Discovering the Characteristics of Mathematical Programs via Sampling

John W. Chinneck
Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada
email: chinneck@sce.carleton.ca

June 13, 2000

Abstract

It is often important to know more about the characteristics of a mathematical program than the simple information that is returned by a solver. For example, to choose an appropriate solution algorithm, you may need to know something about the convexity of the functions in a nonlinear program, or you may like to know which constraints are redundant. For the more complex model forms, particularly nonlinear programs and mixed-integer programs, random sampling can discover a great deal of information. This paper describes how to discover interesting characteristics of mathematical programs via sampling, and describes solutions to several difficulties that arise in practice. Several new techniques for discovering characteristics and for improving the accuracy of the characterizations are described.

1. Introduction

Questions about the characteristics of mathematical programs arise often during the formulation and post-solution phases. This is especially true in the more complex model forms such as nonlinear programs and mixed-integer programs. For example, you may wish to know whether your nonlinear program can be simplified by approximating some of the nonlinear constraints by simpler quadratic or linear forms. You may also wish to know whether a solver is likely to reach the global optimum of your nonlinear program. In any form of mathematical program, you may wish to identify and delete redundant constraints, or to identify and concentrate on accurately modeling those constraints that are particularly effective in removing points from the feasible region.

Mathematical programs have proliferated and grown larger and more complex in recent years, due to the wide availability of inexpensive computing power and the increased need for computer control of complex operations. Experts are creating more sophisticated mathematical programs, but a host of novices are also creating models, enticed via the ready availability of

solvers, which can be found even in ubiquitous spreadsheet programs. Both the experts and the novices frequently need assistance in developing and understanding their models. A good deal of such assistance could be provided by special-purpose algorithms and computer tools, in the same way that computer programmers in standard languages such as C are assisted by debugging tools and visual environments. Unfortunately, very few useful tools are available.

ANALYZE [Greenberg 1993] and MProbe [Chinneck 1999] are perhaps the only readily available general-purpose tools for probing mathematical programs. ANALYZE provides a number of analytic tools that can be used with linear and mixed-integer programs. MProbe provides tools suitable for all forms of mathematical programs, and has special strengths in analyzing nonlinear programs. A number of the tools in MProbe use a sampling technique to discover characteristics of the mathematical model. The purpose of this paper is to describe these sampling techniques and to show how they provide useful information about mathematical programs.

There is relatively little previous work on sampling as a method of discovering information about mathematical programs. Previous work concentrates mostly on the discovery of redundancy. Boneh's groundbreaking PREDUCE system [Boneh 1983] is mainly for the identification of redundant constraints, but it also discovers several general characteristics of mathematical programs, such as boundedness, convexity, and the dimensionality of the feasible region, and information on the size of the facets of the feasible region and the bounds on the variables.

More recent work develops the concept of *hit-and-run* methods for sampling inside a polytope, or more generally inside any convex full-dimensional enclosure (see for example Berbee et al. [1987], Boneh et al. [1993] and Bélisle et al. [1998]). Briefly, hit-and-run methods launch rays from the interior of a full-dimension feasible region to create line segments that span the feasible region. The first constraint hit by a ray must be necessary. Subsequent rays can be launched from points along the spanning line segment, since points on the spanning line segments are known to be feasible. See section 3 for more details.

In the *stand-and-hit* variation of hit-and-run, all of the rays are launched from the same feasible point on the interior of the full-dimension feasible region. The question then arises as to the best point to choose as the source of the rays. Here the idea of the *prime analytic center* comes into play. The prime analytic center is, by various definitions, the approximate center of the feasible region, and is a good choice for the source of the rays (see Caron, Greenberg, and Holder [1999]).

Finally, Feng [1999] has used random sampling to analyze infeasibility in inequality-constrained linear programs. The idea is to collect data on which constraints are satisfied or violated at various random points. A set covering problem is then solved to find the minimum number of constraints which together render all of the sample points infeasible; the members of this set constitute an *Irreducible Infeasible Set (IIS)*.

2. Characteristics Discovered by Sampling

Assume for the moment that we are able to bracket the region of interest (normally the feasible region, if one exists) reasonably closely by specifying upper and lower bounds on the variables

in the mathematical program. This creates a *sampling enclosure* in the shape of a *box*. Uniform sampling inside the enclosure, simple to do when it is a box, permits the discovery of many important characteristics of the mathematical program, as described in following subsections. The use of more advanced sampling enclosures is introduced in Section 3. Constraint functions have the general form f(***x***) {≤,≥,=} *b*, where *b* is a constant.

2.1 Function Shape

While formulating a nonlinear program, it is essential to know the *shape* of the constraint and objective functions, where the shape might be linear, convex, concave, etc. Knowledge of the shapes of the functions affects the choice of solver. It is easy to check the shape of functions of one or two variables by visual inspection of a plot, but more sophisticated methods are required for functions of higher dimension. As pointed out by Pardalos [1994], "there is no known computable procedure to decide convexity", let alone the other shape possibilities. However, it is possible to gain a good idea of the shape, and the extent of the shape (e.g. highly concave or just barely concave) via a sampling procedure.

Convexity and concavity of functions are defined as follows. Connect any two points in the variable space. You can construct an interpolated value of the function at any point on the line segment connecting these two points. A function is convex if the interpolated values at all points of every such line segment have function values that are greater than or equal to the actual function value at the same point. A function is concave if the interpolated values at all points of every such line segment have function values that are less than or equal to the actual function value at the same point.

These basic definitions also provide a way of estimating function shape via sampling. The variable space for a function is sampled in a region of interest by randomly scattering line segments, and then comparing the interpolated function value at various points on the line segment with the actual value of the function at the same point. Defining the difference as (*interpolated function value*) – (*actual function value*), convexity will show as a positive difference, concavity as a negative difference, and linearity as zero difference. Random line segments are constructed by connecting two uniformly distributed random points in the sampling enclosure. Difference calculations are made at a specified number of points arranged at fixed intervals along the line segment, as illustrated in Figure 1.

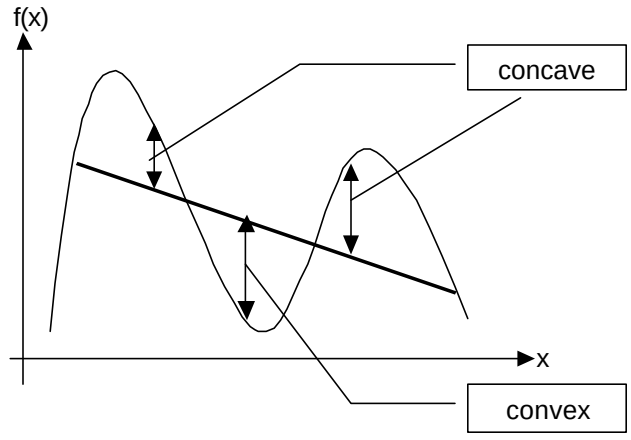

Figure 1: Function shape is assessed via difference measurements along the line segment.

The difference information collected over a large number of random samples is presented as a histogram. This provides useful information on the extent and range of convexity and concavity. For example, a histogram having numerous very large negative values and also numerous very large positive values indicates a function that is highly nonlinear. Another histogram having mostly positive values with only a few small negative values is one that is mostly convex and only slightly concave; it may be a good candidate for approximation by a convex function.

Different thresholds can be used to help identify candidates for simplification. As is necessary in numerical calculations, a small tolerance is needed in any assessment of the equality of two floating-point numbers. A difference histogram with all entries within ± equality tolerance indicates a completely linear function. A somewhat larger tolerance helps to identify functions that are "almost" concave or "almost" convex. Functions whose difference histograms include values in these regions may be candidates for simplification.

Specifying just two tolerances, ± equality tolerance, and ± "almost" tolerance, it is possible to discern several classes of function shape based on histograms collected via random sampling of a function. As a case in point, the shape classes distinguished by the MProbe software are as follows:

- *linear:* all differences are within the equality tolerances.
- *almost linear:* all differences are within the "almost" tolerances.
- *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:* all differences are between the negative "almost" tolerance and the positive "almost" tolerance, and 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.

The sampling approach is the only way to assess the shape of general multivariable nonlinear functions. Analytic methods can only be used in rare instances such as the constant Hessians associated with quadratic functions.

An added advantage is that the empirical shape, as assessed by the sampling method, is much more specific than any analytic information. For example, a particular function may have a very difficult nonlinear algebraic form, but it may happen that in the region of interest (as specified by the sampling enclosure), the function has a simpler form (e.g. linear or convex). This may permit the use of a more effective solver. Empirical evaluation via sampling is valuable even when analytic information about function shape is available.

2.2 Function Value

A large number of points in the sampling enclosure are visited in the course of the function shape evaluation described above. Since the function is evaluated at each of these points, it is simple to collect this information into a histogram and to collect simple statistics such as minimum and maximum value, mean, and standard deviation. This gives an idea of the range of the function inside the sampling enclosure.

2.3 Function Slope

The concept of function slope is straightforward for one-dimensional functions: the change in function value divided by the change in the single variable. A large value indicates a steep function while a small value indicates a flat function. Such a simple concept is not directly available for multidimensional functions that instead rely on the idea of the gradient, a vector concept of the direction of increase of the function.

However, a simple measure of the "flatness" (or "slope") can be constructed for multidimensional functions and can be measured over a large number of samples. This can be done at the same time as collecting data for assessing the function shape because it also relies on uniform sampling via line segments. For each random line segment, calculate |(*difference in function value between line segment end points*)/(*length of line segment*)|. Note that this captures the essential idea of slope: the change in the function value over an interval. The absolute value is used because the sign has no special meaning in a multidimensional space.

As for the other measures, the random samples of multidimensional slope are collected in a histogram. Many large values indicate a very steep function that has dramatic changes in value over very short distances. In contrast, a histogram having all of the slope instances in the smallest cells indicates a very "flat" function. This information has obvious uses in assessing the flatness of a nonlinear objective function in the region surrounding a suspected optimum point, but is also useful in assessing nonlinear constraints.

2.4 Line Segment Length

The length of the line segments used in sampling constraint shape and slope affects the results in a manner similar to scaling. Consider a function that is very flat, but which has many very small peaks and valleys. Long line segments will result in very small slope samples, but short line

segments may show relatively large slope values. Consider also a quadratic with a very gentle curve over a long distance within the sampling enclosure: using only short line segments may make this function appear to be linear.

In the current MProbe implementation, line lengths are random, but experiments with fixed line lengths to assess scaling issues are planned. A histogram of line segment lengths is currently assembled during the sampling phase.

2.5 Constraint Effectiveness

Some constraints are much better than others at excluding points from the sampling enclosure. Inequality *constraint effectiveness* is defined here as the fraction of the sample points that violate the inequality. If an inequality has an effectiveness of 0.75, then 75% of the points sampled in the sampling enclosure violated the constraint. From a modeling perspective, this is an interesting measure: if you need to simplify the model by omitting constraints, omit the inequalities that have low effectiveness.

It is also possible to make observations about overall feasibility (see also Section 2.9). If a constraint has an effectiveness of 1.0, then no sample points in the sampling enclosure satisfied the constraint. This indicates that the overall model is likely infeasible.

It is not possible to use the same approach for equality constraints because the probability of satisfying an equality constraint at a random point is almost zero. Instead, three fractions are reported: fraction of points at which the functional value is less than the constant, fraction of points at which the functional value is greater than the constant, and the fraction of points at which the functional value is equal to the constant (within the equality tolerances). The effectiveness is reported as "possible" in either of two cases: (i) the fraction of points at which the functional value is equal to the constant is greater than zero, or (ii) the fraction of points at which the functional value is less than the constant is greater than zero, and the fraction of points at which the functional value is greater than the constant is also greater than zero. This second case implies that there must exist at least one point in the sampling enclosure at which the constraint is exactly satisfied. If the functional value is greater than the constant at all sample points, or the functional value is less than the constant at all sample points, then the effectiveness of the equality constraint is reported as 1.0.

There may be analytic value in looking at the ratio of the two fractions (functional value less than constant vs. functional value greater than constant). A linear equality constraint that passes through the center of the sampling enclosure should show these two fractions in about a 1:1 ratio, whereas a linear equality constraint that passes through the edge of the sampling enclosure will have an unbalanced ratio. The reasoning is a bit different for nonlinear constraints, though the main idea is the same: equality constraints having a balanced ratio of the two fractions are probably easier to satisfy in the sampling enclosure.

2.6 Constraint Region Effect

A convex set of points is defined as one in which the straight line connecting any two points in the set is contained entirely within the set. Knowing whether the constraints form a convex set is

vital in choosing a nonlinear solver. The feasible region (if one exists) that is created by considering all of the constraints simultaneously may or may not be convex. A first step towards assessing the convexity of any feasible region is to consider the contributing effect of each individual constraint. Once the shape of each constraint is estimated via sampling, the individual "constraint region effect" is deduced from the empirical function shape and the sense of the constraint (≤,≥,=), as follows:

- *Convex:* contributes to a convex constrained region. This is given by (i) any linear constraint, (ii) convex inequalities of ≤ type, (iii) concave inequalities of ≥ type.
- *Almost convex:* given by (i) almost linear equality constraints, (ii) almost convex inequalities of ≤ type, (iii) almost concave inequalities of ≥ type.
- *Nonconvex:* given by all constraints whose empirical shape is not "convex" or "almost convex".

A convex region effect is provided by a combination of the function shape and constraint sense that creates a single contiguous convex feasible region for the variables in the constraint (see Figure 2 for some one-dimensional examples). How these individual constraint effects are combined to provide an assessment of the shape of the overall feasible region is described in Section 2.8

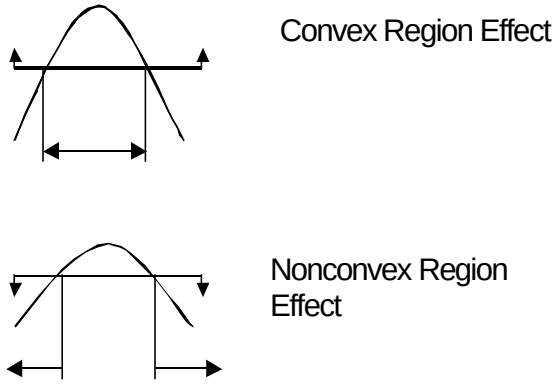

Figure 2: Examples of convex and nonconvex region effects.

2.7 Optimum Effect

Similar to the constraint region effect, the objective function has an optimum effect as a result of its empirical function shape and the sense of the optimization (maximization or minimization). The optimum effect deals with the question of whether a standard gradient-based search method would likely find the global optimum if searching in a contiguous convex feasible region. The possible empirical optimum effects for an objective function are as follows:

- *Global optimum possible:* assigned in three cases: (i) linear objective, (ii) convex objective to be minimized and (iii) concave objective to be maximized.
- *Local optimum likely:* assigned in three cases: (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:* assigned in three cases: (i) almost linear objective, (ii) almost convex objective to be minimized, (iii) almost concave objective to be maximized.

The "local, almost global" characterization is assigned in cases in which the objective shape is very close to allowing the characterization "global optimum possible". It indicates that the objective function is a good candidate for approximation, with little error, by a function that has more favorable properties for optimization.

2.8 Constrained Region Shape

The *constrained region* refers to the interaction of the constraints within the sampling enclosure. This is a broader concept than the feasible region, which, if it exists, is a subset of the constrained region. To assess the feasible region would require that only feasible points be sampled; however it is virtually impossible to randomly generate feasible points, especially when there are equality constraints. Instead, we can draw various conclusions by combining the independently-evaluated region effects of the individual constraints.

The conclusions relate primarily to how a standard gradient-based phase 1 feasibility-seeking algorithm is likely to perform if started at an arbitrary point in the sampling enclosure. In some cases, conclusions can also be drawn about the shape of the feasible region, if one exists. The three primary conclusions that can be drawn are as follows:

- *If all constraints have convex region effects:* a gradient-based phase 1 feasibility seeking algorithm will be able to accurately determine the feasibility of the constraint set. Further, a feasible region, if one exists, will be a convex set. The constrained region shape is denoted as "convex".
- *If some constraints have convex region effects and some have "almost convex" region effects, and none have a nonconvex region effect:* the constraints having an "almost convex" region effect are good candidates for approximation to improve the behaviour of a gradient-based phase 1 feasibility-seeking algorithm. Appropriate approximation also means that a feasible region, if it exists, will be a convex set. The constrained region shape is denoted as "almost convex".
- *If there is at least one constraint having a nonconvex region effect:* a gradient-based phase 1 feasibility-seeking algorithm may not perform well. No conclusions can be drawn about the shape of a possible feasible region (the nonconvexity may occur in a portion of the sampling enclosure that is rendered infeasible by the action of another constraint, so any feasible region might still be a convex set). The constrained region shape is denoted as "nonconvex".

2.9 Feasibility

Conclusions about overall feasibility are drawn by combining the independent effectiveness evaluations of the individual constraints. A definite conclusion of infeasibility is drawn if there is at least one completely effective constraint (effectiveness is 1.0). This represents a constraint that was never satisfied during sampling in the enclosure (or showed no possibility of being satisfied in the case of equality constraints). Hence, given that any possible feasible region must be within the sampling enclosure, the model is most probably infeasible, though there remains the possibility that a feasible point exists which has not been sampled. A simple example is shown in Figure 3.

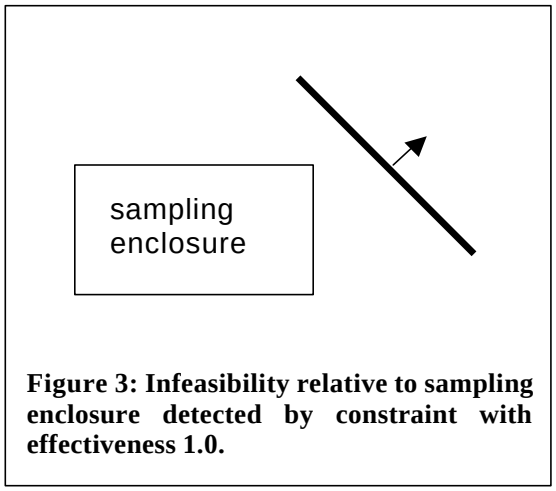

Figure 3: Infeasibility relative to sampling enclosure detected by constraint with effectiveness 1.0.

In the case where all constraints have effectiveness less than 1.0 (or can possibly be satisfied in the case of equality constraints), no definite conclusion can be drawn. The possibility of feasibility remains open, but infeasibility has not been ruled out.

2.10 Constraint Necessity and Redundancy

There are two ways to assess constraint necessity and redundancy via sampling, depending on whether the constraint forms part of the sampling enclosure or not. If the constraint forms part of the sampling enclosure, then an understanding of sampling inside arbitrary convex enclosures is needed; this topic is addressed in Section 3.

Conclusions about the redundancy of constraints that are not part of the sampling enclosure can be drawn based on the constraint effectiveness. Constraints with an effectiveness of 0.0 were satisfied at every sampled point. This indicates that they do not intersect the sampling enclosure, and hence are redundant relative to the enclosure. No definite conclusions can be drawn about constraints that have effectiveness greater than 0.0 but less than 1.0 (or "possible" equality constraints). Redundancy relative to the sampling enclosure is illustrated in Figure 4.

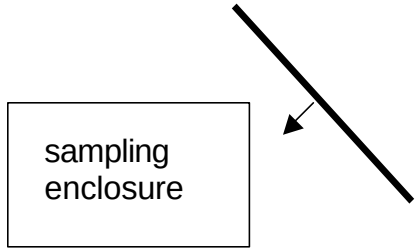

Figure 4: Constraint effectiveness of 0.0 indicates redundancy relative to the sampling enclosure.

2.11 Increasing Confidence in the Analysis

There are a number of ways to increase confidence in the results of the analyses resulting from the sampling procedures described above. The simplest is to increase the number of samples taken before conclusions are drawn. The ability to do so may be limited by the time taken to evaluate each function, or by the overall size of the model.

The quality of the conclusions also depends on how closely the region of interest is approximated by the sampling enclosure. The closer the sampling enclosure is to the region of interest, the more accurate the conclusions. A number of techniques for tightening the sampling enclosure around the region of interest are described in Section 5.

2.12 Binary and Integer Variables

During many of the analyses described above (e.g. the analysis of function shape), binary and integer variables are best treated as real-valued. However, such variables are best treated as their native type during other analyses, for example when assessing constraint effectiveness. This is accomplished by generating real values for such variables, and then *snapping* them to the closest appropriate binary or integer value, when needed.

3. Sampling in General Convex Enclosures

For the most accurate and meaningful results, the sampling enclosure should tightly bound the region of interest, which is normally the feasible region. While boxes defined by bounds on the variables are easy to construct and to sample uniformly, they may not bound the region of interest tightly. A better alternative is to construct a general convex enclosure by choosing appropriate constraints from the model. This has the added advantage of permitting more advanced analyses of the necessity and redundancy of the constraints that form the sampling enclosure. There are two disadvantages: much increased complexity of the procedures for uniform sampling inside the enclosure, and finding an initial point that is feasible relative to the sampling enclosure.

Random sampling procedures of the type described in Section 2 require a full-dimensional convex sampling enclosure. Finding such an enclosure is straightforward if procedures are already in place for sampling inside box enclosures. First sample in the box enclosure and identify all those inequalities that have a convex region effect in the box. These constraints, along with the variable bounds forming the box, then form the convex sampling enclosure. One pitfall in determining a general convex enclosure in this manner is the possibility that an implied inequality may eliminate the full-dimensionality of the resulting enclosure.

Sampling inside this new convex enclosure may in fact show that other constraints that were seen as having nonconvex region effects when sampled in the original box actually have convex region effects when sampled in the smaller convex enclosure. They can then be added to the list of enclosing constraints. A maximum cardinality convex sampling enclosure can be built up in this manner.

Note that equality constraints (even if linear) are excluded from the convex sampling enclosure due to the virtual impossibility of satisfying them during sampling. Constraints that have effectiveness of 1.0 are also not permitted as part of the sampling enclosure since they are impossible to satisfy. The hit-and-run methods described below depend on the ability to generate an initial point that is feasible relative to the enclosure constraints. The convexity of the enclosure is also essential to ensure uniform sampling via hit-and-run methods.

3.1 Hit-and-Run Methods

Hit-and-run methods [Berbee et al. 1987] provide a means of sampling the interior and perimeter of a general full-dimension convex enclosure. Starting at an arbitrary Point 0 ($\boldsymbol{x}_0$) that is feasible relative to the enclosure, a *spanning line segment* is created by generating a random ray rooted at $\boldsymbol{x}_0$. Point 1 ($\boldsymbol{x}_1$), the point at which this ray meets the first enclosure constraint, is noted, as is Point 2 ($\boldsymbol{x}_2$) the point at which the oppositely directed ray meets the first enclosure constraint. $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the two *hit points*, define the spanning line segment.

There are various options for choosing a new $\boldsymbol{x}_0$ for generating the next spanning line segment. It can be chosen at a random point on the last spanning line segment, or at a fixed point on the last spanning line segment (e.g. the center). In a *stand-and-hit* variant, $\boldsymbol{x}_0$ is a single fixed point.

The ray direction from $\boldsymbol{x}_0$ is normally constructed by choosing a random point on the unit hypersphere surrounding $\boldsymbol{x}_0$, though other choices are also possible. An illustration of hit-and-run sampling is given in Figure 5. The darkened squares indicate the various $\boldsymbol{x}_0$ points used to generate hitting rays; the numbers indicate their order of use.

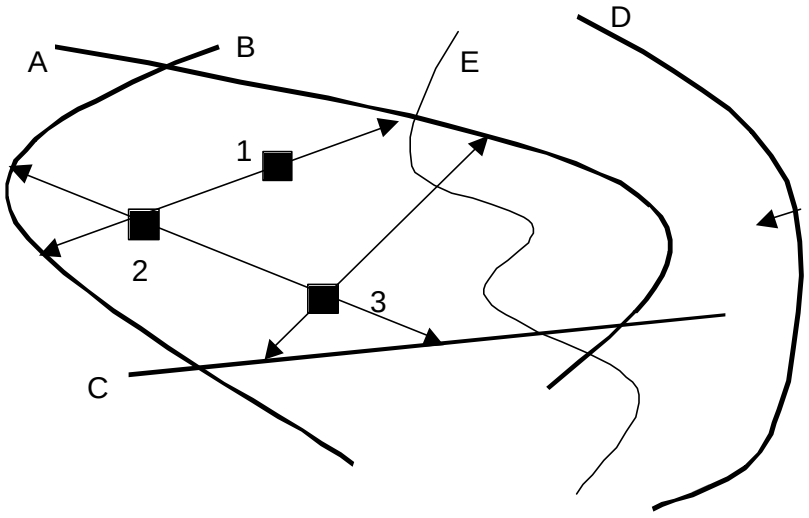

Figure 5: Hit-and-run sampling in a convex enclosure.

3.2 Analyzing Constraint Necessity and Redundancy

At each iteration of the hit-and-run method, Points 1 and 2 are determined by the intersection of the ray from Point 0 with the *first* enclosure constraints that are hit in the forward and reverse directions. As shown by Berbee et al. [1987], the hit constraints are therefore non-redundant relative to the sampling enclosure. The special case of a shared hit when two or more constraints are tied for the hit point (within the equality tolerance) is also tallied.

After sufficient iterations of the hit-and-run method, the enclosure constraints can be divided into three classes:

- *Necessary relative to the enclosure:* The constraint experienced one or more unshared hits during sampling.
- *Strongly redundant relative to the enclosure:* The constraint experienced no hits during sampling.
- *Weakly redundant relative to the enclosure:* The constraint experienced only hits that were shared with another constraint.

Figure 5 illustrates these concepts. The enclosure constraints are A, B, C, and D. After a few hit-and-run iterations, constraints A, B, and C have been identified as necessary, while constraint D is redundant relative to the sampling enclosure. Constraint E is a non-enclosure constraint that

intersects the sampling enclosure; it's shape, effectiveness, and other properties will be assessed as usual by the sampling procedures.

Knowledge of the necessity/redundancy status of the enclosure constraints relative to the enclosure sheds some light on the status of those constraints relative to any feasible region as well. Any constraint that is redundant relative to the sampling enclosure is also redundant relative to the feasible region since the sampling enclosure constraints are normally a subset of the constraints defining the model. In the special case where the sampling enclosure is composed of all of the constraints in the model, then the necessity/redundancy estimates apply to the feasible region for the model itself.

Note that the analysis results also apply to the variable bounds that form the enclosing box since these are included as part of the sampling enclosure.

The fraction of hits that an enclosure constraint receives is a measure of the fraction of the surface of the enclosure that it comprises. For example, a constraint or variable bound that receives 20% of the hits comprises about 20% of the surface of the enclosure. This gives some idea of how important the constraint or bound is to the make-up of the enclosure. Attention should be paid to accurate modeling of constraints that comprise a large fraction of the enclosure surface.

3.3 Finding an Initial Feasible Point

The hit-and-run methods depend on the provision of an initial $\boldsymbol{x}_0$ that is feasible relative to the enclosure constraints; the feasibility of subsequent $\boldsymbol{x}_0$'s is maintained thereafter by the hit-and-run method and the convexity of the enclosure. Attaining initial $\boldsymbol{x}_0$ feasibility can be difficult if numerous enclosure constraints are added simultaneously, as when the enclosure is first set up. Algorithm 1 shows an efficient bootstrapping method for finding an initial feasible point. Random points are generated and tested against the constraints; when a constraint is satisfied, all subsequent random points must also satisfy that constraint. This is assured by using the hit-and-run method. Feasibility relative to all of the enclosure constraints is built up gradually.

The model is deemed infeasible if a point satisfying all of the enclosure constraints simultaneously cannot be found. A reasonably large number of points should be sampled before this conclusion is reached. Note, though, that the procedure becomes more and more accurate in it's sampling as constraints are moved from the *NotSat* set to the *Sat* set.

A serious difficulty in finding an initial feasible point relates to the shape of the sampling enclosure. It is very difficult to find a feasible point, if one exists, when the sampling enclosure is extremely long and thin. When the initial sampling box is very long and thin, then the subsequent sampling enclosures built up in the course of Algorithm 1 also have this shape. The difficulty arises in that $\boldsymbol{x}_0$ does not tend to move along the length of the sampling enclosure because the probability of a random ray being oriented along the length of the thin enclosure is very small. The result is that only one part of the enclosure is sampled, and if the enclosure-feasible region is not in that part, then an initial feasible $\boldsymbol{x}_0$ cannot be found.

This difficulty is very common in practice. Many models have long thin sampling enclosures because binary variables are included along with real variables that are unbounded or have very large bounds. The initial sampling box is then extremely long and thin, as are the subsequent sampling enclosures built up in the course of Algorithm 1. It is possible, however, to take advantage of the fact that these very common long and thin enclosures are axis-aligned.

The solution is to bias the ray-generation probabilities so that there is a much higher probability of generating a ray that points along the length of the long and thin enclosure. In the case of axis-aligned enclosures this is easy to do by multiplying the search direction vector produced by a random hypersphere by the lengths of the variable ranges. This converts the hypersphere to an axis-aligned hyperellipse that has a much larger probability of generating rays that are oriented along the long axes of the enclosure.

Another option is to generate search directions by simply choosing two points in the variable box (even during Step 2 of Algorithm 1). The search direction is then set as the difference of the two points. This also has a much higher probability of generating rays that are oriented along the long axes of the enclosure.

Inputs: *NotSat:* the set of candidate constraints for addition to the sampling enclosure (inequality constraints having convex region effects)

Step 1 (initialization):
Sat = the set of variable lower and upper bounds (the initial sampling box)
Do the following a specified number of times:
Generate a random point $\boldsymbol{x}_0$ satisfying *Sat* using box sampling.
Are any constraints in *NotSat* satisfied at $\boldsymbol{x}_0$? If yes then move the satisfied constraints from *NotSat* to *Sat* and go to Step 2.
Issue an infeasibility message and exit.

Step 2 (satisfy general inequalities):
Do the following a specified number of times:
If *NotSat*=ϕ then exit (success).
Generate a random line segment satisfying *Sat* from $\boldsymbol{x}_0$ using the hit-and-run method.
Select a random point on the line segment, label this $\boldsymbol{x}_0$.
Are any constraints in *NotSat* satisfied at $\boldsymbol{x}_0$? If yes then move the satisfied constraints from *NotSat* to *Sat*.
Issue an infeasibility message and exit.

Algorithm 1: Bootstrapping procedure to achieve initial feasibility of a new convex enclosure.

Consider this experiment. The variable bounds are $0 \le x \le 2100$ and $0 \le y \le 1$, forming a long thin axis-aligned box (y is binary, but treated as real for the purposes of finding an enclosure-feasible point). The constraints in the model essentially create a feasible region between $1000 \le x \le 1100$, i.e. about 5% of the length of the long x-axis and 5% of the volume of the box. We

will compare the effects of the three methods of generating rays by measuring how many line segments are required in Step 2 of Algorithm 1 before enclosure-feasibility is reached. Step 2 is limited to 500 line segments; if Step 2 requires more line segments than that, then it is arbitrarily assigned a score of 500. One hundred tests are performed for each method, with results as follows:

- *Random hypersphere directions*: only 22 of 100 tests achieved a feasible $\boldsymbol{x}_0$. Average 425.4 line segments, standard deviation 156.6.
- *Random hyperellipse directions*: all 100 tests achieved a feasible $\boldsymbol{x}_0$. Average 4.6 line segments, standard deviation 4.1.
- *Random box directions*: all 100 tests achieved a feasible $\boldsymbol{x}_0$. Average 5.0 line segments, standard deviation 5.1.

These results show that generating points from the surface of a random hypersphere is disastrous for long thin enclosures. Biasing by the length of the variable range is very successful for axis-aligned enclosures. The MProbe software described in Section 5 uses the random hyperellipse directions method. The random box directions method is almost as successful, and could be substituted to take advantage of its smaller computational load.

There is no such remedy for long thin enclosures that are not axis-aligned. If an enclosing box that aligns with the long axes of the enclosure can be found, then a similar random hyperellipse directions method can be applied. It is at least fortunate that a remedy is available for the very common difficulty of long and thin axis-aligned enclosures.

Note that the random hypersphere direction method must still be used when testing the redundancy and necessity of constraints and bounds so as not to bias the results. Constraints and bounds that form only a small part of the enclosure surface should retain their small probability of detection and resulting small surface fractions.

3.4 Approximating the Analytic Center

When using a stand-and-hit method to sample inside a convex sampling enclosure, the most accurate results should be returned when $\boldsymbol{x}_0$ is fixed at the *prime analytic center*. The prime analytic center can be thought of as the center of the feasible region, the point as far as possible from each of the necessary constraints. This is a good place from which to generate rays to test the necessity of the enclosure constraints, but unfortunately the necessity of the constraints must be known a priori in order to find the prime analytic center.

When the necessary constraints are known, the prime analytic center is the feasible point that maximizes the function [Caron, Greenberg and Holder 1999]:

$$\sum_{i=1}^{m} \ln(b_i - B_i x)$$

This objective function has a barrier effect of repelling the potential prime analytic center point away from the necessary constraints.

Of course, the complete set of necessary constraints is not known while the hit-and-run method is operating, but the prime analytic center objective function can be used to measure how well any

candidate $\boldsymbol{x}_0$ might approximate the true prime analytic center. A candidate $\boldsymbol{x}_0$ having a higher value of this objective function indicates a better approximation. This suggests a method of moving towards the prime analytic center while the hit-and-run method is operating. The maximum value of the measure given in the function above is tracked as the hit-and-run method proceeds and various $\boldsymbol{x}_0$'s are generated. If a proposed $\boldsymbol{x}_0$ has a greater value of the measure, then the new $\boldsymbol{x}_0$ is accepted and the maximum value is updated. Otherwise, the new $\boldsymbol{x}_0$ is rejected, and the old $\boldsymbol{x}_0$ is again used to generate the next hitting rays. The measure is calculated using only those constraints that have been identified as necessary thus far in the hit-and-run procedure. More constraints will be identified as necessary as the hit-and-run procedure continues.

This constitutes a method that is somewhere between hit-and-run and stand-and-hit. $\boldsymbol{x}_0$ usually moves fairly frequently at the beginning, but gradually settles into longer and longer stays at particular points. Discovery of a new necessary constraint may cause some movement. An important advantage of this procedure is that the repelling effect of the measure tends to force $\boldsymbol{x}_0$ towards constraints that have not yet been identified as necessary, which increases the chances of their identification as necessary. This speeds the identification of all of the necessary constraints, and improves the placement of $\boldsymbol{x}_0$ itself, closer to the true prime analytic center. This method also tends to keep the $\boldsymbol{x}_0$ from becoming stuck in "corners" of the sampling enclosure.

Performance is improved when the new $\boldsymbol{x}_0$ candidates are generated by taking the midpoint of the last spanning line segment, rather than a random point on the last spanning line segment. This is because the midpoint of a spanning line has a greater chance of being close to the prime analytic center than a random point on a spanning line, which could be near the boundaries.

Experimental results support this conjecture. A simple rectangle with two sides of length 100 and two sides of length 1 was used to test how quickly either method arrived at a point within a target 10% error of the prime analytic center (i.e. the point having the first variable between 45 and 55, and the second variable between 0.45 and 0.55). Each method was tested using 1000 spanning line segments, and the iteration at which $\boldsymbol{x}_0$ first arrived within the target zone was noted. If $\boldsymbol{x}_0$ never arrived within the target zone, then the arrival iteration was assigned as 1000. Over 100 tests each, the results were:

- *Candidate $\boldsymbol{x}_0$ is midpoint of spanning line segment*: in all 100 tests, the target zone was reached within 1000 spanning line segments (maximum 780). Average first arrival at target zone: 166.5 iterations, standard deviation 154.5.
- *Candidate $\boldsymbol{x}_0$ chosen randomly along spanning line segment*: in 100 tests, the target zone was reached in only 58 cases. Average first arrival at target zone: 643.5 iterations, standard deviation 392.7.

The target zone is reached much earlier when $\boldsymbol{x}_0$ candidates are generated by taking the midpoint of the spanning line segments. In the example above, with the target zone reached for the first time after 166.5 line segments on average, that leaves 833.5 iterations on average during which the hitting rays are generated from a point that is very close to the prime analytic center.

4. Tightening the Sampling Enclosure

All of the analyses performed by random sampling are improved when the sampling enclosure closely brackets the region of interest. In the best case, the sampling enclosure is identical to the feasible region of the model. In this (rare) case, the conclusions about necessity and redundancy of the constraints apply directly to the feasible region itself. Close bracketing gives more accurate function analyses, which may permit additional constraints to be added to the sampling enclosure as they are discovered to have convex region effects. A chain of such effects may ensue as tightening proceeds. A number of techniques for tightening the sampling enclosure are described below.

4.1 Manual Adjustment of the Variable Bounds

Human insight may permit the manual adjustment of the upper and lower variable bounds, which tightens the sampling box. The sampling box need not bracket the entire feasible region. Depending on the analyst's intentions, it may be sufficient to bracket only the region surrounding the optimum or suspected optimum point, or some other region of interest.

4.2 Classic Presolving Techniques

Classic presolving techniques, e.g. bound tightening, [Brearly, Mitra, and Williams 1975] can be applied before sampling approaches are undertaken. Because such techniques are primarily algebraic in nature, they are best applied to the initial algebraic expression of the model. If the model is originally expressed in a commercial algebraic modeling language, such as AMPL [Fourer, Gay and Kernighan 1993], then advantage should be taken of any presolving routines built into the modeling language.

4.3 Linear Interval Analysis

Where linear constraints (inequalities and equalities) are involved in the model, simple linear interval analysis can be applied when the bounds on the variables are known. The idea is to determine the maximum possible range for each variable given the constraint and the bounds on the other variables in the constraint. This is the sort of analysis performed in presolvers, but it can be performed again whenever a bound has been tightened for any reason, as part of a cascade of bound adjustments resulting from the tightening.

For example, consider the constraint $2x_1 - 5x_2 \le 10$ when $-10 \le x_1,x_2 \le 10$. x_2 is lower bounded by this constraint when x_1 is at it's lower bound, i.e. $2(-10) - 5x_2 \le 10 \Rightarrow x_2 \ge -6$. The conclusion is that the true bounds on x_2 are $-6 \le x_2 \le 10$, rather than the original $-10 \le x_2 \le 10$.

When the tightened interval on a variable returned by this sort of analysis does not overlap with the current bounds on the variable, then the model is found to be infeasible. This is a deterministic, not probabilistic, result.

Integer and binary variables are treated as real-valued during such an analysis, with the resulting bounds rounded inwards to the nearest integer value.

4.4 Nonlinear Interval Analysis via Sampling

Sampling the individual nonlinear constraints can accomplish something similar to the linear interval analysis. Consider nonlinear inequality constraints first. As random points inside the sampling enclosure are generated, the feasibility of the nonlinear inequality is evaluated. Infeasible points are rejected, but at feasible points, the values of the constraint variables are noted. Over numerous feasible sample points, the maximum and minimum value of each variable is recorded. On exit, each minimum/maximum pair provides an overtightening of the bounds on the variable. It is an overtightening because the probability of attaining the true maximum and true minimum of each variable in this way is small. Still, when there are few other options, this approach does help to zero in on the feasible region, if one exists. The probability of serious error is greatest when the variable bounds are unduly large or unbounded. The process is illustrated in Figure 6(a).

As the variable ranges are (over)tightened, it may happen that the interval returned for a variable by one constraint does not overlap with the interval returned for the same variable by a different constraint. Because of the overtightening, it is possible that a feasible value for the variable exists in the gap between the two non-overlapping intervals. For this reason, when two intervals do not overlap, the interval that is returned consists of the gap between those intervals. Infeasibility is not assumed.

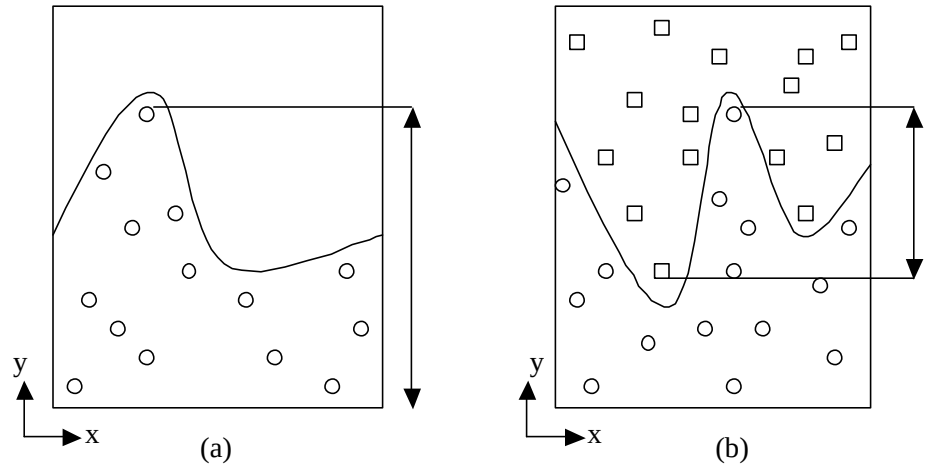

Figure 6: Nonlinear interval analysis via sampling tightens the bounds on variable y. (a) Inequality constraint. (b) Equality constraint.

Nonlinear equality constraints can also be handled this way. The equality is first treated as a $\leq$ constraint and appropriate variable ranges are constructed, then it is treated as a $\geq$ constraint and a second set of variable ranges is constructed. The intersection of the two ranges is returned as the final tightened range after sampling the equality constraint. See Figure 6(b) for an illustration of the process.

4.5 Handling Unbounded Variables

Unbounded variables can cause difficulties for nonlinear interval analysis via sampling. This is because the feasible region for the constraint may comprise only a tiny portion of the variable box when the constituent variables are unconstrained, even when "unbounded" is numerically defined as $\pm 1 \times 10^{20}$, for example. Sampling then turns up no feasible points for the constraint, so that the variable ranges cannot be tightened. Variables are often left unbounded when the model formulator has no idea of the approximate range of the solution values for the variables.

When the interval tightening methods described above are unsuccessful, the opposite approach may work: expanding boxes instead of shrinking them. The main idea is to start with a very small sampling box and to expand the bounds in several stages, looking for the approximate scale of the variables. The initial small box is centered around the origin. If the variable is unbounded in both directions, then the initial box side extends from –1 to +1. If the variable is nonnegative unbounded, then the initial box side extends from 0 to +1. This temporary sampling box is called a *nucleus box*.

For each nonlinear constraint for which no feasible points have been found, various nucleus boxes are sampled, usually at the scale of 1 as above, then increasing powers of 10 (10^1, 10^2, …10^5). The largest box that registers any feasible sample points for the constraint is used to reset the bounds on the variables in the constraint. In the case of equality constraints, the nucleus box is accepted if it registers at least one point at which the functional value is less than the constant, and at least one point at which the functional value is greater than the constant (or the low probability event that the point satisfies the equality constraint).

It is possible that the bounds will be overtightened by this procedure, but they could also be undertightened. Some manual adjustment may be desirable.

4.6 Nonlinear Range Cutting via Sampling

This heuristic works by examining possible cuts on the outside of the sampling box. For a given variable, a cut is proposed (as large as 90% of the total variable range if the range is very large, more commonly 30% of the variable range). The cut is accepted if, after sufficient samples, at least one of the constraints that use the variable has not been satisfied at any of the sample points. This method is similar to nonlinear interval analysis via sampling (Section 4.4), but works from outside the feasible zone of the constraint rather than from the inside.

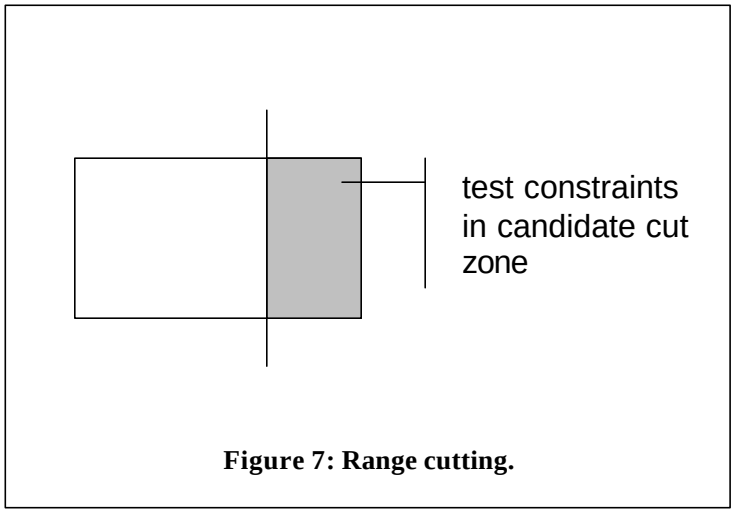

Figure 7: Range cutting.

Equality constraints are considered satisfied in the test cut zone if they register at least one point at which the functional value is greater than the constant and at least one point at which the functional value is less than the constant, or the low probability event that a sample point satisfies the equality constraint.

4.7 Convex Enclosure Sampling

The methods described thus far all operate by examining a single constraint at a time. Bound tightening is more effective when the feasibility of all of the constraints is considered simultaneously at a given point. This idea is difficult to incorporate in a random sampling approach given the presence of equality constraints. However, when a general convex sampling enclosure is in use, the hit-and-run method guarantees that all of the sampling enclosure inequalities are simultaneously feasible. It is then simple to record the minimum and maximum values of the variables over all of the points sampled by the hit-and-run method.

Accuracy is improved by using the end points of the spanning line segments since these are at the extreme edges of the enclosure-feasible region. Overtightening of the variable bounds remains a possibility, but it is much less likely because of the use of the spanning line segment endpoints. All of the sample points used in resetting the maximum and minimum values are on the boundary of the enclosure-feasible region.

4.8 Combining the Techniques

The enclosure tightening techniques described above can be combined in various ways. The combined procedure used in MProbe is as follows:

1. one pass of linear interval analysis,
2. nonlinear interval sampling (one constraint at a time),
3. for those constraints which registered no feasible points during nonlinear interval sampling, get a nucleus box,
4. nonlinear range cutting via sampling.

MProbe also permits use of the other 3 techniques under user control: manual adjustment of variable bounds, the classic presolving techniques available in AMPL, and convex enclosure sampling.

5. MProbe

MProbe [Chinneck 1999] is a software tool for probing mathematical programs of all types. It has particular strengths in examining nonlinear programs using the random sampling techniques described in this paper. The various sampling rates, histogram cell limits, tolerances, etc. can be set by the user. MProbe reads models written in the AMPL language.

MProbe also provides a number of other tools for examining large and complex mathematical programs, including the following:

- spreadsheet-style displays of data about the constraints, variables, and objectives that can be sorted and filtered in various ways,

- profile plotting of functions along a straight line between any two points in the multivariable space,
- ability to navigate in the model, e.g. by seeing all of the variables in a particular function, or all of the functions using a particular variable,
- facilities for keeping a text file record of the analyses made via the graphical user interface.

6. Conclusions

A random sampling approach can provide valuable insights into the characteristics of complex mathematical programs. In fact, random sampling is the only way to obtain some important information, such as estimates of the shape of nonlinear constraints and objective functions. A number of new techniques are introduced in this paper:

- the use of subsets of the constraints to define a convex sampling enclosure which more closely bounds the feasible region, if one exists,
- a method of approximating the prime analytic center so as to improve the performance of the hit-and-run method for determining the necessity of constraints,
- tightening the sampling enclosure via sampling interval methods,
- improved heuristics for obtaining an initial feasible point for a convex sampling enclosure.

Tools for the analysis of optimization models are as necessary to the effective use of mathematical programming as supporting tools such as debuggers and syntax-aware editors are to general programming. Random sampling techniques fill an important niche among model analysis tools, and are finding more frequent use (e.g. MProbe). Opportunities for the development of new methods abound.

References

C. Bélisle, A. Boneh, and R.J. Caron (1998). *Convergence Properties of Hit-and-Run Samplers*, **Commun. Statist.—Stochastic Models** 14(4), pp. 767-800.

H.C.P. Berbee, C.G.E. Boender, A.H.G. Rinooy Kan, C.L. Scheffer, R.L. Smith, and J. Telgen (1987). *Hit-and-Run Algorithms for the Identification of Nonredundant Linear Inequalities*, **Mathematical Programming** 37, pp. 184-207.

A.L. Brearly, G. Mitra, and H.P. Williams (1975). *Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm*, **Mathematical Programming** 8, pp. 54-83.

A. Boneh (1983). *PREDUCE – a Probabilistic Algorithm Identifying Redundancy by a Random Feasible Point Generator (RFPG)* in Karwan, Lotfi, Telgen, Zionts (eds.), **Lecutre Notes in Economics and Mathematical Systems** 206.

A. Boneh, S. Boneh, and R.J. Caron (1993). *Constraint Classification in Mathematical Programming*, **Mathematical Programming** 61, pp. 61-74.

R.J. Caron, H.J. Greenberg, and A.G. Holder (1999). *Analytic Centres and Repelling Inequalities*, University of Windsor, Canada, Dept. of Mathematics and Statistics, WMSR #99-04.

J.W. Chinneck (1999). *Analyzing Mathematical Programs using MProbe*, to appear in *Modeling Languages and Approaches*, a volume of **Annals of Operations Research**. MProbe is available via the web at http://www.sce.carleton.ca/faculty/chinneck/mprobe.html.

J. Feng (1999). **Nonlinear Redundancy: Where is the Information?**, M.Sc., Mathematics, Department of Economics, Mathematics, and Statistics, University of Windsor, Canada.

R. Fourer, D.M. Gay, and B.W. Kernighan (1993). **AMPL: a Modelling Language for Mathematical Programming**, Boyd and Fraser Publishing Company, Danvers, Massachusetts.

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.

P.M. Pardalos (1994). *On the Passage from Local to Global in Optimization*, in J.R. Birge and K.G. Murty (eds.), **Mathematical Programming: State of the Art 1994**, The University of Michigan.