Analyzing Mathematical Programs using MProbe

John W. Chinneck

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

July 28, 1999

Abstract

Just as modern general-purpose programming languages (e.g. C++, Java) are supported by a suite of tools (debuggers, profilers, etc.), mathematical programming languages need supporting tools. MProbe is an example of a suite of tools supporting a mathematical programming language, in this case AMPL. MProbe includes tools for empirically estimating the shape of nonlinear functions of many variables, nonlinearly-constrained region shape, the effect of the objective shape on the ability to find a global optimum, tools for estimating the effectiveness of constraints and for navigating through the model, among others.

1. Introduction

Mathematical programs can be very complex to formulate and solve, and there are many instances when a tool for analyzing the characteristics of the mathematical program, rather than simply expressing it or solving it, can be extremely helpful. Examples of situations calling for analysis include:

- you wish to select an appropriate solver for a nonlinear program, but in order to do so, you need to know whether the constrained region is convex or nonconvex, and whether the objective function shape is likely to limit you to a local optimum, or whether a global optimum is possible,
- you are solving a mixed-integer linear program and wish to order your integer and binary variables according to the number of constraints in which they appear,
- you wish to simplify a complicated nonlinear program, and wish to know which of the constraints are linear or close to linear,
- you are having difficulty solving a nonlinear formulation, and suspect that the solver may be stuck at a local optimum,

- you wish to reduce the number of constraints in the model, keeping only those that are effective in eliminating points from the feasible region, hence you need to analyze and rank the effectiveness of every constraint.

MProbe is a software tool for analyzing mathematical programs to answer questions of this sort. It has special strengths in analyzing the "shape" (convex? concave? etc.) of nonlinear functions of many variables which cannot be plotted and assessed by eye, and a long list of other capabilities. MProbe works with mathematical models specified in the AMPL language [Fourer et al 1993]. As currently implemented, AMPL must first be invoked to specify a model which is subsequently read and analyzed by MProbe. However, closer integration with AMPL or other modeling languages would be straightforward to realize, to the extent that the current MProbe can be considered a preview of formulation and analysis capabilities that might one day be a standard part of every effective modeling language.

Most modern computer languages (e.g. C++, Java) provide more than simple compilation; they are packaged with a suite of tools that assist in the process of producing good code. Common tools include a visual development environment, a debugger, and a profiler. In a similar vein, MProbe provides tools to accompany mathematical programming languages to assist in the mathematical programming process. In fact, the name "MProbe" derives from "*M*athematical *P*rogramming *Probe*".

MProbe belongs to the relatively recent body of work in computer-assisted analysis of mathematical programs best exemplified by Harvey Greenberg's ANALYZE software [Greenberg 1993]. ANALYZE provides numerous tools for the analysis of linear programs and mixed-integer programs, including a selection of model debugging tools, and works in conjunction with the MPS file specification of the model and various formats of solver output. MProbe provides a different selection of tools than ANALYZE, including tools for the analysis of nonlinear programs, and is more closely tied to the modeling language itself, in this case AMPL.

Consult Greenberg [1996] for a comprehensive bibliography of work in the area of computer-assisted mathematical programming, including the antecedents to ANALYZE and MProbe. See also the article by Chinneck and Greenberg [1999] for a background orientation to the field of "intelligent mathematical programming".

This paper summarizes the analytic algorithms used in MProbe, and its other capabilities and features. Section 2 introduces background concepts used in MProbe. Section 3 provides an overview of the tools in the suite, and Section 4 suggests which tools can be used for each class of mathematical program. Brief implementation details and conclusions follow.

Readers are invited to download a demonstration edition of MProbe from the World Wide Web at http://www.sce.carleton.ca/faculty/chinneck/mprobe.html.

2. Background

The state of the art in modeling languages for mathematical programming is quite advanced. Languages such as AMPL can express most forms of mathematical programs (linear programs, mixed-integer programs, nonlinear programs, etc.), and provide services such as presolvers, database interfaces, etc. The languages also provide an interface to a variety of solvers for the mathematical programs.

An analytic tool such as MProbe also needs expressive power and services similar to those provided by modern mathematical programming languages. Hence it is convenient to link MProbe directly with existing well-developed modeling languages rather than duplicating their features internally. A link between MProbe and the modeling language is straightforward to create based on the existing facilities for interfacing with solvers. See Section 5 for futher details.

A significant portion of the information returned by MProbe is obtained by sampling in the variable space of the mathematical model. The concept of assessing some properties of a mathematical program by random sampling is not new. For example, Boneh [1983] describes numerical experiments with a sampling algorithm named PREDUCE (for *P*robabilistic *Reduce*) which is primarily designed to assess redundancy of constraints, but also delivers information on boundedness, convexity, and dimensionality of the feasible region. Early versions of MProbe date to 1994, with the first public release in 1996.

The *variable space* of a mathematical program is the multidimensional "box" defined by the variables and their bounds. The quality of the result obtained by sampling is obviously affected by the size of the variable space; more accurate assessments are returned for smaller variable spaces. Analysts can improve accuracy in several ways:

- by providing the smallest initial variable bounds possible, based on knowledge of the application,
- by applying presolve routines which tighten the variable bounds before analysis begins (using the AMPL presolve routines invoked by MProbe),
- by directly reducing individual variable bounds to focus on a region of interest within the original variable space (via MProbe commands).

For computational reasons, unbounded variables are automatically assigned very large fixed bounds.

MProbe samples the variable space by scattering randomly placed line segments of random length. The placement of each line is determined by first generating two random endpoints in the variable space. Test points are then equally spaced along the interior of the line segment. Assume that there are l random line segments, each having i internal test points. As described in Section 3, information is obtained from the line segments in several ways:

- the test points and the line segment end points are sample points for function value and constraint effectiveness ($l\times(i+2)$ sample points),

- function "shape" samples are obtained at the test points by comparing actual function values with values interpolated from the function values at the line segment end points; see Sec. 3.1 for details ($l \times i$ sample points),
- function "slope" samples are obtained using the difference in function value at the end points and the length of the line; see Sec. 3.1.2 for further details (l sample points).

In practice, this sort of sampling procedure must deal with several kinds of sampling errors. The routines in MProbe handle the following errors:

- Mathematical errors (e.g. divide by zero) are handled automatically. The sample point is rejected, reducing the count of valid test points. When the number of mathematical errors is large (more than 15% of sample points), a warning is returned in place of any conclusions.
- Line segments that are too long are rejected as "line errors" because they invalidate the slope estimates. This reduces the number of valid sample points.
- Too few valid sample points: a warning is returned in place of any conclusions.
- Too few test lines: a warning is issued.

3. MProbe Analytic Capabilities

This section briefly reviews the major analytic capabilities available in MProbe.

3.1 Analyzing Function Shape

Function shape refers to the form that the function assumes in the variable space. The *algebraic shape* (e.g. linear, quadratic, general nonlinear, etc.) is deduced from the algebra of the mathematical formula expressing the function, and hence is not specific to any particular region in the variable space. In contrast, the *empirical shape* (convex, concave, both convex and concave, etc.) refers to the shape actually taken by the function in some specific region of interest. The distinction is important. For example, it may happen that a function has a highly nonlinear algebraic shape, yet within the region of interest it is actually linear, which permits much simpler modeling and solution.

The assessment of the algebraic shape of functions is a service provided by many modeling languages because this information is needed by many of the nonlinear solvers that the modeling language calls. Hence MProbe simply displays the algebraic shape as deduced by the modeling language (AMPL can classify functions as linear, quadratic, or general nonlinear), but performs its own analysis of individual functions to arrive at an estimate of the empirical shape.

The method that MProbe uses to assess empirical shape must be effective for functions of many variables. This rules out simple plotting and visual inspection, which is practical only for functions of one or two variables. MProbe uses the basic definitions of convexity and concavity in conjunction with sampling using the random line segments described above. A number of test points are evenly spaced along the interior of each random line segment. Two values are calculated at each test point: (i) the interpolated

value of the function based on the function values at the two line segment endpoints, and (ii) the actual function value at the test point. Now the difference between (i) and (ii) is calculated. A positive value indicates convexity, while a negative value indicates concavity. Consult a basic textbook on nonlinear programming (e.g. Ecker and Kupferschmid [1988], p. 295) for more information on this simple test of function shape.

A large number of line segments and test points are generated, and the difference results are collected in a histogram. Binary and integer variables are treated as real-valued during these tests. An evaluation of the histogram allows MProbe to arrive at a conclusion about the shape of the function within a specified region. As required, this method is effective for functions of many variables. Histograms are generated at the same time for function values, function "slope" estimators, and the lengths of the test lines. In addition, if the function is a constraint, data are collected about whether or not the constraint is satisfied at each sample point.

Consider the simple nonlinear function *singularity*, whose equation is:

$$z2 - z1/(1 - z2/2) \leq 100, \text{ where } -10 \leq z1,z2 \leq 10$$

The shape histogram for this function over the specified variable space is shown in Fig. 1. A concluding statement about the estimated empirical shape is returned as described below, but the shape histogram is also useful in assessing the extent of the shape. Fig. 1 shows that the *singularity* function has regions that are extremely concave as well as regions that are extremely convex. This function is not a good candidate for linearization in the tested region, for example.

Two special tolerances are used in assessing function shape. As is common practice, the *equality limit* provides a small interval within which two numbers are deemed to be essentially equal. The *"almost" limit* provides a larger interval in which two numbers are deemed to be "almost equal". For example, the equality limit in Fig. 1 is ±0.0001 and the "almost" limit is ±0.1. The user can set both tolerances.

The possible *empirical shape* estimates returned by MProbe are listed below. "Difference" refers to the difference between the interpolated value of the function on a test line and the actual value of the function at the same point.

- **linear**: all differences are within the equality tolerances (negative and positive).
- **convex**: all differences are above the negative equality tolerance and at least one is above the positive "almost" tolerance.
- **convex, almost linear**: all differences are above the negative equality tolerance, at least one is between the positive equality tolerance and the positive "almost" tolerance, and none are above the positive "almost" tolerance.
- **almost convex**: at least one difference is between the negative "almost" tolerance and the negative equality tolerance, and at least one difference is above the positive "almost" tolerance
- **concave**: all differences are below the positive equality tolerance, and at least one is below the negative "almost" tolerance.
- **concave, almost linear**: all differences are below the positive equality tolerance, at least one is between the negative equality tolerance and the negative "almost" tolerance, and none are below the negative "almost" tolerance.

- **almost concave**: at least one difference is between the positive "almost" tolerance and the positive equality tolerance, and at least one difference is below the negative "almost" tolerance.
- **convex and concave**: at least one difference is above the positive "almost" tolerance, and at least one difference is below the negative "almost" tolerance.
- **convex and concave, almost linear**: at least one difference is between the positive equality tolerance and the positive "almost" tolerance, and at least one difference is between the negative equality tolerance and the negative "almost" tolerance.
- **not a function**: there is more than one possible function value for a given set of variable values.
- **excessive math errors**: evaluating the function at various points yielded so many mathematical errors that the empirical shape evaluation is not reliable.
- **error in shape finder**: unspecified errors occurred in the shape analysis.

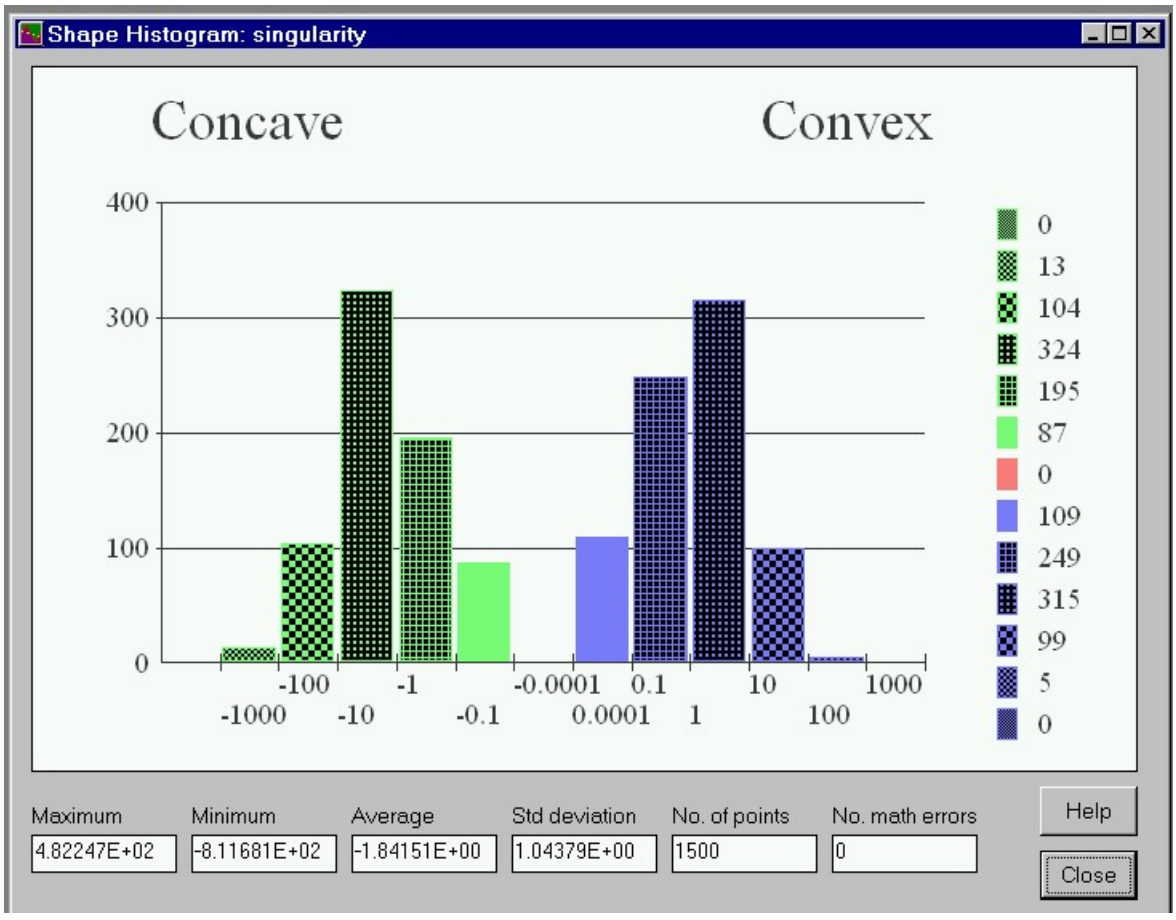

Figure 1: Shape histogram for the *singularity* constraint.

As shown in Fig. 2, the empirical shape of the *singularity* function over the specified variable space is estimated by MProbe to be "convex and concave". What is interesting

for mathematical programmers is that the empirical shape may differ in different variable spaces. For example, the empirical shape of the *singularity* function is estimated to be "convex and concave, almost linear" in the variable space $-10 \leq z1 \leq 10$, $-10 \leq z2 \leq -9$. This opens the possibility of using simpler models of functions depending on the variable space of interest. The degree of simplification can be surprising. In research at Carleton University, various lengthy nonlinear functions expressing transistor behavior (hundreds of lines of AMPL code each) were found by MProbe to be almost linear in shape.

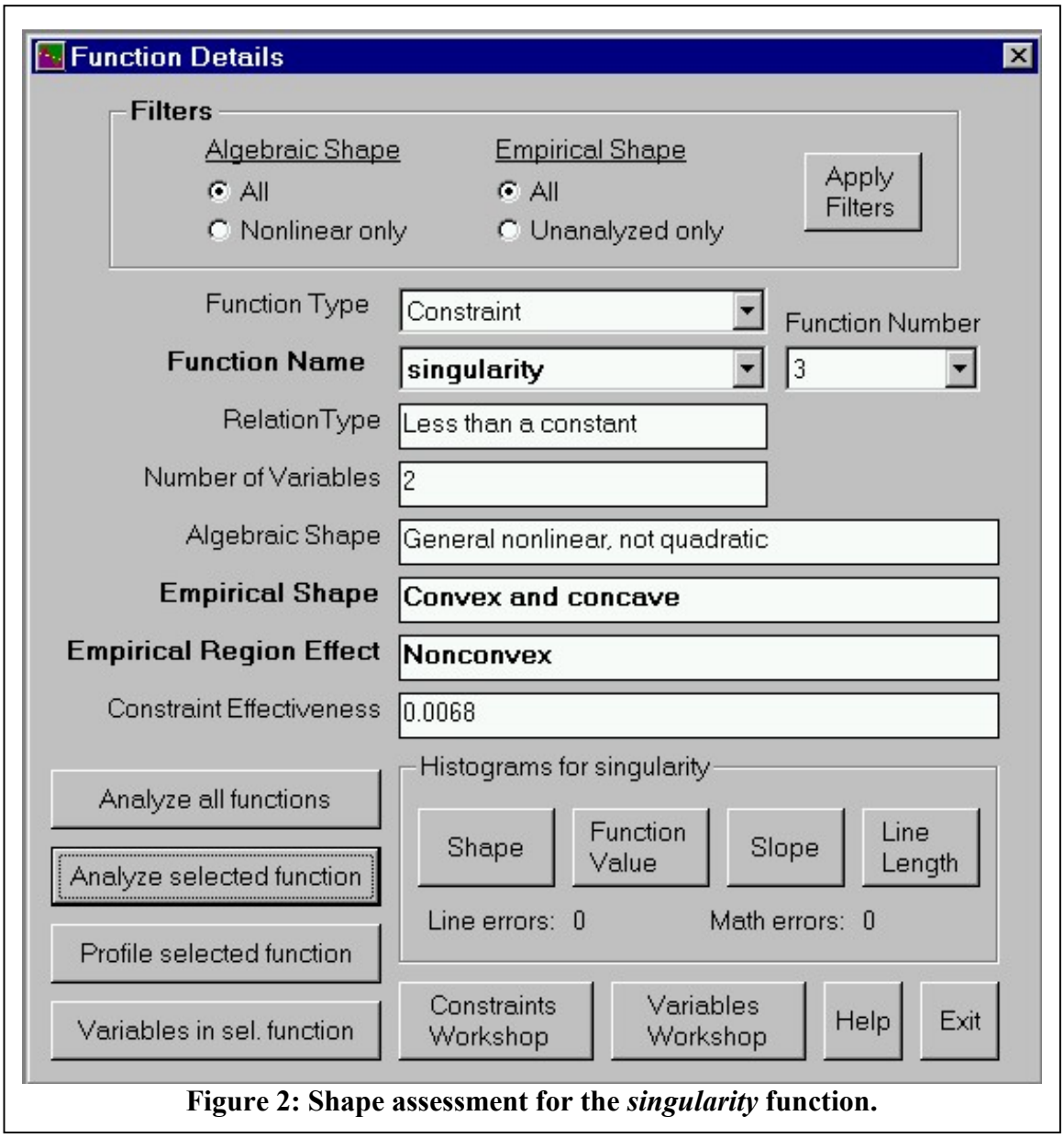

Figure 2: Shape assessment for the *singularity* function.

3.1.1 Function Value Distribution

Because numerous points on the function surface are sampled while assessing the empirical shape of a function, it is simple to construct a histogram of function values at the same time. MProbe allows the user to specify the cell limits, and displays a histogram similar to the shape histogram in Fig. 1, along with other useful data output with all MProbe histograms: maximum value, minimum value, average value, and the standard deviation.

3.1.2 Function "Slope" Distribution

The concept of function slope is intuitive for functions of one or two variables that can be readily plotted and viewed. For functions of higher dimensions, the only available analytic tool is the gradient, which does not have a direct intuitive meaning for most analysts. MProbe introduces and uses a simple slope analog that is valid for functions of many variables. For a random line segment in a multidimensional variable space, the "slope" is the difference in the function value at the end points, divided by the length of the line segment. The absolute value of this quantity is used since the sign has no particular meaning for functions of many variables.

The intuitive meaning of this slope analog is obvious: how much the function value changes over a distance. Large values indicate a "steep" change in the function, while small values indicate a "flat" region. Sample points for the slope analog histogram are collected at the same time as data points for the shape histogram. The resulting slope histogram is useful in assessing whether the multidimensional function is relatively steep or relatively flat in the region of interest. This can be especially helpful in assessing the region around a reported optimum point. If this region is very flat for example, then there may be multiple optima, or alternatively, it may be worth restarting the solver far away from this region to see if a better optimum can be found.

3.1.3 Function Profiling

It is often revealing to see a plot of a function, but this is difficult for functions of many variables. MProbe handles this difficulty by providing plots of a function along a straight line between two arbitrary points in n-space, as illustrated for the *singularity* function in Fig. 3. Note that the function profile in Fig. 3 is along a line segment from Point A in (*z1,z2*) space to Point B in (*z1,z2*) space; it is not simply a plot along the *z1* axis as might be inferred from the displays in the drop-down lists under *Point A* and *Point B*. This one-dimensional plot which cuts through n-space is referred to in MProbe as a *profile* to distinguish it from a full-dimensional plot. The horizontal axis gives the point number for the evenly spaced points along the line segment from point A to point B. The vertical axis gives the function value. If there are mathematical errors along the profiling line, the user is alerted and has the opportunity to recover the points for use in off-line analysis.

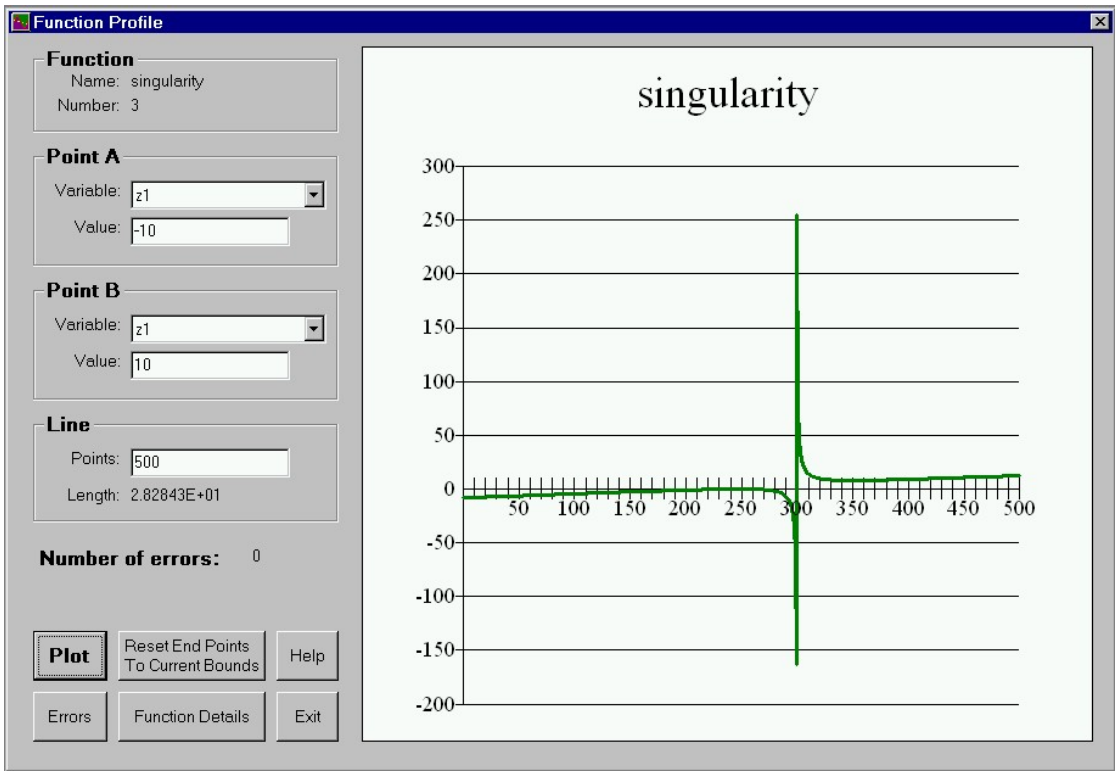

Figure 3: Profile for the *singularity* function.

The line segment end points can be arbitrarily set. Binary and integer variables are again treated as real-valued for the purposes of function profiling.

3.1.4 Test Line Length Distribution

MProbe also collects a histogram of the lengths of the random line segments used in assessing function shape because test line length affects the assessment of the slope analog. Consider a function which is essentially flat but which has numerous small hills and valleys. A scattering of very short test lines may show the function to have some steep "slopes" because a modest change in function value divided by the length of a very short test line may produce a large "slope". On the other hand, a scattering of very long test lines over this function will always produce very small "slopes". These ideas are related to the concept of scaling, and are slated for further development in future versions of MProbe.

3.2 Constraint Effectiveness

The *effectiveness* of a constraint is defined as the fraction of the variable space that it eliminates. This is a useful concept for inequality constraints. Statistics on constraint effectiveness are collected during the random sampling process used to estimate empirical function shape. For example, a constraint effectiveness of 75% is reported if 75% of the tested points did not satisfy the inequality constraint. An inequality constraint effectiveness of 0% indicates that the constraint has no impact on the feasible region, and

can be dropped from the model. An effectiveness of 100% indicates that none of the sampled points in the variable space satisfy the constraint, hence the model is infeasible. To achieve feasibility in this case, either the constraint must be modified, or the variable space must be changed.

Constraint effectiveness is not meaningful for equality constraints. A sampling approach would usually find all equality constraints to be 100% effective due to the small probability of hitting on a combination of variable values that satisfied the equality to within acceptable tolerances. Instead, we can only estimate whether or not it is possible to satisfy the equality within the current variable space. MProbe does this as follows. Each sample point is assigned to one of three categories:

- LT: function value is less than the right hand side,
- EQ: function value is equal to the right hand side within the equality tolerances,
- GT: function value is greater than the right hand side.

It is possible to satisfy the equality constraint in the current variable space in two cases:

- at least one point is in the EQ category, or
- at least one point is in the LT category and at least one point is in the GT category. This shows that at least one point satisfying the equality constraint exists in the variable space: it will lie on a line connecting a point in the LT category to a point in the GT category.

If neither of these two conditions is met (i.e. if all points are in the GT category, or if all points are in the LT category), then MProbe concludes that it is not possible to satisfy the equality constraint in the current variable space.

Knowing the constraint effectiveness can be useful in determining which constraints to eliminate in simplifying a model. Highly effective constraints should be retained, but constraints with low effectiveness estimates may be candidates for elimination from the model. The constraint effectiveness can also be used to set the order of the constraints in a constraint logic programming model: placing the more effective constraints earlier in the constraint list usually results in a smaller search tree.

Fig. 2 shows that the *singularity* constraint is a good candidate for elimination from the model due to its low effectiveness.

3.3 Analyzing Objective Function Effect

The sense of an objective function (maximize or minimize) coupled with its empirical shape (convex, concave, linear, etc.) determines whether a global optimum is likely to be found, or whether a local optimum is probable. MProbe returns the following *empirical optimum effect* outcomes for an objective:

- **Global optimum possible**: given for (i) linear objective, (ii) convex objective to be minimized and (iii) concave objective to be maximized.
- **Local optimum likely**: given for (i) convex (or almost convex) objective to be maximized, (ii) concave (or almost concave) objective to be minimized, (iii) convex and concave objective.

- **Local, almost global**: given for (i) almost linear objective, (ii) almost convex objective to be minimized, (iii) almost concave objective to be maximized.

3.4 Analyzing the Shape of a Constrained Region

Taken together, the empirical shapes of the constraints affect whether or not the constrained region is convex or not. MProbe estimates the *empirical region effect* for each constraint as it is analyzed (see Fig. 2, for example). One of three possible empirical region effects is assigned to each constraint:

- **Convex**: the constraint contributes to a convex constrained region. This is given for (i) any linear constraint, (ii) convex inequalities of less-than-or-equal-to type, (iii) concave inequalities of greater-than-or-equal-to type.
- **Almost convex**: the constraint contributes to a convex constrained region except for a minor nonconvexity within the "almost" tolerance. This is given for (i) almost linear equality constraints, (ii) almost convex inequalities of less-than-or-equal-to type, (iii) almost concave inequalities of greater-than-or-equal-to type.
- **Nonconvex**: the constraint contributes to a nonconvex constrained region. This is given for all constraints whose empirical region effect is not "convex" or "almost convex".

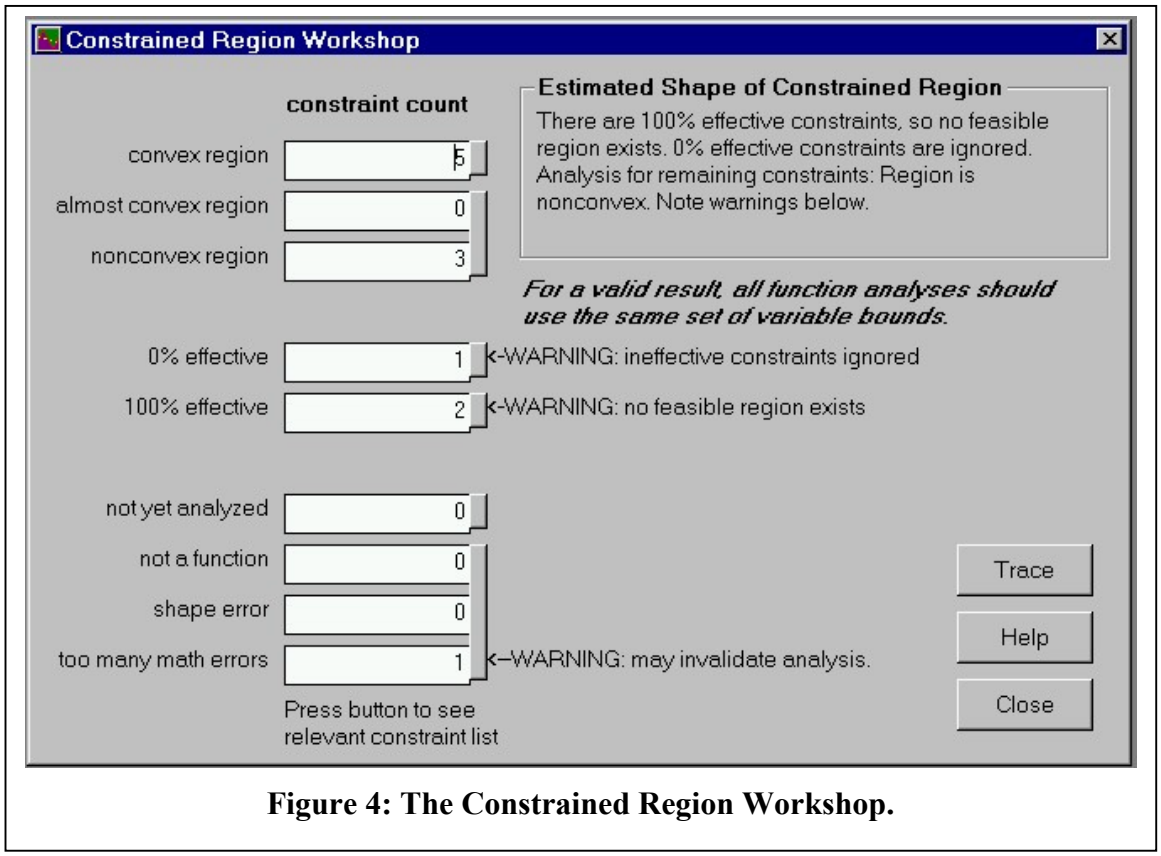

Figure 4: The Constrained Region Workshop.

The overall effect of the individual constraint region effects is summarized in the *Estimated Shape of Constrained Region* box in the *Constrained Region Workshop* window, as illustrated in Fig. 4. Note that the *constrained region* is not to be confused with the *feasible region*. If it exists, the feasible region is normally a subset of the constrained region. The shape of the feasible region affects the progress of a solver towards the optimum once a feasible point has been found, but the shape of the larger constrained region affects the progress of the solver towards finding a feasible point. Note in particular that it is entirely possible for MProbe to report a nonconvex constrained region when a convex feasible region exists.

The Constrained Region Workshop comments on feasibility only when there are 100% effective constraints. In this case it is known that the model is infeasible. Further tools for the analysis of feasible regions are under development.

3.5 Filtering and Sorting Variables and Constraints

A huge volume of information is available in large models, so it is helpful to be able to selectively view and sort subsets of the information in order to focus on the material that is important to the question at hand. MProbe provides tools for filtering and sorting of the information about variables and constraints. *Filtering* refers to the selection of a subset of the information for viewing, based on specified criteria. For example, you may choose to display only constraints that contain only binary variables; these so-called "multiple choice" constraints are often the cause of modeling difficulties in mixed-integer problems. The Constraints Workshop includes a long list of constraint filtering criteria based on algebraic shapes, empirical shapes, types of variables, constraint effectiveness, etc. The Variables Workshop includes a similar list of filtering criteria appropriate for variable information.

Fig. 5 shows the spreadsheet-like display of information found in the Constraints Workshop. The constraints in Fig. 5 have been filtered so that only those having a nonlinear algebraic shape are shown.

Information about variables and constraints can also be sorted in user-defined ways. In Fig. 5, the constraints have been sorted in descending order by constraint effectiveness. This may be useful in determining the ordering of constraints for a constraint logic program for example. Similar sorting of variables can be done, based on data such as type (real, integer, binary) or the number of functions in which the variable appears. The latter sort criterion can be important in creating Special Ordered Sets of variables in mixed-integer linear programs. Subordering is also possible by selecting a subset of the data and re-sorting.

3.6 Other Features

MProbe provides various other tools of use to mathematical programmers:

- A *Statistics* window summarizes the numbers of variables, constraints, and objectives in various categories (e.g. number of binary variables, number of general nonlinear constraints).
- Simple navigation of the model is available via the *Variables Workshop* and the *Constraints Workshop*. You can view all of the constraints that contain a specified variable, or all of the variables that are used in a specified function.
- A text file trace of an analysis session can be written, viewed, and saved.

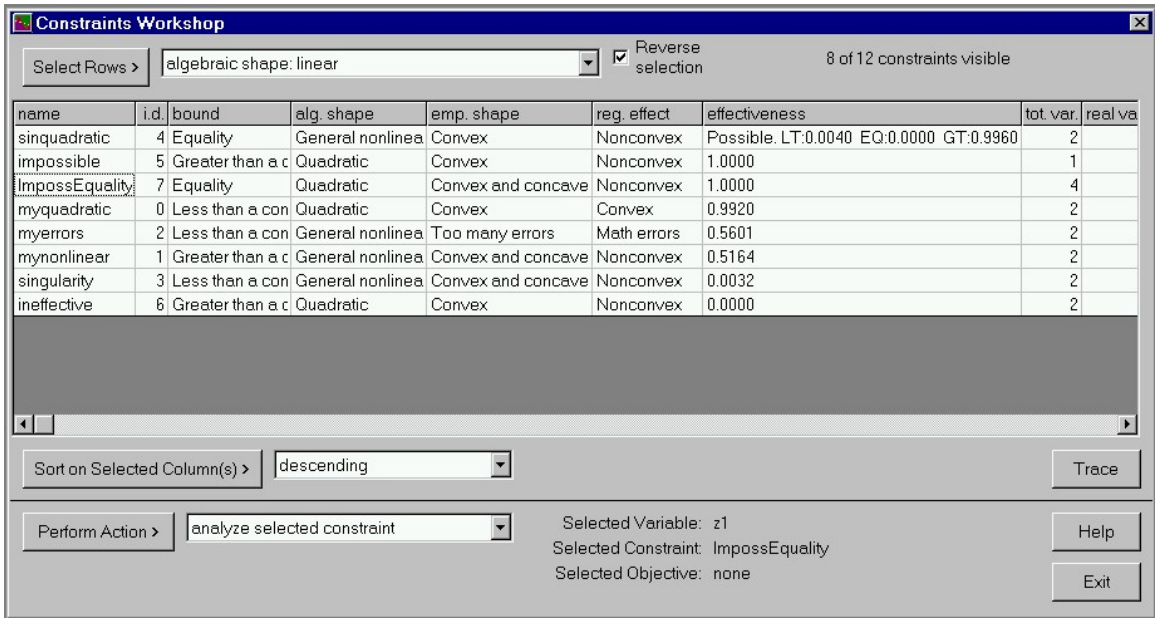

Figure 5: Constraints filtered and sorted.

4. Analyzing Mathematical Programs using MProbe

MProbe can be used with all of the common forms of mathematical programs. Tools that are useful with all forms of mathematical program include:

- listing of statistics about the model,
- simple navigation of the model in the *Variables Workshop* and *Constraints Workshop*,
- analysis of constraint effectiveness,
- ability to filter and sort constraints and variables in user-defined ways,
- optional generation of a trace file record of the analysis session.

While the analytic and empirical function shapes are all known for linear programs, analysts may still be interested in the estimates of constraint effectiveness and the various display and navigation tools listed above.

There is a wider range of tools that are of special interest for nonlinear programming, including:

- analysis of constraint and objective empirical shape in the *Constraints Workshop* and the *Function Details* window, including the detailed histograms of sample points relating to function shape, function value, "slope", and test line length,
- estimation of the shape of the constrained region in the *Constrained Region Workshop*,
- estimation of the effect of the objective function empirical shape on the ability to find a global optimum in the *Function Details* window,
- *function profiling* between any two points in n-space.

Integer, binary, and mixed-integer programmers can use the following techniques:

- sorting the constraints in decreasing order by number of integer, binary (or both) variables, in the *Constraints Workshop*,
- setting up Special Ordered Sets by sorting the integer and binary variables in decreasing order by number of constraints in which they appear,
- identification of "multiple choice" constraints by appropriate filtering in the *Constraints Workshop.*

Constraint logic programmers can sort the constraints in decreasing order by constraint effectiveness. It is usually a good idea to order the constraints in this manner so as to reduce the size of the search tree by encouraging early failure of search paths.

5. Implementation Details

There are three main software elements in MProbe: (*i*) AMPL (32-bit DOS executable), (*ii*) the AMPL solver interface library (32-bit C language DLL), and (*iii*) the MProbe interface (32-bit Visual Basic program). The AMPL solver interface library, available from Bell Labs [Gay 1997], is augmented with additional C routines that return data in the specific formats required by MProbe.

MProbe interacts with the AMPL model description as follows. MProbe first invokes AMPL, allowing the user to create or recall a model. Entering a keyword in the AMPL window causes it to write the model files and close. Thereafter, MProbe makes DLL calls to the modified AMPL solver interface library, which in turn operates on the model files to obtain information about variables and constraints, to evaluate function values at a given point, etc. Other types of interfaces, such as file-passing or object calls, are possible, and may prove more convenient for interaction with other modeling languages.

Because the MProbe interface is compiled Visual Basic, it runs only under the Windows 95/98 or Windows NT (version 3.51 service pack 5 or later, version 4.0 preferred) operating systems. A full help system is included. A fully functional demonstration edition of MProbe version 2.1 is available for download from the Web at http://www.sce.carleton.ca/faculty/chinneck/mprobe.html.

6. Conclusions

MProbe provides a suite of tools that assist in the process of mathematical modeling. These tools complement the mathematical programming language which expresses the model in much the same way that tools such as visual development environments, debuggers, and profilers complement general purpose programming languages such as C++ or Java. MProbe can empirically estimate the shapes of nonlinear functions of many variables and of nonlinearly-constrained regions, estimate the effectiveness of constraints, and estimate the effect of the objective shape on the ability to find a global optimum. Many of the tools in MProbe are original:

- the method of estimating function "slope",
- the estimator for constraint effectiveness,
- use of the equality and "almost" tolerances to arrive at meaningful statements about the shape of nonlinear functions of many variables,
- the filtering and sorting facilities for information about variables and constraints,
- the collection of information about the shapes of individual functions to arrive at a statement about the shape of the constrained region.

It is reasonable to expect that analytic features of the type demonstrated in MProbe will see gradual adoption into the modeling environments accompanying mathematical programming languages.

Acknowledgements

David Gay (Bell Labs, Lucent Technologies), the main author of AMPL, assisted greatly by making key modifications to the AMPL-solver interface library to permit easy connection with MProbe. He was also a skillful beta-tester. Rania Awad (formerly of Carleton University) provided an excellent user interface for the original release of MProbe.

References

A. Boneh (1983). *PREDUCE: a Probabilistic Algorithm Identifying Redundancy by a Random Feasible Point Generator (RFPG)* in **Redundancy in Mathematical Programming: A State of the Art Survey,** Lecture Notes in Economics and Mathematical Systems No. 206, pp. 108-134, M. Karwan, V. Lotfi, J. Telgen, S. Zionts (eds.), Springer-Verlag.

J.W. Chinneck, H.J. Greenberg (1999). *Intelligent Mathematical Programming Software: Past, Present, and Future*, **INFORMS Computing Society Newsletter**, April. Also appeared in the **CORS Bulletin**, vol. 33, no. 2, April.

J.G. Ecker, M. Kupferschmid (1988). **Introduction to Operations Research**, John Wiley & Sons, New York.

R. Fourer, D.M. Gay, B.W. Kernighan (1993). **AMPL: A Modeling Language for Mathematical Programming**, Duxbury Press/Wadsworth Publishing Company.

D.M. Gay (1997). *Hooking Your Solver to AMPL*, technical report, Bell Laboratories, Lucent Technologies, Murray Hill, NJ, USA. Routines are available via the World Wide Web at http://netlib.bell-labs.com/netlib/ampl/solvers/

H.J. Greenberg (1993). **A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE**, Kluwer Academic Publishers, Boston.

H.J. Greenberg (1996). *A Bibliography for the Development of an Intelligent Mathematical Programming System*, **Annals of Operations Research** 65, pp. 55-90. A 1997 update is at http://www.cudenver.edu/~hgreenbe/imps/impsbib/impsbib.html.