Improving Solver Success in Reaching Feasibility for Sets of Nonlinear Constraints

Walid Ibrahim
College of Information Technology
United Arab Emirates University

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

August 3, 2005

Abstract

Whether a given nonlinear solver can reach a feasible point for a set of nonlinear constraints depends heavily on the initial point provided. We develop a range of computationally cheap constraint consensus algorithms that move from a given initial point to a better final point that is then passed to the nonlinear solver. Empirical tests show that this added step greatly improves the success rate of various nonlinear solvers in reaching feasibility, and reduces the effort they expend in doing so. We also develop a new initial point placement heuristic for use when an initial point is not provided by the modeller. Empirical tests show much improved performance for this new heuristic, both alone and in conjunction with the constraint consensus algorithms.

1. Introduction

Finding a feasible point for a set of linear constraints is not difficult, but it can be extremely challenging when the set has one or more nonlinear members. This is an important problem: feasibility is the only goal in some applications, and some nonlinear solution algorithms require a feasible starting point before they can proceed to optimality (e.g. the Generalized Reduced Gradient algorithm [Drud 1994], feasible sequential quadratic programming [Lawrence and Tits 2001], or methods of feasible directions [Lasdon 1970]).

The problem can be stated formally as follows. Given a set of a set of I constraints $c_1 \ldots c_I$, in J variables $x_1 \ldots x_J$ of the form $c_i(\boldsymbol{x})$ $\{\leq, \geq, =\}$ b_i where b_i is a constant, with variable bounds $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$, find $\boldsymbol{x}$ such that all of the constraints and variable bounds are satisfied. An initial point $\boldsymbol{x_0}$ may possibly be provided by the modeller. For the purposes of this paper, at least one constraint is nonlinear. There is no objective function, or if one is present, it is ignored.

The initial point provided by the modeller greatly influences the ability of a nonlinear solver to find a feasible point for a set of constraints that has at least one nonlinear member. In the best case, a knowledgeable modeller provides a good initial point that allows the solver to proceed to

feasibility without difficulty. In the worst case, the initial point is poorly chosen by a novice or inept modeller so that the solver is unable to find a feasible point at all, or perhaps more frequently, no initial point is given whatsoever. The difficulty of achieving feasibility is increased when the modeller fails to provide appropriate bounds on the variables. This is common in practice when the modeller knows little about the model, for example during the original formulation. The variables are often treated as unbounded in one or both directions in this case, so there are few clues as to where to set the initial point. Simple heuristics are usually used in this case.

A common approach to finding a feasible solution for a set of nonlinear constraints, given an initial point, is to use the solver to minimize a penalty function (e.g. sum of the squares of the individual constraint violations) during a phase one procedure whose sole purpose is to find a feasible point (see e.g. section 14.5 of Rardin [1998]). A variety of other approaches, internal to the nonlinear solvers, may also be used. Some algorithms move towards feasibility and optimality simultaneously, such as infeasible-path interior point methods (see e.g. Wright [1997]), filter methods [Gould et al. 2004]), and the common heuristic of optimizing a weighted sum of the original objective function and the phase one objective function.

There are, however, two classes of algorithms that operate *before* the initial point is turned over to the nonlinear solver: (*i*) initial point placement heuristics that are used when the modeller does not provide an initial point, and (*ii*) inexpensive algorithms for improving a given initial point prior to passing it to the nonlinear solver (especially important when the modeller provides an inappropriate initial point that is far from feasibility). This paper develops algorithms in both classes and shows that they can dramatically improve solver success rates in finding feasible points for sets of constraints that have at least one nonlinear member. The heuristics are simple, fast, and reliable. In addition, we examine the case of improving a relatively good initial point by applying an inexpensive heuristic in order to reduce the amount of costly nonlinear solver computation.

The new heuristics for improving a given initial point are variations on the constraint consensus method [Chinneck 2003], briefly reviewed next. This is a type of simultaneous component-averaging gradient projection algorithm (see e.g. Censor [2001]). The main idea is to improve the current point by making a move that arises as a result of the "consensus of opinion" among the currently violated constraints as to the direction and distance in which to move to achieve feasibility. These movements are repeated until a stopping condition is met.

The first step in the constraint consensus method is to find the *feasibility vector* for each constraint that is violated at the current point $\boldsymbol{x}$. This is $\boldsymbol{fv}_i = v_i d_i \nabla c_i(\boldsymbol{x}) / \|\nabla c_i(\boldsymbol{x})\|^2$ for the ith constraint where:

- $\nabla c_i(\boldsymbol{x})$ is the gradient of the constraint, and $\|\nabla c_i(\boldsymbol{x})\|$ is its length.
- v_i is the *constraint violation* $|c_i(\boldsymbol{x}) - b_i|$, or zero for satisfied constraints,
- d_i is +1 if it is necessary to increase $c(\boldsymbol{x})$ to satisfy the constraint, and –1 if it is necessary to decrease $c_i(\boldsymbol{x})$ to satisfy the constraint.

The feasibility vector shows how to move from the current point to the nearest feasible point for an individual constraint. It is exact for linear constraints, but just an estimate for nonlinear constraints. The length of the feasibility vector for the ith constraint is denoted by $\|\boldsymbol{fv}_i\|$.

The second step is to combine the feasibility vectors for all of the violated constraints to arrive at the *consensus vector* that is actually used to make the updating move. This is done in a component-wise manner: only the violated constraints that include a particular variable in $c(\boldsymbol{x})$ are able to "vote" on the movement in that dimension. In the original algorithm the movement in each dimension is obtained by averaging the relevant component of each eligible feasibility vector; the resulting consensus vector specifies both the direction and distance of movement. The current point is updated by applying the consensus vector. The first and second steps repeat until the stopping conditions are met.

The algorithm terminates successfully if the length of every feasibility vector is less than the feasibility distance tolerance α, and unsuccessfully if either (*i*) the first condition is not met and the length of the consensus vector is less than the movement tolerance β or (*ii*) a preset number of iterations μ is exceeded. When successful, the final point is within an estimated Euclidean distance α of satisfying every constraint, where α might be quite large (e.g. 100) depending on the purpose at hand (e.g. finding the order of magnitude of a suitable starting point for the nonlinear solver). The movement tolerance β is used to detect situations in which the algorithm gets stuck or is proceeding very slowly.

The basic constraint consensus method is shown in Algorithm 1. NINF is the number of violated constraints ("Number of INFeasibilities") at the current point, s_j is the sum of the feasibility vector components in the jth dimension, n_j is the number of violated constraints that involve variable j,

Inputs:

- a set of I constraints $c_1 \ldots c_I$, in J variables $x_1 \ldots x_J$
- an initial point $\boldsymbol{x}$,
- a feasibility distance tolerance α,
- a movement tolerance β,
- a maximum number of iterations μ.

1. Repeat μ times:
   - 1.1. NINF = 0; for all j: $n_j = 0$, $s_j = 0$.
   - 1.2. For every constraint c_i:
     - 1.2.1. If c_i is violated then:
       - 1.2.1.1. Calculate feasibility vector $\boldsymbol{fv}_i$ and the feasibility distance $\|\boldsymbol{fv}_i\|$
       - 1.2.1.2. If $\|\boldsymbol{fv}_i\| > \alpha$ then:
         - NINF = NINF + 1.
         - For every variable x_j in c_i: $n_j \leftarrow n_j + 1$; $s_j \leftarrow s_j + fv_{ij}$
   - 1.3. If NINF = 0, then exit successfully.
   - 1.4. For every variable x_j:
     - 1.4.1. If $n_j \neq 0$ then $t_j = s_j / n_j$, else $t_j = 0$.
   - 1.5. If $\|\boldsymbol{t}\| \leq \beta$ then exit unsuccessfully.
   - 1.6. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{t}$.
   - 1.7. If any x_j exceeds its bounds, reset onto the nearest bound.
2. Exit unsuccessfully.

Algorithm 1: Basic Constraint Consensus

and ***t*** is the consensus vector. For simplicity, details of how the basic algorithm tolerates numerical errors are not shown. Briefly, the algorithm ignores constraints that experience a numerical error at the current point (e.g. divide by zero) and carries on, hoping that the problem will not recur at the newly updated point. If the algorithm returns a final point at which at least one constraint experiences a numerical error, then the termination is deemed unsuccessful. See Chinneck [2003] for details.

There is a variety of previous work on relatively complex feasibility-seeking procedures to be used as part of a solver phase-one procedure (see, e.g. Elwakeil and Arora [1995]). However there has been very little previous work on inexpensive methods for improving a given initial point, not necessarily all the way to feasibility. Chen and Kostreva [1999] describe a feasible directions method that is limited to solving nonlinear inequalities, and is to be used prior to optimization via the method of feasible directions. Gertz et al [2003] describe an approach that computes an affine scaling step by solving a system of linear equations related to a Newton iteration. Their algorithm is specifically for interior point methods in that it also provides initial values of other multipliers and parameters used by such methods.

Heuristic methods for selection of the initial point when none is provided have received much less attention, and do not have an extensive literature. Some solvers provide a nonlinear "crash" heuristic to set the initial point, see e.g. the procedure used in MINOS [Murtagh and Saunders 1983], but details differ between implementations and are often confidential. A widely-applied heuristic (referred to here as the "standard" heuristic) is as follows:

- if the variable is doubly bounded: set at midpoint,
- if the variable is singly bounded: set at bound closest to zero,
- if the variable is unbounded in both directions: set at zero.

The remainder of the paper presents new constraint consensus algorithms in Section 2 and a more effective modification of the standard initial point placement algorithm in Section 3. Experiments to determine the effectiveness of these methods in providing good starting points for nonlinear solvers are then described and conclusions are drawn.

2. New Constraint Consensus Algorithms

The basic constraint consensus method treats all of the eligible feasibility vectors equally. However there may be value in choosing to emphasize the effect of the longest or shortest feasibility vector. This is the approach taken in the variations developed in Section 2.1. It may also be valuable to consider the *number* of constraints voting for a movement in the positive versus negative direction in a particular component. This is the basis of the algorithm variations developed in Section 2.2.

2.1 Feasibility-Distance Based Algorithms

The feasibility-distance based algorithms use the length of the feasibility vector associated with each violated constraint to set the consensus vector. In the "near" mode used in the *FDnear* algorithm, the consensus vector is set equal to the shortest feasibility vector on the assumption that it is better to move to satisfy the smallest violation first because this keeps the point in a region where the gradients are good approximations of the functions. In the "far" mode used in the *FDfar* algorithm, the opposite assumption is made and the consensus vector is set equal to

the longest feasibility vector because this is likely to provide the most rapid movement towards feasibility. In both cases, dimensions that do not appear in the selected shortest or longest feasibility vector are set by averaging as in the basic constraint consensus scheme. Details are shown in Algorithm 2, where *fd* is the maximum or minimum feasibility distance, and **z** is the shortest or longest feasibility vector. The FDfar approach is related to the "remotest set control" class of projection algorithms [Censor and Zenios 1997, p. 80].

2.2 Direction-Based Algorithms

The direction-based algorithms conduct a "vote" on whether to move in the positive or the negative direction for each dimension prior to deciding how far to move. In some variants, the direction vote is the simple count of how many violated constraints would prefer an increase in a dimension versus how many would prefer a decrease in that dimension. In other variants, the vote is settled by the size of the largest proposed movement in the positive versus negative direction: whichever direction has the largest proposed movement wins the vote. Once this vote settles the question of whether to increase or to decrease in the dimension, there are several ways to decide how far to move.

The *DBavg* method decides the direction of movement in a dimension by a simple count of the number of votes for positive or negative movement, and the magnitude of the movement is decided by averaging the projections in the winning direction, as shown in Algorithm 3. s^+_j and s^-_j are the sums of the feasibility vector components in the positive and negative directions for variable j and n^+_j and n^-_j are the number of violated constraints that vote for movement in the positive and negative directions for variable j.

In the *DBmax* variant, the direction vote is conducted by looking at the size of the largest proposed movement in each of the positive and negative directions: the largest proposed movement determines both the direction and the size of the component in the consensus vector. See Algorithm 4. This is again related to the "remotest set control" projection algorithm [Censor and Zenios 1997, p. 80], but is applied in a component-wise manner.

In the bound-type direction-based variant *DBbnd*, the direction vote is settled by a simple count of the number of votes for an increase or a decrease in the component. The size of the movement in each component depends on the types of constraints that include that variable. Movements in the selected direction suggested by equality constraints are totalled; for inequalities only the largest movement in the selected direction is added because the largest movement will satisfy all of the inequalities. The resulting total is then reduced to an average. See Algorithm 5. $n^{=+}_j$ and $n^{=-}_j$ represent the number of votes for the positive and negative directions for the jth variable recorded by violated equality constraints and max^+_j and max^-_j represent the largest positive and negative feasibility vector components for the jth variable in violated inequality constraints.

While all of the constraint consensus methods deal well with constraint scaling, they are vulnerable to discrepancies in variable scaling. The effect of uneven variable scaling could be more pronounced in the DBmax and DBbnd variants, so care should be taken with variable scaling prior to application of these heuristics.

Inputs:

- a set of I constraints $c_1 \ldots c_I$, in J variables $x_1 \ldots x_J$
- an initial point $\boldsymbol{x}$
- a feasibility distance tolerance α
- a movement tolerance β
- a maximum number of iterations μ
- *mode* (*near, far*)

1. Repeat μ times:
   - 1.1. NINF = 0; $k = 0$; for all j in $\boldsymbol{x}$: $n_j = 0$, $s_j = 0$, $z_j = 0$.
   - 1.2. If *mode* = *near* then $fd = \infty$, else $fd = 0$.
   - 1.3. For each constraint c_i:
     - 1.3.1. If c_i is violated then:
       - 1.3.1.1. Calculate feasibility vector $\boldsymbol{fv}_i$ and feasibility distance $\|\boldsymbol{fv}_i\|$.
       - 1.3.1.2. If $\|\boldsymbol{fv}_i\| > \alpha$ then:
         - 1.3.1.2.1. NINF = NINF + 1
         - 1.3.1.2.2. For each variable j in c_i:
           - $s_j = s_j + fv_{ij}$; $n_j = n_j + 1$
           - If ((*mode* = *near*) and ($\|\boldsymbol{fv}_i\| < fd$)) or ((*mode* = *far*) and ($\|\boldsymbol{fv}_i\| > fd$)) then:
             - $k = i$
             - $\boldsymbol{z} \leftarrow \boldsymbol{fv}_i$
   - 1.4. If NINF = 0, exit successfully with final point $\boldsymbol{x}$.
   - 1.5. For each variable x_j:
     - 1.5.1. If x_j appears in c_k then $t_j = z_j$.
     - 1.5.2. Else if $n_j \neq 0$ then $t_j = s_j / n_j$, else $t_j = 0$.
   - 1.6. If $\|\boldsymbol{t}\| < \beta$, then exit unsuccessfully.
   - 1.7. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{t}$
   - 1.8. If any x_j exceeds its bounds, reset onto nearest bound.
2. Exit unsuccessfully.

Algorithm 2: Feasibility-Distance Based Constraint Consensus (FDnear, FDfar).

3. An Improved Initial Point Placement Heuristic

There are two main problems with the standard heuristic for initial point placement (as described near the end of the Introduction). First, it sets many variables to zero, which can cause numerical errors, e.g. for a constraint that includes a term such as $1/x$. Second, since many variables are given similar bounds by the modeller (e.g. unbounded or singly bounded), many of the variables are also given the same initial values. This can also cause numerical errors, e.g. in constraints that include terms like $1/(x_1-x_2)$.

Inputs:

- a set of I constraints $c_1 \ldots c_I$, and J variables $x_1 \ldots x_J$
- an initial point $\boldsymbol{x}$
- a feasibility distance tolerance α
- a movement tolerance β
- maximum number of iterations μ

1. Repeat μ times:
   - 1.1. NINF = 0; for all j: $s^+_j = 0$, $s^-_j = 0$, $n^+_j = 0$, $n^-_j = 0$.
   - 1.2. For each constraint c_i:
     - 1.2.1. If c_i is violated then:
       - 1.2.1.1. Calculate feasibility vector $\boldsymbol{fv}_i$ and feasibility distance $\|\boldsymbol{fv}_i\|$.
       - 1.2.1.2. If $\|\boldsymbol{fv}_i\| > \alpha$ then
         - 1.2.1.2.1. NINF = NINF + 1
         - 1.2.1.2.2. For each variable j in c_i:
           - If $fv_{ij} > 0$ then $s^+_j = s^+_j + fv_{ij}$ and $n^+_j \leftarrow n^+_j + 1$
           - If $fv_{ij} < 0$ then $s^-_j = s^-_j + fv_{ij}$ and $n^-_j \leftarrow n^-_j + 1$
   - 1.3. If NINF = 0 then return successfully with final point $\boldsymbol{x}$.
   - 1.4. For each variable x_j:
     - 1.4.1. If $n^+_j = n^-_j$ and $(n^+_j + n^-_j) > 0$ then $t_j = (s^+_j + s^-_j) / (n^+_j + n^-_j)$
     - 1.4.2. Elseif $n^+_j > n^-_j$ then $t_j = s^+_j / n^+_j$
     - 1.4.3. Else $t_j = s^-_j / n^-_j$
   - 1.5. If $\|\boldsymbol{t}\| < \beta$, then exit unsuccessfully.
   - 1.6. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{t}$
   - 1.7. If any x_j exceeds its bounds, reset onto nearest bound.
2. Exit unsuccessfully.

Algorithm 3: Average Direction-Based (DBavg) Constraint Consensus

For these reasons, we propose a simple modification to this heuristic that superimposes a random perturbation Δ on the initial values proposed above. The revised heuristic operates as follows:

- if the variable is doubly bounded: set at midpoint + Δ,
- if the variable has a single lower bound: set at bound + Δ,
- if the variable has a single upper bound: set at bound – Δ,
- if the variable is unbounded in both directions: set at zero + Δ,

where Δ is a uniformly distributed random number between 0 and 1 (or suitably smaller if the bounds on the variable define a smaller range).

Note that we use a positive perturbation when the variable is unbounded in both directions. This avoids numerical problems caused by some functions (e.g. square root) when the variable really should have been specified as nonnegative, or even as positive (e.g. the derivative of the square root blows up at zero).

Inputs:

- a set of I constraints $c_1 \ldots c_I$, and J variables $x_1 \ldots x_J$
- an initial point $\boldsymbol{x}$
- a feasibility distance tolerance α
- a movement tolerance β
- maximum number of iterations μ

1. Repeat μ times:
   - 1.1. NINF = 0; for all j: $s^+_j = 0$, $s^-_j = 0$, $n^+_j = 0$, $n^-_j = 0$.
   - 1.2. For each constraint c_i:
     - 1.2.1. If c_i is violated then:
       - 1.2.1.1. Calculate feasibility vector $\boldsymbol{fv}_i$ and feasibility distance $\|\boldsymbol{fv}_i\|$.
       - 1.2.1.2. If $\|\boldsymbol{fv}_i\| > \alpha$ then
         - 1.2.1.2.1. NINF = NINF + 1
         - 1.2.1.2.2. For each variable j in c_i:
           - If $fv_{ij} > 0$ then
             - $n^+_j \leftarrow n^+_j + 1$
             - If $fv_{ij} > s^+_j$ then $s^+_j \leftarrow fv_{ij}$
           - Else if $fv_{ij} < 0$
             - $n^-_j \leftarrow n^-_j + 1$
             - If $fv_{ij} < s^-_j$ then $s^-_j \leftarrow fv_{ij}$
   - 1.3. If NINF = 0 then return successfully with final point $\boldsymbol{x}$.
   - 1.4. For each variable x_j:
     - 1.4.1. If $n^+_j = n^-_j$ then $t_j = (s^+_j + s^-_j) / 2$
     - 1.4.2. Elseif $n^+_j > n^-_j$ then $t_j = s^+_j$
     - 1.4.3. Else $t_j = s^-_j$
   - 1.5. If $\|\boldsymbol{t}\| < \beta$, then exit unsuccessfully.
   - 1.6. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{t}$
   - 1.7. If any x_j exceeds its bounds, reset onto nearest bound.
2. Exit unsuccessfully.

Algorithm 4: Maximum Direction-Based (DBmax) Constraint Consensus

This revision avoids many of the numerical problems associated with the original heuristic while retaining its main features. As shown later, it provides a useful initial point for the solver more often than the original version.

4. Experimental Setup

The goal of the experiments is to show that the right combination of the new initial point placement and constraint consensus algorithms improves the success of standard NLP solvers in reaching feasibility, both when provided with an initial point by the modeller and when no such initial point is given. When an initial point is provided, we first run a constraint consensus algorithm to find an improved point, and then pass this final point to the solver. When no initial point is provided, we first apply the new randomized standard initial point placement heuristic, then run a constraint consensus algorithm to improve the point, and then pass this final point to the solver. We also examine the effectiveness of the new initial point placement heuristic used alone.

The main performance metric is the fraction of models for which a given NLP solver is able to reach feasibility. As shown later, the methods developed in this paper increase the success fraction greatly. We also look at the effort required by each solver to reach feasibility (generally the number of iterations or number of function and gradient evaluations) with and without first

Inputs:

- a set of I constraints $c_1 \ldots c_I$, and J variables $x_1 \ldots x_J$
- an initial point $\boldsymbol{x}$
- a feasibility distance tolerance α
- a movement tolerance β
- maximum number of iterations μ

1. Repeat μ times:
   - 1.1. NINF = 0; for all j: $s^+_j=0$, $s^-_j=0$, $n^+_j=0$, $n^-_j=0$, $n^{=+}_j=0$, $n^{=-}_j=0$, $max^+_j=0$, $max^-_j=0$.
   - 1.2. For each constraint c_i:
     - 1.2.1. If c_i is violated then:
       - 1.2.1.1. Calculate feasibility vector $\boldsymbol{fv}_i$ and feasibility distance $\|\boldsymbol{fv}_i\|$.
       - 1.2.1.2. If $\|\boldsymbol{fv}_i\| > \alpha$ then
         - 1.2.1.2.1. NINF = NINF + 1
         - 1.2.1.2.2. For each variable j in c_i:
           - If $fv_{ij} > 0$ then
             - $n^+_j \leftarrow n^+_j + 1$
             - If c_j is an equality constraint then $s^+_j \leftarrow s^+_j + fv_{ij}$ and $n^{=+}_j \leftarrow n^{=+}_j + 1$
             - Else if $fv_{ij} > max^+_j$ then $max^+_j \leftarrow fv_{ij}$
           - If $fv_{ij} < 0$ then
             - $n^-_j \leftarrow n^-_j + 1$
             - If c_j is an equality constraint then $s^-_j \leftarrow s^-_j + fv_{ij}$ and $n^{=-}_j \leftarrow n^{=-}_j + 1$
             - Else if $fv_{ij} < max^-_j$ then $max^-_j \leftarrow fv_{ij}$
   - 1.3. If NINF = 0 then return successfully with final point $\boldsymbol{x}$.
   - 1.4. For each variable x_j:
     - 1.4.1. If $max^+_j \neq 0$ then
       - 1.4.1.1. $s^+_j \leftarrow s^+_j + max^+_j$
       - 1.4.1.2. $n^{=+}_j \leftarrow n^{=+}_j + 1$
     - 1.4.2. If $max^-_j \neq 0$ then
       - 1.4.2.1. $s^-_j \leftarrow s^-_j + max^-_j$
       - 1.4.2.2. $n^{=-}_j \leftarrow n^{=-}_j + 1$
     - 1.4.3. If $n^+_j = n^-_j$ then
       - 1.4.3.1. $t_j = (s^+_j + s^-_j)/(n^{=+}_j + n^{=-}_j)$
     - 1.4.4. Else if $n^+_j > n^-_j$ then $t_j = s^+_j / n^{=+}_j$
     - 1.4.5. Else $t_j = s^-_j / n^{=-}_j$
   - 1.5. If $\|\boldsymbol{t}\| < \beta$, then exit unsuccessfully.
   - 1.6. $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{t}$
   - 1.7. If any x_j exceeds its bounds, reset onto nearest bound.
2. Exit unsuccessfully.

Algorithm 5: Direction-Based and Bound-Based (DBbnd) Constraint Consensus

applying the new heuristics. The effort associated with the heuristics themselves is generally minimal.

4.1 Test Models

We used 231 models from the CUTE suite of test models [Bongartz et al 1995]. The selection criteria were: (i) each model has at least one nonlinear constraint, and (ii) each model has fewer than 300 variables and fewer than 300 constraints. The second criterion was necessitated by our use of limited-size editions of some of the nonlinear solvers. Summary statistics for the 231 models are given in Table 1. The original objective function is replaced by a dummy objective function (maximize $f(\mathbf{x}) = 0$) so that the solver halts at the first feasible solution. This choice of objective function may have some effect on solvers that work towards feasibility and optimality simultaneously.

The CUTE models are specified in the AMPL modelling language [Fourer et al 2003]. AMPL has a number of presolving options that can simplify the model prior to submission to a solver, and thereby also affect the initial point sent to the solver. To avoid confounding effects, the AMPL presolver is turned off in all experiments. If the user does not provide an initial value for a variable, then AMPL tentatively sets its value to zero. Of course the solver itself is free to reset any initial point that it is given, prior to launching its main solution algorithm.

	average	**minimum**	**maximum**
number of variables	24.1	2	300
singly bounded	2.5	0	100
doubly bounded	7.4	0	202
Fixed	0.2	0	12
Free	14.0	0	288
number of constraints	24.2	1	256
linear equalities	2.4	0	60
nonlinear equalities	10.7	0	150
linear inequalities	3.5	0	128
nonlinear inequalities	7.6	0	200
nonzeroes in Jacobian	421.1	2	12797
nonzeroes in Hessian	177.2	1	6105

Table 1: Summary Statistics for Test Models.

Most of the models (199/231 or 86%) provide a suggested initial point, presumably selected by a knowledgeable modeller as a good starting point. We compare solver success rate in reaching feasibility when starting at the modeller-supplied initial point versus the solver success rate when using our initial point placement and improvement algorithms instead. We wish to show that these new algorithms can provide performance equal to that of a knowledgeable modeller in selecting initial points from which to launch nonlinear solvers.

4.2 Initial Point Placement Heuristics

An initial point placement heuristic must be applied when no initial point is provided by the modeller, or when the initial point that is provided is not complete in all dimensions. We consider the following initial point placement methods:

- Random placement of initial points within the variable bounds.
- The origin (all variables set at 0.0).
- The standard heuristic (see latter part of Section 1).
- The new randomized standard heuristic (see Section 3).
- The CUTE-supplied initial point, with unspecified variables set to zero (labelled as CUTE/origin point).

4.3 Constraint Consensus Parameters

The feasibility tolerance α is varied in the experiments assessing the new constraint consensus variants (Section 5.1). Values of 100, 10, 1, and 0.1 are used to assess the effect of this parameter. For the later experiments (Sections 5.2 and 5.3), α is fixed at 0.1. The rationale is that it is worth spending a small amount of extra effort during the constraint consensus phase (which is computationally cheap) to provide the nonlinear solver with a starting point that is closer to feasibility. Initial experiments, not reported here, show that using the smaller α does indeed boost the nonlinear solver success rate.

The movement tolerance β is fixed at 0.001 in all experiments. This is sufficient to detect poor performance of the algorithm. The maximum number of iterations μ is fixed at 500 in all experiments.

In the case of unsuccessful termination of the constraint consensus method (movement vector too short or number of iterations exceeded), the current point at termination is passed to the nonlinear solver on the grounds that it may still be an improvement over the initial point.

4.4 Nonlinear Solvers

Five well-known and well-respected commercially available nonlinear solvers were used in the experiments:

- MINOS 5.5 [Murtagh and Saunders 1983] solves a sequence of subproblems using linearized versions of the constraints and an objective function based on the Lagrangian.
- SNOPT 6.1-1 [Gill, Murray and Saunders 1997] uses a sequential quadratic programming algorithm with quasi-Newton approximations to the Hessian of the Lagrangian.
- KNITRO 4.0 [Waltz and Nocedal 2003] solves a sequence of subproblems using a barrier method and trust regions. It also uses Hessian information.
- DONLP2 [Spellucci 1998] uses a sequential quadratic programming method involving an L1 merit function and a quasi-Newton approximation to the Hessian.
- CONOPT 3.13 [Drud 1994] uses an efficient Generalized Reduced Gradient algorithm. Hessian information is used.

Default parameter settings were used for all solvers. The KNITRO *iprint* parameter was set to 1 to reduce the amount of printed output.

5. Experimental Results

5.1 New Constraint Consensus Algorithms

We first assess the new constraint consensus algorithms prior to using them to provide points to launch the nonlinear solvers. In these experiments we measure the fraction of models for which each algorithm terminates successfully (i.e. does not terminate unsuccessfully due to a consensus vector with length less than β or more than μ iterations). One hundred uniformly distributed random initial points within the variable bounds are tested for each model; hence each success fraction is measured over 23,100 trials.

The success rates are summarized in Table 2, along with the fraction of models that terminate unsuccessfully due to numerical errors or excessive iterations (statistics on the relatively rare termination due to the consensus vector being too short are not shown). The best numbers in each column are shown in bold. Table 2 shows that all of the new variants have better success rates than the basic method, with the exception of FDnear, which is uniformly worse. For FDnear, the shorter steps lead to fewer cases of numerical error, but more steps in total, thus more cases in which the iteration limit is reached. The DBmax method dominates the other methods in terms of success rate. As expected, the success fraction drops as α decreases, generally because more solutions exceed the iterations limit.

	Success				**Numerical Error**				**Too Many Iterations**			
α	**100**	**10**	**1**	**0.1**	**100**	**10**	**1**	**0.1**	**100**	**10**	**1**	**0.1**
Basic	0.551	0.531	0.516	0.444	0.230	0.226	0.212	0.205	0.218	0.241	0.268	0.335
DBmax	**0.578**	**0.559**	**0.535**	**0.477**	0.240	0.236	0.224	0.221	**0.183**	**0.205**	**0.242**	**0.301**
DBavg	0.558	0.539	0.519	0.460	0.231	0.227	0.214	0.209	0.211	0.234	0.267	0.330
DBbnd	0.558	0.539	0.521	0.461	0.232	0.228	0.214	0.206	0.209	0.233	0.265	0.330
FDnear	0.520	0.504	0.486	0.418	**0.220**	**0.215**	**0.202**	**0.203**	0.260	0.281	0.311	0.379
FDfar	0.571	0.551	0.531	0.466	0.239	0.236	0.222	0.220	0.189	0.213	0.247	0.314

Table 2: Exit Status for the Constraint Consensus Variants (in Fraction of Models Tested).

Table 4 and Table 5 examine the effort required for the various algorithms to reach feasibility at different values of α. For a fair comparison, we examine effort on only the subset of the models for which every algorithm terminated successfully. Table 3 shows the fraction of the models in this subset at each value of α.

α	**100**	**10**	**1**	**0.1**
Fraction of Models	0.504	0.483	0.466	0.397

Table 3: Fraction of Models for Which All Constraint Consensus Algorithms Terminated Successfully.

Table 4 shows the effort required at various values of α, in terms of the average number of constraint consensus iterations required over the subset of models. The best results in each column are shown in bold. Both the DBmax and the FDfar algorithms require noticeably fewer iterations than the other methods to reach feasibility across the subset of models while FDnear always requires the most. Note that the number of iterations generally increases between α=100 and α=0.1 because it takes longer to get closer to feasibility. However there are fluctuations due to the differing success fractions (shown in Table 3).

α	100	10	1	0.1
Basic	41.1	41.0	48.3	48.2
DBmax	32.4	31.8	37.1	38.8
DBavg	39.6	39.9	46.1	46.8
DBbnd	37.7	37.8	44.3	43.7
FDnear	51.6	52.3	61.7	63.4
FDfar	**31.4**	**31.2**	**36.8**	**38.7**

Table 4: Average Number of Constraint Consensus Iterations over Compared Subset.

Table 5 shows the effort in terms of the number of gradient evaluations for each constraint consensus method over the compared subset of models. The best result in each column is shown in bold. This is the total number of individual gradient evaluations (recall that gradients are evaluated only for the violated constraints). The number of function evaluations is not shown, but this is easily calculated because every function is evaluated at every iteration to determine its current feasibility status.

α	100	10	1	0.1
Basic	713.9	589.6	687.5	558.9
DBmax	**573.1**	**458.8**	**523.2**	**380.2**
DBavg	662.6	547.0	623.3	509.4
DBbnd	657.6	532.7	617.2	482.3
FDnear	791.7	659.0	831.1	645.2
FDfar	595.2	472.2	538.3	461.5

Table 5: Average Number of Gradient Evaluations over Compared Subset.

5.2 Solver Success in Reaching Feasibility

Finding a feasible point involves the combination of (i) a given initial point or initial point placement heuristic, (ii) a constraint consensus algorithm, and (iii) an NLP solver. Table 7 summarizes the solver success fraction for all 175 combinations studied here (5 initial point placement alternatives × 7 constraint consensus alternatives × 5 NLP solvers). The best success fraction in each column is shown in boldface; results within 0.01 of the best success fraction are shown in bold italics. Note that one model corresponds to $1/231 \approx 0.0043$ of the total models, so it takes about 2 models to make a difference of 0.01 in the success fraction results.

Table 8 measures the effort used by each solver for the various combinations of initial point placement and constraint consensus alternatives. It shows the number of solver iterations for a comparable subset of models. The comparable subset consists of all of the models for which a given initial point placement and solver combination reaches feasibility for every constraint consensus alternative. Table 9 shows the number of models that is included in each calculation. As can be seen in Table 8, using any constraint consensus method reduces solver effort in all cases, with just two exceptions (both of which are for algorithm combinations that are not recommended, as shown later). In some combinations, using a constraint consensus method reduces the number of solver iterations by more than half.

More specific analysis follows below.

5.2.1 Initial Point Placement

The effect of the initial point placement method alone can be seen in Table 7 by looking at the row labelled "none" in each block. With few exceptions, the solver success fraction increases in the order shown in the table:

1. random initial point
2. origin initial point
3. standard heuristic initial point
4. randomized standard heuristic initial point
5. CUTE/origin initial point.

As expected, the CUTE/origin initial point normally provides the highest solver success fraction when constraint consensus is omitted, but there are exceptions: the randomized standard heuristic provides better results than the CUTE/origin initial point for SNOPT. It is especially interesting to note the dramatic improvement in solver success fraction between the standard heuristic and the randomized heuristic in all cases.

Note that the solver success fraction using only the randomized standard heuristic (without a subsequent constraint consensus improvement) is close to the overall best success fraction for DONLP2. This shows the impact that a good initial point placement heuristic can have.

5.2.2 Reaching Feasibility without a Nonlinear Solver

Some combinations of the initial point algorithm and a constraint consensus method are able to achieve feasibility directly, eliminating the need to apply a nonlinear solver at all. Table 6 summarizes the fraction of the 231 models in which this happens. We consider a model to be feasible if every constraint is satisfied or violated by no more than 0.000001, a tolerance commonly used in solvers.

As shown in Table 6, applying any constraint consensus method greatly increases the number of models for which a feasible point is found without applying a nonlinear solver.

	Initial Point Placement Method				
CC	**Random**	**Origin**	**Standard**	**Randomized Std**	**CUTE/origin**
none	0.026	0.216	0.212	0.065	0.156
Basic	0.108	0.286	0.281	0.152	0.247
DBmax	0.104	0.281	0.277	0.160	0.251
DBavg	0.108	0.281	0.277	0.152	0.251
DBbnd	0.108	0.281	0.277	0.160	0.251
FDnear	0.121	0.286	0.281	0.152	0.255
FDfar	0.100	0.290	0.286	0.152	0.247

Table 6: Fraction of Models Feasible Before an NLP Solver is Applied.

It is also interesting to note that the simpler initial point placement heuristics (origin and standard heuristic) more often result in feasible points, finding them even more often than the relatively good CUTE/origin point. This likely reflects the fact that the origin itself is more frequently a feasible point in this set of test problems. The randomized standard heuristic is less successful in

choosing a feasible point by itself or in combination with a constraint consensus method, but still provides better launch points for the NLP solver, as shown in Table 7.

CC	MINOS	SNOPT	KNITRO	CONOPT	DONLP2
Random Initial Point					
none	0.364	0.420	0.714	0.580	0.550
Basic	0.541	0.602	0.745	0.654	0.584
DBmax	0.567	0.602	0.753	0.675	0.580
DBavg	0.554	0.593	0.771	0.675	0.567
DBbnd	0.558	0.606	0.749	0.675	0.567
FDnear	0.537	0.576	0.749	0.675	0.563
FDfar	0.541	0.615	0.766	0.662	0.576
Origin Initial Point					
none	0.680	0.671	0.749	0.593	0.675
Basic	0.684	0.688	0.775	0.649	0.667
DBmax	0.688	0.688	0.775	0.649	0.680
DBavg	0.697	0.693	0.771	0.662	0.684
DBbnd	0.693	0.684	0.775	0.649	0.675
FDnear	0.697	0.688	0.766	0.649	0.684
FDfar	0.688	0.680	0.766	0.654	0.671
Standard Heuristic Initial Point					
none	0.688	0.671	0.749	0.602	0.675
Basic	0.710	0.706	0.766	0.675	0.680
DBmax	0.719	0.706	0.762	0.671	0.688
DBavg	0.732	0.706	0.758	0.697	0.697
DBbnd	0.727	0.701	0.762	0.675	0.675
FDnear	0.719	0.706	0.758	0.671	0.688
FDfar	0.714	0.697	0.762	0.680	0.680
Randomized Standard Heuristic Initial Point					
none	0.792	0.896	0.900	0.844	***0.874***
Basic	0.879	**0.922**	0.909	0.887	0.870
DBmax	0.874	**0.922**	0.905	***0.892***	***0.874***
DBavg	0.883	***0.913***	0.905	**0.900**	0.870
DBbnd	***0.892***	0.909	0.909	***0.892***	**0.883**
FDnear	0.883	0.905	0.905	0.879	***0.874***
FDfar	0.883	**0.922**	0.913	***0.896***	***0.874***
CUTE/Origin Initial Point					
none	0.857	0.887	**0.931**	0.861	***0.879***
Basic	0.874	0.892	0.913	***0.892***	***0.874***
DBmax	**0.896**	0.892	0.918	***0.896***	**0.883**
DBavg	***0.887***	0.896	0.918	***0.892***	**0.883**
DBbnd	***0.892***	0.892	0.918	***0.892***	***0.879***
FDnear	***0.887***	0.896	***0.922***	**0.900**	**0.883**
FDfar	0.879	0.887	0.918	0.887	**0.883**

Table 7: Solver Success Fractions.

CC	MINOS	SNOPT	KNITRO	CONOPT	DONLP2
Random Initial Point					
none	55.2	61.5	24.4	28.4	33.1
Basic	33.2	21.1	10.4	15.0	17.4
DBmax	30.2	18.2	13.3	14.8	16.7
DBavg	30.8	18.2	11.2	15.8	17.6
DBbnd	32.9	24.1	13.0	14.8	16.9
FDnear	24.9	29.7	10.9	15.6	18.3
FDfar	32.6	21.2	12.6	15.5	16.5
Origin Initial Point					
none	33.5	28.3	17.9	11.1	7.4
Basic	31.9	25.7	11.7	10.1	5.6
DBmax	30.6	26.6	11.9	10.0	5.9
DBavg	31.0	26.4	11.3	10.1	5.5
DBbnd	31.2	26.0	12.3	10.0	5.6
FDnear	30.2	25.1	13.1	10.0	5.8
FDfar	31.3	28.4	14.3	10.2	6.2
Standard Heuristic Initial Point					
none	35.7	24.0	17.7	12.1	6.6
Basic	20.0	16.9	14.2	9.7	4.6
DBmax	22.3	17.7	15.1	9.3	5.0
DBavg	23.3	18.2	13.7	9.4	4.9
DBbnd	20.9	22.3	14.4	9.5	5.1
FDnear	20.8	17.0	16.5	9.9	4.8
FDfar	22.0	17.9	16.7	9.7	5.1
Randomized Standard Heuristic Initial Point					
none	38.6	26.4	15.0	17.4	21.1
Basic	19.0	17.9	14.3	11.0	8.8
DBmax	18.3	15.3	14.3	11.2	12.3
DBavg	19.1	18.3	16.9	11.3	9.0
DBbnd	19.0	19.0	13.6	11.4	9.7
FDnear	19.8	17.5	13.7	11.3	10.7
FDfar	20.5	19.2	11.6	10.8	9.4
CUTE/Origin Initial Point					
none	25.2	20.7	17.1	12.8	8.5
Basic	14.1	11.8	12.9	9.0	4.2
DBmax	12.5	12.6	13.4	8.8	5.0
DBavg	14.4	14.9	14.0	9.2	4.0
DBbnd	13.9	12.6	14.5	9.4	5.6
FDnear	15.2	13.3	12.6	9.1	5.0
FDfar	14.0	11.5	11.4	9.1	5.0

Table 8: Solver Iterations over Compared Subsets.

5.2.3 When an Initial Point Is Provided

When an initial point is externally supplied by the modeller, the question is: how effective are the various constraint consensus heuristics at improving solver success in reaching feasibility?

We examine this question by using two sources of supplied initial points, and by looking at the solver success rates both with and without applying the constraint consensus methods to the supplied points.

Initial Point Heuristic	MINOS	SNOPT	KNITRO	CONOPT	DONLP2
Random	80	90	148	121	112
Origin	153	154	170	133	147
Standard	152	153	176	134	145
Randomized Standard	180	200	201	188	190
CUTE/Origin	192	199	209	195	197

Table 9: Number of Models in Compared Subset.

We use two sources of supplied initial points to simulate the extremes of modeller skill in choosing initial points:

- *The initial points provided with the CUTE models*. These are presumably good initial points selected by the model donor and are taken as representative of the judgement of a knowledgeable modeller. 33 of the 231 models (14%) do not include initial points; as described in Section 4.2, unspecified variables are assigned a value of zero.
- *Random initial points*. These simulate the worst-case judgement of a novice or inept modeller.

It would be best to try numerous random initial points for each model in the second case, e.g. 100. However this would necessitate (100 random initial points)×(231 models)×(6 heuristic methods)×(5 nonlinear solvers) = 693,000 nonlinear solutions, which is clearly impractical. Instead we have used a single random initial point, uniformly distributed within the variable bounds, for each model-method-solver combination, for a total of 6,930 nonlinear solutions.

The results are summarized in Table 7 in the blocks for "Random Initial Point" and "CUTE/Origin Initial Point". The "none" row under "Random Initial Point" shows that the unassisted solvers differ markedly in their native ability to achieve feasibility from a poorly chosen initial point. KNITRO significantly outperforms the other solvers in this regard. When given a good initial point (the CUTE/Origin point), all of the solvers perform quite well in achieving feasibility without using a constraint consensus method, achieving success fractions about equal to (or greater than) the fraction of models for which an initial point is provided in the CUTE set. KNITRO again outperforms the other solvers.

Applying a constraint consensus algorithm to a poor (random) initial point prior to submission to the nonlinear solver always improves the solver success rate dramatically. MINOS and SNOPT show the biggest boosts in success rate when a constraint consensus algorithm is applied. Which constraint consensus algorithm is applied can make a noticeable difference, depending on the solver, or more accurately, the solver algorithm. As we will see throughout the results, the results for KNITRO often differ significantly from those for the other solvers since it is the only barrier method algorithm in the group.

Considering the four solvers MINOS, SNOPT, CONOPT and DONLP2, we see that the DBmax algorithm very often gives a large improvement in solver success rate when applied to a poor

(random) starting point. DBmax is always the best method to apply to a poor initial point, or within about 0.01 of the best method. DBmax is also uniformly successful when applied to a good (CUTE/origin) initial point, though the improvement is smaller (up to 0.039). DBavg, DBbnd, and FDnear also perform well for a good initial point. Given that the quality of the initial point is not known beforehand, we conclude that, for non-barrier-method solvers, the DBmax constraint consensus algorithm should be used whenever an initial point is supplied by the modeller. Given the good performance of DBmax as a stand-alone method (Table 2), this is not surprising.

As shown in Table 8, applying the DBmax algorithm in either case (random initial point, or CUTE/origin initial point) also always reduces the number of non-barrier-method solver iterations in reaching feasibility. In fact using DBmax usually results in the smallest or close to smallest number of solver iterations for the initial point and solver combination.

For barrier method solvers such as KNITRO, the conclusion is slightly different. This is because barrier method algorithms perform poorly when points are placed directly on the limiting values of constraints – exactly where the constraint consensus methods tend to put them. As Table 7 shows, for a poor (random) initial point, applying any constraint consensus method improves barrier solver success, simply because it moves the point to within a reasonable range of feasibility. However the improvement in solver success rate is relatively smaller than for the other solvers, with a maximum improvement of 0.057 when DBavg is applied. When a constraint consensus method is applied to a good (CUTE/origin) initial point, the barrier solver success rates actually decline (maximum decline is 0.018, and 0.013 for DBavg), because more points are moved directly onto constraint boundaries. Results are similar when DBmax and FDfar are used with good initial points.

When the quality of the supplied point is not known in advance, we conclude that, for barrier-method solvers, it is best to apply the DBavg constraint consensus algorithm. However, when a trusted initial point is used (e.g. the result of a previous solution of the same model under slightly different conditions), then it is best to omit the use of constraint consensus at all.

As shown in Table 8, applying the DBavg method when an initial point is supplied reduces the number of solver iterations for barrier solvers. The reduction is quite marked in the case of a poorly chosen (random) initial point.

5.2.4 When an Initial Point Is Not Provided

When the modeller does not provide an initial point, one must be found algorithmically. A suitable combination of an initial point placement method and a constraint consensus algorithm can yield points from which the NLP solver is able to achieve feasibility reliably and with less effort. The solver success results collected in Table 7 under the blocks labelled “Origin Initial Point”, “Standard Heuristic Initial Point”, and “Randomized Standard Heuristic Initial Point” cover the various combinations of initial point placement algorithms and constraint consensus alternatives.

As Table 7 shows, the best solver success fractions in the three blocks mentioned above are always achieved using a combination of the randomized standard initial point placement

heuristic and a constraint consensus method. Which constraint consensus algorithm is best to use varies slightly by solver, however it is worth noting that the FDfar algorithm is always the best or within 0.01 of best success fraction for all of the solvers. Results for the DBmax variant are not much worse. Again this is not surprising since FDfar and DBmax give the best results when used as stand-alone algorithms (Table 2).

It is especially interesting to compare the results for the "Randomized Standard Heuristic Initial Point" with FDfar constraint consensus and the "CUTE/Origin Initial Point" with no constraint consensus in Table 7, since these are the best results obtained algorithmically and the best results obtained using modeller-supplied initial points, respectively. For all of the non-barrier-method solvers, the algorithmically generated points provide solver success rates that are better (MINOS, SNOPT, CONOPT), or insignificantly worse (DONLP2) than the success rates obtained from the modeller-supplied CUTE/origin initial points. This means that, for non-barrier-method solvers, algorithmic methods eliminate the need for expert modeller judgement as to where to place the initial point. This greatly improves the chances of successfully finding a feasible point for modellers of all skill levels.

For the KNITRO barrier method solver, the best solver success fraction is obtained when the CUTE/Origin initial point is used directly, without first applying a constraint consensus method. The best algorithmic method provides a solver success fraction that is 0.018 lower. Still, when an initial point is not provided by the modeller, using the randomized standard heuristic initial point and any of the constraint consensus algorithms is the best thing to do, providing a solver success rate that is comparable to the best obtained when using a high quality modeller-supplied initial point.

The general conclusion is that when an initial point is not provided by the modeller, the best recourse is the randomized initial point placement heuristic followed by the FDfar algorithm, for all solvers, both barrier and non-barrier type. As shown in Table 8, this also reduces the number of solver iterations significantly.

5.2.5 Constraint Consensus Special Cases: Feasibility and Numerical Error

There are two special cases of constraint consensus output: (i) the output point is feasible within the NLP solver tolerances (as opposed to the looser constraint consensus tolerances), and (ii) the output point has a numerical error.

As previously shown in Table 6, every combination of an initial point placement algorithm and a constraint consensus method produces points that are immediately feasible for some models. We duplicated all of our experiments with these models removed, leaving a subset of "harder" models. While the success fractions are lower than those in Table 7 (as expected), the pattern of the results is virtually identical, hence these results are not given in detail.

The reduction in the success fractions when the trivial models are removed ranges from 0.013 to 0.021 for the CUTE/origin initial point placement and from 0.004 to 0.008 for the randomized standard initial point placement heuristic. The smaller reductions in success fraction for the randomized standard initial point placement heuristic show that this fully algorithmic method is more robust for this harder subset of the problems.

We also studied the success rate when a solver is started from the point produced when the constraint consensus step terminates without being able to recover from a numerical error in a constraint evaluation. First, note that this outcome is quite rare for all of the initial point placement algorithms other than the random initial point. Since random initial point placement is not recommended, this will not be a significant problem in practice.

We used the faulty points that constraint consensus generated when starting from random initial points to test the ability of the solvers to reach feasibility under these difficult conditions. There is quite a variation. The rank ordering from best to worst is KNITRO (52-56% success on these models, depending on the constraint consensus variant), DONLP2 (30-34%), CONOPT (22-28%), SNOPT (9-10%) and MINOS (0%). More significantly, we observed that in virtually all cases, if the solver fails to reach feasibility from the faulty point output by the constraint consensus method, it also fails to reach feasibility from the original initial point (i.e. when the constraint consensus step is omitted). The constraint consensus step does no harm under these conditions.

5.2.6 Relative Time Cost of Constraint Consensus Algorithms

We have generally assumed that the amount of time taken by the simple constraint consensus algorithms is much smaller than the time taken by the solvers. We verified this by running the experiments with software timing probes in place. Over all models (including those in which the point produced by constraint consensus is already feasible) and all solvers, the ratio of average solver time to average constraint consensus time varies from 9.1 to 2209.8.

For the two cases of most interest (CUTE/origin initial point and randomized standard initial point heuristic), the minimum ratio of the average solver time to the average constraint consensus time is 15.7, and the maximum ratio is 906.2.

The constraint consensus methods consume very little time compared to the solvers.

5.2.7 Default vs. Optional Application of Constraint Consensus Algorithms

It is a good idea to use a constraint consensus algorithm if it almost always produces a better result and only rarely degrades performance. Since the constraint consensus algorithms run quickly, the most important measure of performance is their impact on solver success in reaching feasibility. The average statistics on solver success in Table 7 mask the fact that applying a constraint consensus algorithm actually prevents the solver from reaching feasibility for some models. We examine this phenomenon in Table 10. For brevity, the table includes results only for those combinations of initial point placement heuristic and constraint consensus algorithm that have been recommended above.

Each cell in Table 10 includes 3 numbers, of the form *a (b-c)*. *b* is the number of models for which the solver succeeds with the constraint consensus method in place, but fails when it is not used. *c* is the opposite: the number of models for which the solver fails when constraint consensus is used, but succeeds without it. *a* is the difference between *b* and *c*. For the rest of the models in the test set, the solver succeeds both with and without constraint consensus, or fails both with and without constraint consensus. As the table shows, while constraint consensus does occasionally prevent the solver from reaching feasibility, it is far more likely to help it find a

feasible point when it could not otherwise do so. As usual, results are slightly different for the barrier method solver KNITRO. For KNITRO, applying the DBavg constraint consensus method from the CUTE/origin initial point actually has a negative net effect.

initial pt	CC	MINOS	SNOPT	KNITRO	CONOPT	DONLP2
random	DBmax	47 (47-0)	42 (43-1)		22 (27-5)	7 (17-10)
random	DBavg			13 (22-9)		
randomized std	FDfar	21(23-2)	6 (11-5)	3 (7-4)	12 (14-2)	0 (7-7)
CUTE/Origin	DBmax	9 (11-2)	1 (5-4)		8 (9-1)	1 (6-5)
CUTE/Origin	DBavg			-3 (2-5)		

Table 10: Net Success Using Constraint Consensus.

These results show that the constraint consensus method should be applied by default in most cases. Over the entire set of 231 models it is relatively rare to find cases in which the solver would succeed when constraint consensus is omitted, but fail when it is applied. The opposite case is much more common, and hence the net effect of including constraint consensus is quite positive.

6. Conclusions

The most important conclusion arising from this study is that a fully automated method can provide initial points for nonlinear solvers that are as good as, and often better than, initial points provided by a knowledgeable modeller, in terms of solver success in finding a feasible point. In addition, solver effort in reaching feasibility is reduced.

A second general conclusion is that it is always better to apply a constraint consensus algorithm even when an initial point is provided, prior to passing the point to the nonlinear solver. In the case of novice modellers, the probability of solver success is greatly increased. In the case of expert modellers who provide a good initial point, solver computational effort is reduced. The exception is the use of a trusted initial point with a barrier solver, in which case it is better to forego the constraint consensus algorithm.

Several more specific conclusions can be drawn from the experimental results:

- Some of the new constraint consensus variants significantly outperform the original basic algorithm in terms of successful termination within an estimated distance α of every constraint. The best of the new methods in this regard is DBmax, closely followed by FDfar. The common element here is a bias towards moving to satisfy the largest constraint violation.

- Considered in isolation, the randomized standard heuristic is the best of the initial point placement heuristics. It provides a significant improvement in solver success rates compared to the original non-randomized version.

- With few exceptions, applying a constraint consensus method reduces solver effort in reaching feasibility. It also increases the number of models for which a feasible point is found directly, eliminating the need to use the solver at all.

- When an initial point is provided by the modeller, applying a constraint consensus method always increases the solver success fraction and reduces the solver effort. DBmax is the best constraint consensus variant to use for the non-barrier solvers. Results are more mixed for the barrier solvers; DBavg is recommended in this case since it improves solver success significantly for poorly chosen initial points while reducing solver effort. For trusted initial points, it is best to avoid the use of constraint consensus at all in the case of barrier solvers.

- When an initial point is not provided by the modeller, very good success fractions are given by combining the randomized standard initial point placement heuristic and the FDfar constraint consensus variant. Success fractions for this combination are close to and in some cases better than the success fractions provided by the very good modeller-provided starting points in the CUTE set. Solver iterations are also reduced.

As noted above, results differ for barrier and non-barrier solvers. This is likely because the constraint consensus methods tend to find starting points at which many of the constraints are tight, which causes difficulties for barrier-method solvers. It should be straightforward to include a post-processing step to move slightly away from the boundaries prior to forwarding the output point to a barrier solver.

The contributions of this paper include the development of several new and better variants of the basic constraint consensus algorithm, the development of an improved initial point placement heuristic, and the demonstration that a combination of these elements can provide high quality initial points for nonlinear solvers. We have also shown that using a constraint consensus algorithm is always a good idea for non-barrier solvers, even when an initial point is provided.

Numerous avenues of research on this topic remain, most notably further variations of the constraint consensus algorithms that may provide better performance. For example, various weightings of the two elements in the DBbnd algorithm (equality and inequality constraints) should be examined. One suggestion is to weight the inequalities by their cardinality. An exhaustive examination of the effect of the constraint consensus parameters (α,β,μ) should also be undertaken. Preliminary investigations show that good results can sometimes be obtained when the maximum number of steps is severely limited (e.g. μ=10). We also plan to examine in some detail the few cases in which solvers succeed without using a constraint consensus algorithm, but fail with it in place. Finally, it may be interesting to examine the properties of the final points returned by the different methods: are some closer to optimality than others for example?

Acknowledgements

Michael Saunders of Stanford University provided helpful comments on early drafts of this paper, as did Richard Waltz of Northwestern University. The support of this research via a Discovery Grant to John Chinneck from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

Bongartz, I., Conn, A. R., Gould, N., and Toint, P.L. (1995). "CUTE: constrained and unconstrained testing environment", *ACM Transactions on Mathematical Software* 21, no. 1, pp. 123-160. See http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/cute/index.html for CUTE models in AMPL format.

Y. Censor, D. Gordon, and R. Gordon. 2001. Component Averaging: An Efficient Iterative Parallel Algorithm for Large and Sparse Unstructured Problems. *Parallel Computing* 27 777-808.

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

Chen, X.B. and Kostreva, M.M. (1999). "Global Convergence Analysis of Algorithms for Finding Feasible Points in Norm-Relaxed MFD", *Journal of Optimization Theory and Applications* 100, no. 2, pp. 287-309.

Chinneck, J.W. (2003). "The Constraint Consensus Method for Finding Approximately Feasible Points in Nonlinear Programs", *INFORMS Journal on Computing* 16, no. 3, pp. 255-265.

Drud, A.S. (1994). "CONOPT -- A Large Scale GRG Code", *ORSA Journal on Computing* 6, pp. 207-216.

Elwakeil, O.A. and Arora, J.S. (1995). "Methods for Finding Feasible Points in Constrained Optimization", *AIAA Journal* 33, pp. 1715-1719.

Fourer, R., Gay, David M., and Kernighan, B. (2003). **AMPL: A Modeling Language for Mathematical Programming, 2nd Edition**, Thomson Brooks/Cole, Pacific Grove, CA, USA.

Gertz, M., Nocedal, J., and Sartenaer, A. (2003). "A Starting-Point Strategy for Nonlinear Interior Methods," Report OTC 2003/4, Optimization Technology Center, Northwestern University, Evanston, IL, USA , March.

Gill, P.E., Murray, W. and Saunders, M.A. (1997). "SNOPT: An SQP algorithm for large-scale constrained optimization", Report SOL 97-3, Systems Optimization Laboratory, Stanford University. See http://www.sbsi-sol-optimize.com/.

Lasdon, Leon S. (1970). **Optimization Theory for Large Systems**, Macmillan Company, New York.

Lawrence, C.T. and Tits, A.L. (2001). "A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm", *SIAM Journal on Optimization* 11, no. 4, pp. 1092-1118.

Murtagh, B.A. and Saunders, M.A. (1983). "MINOS 5.4 User's Guide", Report SOL 83-20R, Systems Optimization Laboratory, Stanford University (revised February 1995). See http://www.sbsi-sol-optimize.com/.

Rardin, R.L. (1998). **Optimization in Operations Research**, Prentice Hall, Upper Saddle River, NJ, USA.

Spellucci, P. (1998). "An SQP method for general nonlinear programs using only equality constrained subproblems", *Mathematical Programming* 82, no. 3, pp. 413-448.

Waltz, R.A. and Nocedal, J. (2003). "KNITRO User's Manual", Technical Report OTC 2003/05, Optimization Technology Center, Northwestern University, Evanston, IL, USA, April. See http://www.ziena.com/knitro.html.

Wright, S.J. (1997). **Primal-Dual Interior Point Methods**, Siam Press.