INFORMS Journal on Computing
Vol. 16, No. 3, Summer 2004, pp. 255–265
ISSN 0899-1499 | EISSN 1526-5528 | 04 | 1603 | 0255

DOI 10.1287/ijoc.1030.0046
© 2004 INFORMS

The Constraint Consensus Method for Finding Approximately Feasible Points in Nonlinear Programs

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

This paper develops a method for moving quickly and cheaply from an arbitrary initial point at an extreme distance from the feasible region to a point that is relatively near the feasible region of a nonlinearly constrained model. The method is a variant of a projection algorithm that is shown to be robust, even in the presence of nonconvex constraints and infeasibility. Empirical results are presented.

Key words: nonlinear programming; feasibility; approximate algorithms
History: Accepted by W. David Kelton; received October 2002; revised May 2003; accepted June 2003.

1. Introduction

There are numerous nonlinear-programming applications in which it is necessary to move very quickly from an initial point, often very far from feasibility, to a final point that is approximately feasible for a set of nonlinear constraints. A method for doing this is a valuable "nonlinear crash start" in the solution of any nonlinear program, for example. It is also a necessary first step in global optimization using exhaustive search algorithms (Kearfott and Dian 2000). See Pardalos and Resende (2002) for further information on techniques of nonlinear programming and global optimization.

A generic nonlinear program consisting of m constraints in n variables is shown in Equation 1. Our main interest is the case in which one or more of the constraint functions are nonlinear and the bounds on the variables are very wide.

$$\begin{aligned} &\text{Minimize or maximize } f(\mathbf{x}) \\ &\text{s.t.} \quad \mathbf{g}(\mathbf{x})\,\{\leq, \geq, =\}\,\mathbf{b} \qquad (1) \\ &\qquad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

The major motivation here is to find reasonable bounds on variables when these are not supplied by the user. The MProbe software (Chinneck 2001a, b, 2002) analyzes functions for characteristics such as their shape and distribution of values. It does so by random sampling in the multidimensional box defined by the variable bounds. Users often omit variable bounds when they are unsure as to how to set them. The resulting unbounded sampling box is so large that it is extremely unlikely that any sample points will be placed in the region of interest around the feasible region.

For example, consider a three-variable model in which the feasible region is approximately a cube with sides of length 100, for a volume of 1×10^6. If the bounds are not set by the user, MProbe will assume bounds of $\pm 1 \times 10^{10}$ on each variable, for a total sampling box volume of 8×10^{30}. The probability of placing a random point in the feasible region is then $1 \times 10^6/8 \times 10^{30} = 1.25 \times 10^{-25}$, which is vanishingly small. This difficulty worsens as the number of unbounded variables in the model increases. For this reason MProbe needs to be able to identify quickly the approximate location of the feasible region by moving from arbitrary initial points in the original sampling box to points that are relatively near to the feasible region. These approximately feasible points are used as a guide to set variable bounds that define a suitable box for productive sampling.

In the MProbe context it is important that near-feasibility be achieved relatively quickly since this is only a preliminary step towards the core sampling work. On the other hand, it is acceptable for the method to fail occasionally since many random initial points are supplied. However, the method should converge to an approximate feasible point quickly for a reasonable fraction of the initial points.

A note on nomenclature: We assume throughout that constraints have the form LHS $\{\leq, \geq, =\}$ RHS, where RHS is a constant. The LHS is often referred to as the "constraint body" and includes the functional part of the constraint.

2. Measuring Nonlinear Closeness to Feasibility

In the MProbe context, the idea is to find a box in the variable space that is a close outer approximation of the feasible region. As shown below, there is an important distinction between measures of closeness in the variable space and in the function space. Intuitively, the measure of how close an infeasible point is to feasibility is the minimum Euclidean distance between the point and the feasible region, referred to here as the *feasibility distance*. The feasibility distance is not normally used in most optimization solvers to measure the degree of infeasibility; instead, measures based on function values are the norm.

Most optimization solvers consider a constraint to be "feasible" when it is satisfied to within a prespecified tolerance ϵ. For example, a constraint of the form LHS $\leq$ RHS is commonly considered to be satisfied if LHS $\leq$ RHS $+ \epsilon$, where $\epsilon = 1 \times 10^{-6}$, for example. The LHS could be up to ϵ greater than the RHS and the constraint will still be considered satisfied. Since the feasibility measure in this case relates to the value of the function, we refer to this as a *function tolerance test*. The function tolerance test is applied to the *constraint violation*, which is zero for satisfied constraints, RHS $-$ LHS for violated constraints of the form LHS $\geq$ RHS, LHS $-$ RHS for violated constraints of the form LHS $\leq$ RHS, and $|$RHS $-$ LHS$|$ for violated equality constraints.

A function tolerance test is easy to implement, but is unfortunately severely affected by scaling issues. For example, consider the constraint $x^2 \leq 9$ at $x = 3.5$. The LHS evaluates to 12.25, for a constraint violation of $12.25 - 9 = 3.25$, well above the ϵ of 1×10^{-6}. Note, however, that since the constraint is exactly satisfied at $x = 3$, the feasibility distance in this one-dimensional example is $3.5 - 3 = 0.5$. Now consider a simple multiple of the original constraint: $10x^2 \leq 90$. At the same point $x = 3.5$, the constraint violation is $122.5 - 90 = 32.5$. The function tolerance test seems to show that the identical point is much farther from feasibility. In contrast, the feasibility distance remains 0.5. At the opposite extreme, another multiple of the original constraint, $0.0000001x^2 \leq 0.0000009$, has a constraint violation of 0.000000325 at $x = 3.5$, which is less that the ϵ of 1×10^{-6}, and therefore the constraint is considered to be satisfied by this test. In contrast, the feasibility distance retains its constant value of 0.5.

As the example shows, the common function tolerance test is severely affected by scaling, to the extent that a point can be considered to be very far from feasibility, close to feasibility, or even feasible, depending on the constraint multiple that is applied. On the other hand, the feasibility distance does not change with the scaling factor.

When the feasibility of individual constraints is tested via the function tolerance test, the overall model feasibility is normally tested by summing the constraint violations over all of the violated constraints. Minimizing this "sum of the infeasibilities" (often abbreviated as *SINF*) is the usual objective function during a Phase 1 procedure whose goal is to find a feasible point by driving SINF to zero (or below a prespecified tolerance).

Unfortunately, since the constraint violations for the individual constraints are affected by the constraint scaling, when a SINF of zero (or near zero) is achieved by a suitable Phase 1 procedure, the resulting point may in fact violate some of the constraints by a sizable feasibility distance. In fact, because of the scaling issues, the point may violate a number of constraints by various feasibility distances. Depending on the scaling, the SINF objective function may be trying much harder to satisfy some constraints than others because their constraint violations make a much bigger contribution to SINF. Constraints are not treated equally in terms of their effect on the Phase 1 procedure or outcome.

In contrast, the feasibility distance is an unbiased measure of the nearness of the point to feasibility for each constraint.

Overall model feasibility is also commonly measured by the number of violated constraints (often abbreviated as *NINF*). However, judgment of which constraints are violated is normally based on the function tolerance test, and so this measure suffers from the same scaling difficulties mentioned above. In contrast to the feasibility distance, a smaller value of NINF can be associated with a point that is much farther away from the feasible region. This happens when fewer constraints are violated, but their feasibility distances are larger. Some solvers, such as MINOS (Murtagh and Saunders 1993), use both NINF and SINF during their Phase 1 feasibility-seeking procedures.

To reduce the variable bounds appropriately in the MProbe application, it is important that a variable-space measure of distance to feasibility be used, such as the feasibility distance defined earlier. Nonlinear optimization systems could also benefit by replacing their current function-space measures by variable-space measures for the reasons given above. Unfortunately, the true feasibility distance is expensive to find in nonlinearly constrained models. Given an infeasible point, an optimization problem must be solved to find the closest feasible point. Since this is at least as difficult as solving the original Phase 1 problem itself, embedding such a procedure simply to measure the closeness to feasibility of the current trial point is clearly not justified. A more efficient procedure is needed.

3. Projection Algorithms

The Euclidean distance to feasibility underpins *projection algorithms*, originated by Cimmino (1938) and extensively developed in recent years by Censor and collaborators (e.g., Chapter 5 of Censor and Zenios 1997), among others. In projection algorithms, the *orthogonal projection* of an infeasible point is defined as the closest feasible point (Xiao et al. 2003).

We define here the *feasibility vector* for an individual constraint as the vector extending from an infeasible point to its orthogonal projection on the constraint. Both the direction and the distance of movement necessary to achieve feasibility are captured by the feasibility vector. Adding the feasibility vector to an infeasible point yields the closest point that satisfies the constraint, i.e., the orthogonal projection. The length of the feasibility vector is the feasibility distance.

The feasibility vector is easily found for a linear constraint. At a point $\mathbf{x}$ that violates a constraint c, the gradient of the constraint LHS is denoted by ∇c, with length $\|\nabla c\|$. A unit vector in the gradient direction is then given by $\nabla c/\|\nabla c\|$. The direction parameter d has the value $+1$ if it is necessary to increase the LHS value to satisfy the constraint, and -1 if it is necessary to decrease the LHS to satisfy the constraint. Hence a unit vector in the desired direction of motion is given by $d\nabla c/\|\nabla c\|$. If the constraint violation is v, then the feasibility distance is $v/\|\nabla c\|$, and the feasibility vector is $v/\|\nabla c\| \times d\nabla c/\|\nabla c\| = vd\nabla c/\|\nabla c\|^2$, as has been shown by Xiao et al. (2003) and others.

The feasibility vector can be used to construct numerous varieties of projection algorithms (Censor et al. 2001). In *sequential* projection algorithms, the current point is updated by finding and applying the feasibility vector for each constraint in turn. In *simultaneous* projection algorithms, all of the feasibility vectors are calculated simultaneously and some form of weighting is used to determine a final movement vector. Other variations include the use of control sequences and relaxation parameters to adjust the movements. Relaxation parameters are used to lengthen or shorten the suggested movements.

Component averaging (Censor et al. 2001) is an important recent development in simultaneous projection algorithms. In the usual simultaneous projection algorithm, the complete set of feasibility vectors for the violated constraints are combined in a weighted average. Component averaging, on the other hand, realizes that not all of the constraints contain all of the variables. For this reason, the final movement vector is computed component-wise, and only the constraints that contain a particular variable are considered when that movement component is calculated.

In the case of nonlinear constraints, the feasibility vector cannot be found exactly without a great deal of work: Solving a nonlinear optimization problem to minimize the distance between the infeasible point and a feasible point for the constraint. However, it can easily be estimated by linear approximation: use the constraint violation and the gradient at the current point, and apply the linear formula above. Methods that use the gradient in this manner are called *gradient-projection* methods. The accuracy of the resulting estimated feasibility vector is naturally affected by the curvature of the constraint at the estimation point.

Consider again the simple one-dimensional nonlinear constraint $x^2 \leq 9$. At $x = 3.5$, the estimated feasibility vector provided by gradient projection is $[-0.4643]$, with length 0.4643. This is a good approximation for the true feasibility vector of $[-0.5]$ with length 0.5. As for all Newton projection-based methods, the estimates are quite accurate for points that are close to the orthogonal projection, but are less accurate at more distant points. For the same constraint at $x = 10$, the feasibility distance is 7, but the estimated feasibility distance is 4.55.

Projection algorithms have been proved to converge when the constraints are all inequalities and form a convex region (Censor and Zenios 1997). The component-averaging scheme has also been proved to converge for the convex feasibility problem, which includes the case of linear equalities, even in the case of inconsistency (Censor et al. 2001). However, proofs of convergence have not been developed for the general nonlinear nonconvex case.

The constraint consensus method, developed below, is a form of simultaneous component-averaging gradient-projection algorithm. It is a simple version that does not use relaxation parameters or control sequences or unequal weighting. It is specifically designed for use in the nonlinear nonconvex case. The empirical convergence results presented later in this paper may be of interest to theoreticians seeking convergence proofs as well as those interested in practical applications of the method as a nonlinear crash start.

Hereafter, when referring to the "feasibility vector" for a constraint, we mean the exact feasibility vector in the case of linear constraints, and the estimated feasibility vector based on the gradient projection in the case of nonlinear constraints. Similarly the "feasibility distance" means the exact feasibility distance in the case of linear constraints, and the estimated feasibility distance based on the gradient projection in the case of nonlinear constraints.

4. The Constraint Consensus Method

The constraint consensus method developed here is designed to cope with nonconvex sets of constraints. This is an important characteristic in the MProbe context, where convexity cannot be assumed, and where nonconvexity is in fact likely. The main idea is to combine the feasibility vectors in such a way that a poor direction of motion suggested by the feasibility vector for a nonconvex constraint is "outvoted" by the feasibility vectors for the other constraints. This "voting" characteristic gives rise to the "constraint consensus" name.

The challenge is to construct a single vector that combines the movements suggested by the individual feasibility vectors, and that is then used to update the current point. The resulting vector should in some way reflect the consensus of opinion among the feasibility vectors of the violated constraints; hence, it is called the *consensus vector* **t**. The consensus vector is constructed by component-wise averaging of the feasibility vectors for the violated constraints. Let n_j represent the number of violated constraints that have variable x_j as a component, f_{ij} represent the component for variable x_j in the feasibility vector for the ith constraint, and s_j represent the sum of the f_{ij} for variable x_j over the feasibility vectors for all of the violated constraints. The component of the consensus vector for variable x_j is then given by s_j/n_j. If this vector is too short (length is less than a prespecified tolerance β), then the iterations are halted with an unsuccessful outcome.

The constraint consensus algorithm is summarized in Figure 1. Constraints are considered violated only if the feasibility distance is greater than a prespecified feasibility distance tolerance α (Step 2.1.2). To minimize the number of constraints for which a feasibility vector and feasibility distance must be found, a prescreening step eliminates constraints having a constraint violation of zero (Step 2.1). The algorithm halts when every constraint has a constraint violation of zero or a feasibility distance of less than α, i.e., NINF is zero. The feasibility tolerance can be quite large, depending on the purpose at hand. In the MProbe application, a feasibility distance tolerance of 100 is quite acceptable.

For simplicity, a few details are omitted in Figure 1. The most important of these is a method for coping with failed function or gradient evaluations, which can be common when the algorithm is working with points at extreme distances from the feasible region. Constraints that experience function or gradient evaluation errors are simply ignored, but a flag is raised. In many cases, the algorithm is able to progress eventually to a new point at which there are no function or gradient evaluation errors. However, if all of the successfully-evaluated constraints report feasibility (within the α feasibility distance tolerance), but the flag is raised, then the algorithm terminates unsuccessfully.

The constraint consensus method is simple: There are no line searches, matrix inversions, or other expensive routines such as the solution of a linear-programming problem. The main calculation effort is in determining the lengths of the vectors.

Figure 2 shows an example iteration of the constraint consensus method in which the feasibility distance tolerance is $\alpha = 0.5$ and the consensus vector length tolerance is $\beta = 0.1$. The two constraints are $A(\mathbf{x})\colon x_2 \leq 5$ and $B(\mathbf{x})\colon x_1^2 + x_2 \leq 10$. The initial infeasible point $(2.5, 8)$ violates both constraints. The feasibility vectors (shown as dashed arrows) are $[0\ \ -3]$ for $A(\mathbf{x})$ with length 3, and $[-0.817\ \ -0.163]$ for $B(\mathbf{x})$ with length 0.833. Both feasibility distances are larger than α and hence both feasibility vectors are included in the calculation of the consensus vector. $A(\mathbf{x})$ does not involve x_1, so it is ignored when calculating the x_1 component of the consensus vector. The consensus vector, shown as a solid arrow, is $[-0.817\ \ -1.582]$, and has a length of 1.781, which is greater than β, so the vector is accepted. The updated point is (1.683, 6.418), which satisfies B but not A, so the constraint consensus method will continue.

```
Inputs:
  • a set of constraints,
  • an initial point $\mathbf{x}$,
  • a feasibility distance tolerance $\alpha$,
  • a movement tolerance $\beta$.
1.  NINF = 0; for all $j$: $n_j = 0$, $s_j = 0$.
2.  For every constraint $c_i$:
      2.1  If $c_i$ is violated (without considering a tolerance) then:
             2.1.1  Find the feasibility vector and the feasibility distance.
             2.1.2  If the feasibility distance is greater than $\alpha$ then:
                      2.1.2.1  NINF = NINF + 1.
                      2.1.2.2  For every variable $x_j$ in $c_i$:
                                 2.1.2.2.1   $n_j \leftarrow n_j + 1$; $s_j \leftarrow s_j + f_{ij}$
3.  If NINF = 0, then exist successfully.
4.  For every variable $x_j$:
      4.1  $t_j = s_j/n_j$.
5.  If $\|\mathbf{t}\| \leq \beta$ then exit unsuccessfully.
6.  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{t}$.
7.  If necessary, reset $\mathbf{x}$ to respect any violated variable bounds.
8.  Go to Step 1.
```

Figure 1 The Constraint Consensus Algorithm

4.1. Alternative Stopping Conditions

The algorithm in Figure 1 exits successfully when the (estimated) feasibility distances for the violated constraints are all less than a specified feasibility distance tolerance α. This is most similar to the usual conditions for assessing the feasibility of a model. However, other stopping conditions can be postulated. Two such possibilities are described below.

Analogous to the usual Phase 1 objective of minimizing the sum of the infeasibilities, the algorithm could also be stopped when the sum of the feasibility distances over all of the violated constraints is

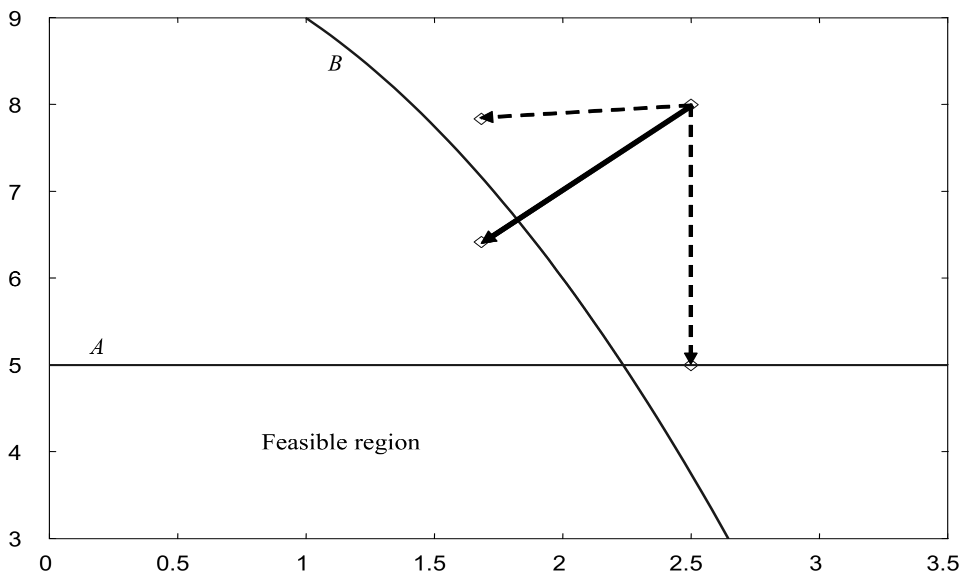

Figure 2 Example Iteration of the Constraint Consensus Method

smaller than a specified tolerance. The important difference from the ordinary Phase 1 objective is that this is a variable-space measure as opposed to the usual function-space measure. A second possibility is to stop the algorithm when the consensus vector is shorter than a specified tolerance.

Both of these alternative stopping conditions consider all of the constraints simultaneously, as opposed to the condition in Figure 1. Both attempt to provide a measure of closeness to simultaneous feasibility of all constraints. Both are also adversely affected by ill-conditioning. The sum of feasibility distances will drive the algorithm to continue iterating under ill-conditioning when very little progress is being made. The length of the feasibility vector will cause early termination in the case of ill-conditioning. In the MProbe context, it is desirable to include ill-conditioned portions of the model in the sampling region so that this property can be examined. For this reason, the stopping condition included in Figure 1 is preferred.

4.2. Characteristics

The gradient projection method of finding an estimated feasibility vector is closely related to Newton's method, and hence has similar characteristics. Principal among these is that the accuracy of the estimates improves the closer the point is to the solution, and the quadratic convergence rate. However, gradient projection methods can also be fooled by specific function shapes or by inappropriate placement of test points. See, e.g., Polak (1997) for further discussion of Newton methods.

A weakness of the constraint consensus algorithm in particular is that the method of combining the individual feasibility vectors can be fooled by specific circumstances, especially when a set of feasibility vectors pulling in one direction is counterbalanced by a set of feasibility vectors pulling in the opposite direction. In this case, the resulting consensus vector is very short, causing the algorithm to terminate unsuccessfully at Step 5 of Figure 1, even though the individual feasibility vectors have reasonable lengths. However, the failure is recognized. Figure 3 shows an example of an infeasible zone in which the two feasibility vectors produce a null consensus vector, thereby halting the iterations.

Short consensus vectors also happen in the case of ill conditioning. Consider two linear inequality constraints that cross, but are close to parallel, as shown in Figure 4. Some infeasible starting points will quickly lead to a point between the constraints at which both are violated, as shown in the figure.

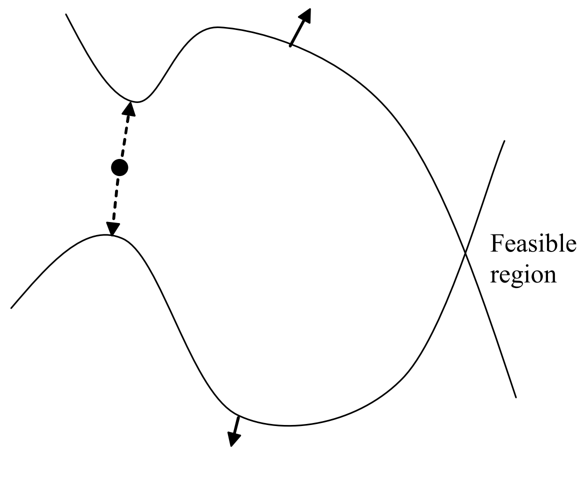

Figure 3 Consensus Vector Is Too Short

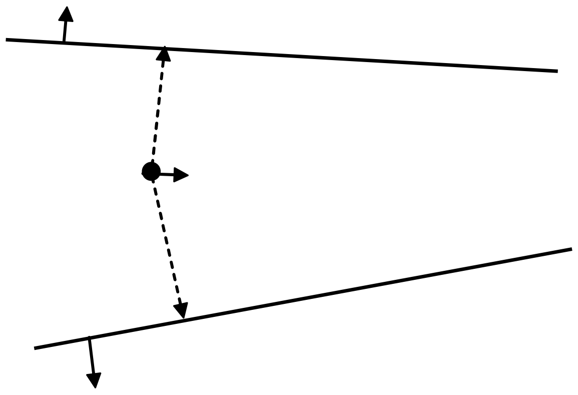
Figure 4 **Ill-Conditioning**

Here the feasibility vectors for the two constraints will be almost opposites of each other. The consensus vector will suggest only a slight movement towards the feasible region. Since the consensus vector is so short, the method will terminate unsuccessfully. However, if there is a third violated linear inequality involved, which is almost perpendicular to the two ill conditioned constraints, then convergence will be rapid.

Experiments show that the constraint consensus method is sometimes not effective for achieving true feasibility of a constraint set. This may happen when the movements suggested by the algorithm fall short of the target due to averaging. This effect is mitigated in other projection algorithms by using relaxation parameters to influence the length of the suggested movement vector. In the constraint consensus algorithm, this effect is instead mitigated by the relatively large size of the approximate feasibility tolerance. If a constraint violates the approximate feasibility condition (i.e., the estimated feasibility distance is greater than α), then the associated feasibility vector is included in the movement calculations. This feasibility vector is attempting to move to a point which actually satisfies the constraint (with a feasibility distance tolerance of zero) as opposed to a point that approximately satisfies the constraint (with a feasibility distance tolerance of α). The attempted movement is thus well beyond what is needed to satisfy a large feasibility distance tolerance, so relaxation parameters are not necessary.

5. Empirical Results

Test models are drawn from a number of well-known sources: Himmelblau (1972), Floudas and Pardalos (1990), Floudas et al. (1999), and Vanderbei's (2002) web repository. The models are chosen to cover a range of constraint types, shapes, and complexities.

An extremely important characteristic of the test models is whether the constraints form a convex or nonconvex set. This can be assessed by determining the *constraint region effect* (Chinneck 2001a, b) for each constraint, which indicates whether or not a constraint contributes to a convex feasible region, if one exists. A constraint that has a *convex region effect* is in one of the following categories:

- convex inequality of $\leq$ type,
- concave inequality of $\geq$ type,
- linear equality.

A constraint that has a *nonconvex region effect* is in one of the following categories:

- convex inequality of $\geq$ type,
- concave inequality of $\leq$ type,
- convex and concave inequality,
- nonlinear equality.

Constraints of these latter types contribute to the creation of a nonconvex feasible region, if one exists.

The MProbe software (Chinneck 2001a, b, 2002) is used to assess constraint region effects. It does so by sampling within the initial set of variable bounds. Constraints having nonconvex region effects are positively identified. However, it is possible that a constraint can be erroneously identified as having a convex region effect through insufficient sampling. Hence the models in Table 1 are at least as nonconvex as shown, if not more.

A number of the test models were chosen because they include constraints that have nonconvex region effects. It is particularly difficult for projection algorithms to find the feasible region in the presence of a constraint that has a nonconvex region effect. This is because the gradient projection of the constraint may point towards different discontiguous feasible zones for the constraint at different intermediate points in the solution.

The relevant model characteristics are summarized in Table 1. Constraints are noted as belonging to one of three categories: linear, quadratic, or general nonlinear (NL), and the number of equalities and inequalities in each category are shown. Note especially the number of unbounded variables and the number of constraints having nonconvex region effects in each model. Brief descriptions of each model follow:

- *FPhe1*: Heat-exchanger network. The four quadratic constraints are convex and concave, and have nonconvex region effects. Source is Floudas and Pardalos (1990, p. 63).
- *FPnlp3*: Nonlinear objective with all linear constraints. Source is Floudas and Pardalos (1990, p. 28).
- *FPnlp6*: Nonlinear test problem. The variables have very small ranges, so every test point is within the approximate feasibility tolerance in every test. Included for completeness in comparison with results by Kearfott and Dian (2000). Source is Floudas and Pardalos (1990, p. 30).

Table 1 Characteristics of the Test Models

	Variables		Constraints		Linear cons.		Quad. cons.		Gen. NL cons.	
Model	Tot.	Unbnded.	Tot.	Nonconv.	=	Ineq.	=	Ineq.	=	Ineq.
FPhe1	16	4	13	4	9	0	4	0	0	0
FPnlp3	4	2	3	0	1	2	0	0	0	0
FPnlp6	2	0	2	2	0	0	0	0	0	2
FPpb1	9	7	6	3	3	0	1	2	0	0
FPqp3	12	3	6	0	0	6	0	0	0	0
Hang	143	113	116	58	58	0	0	0	58	0
Test6	17	0	13	6	1	4	1	3	1	3
Electrns	150	150	50	50	0	0	50	0	0	0
Himmelblau6	43	43	14	0	14	0	0	0	0	0
Himmelblau15	6	0	4	4	0	0	0	0	0	4
Himmelblau20	24	24	20	12	2	0	12	0	0	6
Himmelblau23	100	100	12	0	0	12	0	0	0	0
FEA14.1.1	2	2	2	2	0	0	0	0	2	0
FEA14.1.2	5	5	5	5	0	0	0	2	0	3
FEA8.2.7	5	0	3	3	0	0	2	0	1	0
Sawpatterns	3	3	196	?	0	0	0	1	0	195

Note. Ranged inequalities are counted as two constraints.

- *FPpb1*: Pooling-blending problem. Source is Floudas and Pardalos (1990, p. 59).
- *FPqp3*: Quadratic program. Source is Floudas and Pardalos (1990, p. 8).
- *Hang*: Flight distance calculations for a hang glider. Source is Vanderbei (2002).
- *Test6*: Standard MProbe test model. It is infeasible, and includes functions known to cause calculation errors. Source is Chinneck (2002).
- *Electrns*: Positioning of electrons on a unit sphere. While the 50 quadratic constraints all have convex shapes, the fact that they are equality constraints means that they have a nonconvex region effect. The variable ranges are set at $\pm 10^6$ due to excessive function evaluation errors beyond this range. Source is Vanderbei (2002).
- *Himmelblau6*: A chemical equilibrium problem. The AMPL software (Fourer et al. 1993) used to express the model automatically reduces it slightly, so the variable and equation counts do not match those in the original source. Convergence is very slow unless the AMPL presolver is first applied. Results reported later are after the presolver has been applied. The presolver reduces the number of upper-unbounded variables from 43 to 5. The 5 unbounded variables are further restricted to a range of $\pm 10^6$. Iteration limit reset to 1,000 for $\alpha = 10$. Source is Himmelblau (1972).
- *Himmelblau20*: Minimization of blending costs. The upper-unbounded variables are restricted to a range of zero to 10^6. Source is Himmelblau (1972).
- *Himmelblau23*: All linear constraints. Source is Himmelblau (1972).
- *FEA14.1.1, FEA14.1.2*: Nonlinear systems of equations. These are used directly, not converted to an optimization problem as suggested in the book. The original small box constraints on the variables are removed and replaced by ranges of $\pm 10^5$. There are excessive function evaluation errors beyond this range. Source is Floudas et al. (1999, §§14.1.1 and 14.1.2).
- *FEA8.2.7*: Original box constraints on the variables of ± 5 are replaced by $\pm 10^4$. Above this range, there are numerous function evaluation errors and failures of the method to converge. Source is Floudas et al. (1999, §8.2.7).
- *Sawpatterns*: Unbounded variables restricted to $\pm 1{,}000$ due to excessive function and gradient evaluation errors at larger ranges. Source is Vanderbei (2002). The convexity status of the constraints in the model is unknown (shown as "?" in Table 1) due to excess function evaluation errors during MProbe testing. However, it is highly likely that almost all of the constraints have nonconvex region effects due to their algebraic form.

5.1. Experiment 1: Moving Closer to Feasibility

The first experiment demonstrates the ability of the algorithm to move from a distant point to a point that is much closer to the feasible region. To know the actual distance to feasibility, we require the orthogonal projection. We estimate this point by solving the following nonlinear optimization problem. Given a point $\mathbf{p}$, find the orthogonal projection $\mathbf{x}$ as follows:

$$\text{Minimize } Z = \left[(x_1 - p_1)^2 + (x_2 - p_2)^2 + \cdots + (x_n - p_n)^2\right]^{1/2} \text{ subject to constraints and variable bounds in the model.} \quad (2)$$

Table 2 Evaluation of the Constraint Consensus Method on the *FPhe1* Model

	CC to approx. feasibility			Feasibility distances			MINOS to feasibility		
Trial	Itns	Fcn evals	Grad evals	Initial	Final	Final/Initial	Itns	Fcn evals	Grad evals
1	26	351	351	1.402E+10	2.977E+02	2.124E−08	50	1,300	1,287
2	26	351	351	1.425E+10	3.082E+02	2.164E−08	59	1,872	1,859
3	25	338	338	5.501E+09	2.684E+02	4.879E−08	12	156	143
4	26	351	351	1.005E+10	3.095E+02	3.079E−08	24	663	650
5	26	351	351	1.069E+10	3.205E+02	2.999E−08	26	650	637
6	26	351	351	1.092E+10	3.365E+02	3.082E−08	40	1,157	1,144
7	26	351	351	1.433E+10	3.349E+02	2.337E−08	12	117	104
8	27	364	364	1.176E+10	2.916E+02	2.478E−08	11	143	130
9	27	364	364	1.538E+10	2.430E+02	1.580E−08	13	143	130
10	26	351	351	5.721E+09	2.912E+02	5.091E−08	17	377	364
Avg:	*26.1*	*352.3*	*352.3*	*1.126E+10*	*3.001E+02*	*2.981E−08*	*26.4*	*657.8*	*644.8*

The estimated distance to feasibility of point **p** is given by Z. This nonlinear optimization problem is solved here by MINOS (Murtagh and Saunders 1993).

The starting points for the constraint consensus algorithm are chosen randomly within the variable bounds; unbounded variables have a range of $\pm 10^{10}$. The MINOS estimated distance to feasibility is calculated at both the random initial point and at the final point returned by the constraint consensus algorithm. We expect to find that the final point is close to feasibility. The MINOS estimated distance to feasibility at the final point will not necessarily be less than α, though it should be in the same range.

This experiment uses the *FPhe1* model, which has 16 variables (4 unbounded) and 13 constraints (4 have nonconvex region effects). The results are summarized in Table 2. The constraint consensus method has $\alpha = 100$. The "feasibility distances" columns give the distance to the orthogonal projection, as calculated by solving system (2) using MINOS, for both the initial random point at which the constraint consensus (CC) method begins, and at the final point returned by the constraint consensus method. The "MINOS to feasibility" columns measure the effort that MINOS expends in achieving a first feasible point if started at the same initial point as the constraint consensus method. In all cases, the number of function and gradient evaluations refers to evaluations of individual constraints, not the entire set of constraints.

Both the constraint consensus method and MINOS are successful in achieving final feasibility (approximate feasibility for the constraint consensus method, exact feasibility for MINOS) in all ten trials. Note that the distances to feasibility at the initial random points are quite large, averaging $1.126\text{E} + 10$, but that the distances to feasibility at the final points are much smaller, averaging $3.001\text{E} + 02$. The average ratio of final to initial distance to feasibility is $2.981\text{E} - 08$, a difference of eight orders of magnitude. The constraint consensus method is very effective in moving from a distant point to a point that is quite close to feasibility. Further, it does so with relatively little effort, an average of just 352.3 function evaluations and 352.3 gradient evaluations.

As a comparison, Table 2 also shows the effort expended by MINOS in achieving feasibility from the same initial point. Since the constraint consensus method is only required to achieve approximate feasibility, this is not a fair comparison. However, it is interesting to note that the constraint consensus method expends around half the effort in terms of function and gradient evaluations to achieve approximate feasibility, as compared to effort expended by MINOS to achieve full feasibility. However, MINOS also carries out a linear programming solution and a reduced-gradient solution at each iteration, which adds a great deal more calculation burden than expended by the constraint consensus method. A final observation is that the effort expended by the constraint consensus method is remarkably consistent between trials, even though the model has four constraints that have nonconvex region effects. The effort expended by MINOS is more highly variable, probably due to the same nonconvex effects.

5.2. Experiment 2: Frequency of Success

The second experiment evaluates the success of the method when applied to the entire set of models listed in Table 1. The results are summarized in Table 3. Variables are restricted to various large ranges, as noted; an "unbounded" variable is restricted to a range of $\pm 10^{10}$. In some models, the original bounds on the variables are expanded to test the ability of the constraint consensus algorithm to achieve near-feasibility from larger distances. Results are collected at $\alpha = 100$ and at $\alpha = 10$; $\beta = 0.5$ in all cases. In most cases the method is halted after 500 iterations and the attempt to achieve near-feasibility is deemed to have failed.

Table 3 Empirical Results

	$\alpha = 100$				$\alpha = 10$			
Model	Success	Itns	Fcn evals	Grad evals	Success	Itns	Fcn evals	Grad evals
FPhe1	100	26.2	353.3	353.3	100	29.7	399.0	399.0
FPnlp3	100	135.3	408.9	408.9	100	153.8	464.3	464.3
FPnlp6	100	0.0	2.0	0.7	100	0.0	2.0	0.6
FPpb1	100	209.4	1,262.5	1,084.1	100	242.7	1,462.3	1,224.9
FPqp3	100	174.9	1,055.2	1,055.2	100	198.9	1,199.1	1,199.1
Hang	100	29.6	3,551.9	3,551.9	100	37.4	4,452.1	4,452.1
Test6	90	0.5	17.5	10.6	0	n.a.	n.a.	n.a.
Electrns	100	13.0	702.0	702.0	100	17.0	900.0	900.0
Himmelblau6	66	457.7	6,421.1	6,421.1	100	654.6	9,178.8	9,178.8
Himmelblau20	100	253.0	5,080.2	3,704.1	100	353.5	7,089.8	5,104.4
Himmelblau23	100	76.7	931.9	543.6	100	86.9	1,054.3	615.0
FEA14.1.1	100	22.1	46.2	46.2	100	35.2	72.4	72.4
FEA14.1.2	100	86.5	437.5	437.5	90	108.7	548.5	548.5
FEA8.2.7	100	331.9	998.7	998.7	67	1,376.3	4,132.0	4,132.0
Sawpatterns	77	1.4	460.7	2.4	79	3.6	893.2	4.6

In Table 3, each model is started at 100 random points at $\alpha = 100$, and again for another 100 random points at $\alpha = 10$. The random points are uniformly distributed between the upper and lower bounds on the variables. The *success* columns indicate how many of the 100 trials terminate successfully. The *itns* columns give the average number of iterations for each successful termination. The *fcn evals* and *grad evals* columns show the average number of individual function and gradient evaluations, respectively, per successful termination.

Table 3 shows that the constraint consensus method is quite effective, at least over the 15 models tested. This is true even when the fraction of constraints that have nonconvex region effects is relatively large. For example, 58 of the 116 constraints in the *hang* model have nonconvex region effects, as do all 50 of the constraints in the *electrns* model. This is an important observation for projection algorithms in general: Empirical results for models that include constraints having nonconvex region effects are very good. The method is also unaffected by models having primarily or only equality constraints.

Test6 is a model that is known to be infeasible, and also contains functions that are known to cause calculation errors. Still, the constraint consensus method is able to converge in 90% of the tests at $\alpha = 100$. However, because the model is infeasible, the method does not ever converge at $\alpha = 10$. *Himmelblau6* is also noteworthy in that while it has only linear equality constraints, it does take numerous iterations to converge. This may be due to a degree of ill conditioning.

As expected, tightening the tolerance from $\alpha = 100$ to $\alpha = 10$ increases the average number of iterations required for successful termination. This is usually a relatively modest increase, except in *FEA8.2.7* where the number of iterations required quadruples.

5.3. Experiment 3: Comparison to Another Method
A comparison can be made with results obtained by Kearfott and Dian (2000), who also addressed the issue of finding approximate feasible points from random initial points. They tested three methods: purely random sampling (method *pure-rand*) and two methods that use a generalized Newton method for underdetermined systems. This approach is related to projection methods in that the constraint gradients and the constraint violations are used to adjust the position of the approximate point. However, the method requires the inversion of a matrix, which Kearfott and Dian obtain via a pseudoinverse routine. In one method, all inequality constraints are converted to equality constraints by the addition of slack variables for use in the generalized Newton method (method *slack*), while in the other method satisfied inequality constraints are ignored (method *rand-GN*).

Kearfott and Dian applied their method to eight models, five of which also appear in Table 3: *FPhe1*, *FPnlp3*, *FPnlp6*, *FPpb1*, and *FPqp3*. Note that the terminating condition for Kearfott and Dian is a function-based error tolerance of 10^{-6}, much tighter than that used in Table 3. Note also that the original variable bounds are used for all models; note especially the very tight bounds in *FPnlp6*. Their results for the five comparable models are summarized in Table 4. The "Success" column in Table 4 gives the percentage of trials in which the algorithm terminated successfully.

The results in Tables 3 and 4 are not directly comparable given the differences in the tolerances (a function error of less than 10^{-6} for Kearfott and Dian versus an estimated feasibility distance tolerance of $\alpha = 10$). The effort in each case is also not comparable: It is measured by the number of function and gradient evaluations in Table 3 and is given in "standard time units" in

Table 4 **Method of Kearfott and Dian (2000)**

Model	Method	Random points	Success
FPhe1	*pure-rand*	10^6	0
	slack	100	1
	rand-GN	100	1
FPnlp3	*pure-rand*	10^6	0
	slack	100	18
	rand-GN	100	42
FPnlp6	*pure-rand*	10^6	41
	slack	100	77
	rand-GN	100	100
FPpb1	*pure-rand*	10^6	0
	slack	100	19
	rand-GN	100	29
FPqp3	*pure-rand*	10^6	2
	slack	10,000	3
	rand-GN	100	6

Kearfott and Dian's paper. However, it is interesting to note that the constraint consensus method terminates successfully in 100% of the cases for these five models, and does so at the cost of relatively modest numbers of function and gradient evaluations, without the need for matrix inversions.

Note that *FPnlp6* has very tight bounds on the variables, so it is immediately satisfied at every initial point at $\alpha = 100$ and $\alpha = 10$ when using the constraint consensus method. Kearfott and Dian are also able to achieve 100% success on this model when using the *rand-GN* method.

6. Conclusions

While development continues, there are reasons for optimism about the constraint consensus method. It is simple to implement and has proven effective in the tests to date. Calculation effort is small, consisting almost entirely of function and gradient evaluations, and does not require line searches, linear-programming solutions, reduced-gradient calculations, matrix inversions, etc. Perhaps most importantly, it has proven quite robust, even for models including numerous constraints that have nonconvex region effects. This robustness is thought to be due to the "voting" behavior in which poor movement directions suggested by a few constraints are "outvoted" by better directions suggested by the majority of the constraints. On the negative side, the constraint consensus method suffers from the usual drawbacks of gradient-based methods when faced with ill-conditioning effects and local minima.

Projection methods in general provide a way of exactly determining the variable-space closeness of a linear constraint to feasibility, as opposed to the usual practice of estimating closeness to feasibility based on function-space measures. Adoption of this variable-space measure may improve the accuracy of linear-programming solutions since it eliminates scaling effects. In the same vein, projection methods provide a way of estimating the variable-space closeness to feasibility for nonlinear constraints. This may also prove to be a better method for nonlinear programming for similar reasons.

Finally, there are numerous avenues for future research. Work is already underway on using the constraint consensus method in a random-sampling approach to tighten the variable bounds in nonlinear programs. Adjustments to the algorithm can be expected as the research proceeds. For example, it may be straightforward to detect ill conditioning when a relatively small component movement results even though the constituent feasibility vector components are themselves much larger. This may be the signal to apply a relaxation parameter to lengthen the movement in the recommended direction.

Acknowledgments
This research is supported by a research grant to the author from the Natural Sciences and Engineering Research Council of Canada. Yair Censor (University of Haifa) provided valuable advice on early drafts of the paper, as did Michael Saunders (Stanford University).

References

Censor, Y., T. Elfving, G. T. Herman. 2001. Averaging strings of sequential iterations for convex feasibility problems. D. Butnariu, Y. Censor, S. Reich, eds. *Inherently Parallel Algorithms in Feasibility and Optimization and Their Applications*. Elsevier Science B. V., Amsterdam, The Netherlands, 101–113.

Censor, Y., D. Gordon, R. Gordon. 2001. Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel Comput.* **27** 777–808.

Censor, Y., S. A. Zenios. 1997. *Parallel Optimization: Theory, Algorithms, and Applications.* Oxford University Press, New York.

Chinneck, J. W. 2001a. Analyzing mathematical programs using MProbe. *Ann. Oper. Res.* **104** 33–48.

Chinneck, J. W. 2001b. Discovering the characteristics of mathematical programs via sampling. *Optim. Methods Software* **17** 319–352.

Chinneck, J. W. 2002. MProbe web page, http://www.sce.carleton.ca/faculty/chinneck/mprobe.html.

Cimmino, G. 1938. Calcolo Approssimato per Soluzioni dei Sistemi di Equazioni Lineari. *La Ricerca Sci. XVI, Ser. II, Anno IX* **1** 326–333.

Floudas, C. A., P. M. Pardalos. 1990. *A Collection of Test Problems for Constrained Global Optimization Algorithms.* Springer-Verlag, Berlin, Germany.

Floudas, C. A., P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer, C. A. Schweiger. 1999. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Fourer, R., D. M. Gay, B. W. Kernighan. 1993. *AMPL: A Modeling Language for Mathematical Programming*. Boyd & Fraser Publishing Company, Danvers, MA.

Himmelblau, D. M. 1972. *Applied Nonlinear Programming*. McGraw-Hill Book Company, New York.

Kearfott, R. B., J. Dian. 2000. An iterative method for finding approximate feasible points. Technical report, Department of Mathematics, University of Southwestern Louisiana, Lafayette, LA.

Murtagh, B. A., M. A. Saunders. 1993. MINOS 5.4 User's Guide (Preliminary). Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA.

Pardalos, P. M., M. G. C. Resende. 2002. *Handbook of Applied Optimization*. Oxford University Press, Oxford, U.K.

Polak, E. 1997. *Optimization: Algorithms and Consistent Approximations*. Springer-Verlag, New York.

Vanderbei, R. 2002. Web Repository of AMPL Test Models. http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/index.html.

Xiao, Y., Y. Censor, D. Michalski, J. Galvin. 2003. The least-intensity feasible solution for aperture-based inverse planning in radiation therapy. *Ann. Oper. Res.* **119** 183–203.