Noname manuscript No.
(will be inserted by the editor)

Improved Constraint Consensus Methods for Seeking Feasibility in Nonlinear Programs

Laurence Smith · John Chinneck · Victor Aitken

To appear in Computational Optimization with Applications (2012)

Abstract The Constraint Consensus method moves quickly from an initial infeasible point to a point that is close to feasibility for a set of nonlinear constraints. It is a useful first step prior to launching an expensive local solver, improving the probability that the local solver will find a solution and the speed with which it is found. The two main ingredients are the method for calculating the *feasibility vector* for each violated constraint (the estimated vector to the closest point that satisfies the constraint), and the method of combining the feasibility vectors into a single *consensus vector* that updates the current point. We propose several improvements: (i) a simple new method for calculating the consensus vector, (ii) a predictor-corrector approach to adjusting the consensus vector, (iii) an improved way of selecting the output point, and (iv) ways of selecting subsets of the constraints to operate on at a given iteration. These techniques greatly improve the performance of barrier method local solvers. Quadratic feasibility vectors are also investigated. Empirical results are given for a large set of nonlinear and nonconvex models.

Keywords Seeking feasibility · Constraint Consensus · Heuristics · Nonlinear programming

Laurence Smith
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: lsmith@sce.carleton.ca

John Chinneck
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: chinneck@sce.carleton.ca

Victor Aitken
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: vaitken@sce.carleton.ca

1 Introduction

Constraint Consensus (CC) methods [1–3] very quickly find approximately feasible solutions for sets of nonlinear constraints. The CC output point is passed to a full-scale nonlinear local solver, which is usually able to proceed quickly to a fully feasible and locally optimum solution [2]. Using CC to provide a good launch point improves the probability that the local solver will find a feasible and optimal point, and reduces the overall time to do so (including both the CC time and the local solver time).

There are two main steps in a Constraint Consensus algorithm. The first step is the calculation of the *feasibility vector* for each violated constraint, which estimates the vector from the current point to the closest point that satisfies the constraint. The second step is combining the feasibility vectors into a single *consensus vector* that updates the current point. This cycle repeats until the stopping conditions are satisfied.

This paper presents a number of improvements to Constraint Consensus with the specific goal of providing better starting points for barrier method local solvers for nonlinear models. Barrier method solvers are the focus of interest for two reasons: (i) solvers using a barrier method are among the best of the local nonlinear solvers (see e.g. [4]), and (ii) existing CC methods provide less advantage when paired with barrier method solvers than with solvers based on other algorithms, so improvement is needed. Four new techniques provide advantages relative to the current state of the art in CC algorithms:

- The SUM method: a new way of combining the feasibility vectors to create the consensus vector.
- Augmentation: a predictor-corrector method for adjusting the length of the consensus vector.
- A better way of selecting the CC output point (the last calculated point is not necessarily the best one to return).
- Rules for selecting subsets of the violated constraints to operate on. Operating on particular subsets provides CC output points that are better suited for use in barrier method solvers.

We also examine the use of quadratic feasibility vectors, which are less successful. All of these new techniques are extensively tested on a large set of nonlinear models.

1.1 Background

A nonlinear program (NLP) consists of an objective function and a set of constraints such that at least one of the objective or constraint functions is nonlinear. When the goal is only to find a feasible solution the objective function is ignored, leaving the following *feasibility problem*:

$$\text{Find } \mathbf{x} \text{ such that :} \tag{1}$$

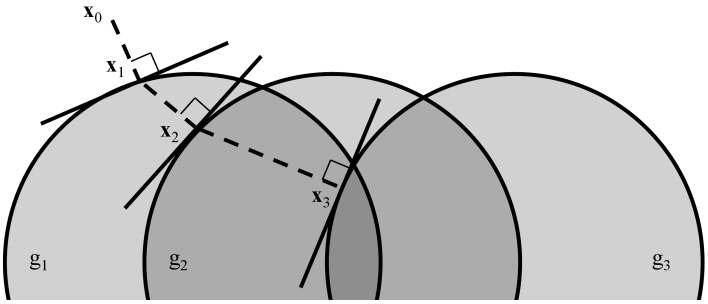


Fig. 1: The method of successive orthogonal projections (figure inspired by [5], p. 82).

$$g_i(\mathbf{x})\{\leq,=\}0, (i = 1, ..., m) \tag{2}$$

$$\ell_j \leq x_j \leq u_j, (j = 1, ..., n). \tag{3}$$

Linear constraints are a special case of nonlinear constraints. The constraints functions $g_i(\mathbf{x})$ are continuously differentiable. The gradient of a constraint function is represented by the vector $\nabla g_i(\mathbf{x})$ and the Hessian is represented by the matrix $\nabla^2 g_i(\mathbf{x})$.

The violation of an equality constraint is defined as the absolute value of the constraint

$$v_i = |g_i(\mathbf{x})|, \tag{4}$$

and the violation of an inequality constraint is the greater of the constraint function value and 0

$$v_i = \max\{0, g_i(\mathbf{x})\}. \tag{5}$$

The maximum constraint violation at some point $\mathbf{x}_k$ is represented by

$$\mathcal{V}(\mathbf{x}_k) = \max\left(v_i(\mathbf{x}_k), ..., v_m(\mathbf{x}_k).\right), \tag{6}$$

A successful solution of the feasibility problem will find a point at which $\mathcal{V}$ is less than a small tolerance, typically 10^{-6} or smaller.

There are numerous *projection algorithms* [5] for seeking feasible solutions for sets of constraints, typically sets of linear or convex nonlinear constraints. Projection algorithms assume that the feasibility vector for a violated constraint is the orthogonal projection of the current infeasible point onto the closest point that satisfies the constraint. In *sequential projection algorithms* a single feasibility vector is used as the consensus vector: each iteration uses the feasibility vector from a different violated constraint. A control sequence governs the order in which the constraints are selected. There are numerous choices for the control sequence, including simple cyclic control, and most violated constraint control which uses the longest feasibility vector at the current point. The length of the consensus vector may also be adjusted by a scalar relaxation factor, λ, which could be smaller or greater than 1. The Sequential

Orthogonal Projection method [6] is the most basic form of sequential projection algorithm. The control sequence is a simple cycle and the relaxation factor is equal to 1. This method is illustrated in Fig. 1. The consensus vector at each iteration is the feasibility vector for the next violated constraint in the cycle.

In fully *simultaneous projection algorithms*, the consensus vector is formed by combining all of the feasibility vectors in some manner, an idea first suggested for the linear case by Cimmino [7]. Block-iterative projection algorithms [8] are simultaneous algorithms in which the consensus vector is constructed by combining a subset of the feasibility vectors using a weighting system. Variations include how the feasibility vectors are selected and how the weights are chosen.

The calculation of an exact feasibility vector for a violated nonlinear constraint can be extremely time-consuming, in general requiring the solution of an entire nonlinear program. For this reason, most projection methods substitute an estimated feasibility vector, often using a linear approximation based on the gradient of the violated constraint at the current point. Methods in this class include the Method of Cyclic Subgradient Projections [9], and generalizations of Kaczmarz's Algorithm [10] to solve systems including nonlinear equations [11–13].

The *linear feasibility vector* moves in a direction parallel to the gradient of the violated constraint. The step size is determined by using a first order Taylor series expansion of the constraint function around the current location. Consider a first order Taylor series expansion of some constraint function $g(\mathbf{x}) : \mathcal{R}^n \rightarrow \mathcal{R}$

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{7}$$

The search direction is chosen to be parallel to the gradient of g at $\mathbf{x}_k$, therefore

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \rho_l \nabla g(\mathbf{x}_k)^T \tag{8}$$

where ρ_l is some scalar step size for this linear approximation. ρ_l will be negative for violated inequality constraints and for equality constraints with $g(\mathbf{x}_k) > 0$, but will take a positive value for violated equality constraints with $g(\mathbf{x}_k) < 0$. The step size is determined by setting $g(\mathbf{x}_{k+1}) = 0$ and solving for ρ_l

$$\begin{aligned} 0 &= g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) \\ 0 &= g(\mathbf{x}_k) + \nabla g(\mathbf{x}_k)(\rho_l \nabla g(\mathbf{x}_k)^T) \\ 0 &= g(\mathbf{x}_k) + \rho_l ||\nabla g(\mathbf{x}_k)^T||^2 \\ \rho_l &= \frac{-g(\mathbf{x}_k)}{||\nabla g(\mathbf{x}_k)^T||^2}. \end{aligned} \tag{9}$$

Therefore, the linear feasibility vector is

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \frac{-g(\mathbf{x}_k)}{||\nabla g(\mathbf{x}_k)^T||^2} \nabla g(\mathbf{x}_k)^T. \tag{10}$$

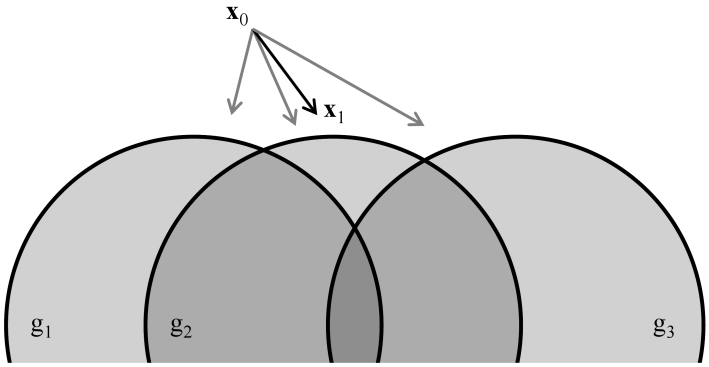


Fig. 2: Example of Basic Constraint Consensus. The gray arrows are the feasibility vectors and the black arrow is the resulting consensus vector.

1.2 Constraint Consensus Methods

Much of the theoretical development of projection algorithms relates to solving the feasibility problem for sets of convex constraints, including proofs of convergence [5]. Constraint Consensus methods, on the other hand, use various forms of projection algorithms as heuristic methods for finding near-feasible points for general sets of nonlinear constraints, including nonconvex constraints. The original Constraint Consensus method [1] (hereafter *Basic Constraint Consensus*) is a block-iterative algorithm with $\lambda = 1$ and uniform weighting in which the consensus vector is formed component-wise. That is, each element of the consensus vector is formed as the average of the elements in the feasibility vectors for violated constraints that include that variable at the current point. This is the same as the Component Averaging projection method proposed by Censor et al. [14]. The Basic Constraint Consensus method is illustrated in Fig. 2.

Constraint Consensus methods are distinguished from other block-iterative algorithms by the way they select the subset of feasibility vectors used to form the consensus vector. Specifically, Constraint Consensus filters the feasibility vectors using a *feasibility tolerance* (denoted α). Only feasibility vectors with a length greater than the feasibility tolerance are selected to help form the consensus vector. The algorithm terminates successfully when all feasibility vectors are shorter than the feasibility tolerance.

The linear feasibility vectors are exact for linear constraints, but are only estimates for the nonlinear constraints. The general idea is that a few inaccurate feasibility vectors for nonlinear constraints are usually *outvoted* by a larger number of well-directed feasibility vectors when the consensus vector is constructed, so the update usually moves in a useful direction. Most CC methods make very rapid initial progress towards a feasible point for a general set of nonlinear constraints, but progress may slow significantly as a feasible region is approached. For this reason CC methods are generally used to quickly

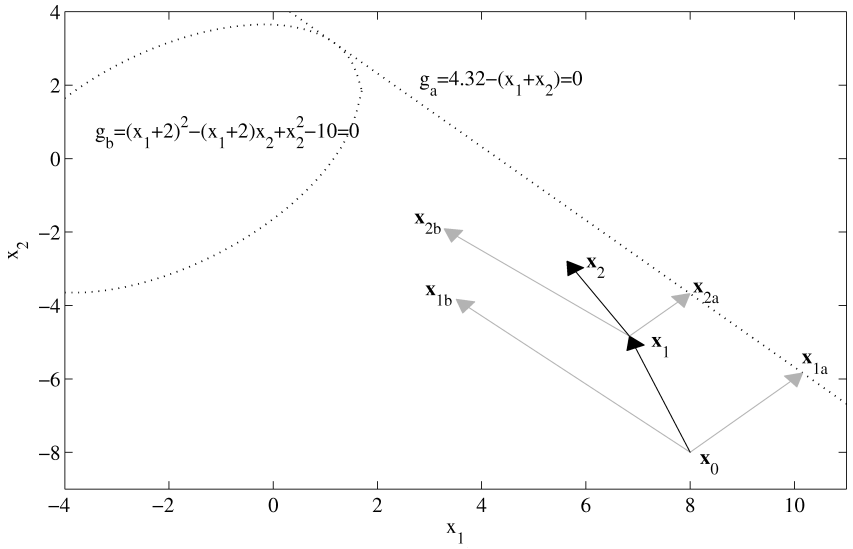


Fig. 3: The linear feasibility vector calculation.

provide only an approximately feasible point, however this is highly valuable because launching a local solver from such a point greatly improves its success and speed [2]. CC methods have very useful properties as heuristics for sets of general nonlinear constraints: they are very fast because they do not rely on time-consuming line searches or matrix inversions. Since they are so fast, a failed CC run is easily replaced by a simple restart at a different initial point.

A numerical example of the construction of the feasibility vectors and the consensus vector in the Basic Constraint Consensus method is provided in Fig. 3. Grey arrows are feasibility vectors and black arrows are the resulting consensus vectors. There is a feasible point near $\mathbf{x} = [1.216\ \ 3.104]^T$. The constraint violations at the intial point $\mathbf{x}_0$ are $v_a = 4.320$ and $v_b = 234.0$. Note the exact feasibility vector $(\mathbf{x}_{1a} - \mathbf{x}_0)$ for the linear constraint, and the approximate feasibility vector $(\mathbf{x}_{1b} - \mathbf{x}_0)$ for the nonlinear constraint. The consensus vector $(\mathbf{x}_1 - \mathbf{x}_0)$ is also shown. The point $\mathbf{x}_1$ is closer to the feasible region but still violates the constraints, so the cycle repeats using $\mathbf{x}_1$ as the starting point. At the point $\mathbf{x}_2$ the violations of constraints g_a and g_b are significantly reduced at 1.476 and 77.479 respectively.

Constraint Consensus iterations continue until certain stopping conditions are met. Three conditions are normally used [2]: (i) stop successfully when every feasibility vector is shorter than α (e.g. 10^{-6}), (ii) stop when the consensus vector is shorter than β, which indicates stalling, or (iii) stop when more than μ iterations have been completed. The last two conditions indicate unsuccessful termination and are triggered when convergence is too slow. It is also possible to exit unsuccessfully when the consensus vectors are not shortening between iterations at a fast enough rate [3]. Slow convergence can

result from ill-conditioning which causes feasibility vectors that almost cancel out, resulting in a very short consensus vector. Slow convergence can also be caused by consensus vectors that zig-zag as they alternately satisfy one set of constraints and then another [1]. Another source of difficulty is poor linear feasibility vector approximations to highly nonlinear constraints.

A number of other Constraint Consensus methods have been developed, varying in the way that the consensus vector is constructed [2]. The *Feasibility Distance far* (FDfar) algorithm is a sequential method that chooses the longest feasibility vector as the consensus vector. The *Direction Based average* (DBavg) algorithm constructs the consensus vector component-wise. For each element, the direction of movement (positive or negative) is determined by the most common sign among the elements for that component in the feasibility vectors. The distance of movement is given by the average of the magnitudes of the elements in the winning direction for that component in the feasibility vectors. *Direction Based maximum* (DBmax) is similar to DBavg except that the largest element determines the distance of movement in the winning direction in each component. A number of other consensus schemes are also available, though empirical testing shows the superiority of the DBmax and FDfar variants [2].

2 New Constraint Consensus Techniques

Ibrahim and Chinneck [2] show that CC can improve solver success in reaching feasibility. Specifically, they provide numerical evidence showing how different combinations of initial point placement routines and CC variants improve the success rates of the local NLP solvers MINOS 5.5, SNOPT 6.1-1, KNITRO 4.0, DONLP2, and CONOPT 3.13. Performance is significantly improved for all non-barrier method solvers, but improvement is only moderate in the case of the KNITRO barrier method solver. Since barrier method solvers are among the best NLP solvers available, it is especially important that CC methods provide useful launch points for solvers in this class.

The issue with barrier method solvers appears to be that CC tends to move points directly onto the limiting values of inequality constraints, which can cause difficulties for the barrier algorithm. The barrier algorithm prefers a starting point that is very close to satisfying the equality constraints but which oversatisfies the inequality constraints. Some of the new CC methods developed here (the SUM method and augmentation) more often provide points that are better suited to barrier methods, while the other new techniques generally improve the performance of all types of solvers.

Note that while the feasibility vectors are identical in all CC methods, the update step applied to the current point differs between methods because the feasibility vectors are combined in different ways to calculate the consensus vector. In general, both the update direction and the step size differ between methods.

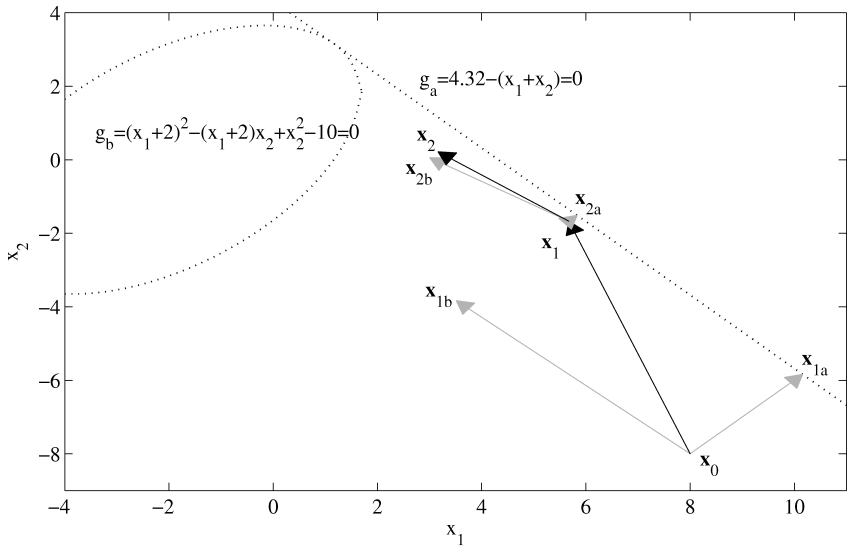


Fig. 4: The SUM consensus calculation.

2.1 The SUM Consensus Method

The FDfar variant is one of the most successful CC methods [2], however it sometimes cycles between the constraints that provide the longest feasibility vector. The new SUM consensus method helps CC avoid cycling while still taking advantage of long feasibility vectors. As its name suggests, the SUM method forms the consensus vector by summing all the feasibility vectors at the current iterate. This allows longer feasibility vectors to have more influence than shorter vectors as in FDfar, but every violated constraint still provides some input to the consensus vector. The method is block-iterative; a feasibility tolerance is used to select the subset of feasibility vectors at each iterate.

The SUM method is illustrated using the model presented previously. Both constraints g_a and g_b are violated at the initial point $\mathbf{x}_0 = [8 \;\; -8]^T$. The linear consensus vectors are

$$(\mathbf{x}_{1a} - \mathbf{x}_0) = \begin{bmatrix} 2.160 \\ 2.160 \end{bmatrix} \tag{11}$$

and

$$(\mathbf{x}_{1b} - \mathbf{x}_0) = \begin{bmatrix} -4.488 \\ 4.167 \end{bmatrix}. \tag{12}$$

The SUM method calculates the next point $\mathbf{x}_1$ by adding the feasibility vectors to the previous point, $\mathbf{x}_0$

$$\mathbf{x}_1 = \begin{bmatrix} 8 \\ -8 \end{bmatrix} + \begin{bmatrix} 2.160 \\ 2.160 \end{bmatrix} + \begin{bmatrix} -4.488 \\ 4.167 \end{bmatrix} = \begin{bmatrix} 5.672 \\ -1.673 \end{bmatrix} \tag{13}$$

Two iterations of the SUM consensus method are depicted in Fig. 4. The constraint violations are much smaller after the two iterations than after the two iterations of the Basic CC method depicted in Fig. 3.

2.2 Augmentation

Augmentation uses a predictor-corrector approach to find an adjustment to the length of the last consensus vector. This is similar to the Secant method for root finding [15], and is in the spirit of a number of algorithms that adjust the length of the consensus vector automatically via a relaxation factor (see Sec. 5.10 in [5] or [16] for some examples). The predictor step is the usual consensus vector constructed as in Basic CC. A correcting relaxation factor for the consensus vector is then calculated independently for each violated constraint. The average of the relaxation factors found for each violated constraint is then applied to form the consensus vector.

Note that our augmentation corrector is simply an adjustment of the length of the predictor vector, without any change in direction. This is in contrast to the classical predictor-corrector scheme for interior point methods [17] in which the corrector is a separate vector that has both a different length and a different direction.

Each violated constraint is initially approximated by a first order Taylor series expansion

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+2} - \mathbf{x}_{k+1}). \tag{14}$$

The length of the initial consensus vector $(\mathbf{x}_{k+1} - \mathbf{x}_k)$ (the predictor) is adjusted by an augmentation relaxation value ρ_a (the corrector), i.e.,

$$(\mathbf{x}_{k+2} - \mathbf{x}_{k+1}) = \rho_a(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{15}$$

Substituting Eqn. 15 into Eqn. 14, the equation approximating the violated constraint becomes

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \rho_a \nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+1} - \mathbf{x}_k). \tag{16}$$

Start again with the Taylor series expansion about $\mathbf{x}_{k+1}$

$$g(\mathbf{x}_k) \approx g(\mathbf{x}_{k+1}) + \nabla g(\mathbf{x}_{k+1})(\mathbf{x}_k - \mathbf{x}_{k+1}) \tag{17}$$

and rearrange such that

$$\nabla g(\mathbf{x}_{k+1})(\mathbf{x}_{k+1} - \mathbf{x}_k) \approx g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k), \tag{18}$$

then substitute the result into Eqn. 16 to give

$$g(\mathbf{x}_{k+2}) \approx g(\mathbf{x}_{k+1}) + \rho_a(g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)). \tag{19}$$

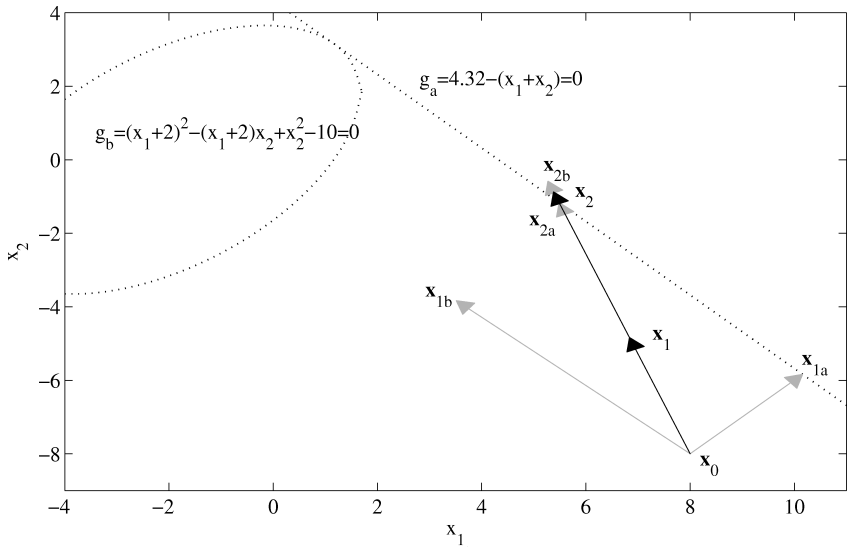


Fig. 5: The augmented feasibility vector calculation.

The goal is to find the root of the violated constraint function, i.e., $g(\mathbf{x}_{k+2}) = 0$, so the step size is determined by setting Eqn. 19 to zero and solving:

$$\rho_a = \frac{-g(\mathbf{x}_{k+1})}{g(\mathbf{x}_{k+1}) - g(\mathbf{x}_k)}. \tag{20}$$

The final augmentation relaxation factor for the consensus vector is the average of the relaxation factors for the violated constraints.

An example of augmentation is shown in Fig. 5. The first step $(\mathbf{x}_1 - \mathbf{x}_0)$ is the predictor, i.e. the consensus vector calculated previously for this model. The second step $(\mathbf{x}_2 - \mathbf{x}_1)$ is the corrector, i.e. the consensus vector formed using the predictor consensus vector multiplied by the average of the relaxation factors found for the violated constraints. For example, given points $\mathbf{x}_0$ and $\mathbf{x}_1$, the augmented step size for constraint g_b is

$$\rho_a = \frac{-g_b(\mathbf{x}_1)}{g_b(\mathbf{x}_1) - g_b(\mathbf{x}_0)} \tag{21}$$

$$= \frac{-134.2}{134.2 - 234.0} \tag{22}$$

$$= 1.345 \tag{23}$$

and the augmented feasibility vector is

$$(\mathbf{x}_{2a} - \mathbf{x}_1) = \rho_a(\mathbf{x}_1 - \mathbf{x}_0) \tag{24}$$

$$= \begin{bmatrix} -1.565 \\ 4.254 \end{bmatrix}. \tag{25}$$

The same operation is applied to the other constraint g_a and then the consensus vector is formed using Basic consensus. Fig. 5 illustrates the result. After two iterations (one linear iteration and one augmenting iteration) the violations of the constraints g_a and g_b at $\mathbf{x}_2$ are 0.185 and 51.653, respectively. The augmented method requires fewer calculations and finds a point with smaller constraint violations as compared to Basic CC.

Re-correcting a corrected step is not productive, so it is not useful to augment every CC step. In fact, performing an augmentation only every several steps is more effective. We study this issue in depth in the Online Appendix. For the numerical experiments reported later, one CC variant uses an augmented consensus vector every third step, a recurrence period that worked well in our experiments.

2.3 Selecting the Subset of Constraints

Many nonlinear local solvers linearize the nonlinear constraints and manipulate this linear system as a main step, so the original linear constraints are dealt with exactly. For this reason, it is more important for CC to reduce the violation of the nonlinear constraints prior to starting the local solver. This is especially important when the amount of time allowed for Constraint Consensus is limited. Limiting CC to operating on only the violated nonlinear constraints provides higher-quality launch points for the local solver. Section 4 provides empirical evidence of this.

2.4 Choosing the CC Output Point

Experimental data shows that a positive correlation exists between local solver launch points with low $\mathcal{V}$ and solver success [2]. However CC methods may not monotonically reduce the violation at each iteration; it may fluctuate. For this reason, we record the violation after each CC iteration and then launch the local solver at whichever point has the lowest $\mathcal{V}$.

MacLeod [18] studied related ideas, in his case for measuring CC progress in order to decide when to terminate. He used consensus vector length as the measure of progress, terminating the algorithm when this failed to decrease, and returning this final point. In contrast, we allow the algorithm to proceed until it meets one of the original termination conditions proposed by Chinneck [1]. All of the CC iterates are candidate points, and the one with the lowest $\mathcal{V}$ is chosen as the launch point for the local solver.

2.5 Quadratic Feasibility Vectors

Using a similar Taylor series expansion of each violated constraint function, it is possible to derive feasibility vectors that are exact for quadratic constraints (and hence also exact for linear constraints). We show how to do this

Table 1: Summary statistics for test models

	Avg.	Min.	Max.
Linear Constraints	661.1	0	12000
Quadratic Constraints	853.7	0	13798
General Nonlinear Constraints	686.5	0	5001
All Constraints	2201.3	11	14000
Variables	2486.8	2	20008
Nonzeros in Jacobian	13113.3	39	128004

in the Online Appendix. However the calculation load is significantly higher than for linear feasibility vectors and our experimental results show that the added complexity does not pay off in faster or more accurate solutions. Several less accurate linear iterations can be carried out in the time taken for one quadratic iteration, with greater overall accuracy on average. The complete set of experimental results, including all methods using both linear and quadratic feasibility vectors, is given in the Online Appendix.

3 Experimental Setup

3.1 Hardware and Software

The experiments were run on a machine with an Intel®Core™2 Duo Processor E6600 (4M Cache, 2.40 GHz, 1066 MHz FSB) and 3GB of memory. The operating system was Linux Fedora Core 6.

The Constraint Consensus algorithms were written in the C language and compiled using gcc 4.1.2. Ipopt 3.9 [19] was used as the local solver. This barrier method solver performs very well in independent testing [4], and is free and open source, an important consideration given that we frequently ran several IPOPT solutions in parallel in order to handle a long list of experiments and test models.

3.2 Test Models

The models used in the experiments were taken from the COntinuous CONstraints - Updating the Technology (COCONUT) benchmark [20], a collection of over 1000 problems from a variety of fields, including computational chemistry, robotics, operations research, and economics. All the models are in the AMPL modelling language format [21].

Models with 10 or fewer nonlinear constraints were removed from the test set. The *argauss* and *grouping* models were removed because they caused Ipopt to return the runtime exception message *Too few degrees of freedom.* The models *argtrig*, *bratu*2*d*, *bratu*2*dt*, *bratu*3*d*, *camcge*, *catenary*, *cbratu*2*d*, *ex*8_3_13, *ex*8_3_14, *ex*8_6_1, *oet*7, *pindyck*, and *polak*3 were removed because they caused the AMPL interface function *conval*() to return an error.

The models were separated into three subsets based on the number of nonlinear constraints:

- Problem Set I (PS I) has 79 models with more than 10 nonlinear constraints and fewer than 101 nonlinear constraints.
- Problem Set II (PS II) has 62 models with more than 100 nonlinear constraints and fewer than 1001 nonlinear constraints.
- Problem Set III (PS III) has 72 models, each with more than 1000 nonlinear constraints.

The statistics of the selected test models are presented in Table 1.

3.3 Starting Points

Every algorithm is run ten times for each model (2130 runs in total). Suggested start points provided with the models were ignored and each of the ten start points was chosen randomly. This increases the difficulty of the problem set because the provided start points are often very promising. All of the algorithms are launched from the same randomly selected start points for each model to ensure fair starting conditions.

The random start points were chosen within the variable bounds. If bounds are not provided, the start point is randomly chosen within $\pm 10^4$, a range found useful in previous research [18].

3.4 Performance Metrics

The main metric used to judge CC performance is the maximum constraint violation $\mathcal{V}$. The CC algorithms try to find a point for which $\mathcal{V}$ is less than a specified tolerance such as 10^{-6}, a typical value. CC does not have to find a point that is completely feasible to be effective but lower values of $\mathcal{V}$ are usually better.

Each algorithm variant was run 10 times for every model. The average time, average number of iterations, and median violation of the best points found are reported in Table 2. The median violation is an important metric that helps measure quality of the solutions found by the algorithms.

When CC is paired with a local solver, relevant metrics include:

- **feasibility fraction**: the fraction of models for which Ipopt found a feasible solution,
- **infeasibility fraction**: the fraction of models for which Ipopt converged to an infeasible solution,
- **timeout fraction**: the fraction of models for which Ipopt ran out of time,
- **fail fraction**: the fraction of models for which Ipopt returned another return code (e.g., Restoration_Failed see [22] for a complete list of possibilities and full definitions).

3.5 Constraint Consensus Variants

The CC algorithms tested in the experiments are the Basic, FDfar, and SUM variants. These are augmented or not augmented, and use either linear or quadratic feasibility vectors. The naming convention is $a\_b\_c\_d$ where a is L or Q (for linear or quadratic feasibility vector), b is the name of the CC algorithm variant, c is either A or blank for augmentation or not, and d is present only if c is A and represents the recurrence period (T) of augmentation. For instance, if $T = 2$ then linear feasibility vectors are used for the first iteration of constraint consensus and the augmented feasibility vectors are used for the second iteration. Then the whole process is repeated. If $T = 3$ then two iterations with linear feasibility vectors and one iteration with augmented feasibility vectors are used for a cycle of the algorithm. For consistency the augmented calculations always take place in the second iteration of a cycle.

Results are reported here for the following algorithm combinations:

- **L_Basic**: linear feasibility vector, Basic Consensus, no augmentation.
- **L_FDfar**: linear feasibility vector, FDfar Consensus, no augmentation.
- **L_SUM**: linear feasibility vector, SUM Consensus, no augmentation.
- **L_Basic_A_3**: linear feasibility vector, Basic Consensus, augmented with recurrence period 3. This augmentation recurrence period returned the best results among numerous periods tested as shown in the Online Appendix.

Results for numerous other algorithm combinations, mainly with different augmentation recurrence periods and including quadratic feasibility vectors, are given in both the Online Appendix and in [23].

3.6 Experiments and Constraint Consensus Parameter Settings

Four experiments were conducted. Experiment A examines the relative ability of the algorithms to find points close to feasible regions given a short pre-set maximum run time and maximum number of iterations. A very tight feasibility tolerance of $\alpha = 10^{-16}$ and movement tolerance of $\beta = 10^{-16}$ insure that the CC variants will not terminate unless they find a feasible point or have stopped making progress. The maximum number of iterations is $\mu = 100$. The time limit (max_time) varies by problem set, allowing more time for models having more nonlinear constraints. For PS I $max\_time = 0.05s$, for PS II $max\_time = 0.5s$, and for PS III $max\_time = 5.0s$. These time limits are based on preliminary experiments not reported here.

Experiment B tests how often a CC intermediate point has a lower violation than the CC end point. Two sets of CC parameters were used: $\alpha = 10^{-16}$ and $\beta = 10^{-16}$, and $\alpha = 10^{-3}$, $\beta = 10^{-6}$. These settings were chosen to determine whether using intermediate CC points is advantageous over a wide range of tolerances. All other relevant parameters settings were the same as in Experiment A.

Experiment C examines which CC variant improves the performance of the barrier method solver the most. Each randomly selected start point is used to

Table 2: Finding approximately feasible points.

Results for PS I ($max\_time = 0.05s$).
The median $\mathcal{V}$ for the start points is $1,710,000$.

Variant	Avg. Time(s)	Avg. Its	Median $\mathcal{V}$
L_Basic_A_3	0.01	87.47	117
L_FDfar	0.01	88.40	125
L_SUM	0.01	87.69	323
L_Basic	0.01	93.07	29,800

Results for PS II ($max\_time = 0.5s$).
The median $\mathcal{V}$ for the start points is $1,255,000$.

Variant	Avg. Time(s)	Avg. Its	Median $\mathcal{V}$
L_Basic_A_3	0.24	77.14	140
L_FDfar	0.24	79.62	1,075
L_SUM	0.24	76.82	1,315
L_Basic	0.27	76.66	280,500

Results for PS III ($max\_time = 5.0s$).
The median $\mathcal{V}$ for the start points is $1,580,000$.

Variant	Avg. Time(s)	Avg. Its	Median $\mathcal{V}$
L_Basic_A_3	3.87	52.24	1,765
L_SUM	3.77	51.80	3,760
L_FDfar	3.80	52.56	8,965
L_Basic	4.03	46.87	531,500

launch CC. The point having the lowest violation during the CC run is then used as a launch point for the barrier method solver. For this experiment the feasibility tolerance was set at $\alpha = 10^{-3}$ and the movement tolerance was set at $\beta = 10^{-6}$ to ensure that the barrier method solver is invoked quickly if the CC variants either find a point close to feasibility or stop making progress. The maximum number of CC iterations is 100. The maximum time allowed for CC is the same as in Experiment A. Parameter settings for Ipopt were: $honor\_original\_bounds = yes$, $bound\_relax\_factor = 0$, $max\_iter = 9999999$, $constr\_viol\_tol = 10^{-6}$, and $max\_cpu\_time = 60$.

Experiment D compares the performance of the local solver when the CC algorithm constructs feasibility vectors for only the violated nonlinear constraints. The parameters settings are the same as in Experiment C.

4 Numerical Results

4.1 Experiment A: Finding Approximately Feasible Points

The results for Experiment A are reported in Table 2. The four tested algorithm variants are listed in increasing order of median $\mathcal{V}$. The number of

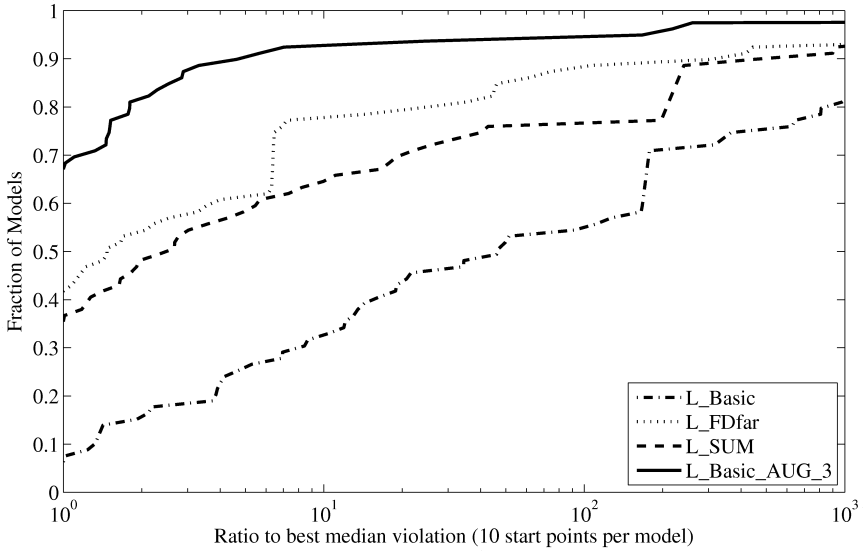


Fig. 6: Performance Profile (PS III)

iterations and the average times do not show significant variation between the methods.

The original L_Basic method has the worst median $\mathcal{V}$ in all three problem sets by a significant margin. L_FDfar provides significantly better results as expected [2], as does the new SUM method. However L_Basic_A_3 provides the smallest median $\mathcal{V}$ in all three problem sets, showing the power of the augmentation technique. The relative peformances of the four CC variants are not greatly affected by the number of nonlinear constraints.

Figure 6 provides a more detailed look at the relative performance of the CC variants on the large models in PS III that are of greatest interest, via a performance profile [24]. Figure 6 shows that L_Basic_A_3 dominates the other methods, providing the smallest median violation for about two-thirds of the models, and having the largest fraction of models below any given ratio to best median violation.

4.2 Experiment B: The CC Output Point

This experiment examines the median violation of the CC end points and the median violation of the point having the lowest violation at any iteration in the CC run (the *best point*). There are several conclusions to be drawn from the data in Table 3:

– The median violation is considerably less for all algorithms when they use the best point rather than the end point.

Table 3: Using Best and End Points. 213 models.

Tight Parameter Settings ($\alpha = 10^{-16}$, $\beta = 10^{-16}$)

	Median Violation		Fraction of Runs
CC Type	**Best**	**End**	**with Best < End**
L_Fdfar	2155.0	2875.0	0.38
L_SUM	1985.0	5075.0	0.49
L_Basic_A_3	961.0	1980.0	0.61
L_Basic	198500.0	225000.0	0.14

Loose Parameter Settings ($\alpha = 10^{-3}$, $\beta = 10^{-6}$)

	Median Violation		Fraction of Runs
CC Type	**Best**	**End**	**with Best < End**
L_Fdfar	2196.3	2935.9	0.41
L_SUM	2154.9	6474.8	0.52
L_Basic_A_3	1088.1	3089.1	0.69
L_Basic	195445.0	225930.0	0.16

- The fraction of runs for which the best point offers an advantage is substantial for the three new algorithms, especially L_Basic_A_3. The fraction is much smaller for L_Basic. This shows the extent to which the more aggressive movements of the new algorithms lead to a less monotonic pattern of improvement.
- The results are consistent for both the tight and the loose parameter settings.

Since the best point is defined as the point with the lowest violation it is not surprising that these points return a lower median violation on average. What is interesting is the percentage of runs for which using the best point offers an advantage. For L_Basic_A_3 approximately 2 in 3 runs are improved by simply taking the best intermediate point. The advantage is greater for all algorithms when the feasibility and movement tolerances are loosened.

4.3 Experiment C: Constraint Consensus Paired with a Barrier Method Solver

The purpose of Experiment C is to identify which CC variant most improves the results for the barrier method solver. The L_FDfar, L_SUM, L_Basic_A_3 and the original L_Basic are compared, along with a base case in which Ipopt is run without CC. The results are summarized in Table 4. The average times are in seconds.

The first section of Table 4 lists the results for the 175 models that did not cause Ipopt to timeout or return a failure. The second section of Table 4 contains results for the 25 models for which Ipopt exceeded the maximum time limit in at least one of the 10 runs and did not report an failure. The

Table 4: Using Constraint Consensus with Ipopt

No timeouts or failures (175 models)					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		0.53	0.53	0.55	0.59
Median Violation		9.36×10^2	1.59×10^3	$\mathbf{4.90 \times 10^2}$	4.60×10^4
Feasibility Fraction		0.012	0.031	**0.038**	0.009
Ipopt					
Avg.Time per run	10.71	9.52	8.97	**7.80**	10.21
Avg. Iterations	401.23	323.39	307.57	**246.07**	372.60
Feasibility Fraction	0.726	0.758	0.761	**0.779**	0.738
Infeasibility Fraction	0.274	0.242	0.239	**0.221**	0.262
Total					
Avg.Time per run	10.71	10.06	9.50	**8.35**	10.80

Timeouts (25 Models)					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		2.36	2.34	2.34	2.41
Median Violation		$\mathbf{2.68 \times 10^5}$	6.89×10^5	4.66×10^5	1.29×10^5
Feasibility Fraction		0.00	0.00	0.00	0.00
Ipopt					
Avg.Time per run	56.84	52.24	52.64	**46.62**	55.03
Timeout Fraction	0.824	0.820	0.800	**0.732**	0.856
Feasibility Fraction	0.152	0.156	0.164	**0.248**	0.128
Infeasibility Fraction	0.024	0.024	0.036	0.020	**0.016**
Total					
Avg.Time per run	56.84	54.60	54.99	**48.97**	57.45

Failures (13 Models)					
Ipopt					
Fail Fraction	0.92	0.86	0.71	**0.52**	0.72
Timeout Fraction	**0.00**	0.01	0.05	0.10	0.04
Feasibility Fraction	0.03	0.08	0.09	**0.26**	0.08
InFeasibility Fraction	**0.05**	0.05	0.15	0.12	0.16

third section of Table 4 reports the results for the 13 models for which Ipopt returned at least one error during one of the 10 runs. The best results are shown in boldface.

The first section of Table 4 confirms the results from Experiment A, that L_Basic_A_3 is the most effective CC variant. L_Basic_A_3 increases Ipopt's feasibility fraction by more than 5 percentage points while reducing the average combined total run time by 22%. The average number of Ipopt iterations is also reduced substantially. L_FDfar and L_SUM also improve the perfor-

mance of Ipopt by smaller margins. L_Basic improves the feasibility fraction but increases average overall runtime slightly.

L_Basic_A_3 also provides the best performance for the 25 timeout models in Table 4, achieving the lowest Ipopt timeout fraction and highest Ipopt feasibility fraction. L_Basic_A_3 is again the the most effective for the 13 models for which Ipopt returned an error during one of the 10 runs. It improved Ipopt's feasibility fraction substantially while decreasing Ipopt's fail fraction.

The two new methods L_Basic_A_3 and L_SUM provide improvements over the two existing methods L_Basic and L_FDfar. L_Basic_A_3 has the best results over all categories of outcomes, while L_SUM has the next best results. Paired with L_Basic_A_3, Ipopt was successful for 1451 of the 2130 runs $((175+25+13)\times 10)$. The next most successful pairing was Ipopt with L_SUM which achieved success for 1382 of the runs. These are noteworthy increases in performance compared to Ipopt by itself which was successful in only 1311 of the runs.

Considering each of the 2130 runs, the total time taken by Ipopt paired with L_Basic_A_3 is the same as for Ipopt alone for 115 models, and is smaller than the time taken by Ipopt alone for 1255 models. Thus the L_Basic_A_3 with Ipopt combination provides an advantage (or at least no disadvantage) in terms of speed for about two-thirds of all runs (1370 of 2130 runs, or 64.3%).

4.4 Experiment D: Constraint Consensus Applied to Nonlinear Constraints Only

Experiment D tests whether a local solver performs better if CC is applied only to nonlinear constraints. The 117 models that have a mixture of linear and nonlinear constraints were used in this experiment (the omitted models consist entirely of nonlinear constraints and hence would return the same output points in either case).

Table 5 shows the results, and uses the same definitions as for Table 4. The top section gives the results when CC operates on all violated constraints, and the bottom section lists the results when CC is applied to only the violated nonlinear constraints. Table 5 shows that the nonlinear solver is able to find a feasible solution for more models in all cases when CC is applied to only the violated nonlinear constraints. The feasibility fraction is increased for all methods, and the total time is reduced for all methods except L_Basic_A_3, which sees a small increase.

5 Discussion

5.1 Constraint Consensus and Barrier Method Solvers

As mentioned earlier, the barrier method algorithm prefers a launch point that is close to satisfying the equality constraints and that oversatisfies the

Table 5: Applying CC to nonlinear constraints only. Results from the 117 models having both linear and nonlinear constraints.

Constraint Consensus Applied to all Constraints					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		0.83	0.83	0.85	0.87
Median Violation		3.83×10^3	5.64×10^3	1.99×10^3	5.15×10^5
Feasibility Fraction		0.01	0.03	0.04	0.00
Ipopt					
Avg.Time per run	16.75	15.37	13.97	11.78	16.33
Avg. Iterations	436.68	318.75	323.69	257.67	416.09
Feasibility Fraction	0.65	0.68	**0.70**	**0.70**	0.65
Total					
Avg.Time per run	16.75	16.21	14.80	**12.62**	17.20

Constraint Consensus Applied Only to Nonlinear Constraints					
CC	**None**	**L_FDfar**	**L_SUM**	**L_Basic_A_3**	**L_Basic**
Avg.Time per run		0.73	0.72	0.75	0.81
Median Violation		3.39×10^3	2.60×10^3	2.28×10^3	2.73×10^5
Feasibility Fraction		0.00	0.00	0.01	0.00
Ipopt					
Avg.Time per run	16.75	14.04	12.75	12.42	15.06
Avg. Iterations	436.68	295.33	274.51	295.98	407.76
Feasibility Fraction	0.65	0.71	**0.73**	0.71	0.67
Total					
Avg.Time per run	16.75	14.76	13.47	**13.18**	15.87

inequality constraints. In particular, a launch point that exactly satisfies an inequality constraint can lead to a blow-up in the barrier function. With this as a working hypothesis, we examine the characteristics of the launch points furnished by the CC methods and the success of the barrier method solver as reported in Experiment C.

In the subsequent analysis, each CC output point is categorized by the degree to which it satisfies each equality and inequality constraint in the model. The statistics reported are as follows:

- **Viol**: The average number of violated constraints.
- **OverSat**: The average number of inequality constraints satisfied by at least 1 (i.e., $v_i(\mathbf{x}_k) \leq -1$).
- **JustSat**: The average number of inequality constraints satisfied by less than 1 (i.e., $-1 < v_i(\mathbf{x}_k) \leq 0$).
- **Wtol**: The average number of inequality constraints within the tolerance (i.e., $0 < v_i(\mathbf{x}_k) \leq 10^{-6}$).

– **Close**: The average number of equality constraints close to being satisfied (i.e., $|v_i(\mathbf{x}_k)| \leq 1$).

Note that in Table 6, the number of inequalities is the sum of Viol, OverSat, JustSat, and Wtol, but the number of equalities is not equal to the sum of Viol and Close. This is because some of the equalities in the Close category actually violate the strict feasibility tolerance and hence are counted as both Viol and Close.

The characteristics of the output points returned by the CC algorithms for the 175 models having no timeouts or failures in Experiment C are given in Table 6, which uses the categories given above. The analysis distinguishes nonlinear constraints from linear constraints and equality constraints from inequality constraints. Equality constraints predominate in the test models (87% of constraints are equality) and nonlinear constraints outnumber linear constraints by a large margin (61% of the constraints are nonlinear). Over 75% of the models have at least one nonlinear equality constraint.

The data in Table 6 show that L_Basic_A_3 excels at both satisfying and oversatisfying nonlinear inequality constraints. L_Basic_A_3 decreases the number of violated nonlinear inequality constraints the most. The same observations can be made about the linear inequality constraints. None of the methods are particularly good at reducing the number of violated equality constraints, however, L_Basic_A_3 does the best job of finding points that are relatively close to satisfying nonlinear equality constraints.

We know from Experiment C that the barrier method solver is the most successful when paired with the L_Basic_A_3 CC variant. As shown by the data in Table 6, L_Basic_A_3 more often produces output points that have the characteristics preferred by the barrier method algorithm. Its output points satisfy or oversatisfy more of the nonlinear inequalities and come close to satisfying more of the nonlinear equality constraints. The output points from L_Basic_A_3 also perform relatively well on the linear constraints, but this is much less important due to way in which linear constraints are handled in the barrier algorithm.

5.2 Why Augmentation Works

Experiment A showed that augmentation generally improves the performance of CC variants that use Basic consensus. Fig. 7 analyzes the COCONUT model *airport*, composed of 84 variables and 42 quadratic constraints, to provide some insight as to why augmentation works. Fig. 7a and Fig. 7b depict the results for the L_Basic and L_Basic_A_3 algorithms, respectively. $||Cvec||$ is the length of the consensus vector. Fig. 7a shows that the measure of step size ($log(||Cvec||)$) for L_Basic is constantly small so it is unable to find a location with a substantially lower constraint violation ($log(\mathcal{V})$). In contrast, Fig. 7b shows that L_Basic_A_3 takes a relatively large step at every augmentation iteration causing $\mathcal{V}$ to decrease rapidly. In 27 iterations the L_Basic_A_3 algo-

Table 6: Characteristics of Constraint Consensus Output Points

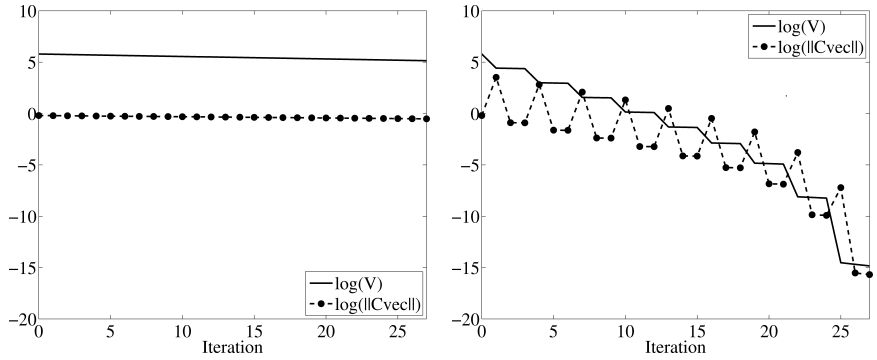
Nonlinear	**Inequality** (Avg. 173.30)				**Equality** (Avg. 812.11)	
CC	**Viol**	**OverSat**	**JustSat**	**Wtol**	**Viol**	**Close**
None	86.50	29.80	57.00	0.00	812.02	5.94
L_FDfar	30.40	69.67	73.01	0.22	810.55	51.73
L_SUM	33.61	**79.11**	51.60	**8.98**	**806.27**	59.33
L_Basic_A_3	**24.95**	71.31	**76.29**	0.74	810.80	**94.18**
L_Basic	52.75	58.35	62.19	0.00	811.66	9.02
Linear	**Inequality** (Avg. 31.71)				**Equality** (Avg. 582.31)	
CC	**Viol**	**OverSat**	**JustSat**	**Wtol**	**Viol**	**Close**
None	20.85	10.25	0.59	0.01	582.31	10.88
L_FDfar	14.24	16.52	0.93	0.03	**582.02**	**31.26**
L_SUM	20.35	9.10	1.20	**1.05**	582.04	8.91
L_Basic_A_3	**13.10**	**16.76**	**1.83**	0.01	582.27	15.22
L_Basic	15.25	15.42	1.03	0.01	582.28	11.04

rithm decreases $\mathcal{V}$ from 3.35×10^2 to 4.16×10^{-17}, as compared to L_Basic which has made little progress.

As shown in the results in the Online Appendix, augmentation generally improves the algorithms using Basic consensus. The L_Basic_A_3 variant uses augmentation the most frequently and it achieves the lowest median violation. Augmentation had mixed results for the methods using FDfar and SUM consensus. This has to do with how the consensus vector is formed in the first place. The FDfar and SUM consensus calculations are both much better at estimating step size than Basic consensus, so augmentation may often be redundant, or at least of little use for these consensus methods. For instance, FDfar is designed to approximately satisfy a single violated constraint at each iteration. If the approximation is reasonably accurate, then the augmenting step will not make a significant improvement, if any. The Basic consensus method is designed to try and satisfy all the violated constraints at each iteration using an averaging scheme. The approximations it makes are likely less accurate than the FDfar method and hence have more potential for improvement from augmentation.

5.3 Quadratic Feasibility Vectors

While quadratic feasibility vectors are exact for quadratic constraints, the results in the Online Appendix make clear that the additional calculation effort does not pay off. It may take several iterations using linear feasibility vectors to match the accuracy using quadratic feasibility vectors, but the linear iterations are so much faster that they take less time in total. The asymptotic



(a) L_Basic. (b) L_Basic_A_3.

Fig. 7: The effect of augmentation on the *airport* model. The value of the violation $\mathcal{V}$ and length of the consensus vector $||Cvec||$ are shown at each iteration.

time complexities of the linear and quadratic feasibility vector calculations are $O(n)$ and $O(n^2)$, respectively, for a single constraint having n variables. The average times for the Constraint Consensus variants using quadratic feasibility calculations are much higher than the other algorithms. As a result, the algorithms that use quadratic feasibility vector calculations are less successful on average at decreasing $\mathcal{V}$ in a short period of time.

5.4 SUM Consensus

Experiment C shows that SUM consensus works relatively well compared to the FDfar and Basic consensus methods. There are several explanations. First, FDfar moves to satisfy the most violated constraint at each iteration, but requires the same amount of computation time as the SUM and Basic methods because feasibility vectors for each violated constraint must be calculated in order to determine the most violated constraint. However FDfar may end up cycling between the same subsets of constraints because it uses only one constraint when forming the consensus vector whereas the SUM method uses all the feasibility vectors at every iteration, and so is less likely to cycle.

Second, Basic consensus averages the feasibility vectors. Although the direction of the Basic consensus vector may be accurate, the step size is less than ideal (as shown in Fig. 7). The SUM method adds the feasibility vectors together, and thus effectively takes more aggressive steps. Experiment C shows that this generally leads to better performance.

5.5 The Impact of Time Limits

The feasibility fractions achieved in the experiments are generally well below 100%. This is mainly because the algorithms (both the CC algorithms and the barrier method solver) were restricted to relatively short run times so that we could examine a large number of algorithm variants (see the Online Appendix) over a large number of test models. The short time limits have an impact because the models are nonlinear and large-scale, and hence inherently difficult to solve. Longer time limits for both CC and the local solver would likely result in higher feasibility fractions.

6 Conclusions and Future Research

Our experiments showed that the new Constraint Consensus techniques developed in this paper improve CC itself (in terms of finding points that are closer to feasibility in less time), and also improve the results returned by a barrier method local NLP solver. In particular:

- The SUM method is better than the existing Basic and FDfar CC algorithms.
- Augmentation can greatly improve the results of the Basic algorithm. In fact, the L_Basic_A_3 method returned the best overall results on all measures.
- Using the lowest violation point found after any iteration of a CC method is generally better than using the final point.
- Applying CC to only the nonlinear constraints in the model provides better performance. This allows for greater progress towards satisfying the difficult nonlinear constraints and leaves the linear constraints to be handled by the standard matrix methods included in nonlinear solvers.

We also observed that using quadratic feasibility vectors does not pay off in practice. It may be that in certain situations the added complexity is warranted, e.g. if the nonlinear constraints have particular properties, but it is not known what these properties might be. This is a topic for future research.

Based on these results, we recommend the use of the following CC algorithm for generating launch points for barrier method local solvers: the L_Basic_A_3 method applied to only the nonlinear constraints, and returning the point with the lowest violation found during the CC run. This will improve the probability that the barrier method solver is able to find a feasible solution, and will reduce the effort it expends in doing so.

An interesting result is the extent to which the success of a barrier method local solver is influenced by the characteristics of its launch point. We observed that the barrier method solver does best when the initial point satisfies or oversatisfies more of the nonlinear inequality constraints and comes close to satisfying more of the nonlinear equality constraints, as would be expected based on theory. This observation may be useful in the design of future Constraint Consensus methods or other launch point generators.

We plan to continue our development of Constraint Consensus methods. One direction is new methods of augmentation, perhaps based on a superposition of the last several consensus vectors. Dynamic stopping conditions based on measures of progress are also under consideration. In the same vein, if slow progress is detected it may be a signal to switch to a different CC algorithm or a different starting point.

References

1. Chinneck JW, *The Constraint Consensus Method for Finding Approximately Feasible Points in Nonlinear Programs*, INFORMS Journal on Computing, 16 (3), 255–265 (2004)
2. Ibrahim W and Chinneck JW, *Improving Solver Success in Reaching Feasibility for Sets of Nonlinear Constraints*, Computers and Operations Research, 35 (5), 1394–1411 (2008)
3. Chinneck JW, *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, Vol. 118, International Series in Operations Research and Management Sciences, Springer (2008)
4. Mittelmann HD, *Benchmarks for Optimization Software: AMPL-NLP Benchmark*, Accessed Online: October 29, (2010) `http://plato.asu.edu/ftp/ampl-nlp.html`
5. Censor Y, Zenios S, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford Univ. Press, New York (1997)
6. Gubin LG, Polyak BT and Raik EV, *The Method of Projections for Finding the Common Point of Convex Sets*, USSR Computational Mathematics and Mathematical Physics, 7, 1-24 (1967)
7. Cimmino , *Calcolo Approssimato per Soluzioni dei Sistemi di Equazioni Lineari*, La Ricerca Sci. XVI, Ser. II, Anno IX 1, 326-333
8. Aharoni R, Censor Y, *Block-iterative Projection Methods for Parallel Computation of Solutions to Convex Feasibility Problems*, Linear Algebra Appl. 120, 165-175 (1989)
9. Censor Y, Lent A, *Cyclic Subgradient Projections*, Math. Program. 24, 233-235 (1982)
10. Kaczmarz S, *Angenherte Auflsung von Systemen Linearer Gleichungen*, Bulletin International de l'Academie Polonaise des Sciences et des Lettres, series A, 35, 335–357 (1937)
11. McCormick SF, *An iterative procedure for the solution of constrained nonlinear equations with application to optimization problems*, Numerische Mathematik, Volume 23, Issue 5, 271–385, (1975)
12. Meyn KH, *Solution of underdetermined nonlinear equations by stationary iteration methods*, Numerische Mathematik, Volume 42, Issue 2, 161–172, (1983)
13. Martinez JM, *Solving Systems of Nonlinear Equations by Means of an Accelerated Successive Orthogonal Projections Method*, Journal of Computational and Applied Mathematics, Volume 16, Issue 2, 169–179, (1986)
14. Censor Y, Gordon D, and Gordon R, *Component Averaging: An Efficient Iterative Parallel Algorithm for Large and Sparse Unstructured Problems*, Parallel Comput. 27, 6, 777–808 (2001)
15. Tjalling YJ, *Historical Development of the Newton-Raphson Method*, SIAM Rev. 37 (4), 531–551 (1995)
16. Combettes PL, *The Convex Feasibility Problem in Image Recovery*, in: Hawkes P, ed., Advances in Imaging and Electron Physics 95, 155-270 (1996).
17. Mehrotra S, *On the Implementation of a Primal-dual Interior Point Method*, SIAM Journal on Optimization 2 (4): 575-601 (1992)
18. MacLeod M, *Multistart Constraint Consensus for Seeking Feasibility in Nonlinear Programs*, MASc Thesis, Systems and Computer Engineering, Carleton University, Ottawa, Canada (2006)
19. Wachter A and Biegler LT, *On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming*, Mathematical Programming, 106 (1), 25-57 (2006)

20. Shcherbina O, Neumaier A, Sam-Haroud D, Vu XH, and Nguyen TV, *Benchmarking Global Optimization and Constraint Satisfaction Codes*, Bliek C et al. (Eds.): COCOS 2002, LNCS 2861, 211222 (2003)
21. Fourer R, Gay DMM, Kernighan BW, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press (2002)
22. Wachter A, Laird C, Kawajir Y, *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt.*, Document Revision 1830, Accessed Online: July 25, (2011) `https://projects.coin-or.org/Ipopt`
23. Smith LR, *Improved Placement of Local Solver Launch Points for Large-scale Global Optimization*, PhD. Thesis, Systems and Computer Engineering Department, Carleton University, Ottawa, Ontario, Canada, (2011)
24. Dolan ED, More JJ, *Benchmarking Optimization Software with Performance Profiles*, Mathematical Programming 91, 201–213 (2002)

Noname manuscript No.
(will be inserted by the editor)

Online Appendix for: Improved Constraint Consensus Methods for Seeking Feasibility in Nonlinear Programs

Laurence Smith · John Chinneck · Victor Aitken

To appear in Computational Optimization with Applications (2012)

1 Quadratic Feasibility Vectors

The formula for a quadratic feasibility vector can be derived based on a second order Taylor series expansion in a manner similar to the derivation of the linear feasibility vector formula using a first order Taylor series expansion. We expand the constraint function g at the location $\mathbf{x}_k$:

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k)+\nabla g(\mathbf{x}_k)(\mathbf{x}_{k+1}-\mathbf{x}_k)+\frac{1}{2}(\mathbf{x}_{k+1}-\mathbf{x}_k)^T\nabla^2 g(\mathbf{x}_k)(\mathbf{x}_{k+1}-\mathbf{x}_k). \quad (1)$$

The desired movement direction is parallel to the gradient of the violated constraint at $\mathbf{x}_k$

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \rho_q \nabla g(\mathbf{x}_k)^T, \quad (2)$$

therefore, $g(\mathbf{x}_{k+1})$ is approximately

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \rho_q ||\nabla g(\mathbf{x}_k)^T||^2 + \frac{\rho_q^2}{2}\nabla g(\mathbf{x}_k)\nabla^2 g(\mathbf{x}_k)\nabla g(\mathbf{x}_k)^T. \quad (3)$$

The step size ρ_q is determined by setting $g(\mathbf{x}_{k+1}) = 0$ and solving

$$\rho_q = \frac{-b \pm \left[b^2 - 4ac\right]^{\frac{1}{2}}}{2a}, \quad (4)$$

Laurence Smith
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: lsmith@sce.carleton.ca

John Chinneck
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: chinneck@sce.carleton.ca

Victor Aitken
Systems and Computer Engineering, Carleton University, Ottawa, ON., Canada
E-mail: vaitken@sce.carleton.ca

where

$$a = \frac{1}{2}\nabla g(\mathbf{x}_k)\nabla^2 g(\mathbf{x}_k)\nabla g(\mathbf{x}_k)^T \tag{5}$$

$$b = ||\nabla g(\mathbf{x}_k)^T||^2 \tag{6}$$

$$c = g(\mathbf{x}_k). \tag{7}$$

The resulting feasibility vector is defined as

$$(\mathbf{x}_{k+1} - \mathbf{x}_k) = \rho_q \nabla g(\mathbf{x}_k)^T \tag{8}$$

with the definition of ρ_q as shown in Eqn. 4.

Due to the quadratic nature of the solution, there may be two distinct step sizes, and the step size may be complex. Both issues are related to the discriminant term of Eqn. 4

$$\gamma = b^2 - 4ac. \tag{9}$$

Three possibilities exist for the step size

1. $\gamma > 0$: There are two distinct real solutions because the quadratic approximation to the constraint function (Eqn. 3) is equal to zero at two points. For instance, the parabola $f(x) = x^2 - 1$ crosses zero at $x = \pm 1$.
2. $\gamma = 0$: There is one distinct real solution because the quadratic approximation to the constraint function is equal to zero at only one point. An example is the parabola $f(x) = x^2$ which equals zero only at $x = 0$.
3. $\gamma < 0$: There are two distinct complex solutions because the quadratic approximation to the constraint function does not equal zero in the real space. An example is the parabola $f(x) = x^2 + 1$.

When $\gamma > 0$ we choose the real root with the smallest magnitude in order to reduce the possibility of cycling between multiple solutions. When $\gamma < 0$ there are no real roots $\mathbf{x}_{k+1}$ such that $g(\mathbf{x}_{k+1}) = 0$. In this case we try to find a critical point of the quadratic approximation to the constraint function $g(\mathbf{x}_{k+1})$ instead. This is achieved by setting the derivative of Eqn. 3 with respect to the step size to zero, i.e.

$$0 = ||\nabla g(\mathbf{x}_k)^T||^2 + \rho_q \nabla g(\mathbf{x}_k)\nabla^2 g(\mathbf{x}_k)\nabla g(\mathbf{x}_k)^T, \tag{10}$$

and solving

$$\rho_q = \frac{-||\nabla g(\mathbf{x}_k)^T||^2}{\nabla g(\mathbf{x}_k)\nabla^2 g(\mathbf{x}_k)\nabla g(\mathbf{x}_k)^T}. \tag{11}$$

This is the same as taking the real part of Eqn. 4 when $\gamma < 0$, so we use this simpler approach.

The quadratic feasibility vector calculation can be used in place of the linear feasibility vector calculation in the CC algorithm, though the linear feasibility vector should be used when the constraint is linear. There is nothing to be gained using the quadratic feasibility calculation for linear constraints

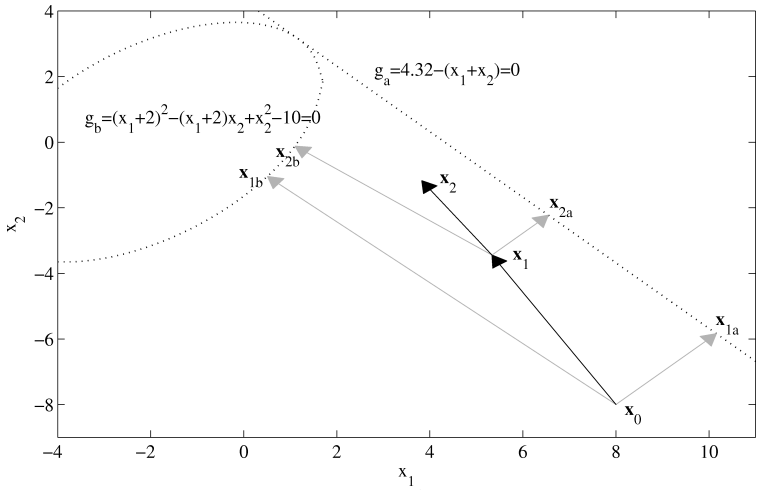


Fig. 1: The quadratic feasibility vector calculation.

because the Hessian is zero. The quadratic method also requires more computations per iteration. This should be taken into account when applying the quadratic feasibility calculation to fully simultaneous or block-iterative methods such as CC that may require many feasibility vector calculations per iteration. Although the quadratic feasibility vector is more accurate for quadratic constraints than the linear calculation, it is not necessarily more accurate for higher order nonlinear constraints.

As an example, consider again the example model in the main paper. The Hessian matrix for the nonlinear constraint is

$$\nabla^2 g_b(\mathbf{x}) = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}. \tag{12}$$

Given the initial point $\mathbf{x}_0 = [8 \quad -8]^T$ the quadratic parameters are:

$$\begin{aligned} a &= \frac{1}{2}\nabla g_b(\mathbf{x}_0)\nabla^2 g_b(\mathbf{x}_0)\nabla g_b(\mathbf{x}_0)^T && (13)\\ &= \frac{1}{2}\begin{bmatrix}28 & -26\end{bmatrix}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} 28 \\ -26 \end{bmatrix} && (14)\\ &= 2188, && (15)\\ b &= ||\nabla g_b(\mathbf{x}_0)^T||^2 && (16)\\ &= \begin{bmatrix}28 & -26\end{bmatrix}\begin{bmatrix} 28 \\ -26 \end{bmatrix} && (17)\\ &= 1460, && (18)\\ c &= g_b(\mathbf{x}_0) && (19)\\ &= 234. && (20) \end{aligned}$$

In this case, $\gamma = 83632 > 0$ so there are two real roots and therefore two possible values for ρ_q (-0.400 and -0.268). The step size with smallest magnitude, $\rho_q = -0.268$, is chosen. Therefore the quadratic feasibility vector is

$$(\mathbf{x}_{1b} - \mathbf{x}_0) = \rho_q \nabla g_b(\mathbf{x}_0)^T \tag{21}$$

$$= \begin{bmatrix} -7.492 \\ 6.956 \end{bmatrix}. \tag{22}$$

The quadratic feasibility vector is shown in Fig. 1. In this case both of the feasibility vectors are exact, however, the consensus vector $(\mathbf{x}_1 - \mathbf{x}_0)$ is still an estimate. At the point $\mathbf{x}_2$ in Fig. 1 the violations of constraints g_a and g_b are 1.669 and 32.138, respectively.

2 Additional Experimental Results

2.1 Additional Constraint Consensus Variants

In addition to the four CC variants tested in the main paper, we developed and analyzed eight other variants with different augmentation recurrence periods:

- **L_Basic_A_11**: linear feasibility vector, basic consensus, augmented with recurrence period of 11.
- **L_Basic_A_17**: linear feasibility vector, basic consensus, augmented with recurrence period of 17.
- **L_FDfar_A_3**: linear feasibility vector, FDfar consensus, augmented with recurrence period of 3.
- **L_FDfar_A_11**: linear feasibility vector, FDfar consensus, augmented with recurrence period of 11.
- **L_FDfar_A_17**: linear feasibility vector, FDfar consensus, augmented with recurrence period of 17.
- **L_SUM_A_3**: linear feasibility vector, SUM consensus, augmented with recurrence period of 3.
- **L_SUM_A_11**: linear feasibility vector, SUM consensus, augmented with recurrence period of 11.
- **L_SUM_A_17**: linear feasibility vector, SUM consensus, augmented with recurrence period of 17.

The periods of 3, 11 and 17 are chosen to gauge performance over a wide range of recurrence periods.

We also examined CC variants that use quadratic feasibility vectors. Variants denoted with q apply the quadratic feasibility calculation only to quadratic constraints and use linear feasibility vector calculations for all other constraints. Variants denoted with n apply the quadratic feasibility calculation to any nonlinear constraint that is violated. In all cases, linear feasibility vectors are used for all linear constraints. Six variants using quadratic feasibility vectors were tested:

Table 1: Results for PS I ($max\_time = 0.05s$). The median $\mathcal{V}$ for the start points is $1,710,000$.

Variant	**Avg. Time(s)**	**Avg. Its**	**Median $\mathcal{V}$**
L_Basic_A_3	0.01	87.47	117
L_FDfar	0.01	88.40	125
L_SUM	0.01	87.69	323
L_FDfar_A_17	0.01	87.01	390
L_Basic_A_11	0.01	90.23	537
L_FDfar_A_11	0.01	86.80	723
L_Basic_A_17	0.01	90.76	742
L_SUM_A_11	0.01	86.55	857
L_SUM_A_17	0.01	86.83	1,525
L_SUM_A_3	0.01	90.31	1,740
L_FDfar_A_3	0.01	87.20	2,140
Q_SUM_q	0.03	68.90	3,197
Q_FDfar_q	0.03	69.23	5,114
Q_FDfar_n	0.03	65.39	7,914
Q_SUM_n	0.03	64.65	8,212
L_Basic	0.01	93.07	29,800
Q_Basic_q	0.03	71.47	67,987
Q_Basic_n	0.04	66.80	148,440

- **Q_Basic_q**: Feasibility vector: quadratic, basic consensus, no augmentation, quadratic constraints only.
- **Q_Basic_n**: Feasibility vector: quadratic, basic consensus, no augmentation, all nonlinear constraints.
- **Q_FDfar_q**: Feasibility vector: quadratic, FDfar consensus, no augmentation, quadratic constraints only.
- **Q_FDfar_n**: Feasibility vector: quadratic, FDfar consensus, no augmentation, all nonlinear constraints.
- **Q_SUM_q**: Feasibility vector: quadratic, SUM consensus, no augmentation, quadratic constraints only.
- **Q_SUM_n**: Feasibility vector: quadratic, SUM consensus, no augmentation, all nonlinear constraints.

We distinguish between quadratic and nonlinear constraints because of the tradeoff between time and accuracy. The quadratic feasibility vector calculation requires more time than the linear feasibility vector calculation, but is more accurate for nonlinear constraints.

2.2 Additional Numerical Results

The complete results for Experiment A, including all tested methods, are reported in Tables 1-3. The goal of the algorithms in this experiment is to reduce the maximum violation as much as possible given the time and iteration restrictions.

Table 2: Results for PS II ($max\_time = 0.5s$). The median $\mathcal{V}$ for the start points is $1,255,000$.

Variant	**Avg. Time(s)**	**Avg. Its**	**Median $\mathcal{V}$**
L_Basic_A_3	0.24	77.14	140
L_FDfar_A_17	0.24	78.82	455
L_FDfar_A_11	0.24	78.78	456
L_SUM_A_17	0.23	73.18	501
L_SUM_A_11	0.23	74.36	502
L_FDfar_A_3	0.22	80.18	1,065
L_FDfar	0.24	79.62	1,075
L_Basic_A_11	0.27	77.08	1,275
L_SUM	0.24	76.82	1,315
L_SUM_A_3	0.22	81.67	1,540
L_Basic_A_17	0.27	76.64	1,725
Q_SUM_q	0.42	45.71	4,023
Q_FDfar_q	0.43	46.07	5,351
Q_SUM_n	0.46	29.67	20,609
Q_FDfar_n	0.47	29.26	174,120
L_Basic	0.27	76.66	280,500
Q_Basic_q	0.44	42.74	412,720
Q_Basic_n	0.47	28.61	768,010

Table 3: Results for PS III ($max\_time = 5.0s$). The median $\mathcal{V}$ for the start points is $1,580,000$.

Variant	**Avg. Time(s)**	**Avg. Its**	**Median $\mathcal{V}$**
L_Basic_A_3	3.87	52.24	1,765
L_SUM_A_11	3.78	53.78	2,540
L_SUM	3.77	51.80	3,760
L_SUM_A_17	3.79	53.49	3,760
L_SUM_A_3	3.67	58.87	4,110
L_Basic_A_11	3.96	48.13	5,595
L_FDfar	3.80	52.56	8,965
Q_SUM_q	4.50	24.82	19,170
L_Basic_A_17	3.99	47.90	91,400
L_FDfar_A_3	3.71	59.70	251,000
L_FDfar_A_11	3.80	54.28	335,000
L_FDfar_A_17	3.81	53.73	481,000
L_Basic	4.03	46.87	531,500
Q_FDfar_q	4.52	25.36	679,260
Q_SUM_n	4.88	8.51	935,445
Q_FDfar_n	4.96	6.25	971,490
Q_Basic_q	4.61	22.39	1,034,500
Q_Basic_n	4.97	6.32	1,163,200

Algorithms using Basic consensus and linear feasibility vectors improve as augmentation is used more frequently. Table 1 shows that L_Basic_A_3 performed the best on PS I while the original L_Basic was the worst. A similar trend is seen for PS II and PS III, found in Table 2 and Table 3, respectively. The L_Basic_A_3 variant was the top algorithm for each problem set and it uses augmentation the most frequently.

There is no clear trend for the frequency of augmentation for FDfar and SUM with linear feasibility vectors.

Tables 1-3 show that the quadratic feasibility vector calculations are much more expensive than the linear feasibility vector calculations: the average numbers of iterations completed within the time limit are always much smaller for the quadratic versions of the algorithms. As a consequence, the median violations of the quadratic methods are substantially higher than most of the linear variants. Applying quadratic feasibility vectors only for quadratic constraints is slightly better than finding quadratic feasibility vectors for all nonlinear constraints. The best performing quadratic variant in all three problem sets is Q_SUM_q, but its relative peformance is much worse than many linear variants.