system of nonsmooth nonlinear equations and solving these using a damped Newton or Gauss-Newton approach [6], [8], [10], [11], [13], [16], [19], [20], [21], [23], [24], [25], [26], [27], [29], [34], [35].

We are concerned in this paper with computational testing and comparison of such algorithms. We see several problems with the current state of affairs in the way solvers are developed and compared.

1. Codes are tweaked to solve particular problems, with different choices of control parameters being used to solve different problems. This is contrary to how solvers are used in practice. In general, modelers are not interested in parameter adjustment; instead, they usually run codes only with default options. A good code will have a set of default parameters that performs well on most problems.

2. Even when a consistent set of control parameters is used, codes are developed and tuned using the same limited set of test problems for which computational results are reported. Consequently, the results do not give a fair picture of how the codes might behave on other problems. Enlarging the test suite and adding real world problems alleviates some of these difficulties.

3. There is no clear understanding of what makes problems difficult. Thus, test cases reported do not necessarily reflect the various difficulties that can cause algorithms to fail. As a result, it is extremely difficult for a modeler to determine which algorithm will work best for his particular class of problems.

4. The majority of papers written are theoretical in nature and provide computational results only for naive implementations of the algorithms. While this can exhibit the potential of a particular approach, it is inadequate for evaluating how an algorithm will work in practice. Instead, computational results need to be reported for sophisticated implementations of the algorithms. In particular, algorithm specific scaling, preprocessing or heuristics are crucial for improved robustness and developer supplied *default settings* should be used in all solver comparisons.

5. Test problems do not reflect the interests of users with real-world applications. Thus, algorithms are developed which are good at solving "toy" problems, but are not necessarily good at solving problems of practical importance.

These problems in the way solvers are currently tested result in two major deficiencies in the usefulness of test results. First, the reported results are inadequate for modelers to determine which codes will be most successful for solving their problems. Second, it is difficult for algorithm developers to determine where additional research needs to be directed.

In order to overcome these difficulties, this paper proposes that a testing environment for large scale mixed complementarity problems be developed. The goals of this environment are again twofold: first, it should provide a means of more accurately evaluating the strengths and weaknesses of various codes, and second, it should help direct algorithm developers toward addressing the issues of greatest importance. A preliminary version of such an environment is described in Section 2 and was used to generate the computational results reported in Section 4. A brief description of each of the codes tested is provided in Section 3.

2. Testing Environment

This section describes a testing environment that aims to correct many of the problems discussed in the introduction concerning how codes are developed and tested. This environment has four main components: a library of test problems, GAMS and MATLAB interfaces that allow these problems to be easily accessed, a tool for verifying the correctness of solutions, and some awk scripts for evaluating results.

2.1. Test Library

The centerpiece of the testing environment is a large publicly available library of test problems that reflects the interests of users with real-world applications, and that also includes problems having known types of computational difficulties. Many of these problems are contained in the standard GAMS distribution [3], while others are part of the expanding collection of problems called MCPLIB[7]. All of the problems that are used in this work are publicly available and can be accessed both from within the GAMS modeling system[3] and from within MATLAB[14].

Because most of the problems in the test library come from real-world applications, the library reflects, as much as possible, the needs of the user community. As this library has become more popular among code developers, we have observed an increased interest among modelers to contribute more and more challenging problems to the library. The motivation is simple: modelers want to encourage the development of codes capable of solving their most difficult problems.

We note that many of the problems contained in the test library are difficult for varying reasons. We believe that it is important to identify the characteristics that make problems hard. This is a daunting task; toward this end, we give an incomplete classification of the types of problem difficulties that may prove challenging for different algorithms.

1. Nonlinearity. We characterize the nonlinearity of a problem by how well a local linearization of the function models the original problem. One difficulty encoun-

tered in highly nonlinear problems is the presence of local minima of the underlying merit function that do not correspond to solutions. Several algorithms include features that allow them to escape such local minima, for example, perturbational schemes and nonmonotone watchdog procedures. Thus, we expect that certain algorithms will be more sensitive to the degree of nonlinearity than others.

2. Active Set Determination. For many problems, once the active set is determined, (that is, once we determine which variables are at their upper and lower bounds) the given algorithm is quick to converge. Thus, quick identification of the active set can greatly improve the performance of the algorithm. This seems to be particularly true for problems that are nearly linear.

3. Problem Size. Some algorithms may be better at exploiting problem structure than others, making them less sensitive to the size of the problem. One weakness of our current test suite is that it does not address the issue of size very well. We have attempted to include problems of reasonable size, but it is clear that the test library needs to be expanded in this area.

4. Sensitivity to Scaling. Our experience is that modelers, of necessity, tend to become very good at scaling their models so that relevant matrices are reasonably well-conditioned. Indeed, most of the problems in our model library are well scaled. However, models under development are often poorly scaled. Frequently, solutions are used to scale models properly and to aid in the model construction. Thus, sensitivity to scaling is quite important. In general it is very difficult to scale highly nonlinear functions effectively, so that an algorithm that is less sensitive to scaling may prove to be more practical for highly nonlinear problems.

5. Others. Several other problem characteristics have been proposed, but have not been well studied in the context of real models. These include monotonicity, multiple solutions, and singularity at the solution.

Tables 1 and 2 describe the problems that are included in the test library. Further documentation on these problems can be found in [31] and [7] respectively. Since the starting point can greatly influence the performance of an algorithm, the library includes multiple starting points for most problems. We note that many of the economic problems have the first starting point very close to a solution. This is the "calibration" point and is used by a modeler to test whether the model reproduces benchmark data. The following abbreviations are used when referring to the type of the problem:

MCP	General mixed complementarity problem
LMCP	Linear mixed complementarity problem
NCP	Nonlinear complementarity problem
LCP	Linear complementarity problem
MPSGE	General economic equilibrium problems defined with the MPSGE macro language
NE	Nonlinear equations
NLP	Optimality conditions of a nonlinear program

The tables also include a column labeled "other". In this column we have added some known characteristics of the problems. Thus "M" is entered in this column if the problem is known to be monotone. Similarly a digit "4" for example indicates the number of known solutions. If an "S" occurs in this column then the submatrix of the Jacobian corresponding to the "active constraints" is known to have condition number greater than 10^8 at a solution. The fact that one of these entries does not appear in the table only signifies that the authors do not know whether the problem has this particular characteristic.

2.2. Interfaces

To make the test library useful, two interfaces are provided that make the problems easily accessible both for testing of mature codes and for evaluating prototype algorithms.

The first interface is a means for programs to communicate directly with the GAMS modeling language [3]. For realistic application problems, we believe that the use of a modeling system such as AMPL[17] or GAMS is crucial. In earlier work with Rutherford[9], we developed the GAMS/CPLIB interface that provides simple routines to obtain function and Jacobian evaluations and recover problem data. This makes it easy to hook up any solver that is written in Fortran or C as a subsystem of GAMS. The advantages of using a modeling system are many; some of the most important advantages include automatic differentiation, easy data handling, architecture-independent interfaces between models and solvers, and the ability to extend models easily to answer new questions arising from solutions of current models. In addition, modeling languages provide a ready library of examples on which to test solvers. GAMS was chosen for our work instead of AMPL because it is a mature product with many users, resulting in the availability of many real-world problems.

While we believe that any mature code should be connected with a modeling language, we also feel that there should be an easier means for making the library of test problems available to prototype algorithms. The MATLAB interface described in [14] provides such a means. Using MATLAB, it is possible to quickly implement a prototype version of a new algorithm, which can be tested on the entire suite of test problems with the MATLAB interface. Thus, the test library can play an

Table 1. GAMSLIB Models

Model	Type	n	nnz	density	other
cafemge	MPSGE	101	900	8.82%	
cammcp	NCP	242	1621	2.77%	
cammge	MPSGE	128	1227	7.49%	
cirimge	MPSGE	9	33	40.74%	
co2mge	MPSGE	208	1463	3.38%	
dmcmge	MPSGE	170	1594	5.52%	
ers82mcp	MCP	232	1552	2.88%	
etamge	MPSGE	114	848	6.53%	
finmge	MPSGE	153	1915	8.18%	
gemmcp	MCP	262	2793	4.07%	
gemmge	MPSGE	178	3441	10.86%	
hansmcp	NCP	43	398	21.53%	
hansmge	MPSGE	43	503	27.20%	
harkmcp	NCP	32	131	12.79%	
harmge	MPSGE	11	60	49.59%	
kehomge	MPSGE	9	75	92.59%	3
kormcp	MCP	78	423	6.95%	
mr5mcp	NCP	350	1687	1.38%	
nsmge	MPSGE	212	1408	3.13%	
oligomcp	NCP	6	21	58.33%	
sammge	MPSGE	23	117	22.12%	
scarfmcp	NCP	18	150	46.30%	
scarfmge	MPSGE	18	181	55.86%	
shovmge	MPSGE	51	375	14.42%	
threemge	MPSGE	9	77	95.06%	
transmcp	LCP	11	34	28.10%	
two3mcp	NCP	6	29	80.56%	
unstmge	MPSGE	5	25	100.00%	
vonthmcp	NCP	125	760	4.86%	S
vonthmge	MPSGE	80	594	9.28%	
wallmcp	NE	6	25	69.44%	

Table 2. MCPLIB Models

Model	Type	n	nnz	density	other
bertsekas	NCP	15	74	32.89%	
billups	NCP	1	1	100.00%	1
bert_oc	LMCP	5000	21991	0.09%	M
bratu	NLP	5625	33749	0.11%	
choi	NCP	13	169	100.00%	
colvdual	NLP	20	168	42.00%	
colvnlp	NLP	15	113	50.22%	
cycle	LCP	1	1	100.00%	M1
ehl_kost	MCP	101	10201	100.00%	
explcp	LCP	16	152	59.38%	1
freebert	MCP	15	74	32.89%	
gafni	MCP	5	25	100.00%	
hanskoop	NCP	14	129	65.82%	
hydroc06	NE	29	222	26.40%	
hydroc20	NE	99	838	8.55%	
josephy	NCP	4	16	100.00%	1
kojshin	NCP	4	16	100.00%	2
mathinum	NCP	3	9	100.00%	
mathisum	NCP	4	14	87.50%	
methan08	NE	31	225	23.41%	
nash	MCP	10	100	100.00%	
obstacle	LMCP/NLP	2500	14999	0.24%	M1
opt_cont	LMCP	288	4928	5.94%	M1
pgvon105	NCP	105	796	7.22%	S
pgvon106	NCP	106	898	7.99%	S
pies	MCP	42	183	10.37%	
powell	NLP	16	203	79.30%	S
powell_mcp	NCP	8	54	84.38%	
scarfanum	NCP	13	98	57.99%	
scarfasum	NCP	14	109	55.61%	
scarfbnum	NCP	39	361	23.73%	
scarfbsum	NCP	40	614	38.38%	
sppe	NCP	27	110	15.09%	
tobin	NCP	42	243	13.78%	

active role in influencing the development of new algorithms. It must be noted, however, that there are subtle differences between the MATLAB models and the GAMS models. In particular, many GAMS models vary not only the starting point for different runs, but also some of the underlying nonlinearities, whereas the MATLAB models vary only the starting point. Thus, a completely accurate comparison must be carried out exclusively in GAMS or exclusively in MATLAB.

2.3. Verification of Solutions

Since stopping criteria vary from algorithm to algorithm, a standardized measure is needed to ensure that different algorithms produce solutions that have some uniformity in solution quality. To achieve this goal, we developed an additional solver, accessible through GAMS, that evaluates the starting point and returns the value of the following merit function:

$$\|F(\pi_{\mathbf{B}}(x)) + x - \pi_{\mathbf{B}}(x)\|_2 , \tag{1}$$

where $\pi_{\mathbf{B}}$ represents the projection operator onto the set $\mathbf{B}$. To use this verification test, we first solve the problem with the algorithm we are testing, and pass the solution to our "special" solver to verify that the standardized residual is not too large. Since the special solver is callable from GAMS, this can be achieved by adding a few lines to the GAMS problem files.

2.4. Data Extraction

The output of MCP codes is typically quite extensive and varies from solver to solver. To extract pertinent information from this output, we have written several awk scripts that read through the files, and then generate data tables. These scripts require slight modifications for each solver, but are a tremendous help in extracting data to produce meaningful information.

3. Description of Algorithms

Ideally, the computational study of algorithms should be performed using only mature, sophisticated codes, so that the strengths and limitations of each algorithm would be accurately reflected in the numerical results. Unfortunately, many of the algorithms proposed for complementarity problems are not accompanied by such mature codes. Of the algorithms described below, the implementations of MILES, PATH, and SMOOTH are the most mature. For the remaining algorithms, we have developed our own implementations which incorporate the GAMS interface.

All of the algorithms outlined have been coded to take explicit advantage of the MCP structure; however, several of them were originally devised for the special case of the nonlinear complementarity problem (NCP)

$$z \geq 0, F(z) \geq 0, z^T F(z) = 0$$

and will be described below in this context. We now give a brief description of the codes that were tested and indicate pertinent references for further details.

3.1. MILES

MILES [30] is an extension of the classical Josephy-Newton method for NCP in which the solution to each linearized subproblem

$$0 \in F(z^k) + \nabla F(z^k)(z - z^k) + N_{\mathbf{B}}(z)$$

is computed via Lemke's almost-complementary pivot algorithm. This Newton point is used to define the Newton direction, which is then used in a damped linesearch. The merit function used measures both the violation in feasibility and in complementarity. MILES also employs a restart procedure in cases where the Newton point cannot be computed due to termination in a secondary ray. Every linearized subproblem is rescaled to equilibrate the elements appearing in the data of the subproblem.

3.2. PATH

The PATH solver [8] applies techniques similar to those used in Newton methods for smooth systems to the following reformulation of the MCP

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

Here $\pi_{\mathbf{B}}$ represents the projection operator onto the set $\mathbf{B}$, which is in general not differentiable. The algorithm consists of a sequence of major iterations, each consisting of an approximation or linearization step similar to that of MILES, the construction of a *path* to the Newton point (the solution to the approximation), and a possible search of this path. When the Newton point does not exists or the path cannot be entirely constructed, a step along the partially computed path is taken before the problem is relinearized. A nonmonotone watchdog strategy is employed in applying the path search; this helps avoid convergence to local minima of the merit function (1), and keeps the number of function evaluations required as small as possible.

Other computational enhancements employed by PATH are a projected Newton preprocessing phase (used to find an initial point that better corresponds to the optimal active set) and the addition of a diagonal perturbation term to the Jacobian matrix when rank deficiency is detected. The Jacobian elements are also automatically scaled by the algorithm at each major iteration.

3.3. NE/SQP

The NE/SQP algorithm [26] is based upon reformulating the NCP as the system of nonsmooth equations

$$0 = H(z) := \min\{z, F(z)\}.$$

In [2] the NE/SQP algorithm is extended to the MCP by using the reformulation

$$0 = H(z) := \min\{z - \ell, \max\{z - u, F(z)\}\} \tag{2}$$

Both algorithms use a Gauss-Newton approach that attempts to minimize

$$\theta(z) := \|H(z)\|^2 \tag{3}$$

to find a zero of H. The nonsmoothness of the equations is handled using directional derivatives of H. Specifically, at each iteration, a search direction is calculated by minimizing a convex quadratic program whose objective function is formed by squaring a linear approximation of H. At points where the derivative is not well defined, the linear approximation is created by choosing a particular element of the subdifferential. Once this direction is determined, an Armijo-type linesearch is used to calculate the step size to be taken along that direction. The advantage of this approach is that the direction finding subproblems are always solvable. This is in contrast to Newton-based approaches, which may fail due to a singular Jacobian matrix, and to PATH and MILES, which determine the search direction by attempting to solve a linear complementarity problem, which may, in fact, be unsolvable.

One weakness of the algorithm is that it is vulnerable to converging to local minima of the merit function θ that are not solutions to the problem. The code uses scaling of the subproblems and enforces a small cushion between the iterates and the boundary of $\mathbf{B}$ as suggested in [26].

3.4. SMOOTH

The SMOOTH algorithm [4] is based upon reformulating the NCP as a system of nonsmooth equations

$$x = \pi_{\mathbf{R}^n_+}(x - F(x)),$$

and then approximately solving a sequence of smooth approximations, which lead to a zero of the nonsmooth system. More precisely, at each iteration, a smooth approximation to the original system is formed where the accuracy of the approximation is determined by the residual of the current point, that is $\left\| x - \pi_{\mathbf{R}^n_+}(x - F(x)) \right\|$. The smooth approximation p_α to $\pi_{\mathbf{R}^n_+}$ corresponds to an integration of the sigmoid function that is commonly used in machine learning. Applying a single step of Newton's method to this smooth function generates a search direction. The next iterate is then generated by performing an Armijo-type linesearch of the merit function

$$\|x - p_\alpha(x - F(x))\|$$

along this direction. Assuming this new point produces an improved residual, the next iteration is based upon a tighter approximation of the nonsmooth equations.

An initial scaling of the data is used in the code, and the PATH preprocessor is used. However, in SMOOTH, the preprocessor is used to try to solve the MCP instead of merely to identify the active set. If this technique fails, the code is restarted and the smoothing technique is then used to find a solution.

3.5. QPCOMP

QPCOMP [2] is an enhancement of the NE/SQP algorithm, which adds a proximal perturbation strategy that allows the iterates to escape local minima of the merit function θ defined in (3). In essence, the algorithm detects when the iterates appear to be converging to a local minimum, and then approximately solves a sequence of perturbed problems to escape the domain of convergence of that local minimum. The perturbed problems are formed by replacing F with the perturbed function

$$F^{\lambda,\bar{z}} := F(z) + \lambda(z - \bar{z}), \tag{4}$$

where the centering point $\bar{z}$ is generally chosen to be the current iterate, and the perturbation parameter λ is chosen adaptively in a manner that guarantees global convergence to a solution when F is both continuously differentiable and pseudomonotone at a solution. In general, the perturbed function is updated after each iteration. Thus, the perturbed problems are not solved exactly; they are just used to determine the next step.

An important aspect of the algorithm is that F is perturbed only when the iterates are not making good progress toward a zero of the merit function. In particular, during the perturbation strategy, whenever an iterate is encountered where the merit function (of the unperturbed problem) has been sufficiently reduced, the algorithm reverts to solving the unperturbed problem. Thus, near a solution, the algorithm maintains the fast local convergence rates of the underlying NE/SQP algorithm.

We note that NE/SQP is equivalent to QPCOMP without the proximal perturbation strategy. Thus, to test NE/SQP, we simply ran the QPCOMP algorithm with the proximal perturbation strategy turned off.

3.6. PROXI

PROXI [1], like NE/SQP and QPCOMP is based upon reformulating the MCP as the system of nonsmooth equations (2). However, instead of solving this system using a Gauss-Newton approach, PROXI uses a nonsmooth version of Newton's method. Specifically, at each iteration, the search direction is calculated by solving a linear system that approximates H at the current iterate. Again, if H is not differentiable at the current iterate, the linear approximation is created by choosing a particular element of the subdifferential.

Like QPCOMP, PROXI uses a proximal perturbation strategy to allow the iterates to escape local minima of the merit function θ defined in (3). This strategy also allows the algorithm to overcome difficulties resulting from singular Jacobian matrices. In particular, if the Newton equation is unsolvable at a particular iteration, the algorithm simply creates a slightly perturbed problem using (4) with a very small λ. The resulting Newton equation for the perturbed function will then be solvable. This strategy for dealing with unsolvable Newton subproblems is considerably more efficient than the Gauss-Newton approach used by NE/SQP and QPCOMP.

3.7. SEMISMOOTH

SEMISMOOTH [1] is an implementation of an algorithm described in [6]. This algorithm is based upon the function

$$\phi(a,b) = \sqrt{a^2+b^2} - (a+b),$$

which was introduced by [15]. This function has the property that

$$\phi(a,b) = 0 \Longleftrightarrow a \geq 0, b \geq 0, ab = 0.$$

Using this function, the NCP is reformulated as the semismooth system of equations

$$0 = \Phi(z),$$

where $\Phi_i(z) := \phi(z_i, F_i(z))$. This reformulation has the nice feature that the natural merit function $\Psi(z) := \|\Phi(z)\|^2$ is continuously differentiable. The SEMISMOOTH algorithm described in [1] extends the approach to the MCP by using the reformulation of MCP given by

$$\Phi_i(z) := \phi(z_i - \ell_i, \phi(u_i - z_i, -F_i(z))).$$

To solve the reformulated system of equations, a generalization of Newton's method is used wherein at each iteration, the search direction d^k is found by solving the system

$$H^k d = -\Phi(z^k),$$

where H^k is an element of the B-subdifferential of Φ. The next point z^{k+1} is then chosen by performing a nonmonotone, Arimijo linesearch along the direction d^k.

3.8. SEMICOMP

SEMICOMP [1] is an enhancement of the SEMISMOOTH algorithm, which, like QPCOMP and PROXI, adds a proximal perturbation strategy to allow iterates to escape local minima of the merit function. The algorithm is identical to SEMISMOOTH except when the iterates stop making satisfactory progress toward a zero of Φ. In this case, the proximal perturbation strategy described for the QPCOMP algorithm is employed to allow the iterates to escape the troublesome region. Specifically, at each iteration, a perturbed function is created by (4), and then the SEMISMOOTH algorithm is used to calculate a new point based on this perturbed function. The perturbed function is then updated and the process repeats. The process continues until a new point is encountered where the merit function is sufficiently smaller than the merit function at any previous point. At this point, the algorithm reverts back to the unperturbed SEMISMOOTH algorithm.

4. Computational Comparison

With the exception of NE/SQP and QPCOMP, each of the eight algorithms described in the previous section was run on all of the problems in the test library from all of the starting points. Since NE/SQP and QPCOMP were implemented using a dense QP code, we only ran the problems with fewer than 110 variables for these solvers. Table A.1 in appendix A shows the execution time needed by each algorithm on a SPARC 10/51, while Table A.2, also in Appendix A, shows the number of function and Jacobian evaluations required by each algorithm. To abbreviate the results, we excluded any problems that were solved in less than 2 seconds by all of the algorithms we tested.

Each algorithm minimizes its own merit function as described in Section 3 and all were terminated when this measure was reduced below 10^{-6}. Since the merit functions are different for each code, we tested the solutions to ensure that the standardized residual given by (1) was always less than 10^{-5}. It is possible that one more or one less "Newton" step would be carried out if the same merit function was used for every algorithm. Since this is impractical, the method we now outline for reporting our results makes these small changes entirely irrelevant.

How one chooses to summarize data of this nature depends on what one's goals are. From a modeling standpoint, one could determine which models were the most difficult to solve by aggregating results for each model. From a computational standpoint, one can compare the solvers using many different criteria, including number of successes/failures, cumulative solution time required, number of cases where solution time is "acceptable", number of function/gradient evaluations required, etc. As examples of useful metrics, we have chosen the following:

success Success is achieved if a solution is computed.

competitive We say the time T_C for a code C is "competitive" with the time T_{min} taken by the best code on that run if $T_C \leq 2\,T_{\text{min}}$, and "very competitive" if $T_C \leq \frac{4}{3}\,T_{\text{min}}$.

Tables 3 and 4 summarize our results for two sets of models, the large ones (≥ 110 variables) for which only the sparsity-exploiting solvers were run, and the smaller ones on which all solvers were run.

Table 3. Code Comparisons – Large Models

	MILES	PATH	PROXI	SEMI-COMP	SEMI-SMTH	SMTH
very comp.	37%	65%	50%	22%	17%	59%
competitive	56%	80%	67%	41%	43%	76%
success	83%	98%	89%	91%	86%	98%

Table 4. Code Comparisons – Small Models

	MILES	NE/SQP	PATH	PROXI	QP COMP	SEMI-COMP	SEMI-SMTH	SMTH
very comp.	32%	0%	43%	34%	0%	26%	25%	24%
competitive	45%	2%	67%	54%	1%	44%	40%	52%
success	84%	67%	94%	95%	90%	88%	65%	92%

5. Conclusions

The testing environment we have described addresses many of the problems we have observed about how codes are developed and tested. In particular, with a large collection of test problems available, it is more difficult to tune a code to the test set. Moreover, even if such tuning is successful, the resulting code will be good at solving the types of problems that are represented in the library, namely, the problems that are of interest to the user community. The inclusion of problems with known difficulties allows codes to be compared by how well they solve different classes of problems, thus allowing users to more accurately choose codes that meet their needs. Finally, by categorizing problems with different computational difficulties, the library can be used to highlight the areas where research energies most need to be directed.

Our testing indicates superior performance by the PATH, SMOOTH, and PROXI algorithms. However, as the codes continue to mature, it is possible that their relative performance will change. It is not our intention to declare a winner, but rather to "clarify the rules" so that code developers will focus on the right issues when developing algorithms. To a large extent, we have accomplished this with our testing environment.

It is unfortunate that the scope of our testing could not have been more broad. Some of the algorithms mentioned above were coded by the authors of this paper (not the originators of the algorithm), while there are numerous other algorithms that we were not able to test at all. This is due primarily to the fact that these algorithms do not have GAMS interfaces. It is our hope that as the CPLIB interface becomes more widely known, other code developers will hook up their solvers to GAMS. This will allow their algorithms to be easily compared with other codes using our testing environment.

Lastly, we wish to emphasize that the test library is continually being expanded. In particular, we are always eager to add challenging new real world models to the library. To this end, we have begun to augment the MCPLIB by adding new models that have recently come to our attention. The 10 models listed in Table 5 have been used in various disciplines to answer questions that give rise to complementarity problems. Some of these models are solved from many different starting points, indicated by the "solves" column. The first 6 are economic models, the next two arise from applications in traffic equilibrium and multi-rigid-body contact problems, the final two correspond to complementarity problems for which all solutions are required. The numbers of solutions for the last two problems are known to be odd, the number listed below is a lower bound. These problems appear to be more difficult than most of the problems solved in this paper. Certainly, some are much larger, while others have singularities either at solutions or starting points. Most of these problems do not have underlying monotonicity.

The results that we present in Tables 6 and 7 for these models are somewhat different to the results in Appendix A and are motivated more by the models themselves. For the games and tinloi models, it is important to find all solutions of the model, and so after a fixed number of runs from a variety of starting points, we report the number of distinct solutions found for these models in Table 6.

For the remaining problems, we just report one statistic in Table 7 for each model. If every problem was solved, we report the total resources used to solve the complete model, otherwise we report an error using a letter to signify some sort of failure "F", memory error "M", time limit exceeded "T" or iteration limit exceeded "I". Only the first error is listed per problem, while the numbers in parentheses are the number of problems that failed to solve.

It is our intention to add these models and newer models that are brought to our attention to MCPLIB. In this way we hope that the problem library will continue

Table 5. New Models

Model	Type	n	nnz	density	solves	other
shubik	MCP	33	207	19.01%	48	S
jmu	MCP	2253	10123	0.20%	1	
asean9a	NE	10199	72320	0.07%	1	
eppa	MPSGE	1269	10130	0.63%	8	
uruguay	MPSGE	2281	90206	1.73%	2	
hanson	NE	487	3868	1.63%	2	S
trafelas	MCP	2904	15000	0.18%	2	
lincont	LCP	419	23626	13.46%	1	
games	NCP	16	256	60.94%	25	5
tinloi	LCP	146	5694	26.71%	64	3

Table 6. Distinct Solutions Found

Model	MILES	PATH	PROXI	S/COMP	S/SMTH	SMOOTH
games	3	5	2	3	3	4
tinloi	2	3	1	1	1	2

Table 7. Summary for New Models

Model	MILES	PATH	PROXI	S/COMP	S/SMTH	SMOOTH
shubik	I(13)	F(9)	F(25)	I(34)	I(42)	I(15)
jmu	I	110.81	F	F	T	214.32
asean9a	T	62.08	M	92.85	94.3	91.62
eppa	249.61	203.79	M	T(7)	F(7)	239.73
uruguay	I(1)	2760.17	M	M	68161.10	4519.53
hanson	F(1)	39.36	F(1)	F(1)	I(1)	4.94
trafelas	T(2)	150.55	F(1)	T(2)	T(2)	346.23
lincont	9.99	10.76	F	F	T	718.27

to serve as a guide for code developers so that they will direct their energies into areas that will best serve the users.

References

1. S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations.* PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.
2. S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. Mathematical Programming Technical Report 95–09, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.
3. A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* The Scientific Press, South San Francisco, CA, 1988.
4. Chunhui Chen and O. L. Mangasarian. A class of smoothing functions for nonlinear and mixed complementarity problems. *Computational Optimization and Applications*, 5:97–138, 1996.
5. R. W. Cottle. *Nonlinear programs with positively bounded Jacobians.* PhD thesis, Department of Mathematics, University of California, Berkeley, California, 1964.
6. T. De Luca, F. Facchinei, and C. Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. Submitted to Mathematical Programming, 1995.
7. S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.
8. 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.
9. 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/.
10. F. Facchinei and J. Soares. A new merit function for nonlinear complementarity problems and a related algorithm. Technical Report 15.94, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Rome, Italy, 1994.
11. F. Facchinei and J. Soares. Testing a new class of algorithms for nonlinear complementarity problems. In F. Giannessi and A. Maugeri, editors, *Variational Inequalities and Network Equilibrium Problems*, pages 69–83. Plenum Press, New York, 1995.
12. M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. Discussion Papers in Economics 95–4, Department of Economics, University of Colorado, Boulder, Colorado, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.
13. M. C. Ferris and D. Ralph. Projected gradient methods for nonlinear complementarity problems via normal maps. In D. Du, L. Qi, and R. Womersley, editors, *Recent Advances in Nonsmooth Optimization*, pages 57–87. World Scientific Publishers, 1995.
14. M. C. Ferris and T. F. Rutherford. Accessing realistic complementarity problems within Matlab. In G. Di Pillo and F. Giannessi, editors, *Proceedings of Nonlinear Optimization and Applications Workshop, Erice June 1995*, New York, 1996. Plenum Press.
15. A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.
16. 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.
17. R. Fourer, D. Gay, and B. Kernighan. *AMPL.* The Scientific Press, South San Francisco, California, 1993.
18. C. B. Garcia and W. I. Zangwill. *Pathways to Solutions, Fixed Points, and Equilibria.* Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1981.

19. C. Geiger and C. Kanzow. On the resolution of monotone complementarity problems. Preprint 82, Institute of Applied Mathematics, University of Hamburg, April 1994. Bundesstrasse 55, D-20146 Hamburg Germany.
20. S.-P. Han, J. S. Pang, and N. Rangaraj. Globally convergent Newton methods for nonsmooth equations. *Mathematics of Operations Research*, 17:586–607, 1992.
21. P. T. Harker and B. Xiao. Newton's method for the nonlinear complementarity problem: A B–differentiable equation approach. *Mathematical Programming*, 48:339–358, 1990.
22. Lars Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.
23. J. J. Moré. Global methods for nonlinear complementarity problems. Technical Report MCS-P429-0494, Argonne National Laboratory, Argonne, Illinois, April 1994.
24. J. S. Pang. Newton's method for B–differentiable equations. *Mathematics of Operations Research*, 15:311–341, 1990.
25. J. S. Pang. A B–differentiable equation based, globally and locally quadratically convergent algorithm for nonlinear programs, complementarity and variational inequality problems. *Mathematical Programming*, 51:101–132, 1991.
26. J. S. Pang and S. A. Gabriel. NE/SQP: A robust algorithm for the nonlinear complementarity problem. *Mathematical Programming*, 60:295–338, 1993.
27. D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 19:352–389, 1994.
28. S. M. Robinson. Generalized equations. In A. Bachem, M. Grötchel, and B. Korte, editors, *Mathematical Programming: The State of the Art, Bonn 1982*, pages 346–367. Springer Verlag, Berlin, 1983.
29. S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.
30. T. F. Rutherford. MILES: A mixed inequality and nonlinear equation solver. Working Paper, Department of Economics, University of Colorado, Boulder, 1993.
31. T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control, forthcoming*, 1995.
32. H. E. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal on Applied Mathematics*, 15:1328–1343, 1967.
33. M. J. Todd. *Computation of Fixed Points and Applications*, volume 124 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Heidelberg, 1976.
34. B. Xiao and P. T. Harker. A nonsmooth Newton method for variational inequalities: I: Theory. *Mathematical Programming*, 65:151–194, 1994.
35. B. Xiao and P. T. Harker. A nonsmooth Newton method for variational inequalities: II: Numerical results. *Mathematical Programming*, 65:195–216, 1994.

Appendix A

Table A.1. Execution Times (sec.)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP- COMP	SEMI- COMP	SEMI- SMTH	SMTH
bert_oc	1	6.15	–	2.63	2.61	–	11.38	13.50	3.23
bert_oc	2	7.07	–	3.13	3.24	–	46.44	54.41	2.57
bert_oc	3	fail	–	2.10	2.78	–	15.52	17.99	2.55
bert_oc	4	136.4	–	2.29	2.67	–	5.80	5.91	2.62
bertsekas	1	0.07	fail	0.08	0.39	2.83	0.64	fail	0.24
bertsekas	2	0.28	fail	0.04	0.27	2.41	0.59	fail	0.05
billups	1	fail	fail	fail	0.02	0.11	0.10	0.90	fail
bratu	1	fail	–	138.52	149.37	–	7452.38	fail	135.48
cafemge	1	0.18	18.16	0.29	0.50	20.11	0.50	0.66	0.41
cafemge	2	0.23	16.57	0.26	0.35	14.19	0.50	0.39	0.25
cammcp	1	0.50	–	0.21	2.89	–	fail	fail	0.23
choi	1	8.13	2.00	2.09	2.03	2.28	2.95	2.93	2.10
co2mge	2	0.43	–	0.50	0.48	–	2.02	2.42	0.52
co2mge	6	0.62	–	0.46	fail	–	fail	fail	1.96
colvdual	1	0.05	fail	0.11	0.25	5.76	0.12	0.10	0.11
colvdual	2	0.07	fail	0.09	0.50	5.39	fail	fail	0.10
colvnlp	1	0.03	fail	0.05	0.09	2.13	0.08	0.09	0.06
colvnlp	2	0.05	fail	0.03	0.05	1.62	0.06	0.05	0.05
dmcmge	1	0.20	–	3.75	fail	–	fail	fail	5.42
dmcmge	2	0.50	–	0.55	fail	–	133.73	fail	0.60
ehl_kost	1	23.58	fail	3.86	18.50	611.41	18.99	15.02	4.73
ehl_kost	2	23.92	248.79	13.56	37.67	250.28	49.06	58.25	12.58
ehl_kost	3	24.15	fail	9.76	64.88	866.08	233.23	240.12	90.38
finmge	2	0.38	–	1.95	11.34	–	fail	fail	5.16
finmge	4	0.48	–	1.72	12.34	–	fail	fail	9.18
freebert	1	0.03	fail	0.07	0.39	2.72	0.51	fail	0.04
freebert	3	0.10	fail	0.05	0.25	2.86	0.55	fail	0.04
freebert	4	fail	fail	0.09	0.31	2.47	0.60	fail	fail
freebert	5	fail	fail	0.04	0.12	1.38	0.15	0.12	0.04
freebert	6	fail	fail	0.08	0.33	3.02	0.53	fail	fail
gemmcp	1	2.12	–	0.21	0.16	–	0.19	0.18	0.24
gemmge	2	0.47	–	3.24	3.31	–	3.31	3.60	4.18
gemmge	3	0.55	–	1.85	1.89	–	2.88	3.92	1.85
gemmge	4	0.52	–	2.51	2.37	–	2.84	3.22	1.84
gemmge	5	0.55	–	8.85	5.00	–	5.32	6.93	2.28
hanskoop	1	0.07	0.37	0.05	0.10	0.37	fail	fail	0.33
hanskoop	2	0.08	0.04	0.06	0.01	0.05	fail	fail	0.02

Table A.1. Execution Times (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
hanskoop	3	fail	0.34	0.11	0.09	0.42	fail	fail	0.23
hanskoop	4	1.10	0.06	0.05	0.01	0.05	fail	fail	0.02
hanskoop	5	0.07	fail	0.09	0.10	0.70	0.07	0.08	0.30
hanskoop	7	0.07	fail	0.05	0.09	0.86	fail	fail	0.22
hanskoop	9	fail	0.50	0.10	0.24	0.43	0.09	0.10	0.23
hansmcp	1	0.10	fail	0.47	0.14	fail	0.16	0.16	0.13
hansmge	1	0.10	3.14	0.36	0.70	2.86	0.84	0.88	0.64
harkmcp	4	0.07	6.96	0.12	0.21	9.31	fail	fail	0.37
harmge	1	0.03	fail	0.06	0.44	1.86	1.52	fail	0.09
harmge	2	0.80	fail	0.03	0.02	0.14	0.01	fail	0.03
harmge	3	0.07	fail	0.04	0.03	0.13	0.02	fail	0.04
harmge	4	0.08	fail	0.05	0.03	0.15	0.01	fail	0.04
harmge	5	0.08	fail	0.05	0.04	0.16	0.02	fail	0.04
harmge	6	fail	fail	0.06	fail	3.24	0.02	fail	2.08
hydroc20	1	fail	16.11	0.38	0.44	13.31	0.54	0.41	0.36
josephy	1	fail	fail	0.03	0.02	0.08	0.02	0.01	0.03
josephy	2	fail	fail	0.04	0.02	0.07	0.02	0.02	0.02
josephy	4	fail	fail	0.02	0.01	0.04	0.01	0.01	0.02
josephy	6	fail	0.04	fail	0.02	0.05	0.01	0.01	0.02
kojshin	1	fail	fail	0.03	0.01	0.07	0.02	0.03	0.03
kojshin	3	0.03	fail	0.06	0.05	0.12	0.06	0.07	0.11
kormcp	1	0.23	2.82	0.08	0.06	2.82	0.07	0.05	0.05
mr5mcp	1	0.60	–	0.62	2.17	–	2.09	2.01	0.62
nsmge	1	0.25	–	0.91	1.64	–	1.69	1.65	2.40
obstacle	1	2.37	–	2.36	3.40	–	6.86	5.59	2.39
obstacle	2	fail	–	5.90	7.33	–	18.01	15.56	6.39
obstacle	3	fail	–	5.03	8.85	–	11.77	9.45	6.27
obstacle	4	3.98	–	4.84	9.29	–	11.01	10.66	6.12
obstacle	5	fail	–	8.04	4.52	–	15.08	14.59	7.13
obstacle	6	fail	–	8.86	9.92	–	19.62	21.14	10.07
obstacle	7	fail	–	7.39	7.57	–	12.84	15.52	7.97
obstacle	8	fail	–	13.84	7.54	–	14.76	14.32	10.58
opt_cont127	1	8.52	–	8.14	9.91	–	46.05	45.58	6.38
opt_cont255	1	fail	–	14.86	18.71	–	107.97	110.61	13.80
opt_cont31	1	2.10	–	1.36	1.51	–	5.55	4.45	1.65
opt_cont511	1	fail	–	39.51	43.19	–	348.63	360.42	37.52
pgvon105	1	fail	fail	1.54	7.99	fail	fail	fail	fail
pgvon105	2	0.42	41.51	0.77	2.18	50.91	fail	fail	fail

Table A.1. Execution Times (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
pgvon105	3	fail	33.47	1.58	52.13	58.80	fail	fail	fail
pgvon105	4	fail	fail	fail	fail	fail	28.09	fail	fail
pgvon106	1	fail	fail	19.77	13.21	fail	fail	fail	125.46
pgvon106	2	fail	fail	1.80	fail	fail	fail	fail	5.37
pgvon106	3	fail	fail	1.29	fail	fail	fail	fail	8.48
pgvon106	4	fail	fail	fail	2.46	fail	38.30	fail	fail
pgvon106	5	5.33	fail	fail	fail	fail	fail	fail	fail
pgvon106	6	fail	fail	fail	fail	fail	fail	fail	3.76
pies	1	0.07	fail	0.13	0.29	7.26	0.11	0.13	0.27
sammge	1	0.07	fail	0.01	0.01	fail	0.00	0.00	0.00
sammge	3	0.10	0.27	0.05	0.16	0.26	0.17	fail	0.18
sammge	5	0.12	0.42	0.07	0.12	0.48	0.36	fail	0.13
sammge	6	0.13	0.45	0.05	0.27	0.58	0.40	fail	0.13
sammge	7	0.18	0.69	0.06	0.13	0.58	0.23	fail	0.20
sammge	8	0.10	0.78	0.05	0.63	0.74	0.39	fail	0.19
sammge	9	0.10	0.71	0.07	0.45	0.69	0.65	fail	0.20
sammge	10	0.17	fail	0.01	0.01	fail	0.01	0.00	0.01
sammge	13	0.05	0.30	0.12	0.20	0.28	0.23	fail	0.23
sammge	14	0.12	0.29	0.11	0.17	0.35	0.23	fail	0.18
sammge	15	0.05	0.27	0.06	0.48	0.31	0.38	fail	0.25
sammge	16	0.05	0.47	0.11	0.26	0.46	0.31	fail	0.10
sammge	17	0.10	0.62	0.09	0.57	1.05	0.20	fail	0.17
sammge	18	0.08	0.37	0.11	0.46	0.45	0.50	fail	0.16
scarfasum	2	0.15	fail	0.04	0.15	1.51	0.15	0.12	0.10
scarfasum	3	0.13	0.29	0.07	0.15	0.37	fail	fail	0.05
scarfbnum	1	0.08	6.27	0.39	0.57	6.42	1.01	fail	0.32
scarfbnum	2	0.10	6.01	0.44	0.43	6.09	7.36	fail	0.32
scarfbsum	1	0.17	fail	fail	0.49	8.77	0.39	0.31	0.24
scarfbsum	2	0.18	fail	3.43	5.16	31.11	1.22	fail	0.66
threemge	7	0.08	–	0.06	fail	–	0.14	0.13	0.05
threemge	8	0.07	–	0.06	fail	–	0.12	0.14	0.05
threemge	11	0.12	–	0.05	fail	–	0.82	fail	0.05
transmcp	1	0.03	fail	0.04	0.09	1.22	0.23	fail	0.05
transmcp	2	0.03	fail	0.01	0.00	fail	0.00	fail	0.00
transmcp	3	0.03	0.02	0.02	0.01	0.02	0.03	fail	0.02
transmcp	4	0.03	0.11	0.02	0.01	0.10	0.04	fail	0.02
vonthmcp	1	fail	–	fail	fail	–	fail	fail	fail
vonthmge	1	0.08	fail	1.06	fail	fail	fail	fail	17.14

Table A.2. Function and Jacobian Evaluations f(j)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
bert_oc	1	13(12)	–	4(4)	4(3)	–	21(11)	21(11)	4(4)
bert_oc	2	21(13)	–	4(4)	4(3)	–	143(42)	143(42)	4(4)
bert_oc	3	fail	–	4(4)	4(3)	–	41(15)	41(15)	4(4)
bertsekas	1	259(36)	fail	27(6)	138(37)	151(44)	251(42)	fail	113(27)
bertsekas	2	5(4)	fail	5(5)	83(31)	126(40)	327(38)	fail	7(7)
bertsekas	3	12(11)	9(8)	12(12)	21(20)	9(8)	181(39)	181(39)	67(24)
billups	1	fail	fail	fail	23(22)	23(22)	631(76)	6903(345)	fail
bratu	1	fail	–	48(26)	48(25)	–	3164(538)	fail	48(26)
cafemge	1	8(6)	16(10)	9(7)	23(9)	16(10)	18(10)	18(10)	9(8)
cafemge	2	6(5)	15(8)	6(6)	17(7)	15(8)	11(8)	11(8)	6(6)
cammcp	1	4(3)	–	4(4)	77(23)	–	fail	fail	4(4)
choi	1	5(4)	5(4)	5(5)	5(4)	5(4)	6(5)	6(5)	5(5)
co2mge	2	9(6)	–	9(7)	7(5)	–	62(15)	63(16)	7(6)
co2mge	6	6(5)	–	7(7)	fail	–	fail	fail	81(13)
colvdual	1	4(3)	fail	15(13)	201(36)	252(78)	44(16)	44(16)	40(15)
colvdual	2	4(3)	fail	16(12)	250(55)	184(59)	fail	fail	52(17)
colvnlp	1	4(3)	fail	10(7)	77(16)	178(54)	46(16)	46(16)	37(14)
colvnlp	2	4(3)	fail	5(5)	29(12)	137(30)	26(15)	26(15)	23(10)
dmcmge	1	99(27)	–	34(18)	fail	–	fail	fail	97(23)
dmcmge	2	13(8)	–	6(6)	fail	–	3099(661)	fail	6(6)
ehl_kost	1	6(5)	fail	6(6)	25(14)	108(105)	32(15)	32(15)	6(6)
ehl_kost	2	8(7)	97(30)	19(19)	95(28)	97(30)	125(34)	125(34)	21(12)
ehl_kost	3	16(11)	fail	11(11)	144(44)	409(79)	671(114)	671(114)	262(55)
finmge	2	5(4)	–	7(7)	151(25)	–	fail	fail	60(13)
finmge	4	5(4)	–	8(8)	135(28)	–	fail	fail	110(20)
freebert	1	4(3)	fail	5(5)	138(37)	151(44)	266(46)	fail	6(6)
freebert	3	4(3)	fail	5(5)	106(35)	173(45)	206(42)	fail	6(6)
freebert	4	fail	fail	27(6)	138(37)	151(44)	240(42)	fail	fail
freebert	5	fail	fail	5(5)	53(14)	116(23)	49(14)	49(14)	5(5)
freebert	6	fail	fail	27(6)	106(35)	173(45)	200(40)	fail	fail
gemmcp	1	2(1)	–	2(2)	2(1)	–	2(1)	2(1)	2(2)
gemmge	1	fail	–	fail	2(1)	–	2(1)	2(1)	2(2)
gemmge	2	7(5)	–	18(7)	22(7)	–	16(9)	16(9)	21(6)
gemmge	3	6(5)	–	6(6)	6(5)	–	10(8)	10(8)	6(6)
gemmge	4	7(6)	–	6(6)	7(6)	–	8(7)	8(7)	6(6)
gemmge	5	10(7)	–	26(21)	25(11)	–	31(13)	31(13)	13(7)
hanskoop	1	6(4)	15(10)	13(7)	42(16)	15(10)	fail	fail	110(36)
hanskoop	2	2(1)	2(1)	14(6)	2(1)	2(1)	fail	fail	2(1)

Table A.2. Function and Jacobian Evaluations (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP- COMP	SEMI- COMP	SEMI- SMTH	SMTH
hanskoop	3	fail	18(11)	23(14)	44(13)	18(11)	fail	fail	78(31)
hanskoop	4	2(1)	2(1)	14(6)	2(1)	2(1)	fail	fail	2(1)
hanskoop	5	6(5)	fail	19(11)	68(15)	27(11)	20(8)	20(8)	102(34)
hanskoop	7	7(5)	fail	11(6)	37(15)	45(13)	fail	fail	83(25)
hanskoop	9	fail	23(13)	16(13)	187(41)	23(13)	20(15)	20(15)	95(31)
hansmcp	1	6(4)	fail	45(18)	18(9)	fail	24(13)	24(13)	10(8)
hansmge	1	4(3)	11(8)	12(8)	37(15)	11(8)	47(17)	47(17)	26(13)
harkmcp	4	5(4)	29(13)	13(6)	23(14)	27(14)	fail	fail	31(17)
harmge	1	284(60)	fail	11(7)	222(38)	132(57)	672(75)	fail	33(11)
harmge	2	5(4)	fail	5(5)	5(4)	5(4)	3(2)	fail	5(5)
harmge	3	5(4)	fail	5(5)	5(4)	5(4)	3(2)	fail	5(5)
harmge	4	8(5)	fail	8(6)	5(4)	5(4)	3(2)	fail	8(6)
harmge	5	8(5)	fail	8(6)	8(5)	8(5)	3(2)	fail	8(6)
harmge	6	fail	fail	13(8)	fail	379(78)	3(2)	fail	1117(139)
hydroc20	1	fail	10(8)	11(9)	10(8)	10(8)	12(9)	12(9)	10(9)
josephy	1	fail	fail	7(7)	37(14)	13(7)	10(7)	10(7)	24(9)
josephy	2	fail	fail	15(11)	15(7)	15(7)	12(7)	12(7)	9(6)
josephy	4	fail	fail	4(4)	5(4)	5(4)	6(5)	6(5)	5(4)
josephy	6	fail	4(3)	fail	12(6)	12(6)	13(7)	13(7)	9(6)
kojshin	1	fail	fail	6(6)	18(9)	16(7)	22(10)	22(10)	33(10)
kojshin	3	11(10)	fail	17(17)	97(22)	35(10)	92(23)	122(30)	189(27)
kormcp	1	4(3)	4(3)	4(4)	4(3)	4(3)	4(3)	4(3)	4(4)
mr5mcp	1	7(6)	–	7(7)	64(15)	–	26(13)	26(13)	7(7)
nsmge	1	82(17)	–	10(8)	35(14)	–	23(12)	23(12)	44(18)
obstacle	1	50(14)	–	11(11)	11(10)	–	15(14)	15(14)	11(11)
obstacle	2	fail	–	12(12)	12(11)	–	17(14)	17(14)	12(12)
obstacle	3	fail	–	17(11)	21(13)	–	14(13)	14(13)	17(11)
obstacle	4	2(1)	–	12(11)	23(16)	–	17(16)	17(16)	12(11)
obstacle	5	fail	–	7(7)	8(6)	–	8(7)	8(7)	7(7)
obstacle	6	fail	–	16(10)	16(9)	–	20(13)	20(13)	16(10)
obstacle	7	fail	–	12(10)	17(9)	–	17(12)	17(12)	12(10)
obstacle	8	fail	–	17(11)	9(6)	–	10(7)	10(7)	17(11)
opt_cont127	1	8(3)	–	6(6)	6(5)	–	27(12)	27(12)	6(6)
opt_cont255	1	fail	–	6(6)	6(5)	–	31(14)	31(14)	6(6)
opt_cont31	1	2(1)	–	6(6)	5(4)	–	11(9)	11(9)	6(6)
opt_cont511	1	fail	–	6(6)	6(5)	–	73(20)	73(20)	6(6)
pgvon105	1	fail	fail	64(16)	403(75)	fail	fail	fail	fail
pgvon105	2	33(14)	199(39)	27(10)	135(23)	213(30)	fail	fail	fail

Table A.2. Function and Jacobian Evaluations (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP- COMP	SEMI- COMP	SEMI- SMTH	SMTH
pgvon105	3	fail	153(32)	63(14)	3353(338)	322(40)	fail	fail	fail
pgvon105	4	fail	fail	fail	fail	fail	352(42)	fail	fail
pgvon106	1	fail	fail	772(101)	739(88)	fail	fail	fail	6428(482)
pgvon106	2	fail	fail	48(36)	fail	fail	fail	fail	109(37)
pgvon106	3	fail	fail	39(20)	fail	fail	fail	fail	233(49)
pgvon106	4	fail	fail	fail	86(28)	fail	412(65)	fail	fail
pgvon106	5	47(23)	–	fail	fail	fail	fail	fail	fail
pgvon106	6	fail	–	fail	fail	fail	fail	fail	58(27)
pies	1	3(2)	fail	13(13)	73(23)	54(49)	22(13)	22(13)	41(14)
sammge	1	1(0)	fail	1(1)	1(1)	fail	1(1)	1(1)	1(1)
sammge	3	4(3)	4(3)	6(4)	46(7)	4(3)	41(8)	fail	66(10)
sammge	5	14(6)	11(5)	11(6)	17(8)	11(5)	84(14)	fail	33(11)
sammge	6	4(3)	11(5)	9(5)	52(19)	11(5)	103(17)	fail	25(9)
sammge	7	6(4)	16(7)	5(5)	23(10)	16(7)	54(11)	fail	47(18)
sammge	8	4(3)	9(5)	9(5)	146(38)	9(5)	114(16)	fail	40(11)
sammge	9	17(7)	13(7)	5(5)	139(26)	13(7)	168(29)	fail	50(18)
sammge	10	1(0)	fail	1(1)	1(1)	fail	1(1)	1(1)	1(1)
sammge	13	4(3)	4(3)	17(6)	47(13)	4(3)	61(14)	fail	51(15)
sammge	14	4(3)	4(3)	16(6)	50(13)	4(3)	74(11)	fail	50(14)
sammge	15	4(3)	4(3)	5(5)	83(23)	4(3)	122(20)	fail	76(15)
sammge	16	4(3)	7(5)	8(8)	40(15)	7(5)	80(12)	fail	14(7)
sammge	17	4(3)	24(7)	6(6)	75(22)	43(7)	61(11)	fail	37(10)
sammge	18	4(3)	7(5)	8(8)	131(26)	7(5)	148(22)	fail	33(12)
scarfasum	2	12(7)	fail	5(5)	25(9)	73(26)	23(12)	23(12)	23(6)
scarfasum	3	5(4)	9(6)	8(6)	34(13)	9(6)	fail	fail	9(6)
scarfbnum	1	5(4)	70(20)	24(14)	164(46)	76(21)	241(51)	fail	71(20)
scarfbnum	2	5(4)	97(22)	25(15)	165(35)	58(19)	1497(341)	fail	95(24)
scarfbsum	1	4(3)	fail	462(57)	60(25)	26(22)	37(18)	37(18)	24(11)
scarfbsum	2	4(3)	fail	162(21)	1062(117)	157(83)	276(43)	fail	103(24)
threemge	7	6(5)	–	6(6)	fail	–	32(12)	32(12)	6(6)
threemge	8	6(5)	–	6(6)	fail	–	30(12)	30(12)	6(6)
threemge	11	6(5)	–	6(6)	fail	–	215(26)	fail	6(6)
transmcp	1	2(1)	fail	12(12)	92(26)	69(67)	193(105)	fail	24(15)
transmcp	2	1(0)	fail	1(1)	1(1)	fail	1(1)	fail	1(1)
transmcp	3	2(1)	2(1)	3(3)	3(2)	2(1)	15(8)	fail	4(3)
transmcp	4	6(5)	6(5)	3(3)	3(2)	6(5)	43(10)	fail	5(5)
two3mcp	1	6(5)	16(8)	6(6)	16(8)	16(8)	13(8)	13(8)	13(8)
two3mcp	2	5(4)	7(4)	5(5)	7(4)	7(4)	7(5)	7(5)	5(4)
unstmge	1	10(8)	10(8)	16(15)	11(8)	10(8)	11(9)	11(9)	8(7)
vonthmcp	1	fail	–	fail	fail	–	fail	fail	fail
vonthmge	1	18(14)	fail	34(22)	fail	fail	fail	fail	730(278)