Optimization Methods and Software
Vol. 00, No. 00, November 2011, 1–30

RESEARCH ARTICLE

Constraint Consensus Concentration for Identifying Disjoint Feasible Regions in Nonlinear Programs

Laurence Smith[a], John Chinneck[a*], and Victor Aitken[a]

[a]*Systems and Computer Engineering, Carleton University, Ottawa, On., Canada*

(*Preprint. Final version to appear in Optimization Methods and Software (accepted 2011)*)

It is usually not known in advance whether a nonlinear set of constraints has zero, one, or multiple feasible regions. Further, if one or more feasible regions exist, their locations are usually unknown. We propose a method for exploring the variable space quickly using Constraint Consensus to identify promising areas that may contain a feasible region. Multiple Constraint Consensus solution points are clustered to identify regions of attraction. A new inter-point distance frequency distribution technique is used to determine the critical distance for the single linkage clustering algorithm, which in turn determines the estimated number of disjoint feasible regions. The effectiveness of multistart global optimization is increased due to better exploration of the variable space, and efficiency is also increased because the expensive local solver is launched just once near each identified feasible region. The method is demonstrated on a variety of highly nonlinear models.

Keywords: nonlinear constrained global optimization; Constraint Consensus; clustering; concentration; disjoint feasible regions

AMS Subject Classification: 90C06; 90C30; 90C99;

1. Introduction

When a model has one or more nonlinear constraints it can be very difficult to find a feasible point in a nonlinear program (NLP). It may not be known a priori whether the model is feasible, or if it is, how many discontiguous feasible regions there are and where they are located. In addition, some nonlinear optimization algorithms require a feasible starting point, and many others are much more efficient and effective if started at a point that is either feasible or close to feasibility. In fact the problem of finding a feasible point is just as hard as the global optimization problem in that it is equivalent to finding a global minimum of an objective that expresses the sum of the constraint violations. The global minimum of this objective is zero, i.e. a feasible point.

The Constraint Consensus (CC) algorithm rapidly and inexpensively moves an infeasible initial point that may be very far away from feasibility to a final point that is close to feasibility in large nonlinear models [3]. Running the CC algorithm prior to launching an NLP solver greatly reduces the total solution time in most cases, and improves the probability that a nonlinear solver will find a feasible point [10]. CC calculates a *feasibility vector* for each constraint that is violated at the current point; this vector estimates the smallest update needed to satisfy the

*Corresponding author. Email: chinneck@sce.carleton.ca

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/1055678xxxxxxxxxxxxx
http://www.informaworld.com

violated constraint. The feasibility vectors for all of the violated constraints are then combined into a single *consensus vector* that is added to the current point to move it closer to feasibility. The process is repeated until the stopping conditions are met, and the output point is passed to a full-scale NLP solver as its launch point.

We combine CC with a clustering technique to rapidly identify the disjoint feasible regions in a constrained NLP. A multistart method generates a population of initial points. CC is started at these points and outputs final points that are closer to feasibility. The CC final points cluster near *regions of attraction* (local minima of the sum of absolute constraint violations), which are frequently feasible regions. The clusters are identified by a single linkage clustering technique. The critical distance used to separate the clusters is determined by a new automated inter-point frequency distribution technique.

The clusters are identified as the first step in a probabilistic global optimization algorithm. The intent is to identify disjoint feasible regions so that a multistart method can launch a local solver near each of them. This allows a very large solution space to be explored inexpensively. Overall speed and efficiency is increased by launching the expensive local solver only at points that are in or near probable feasible regions, and by launching the local solver only once near each of them.

Our focus is on the development of multistart methods for global optimization that are effective and efficient for very large nonlinearly constrained optimization models. The methods must scale well. The main contributions are the development of an inexpensive method to explore the variable space in a complex high-dimension nonlinear program to identify disjoint feasible regions, and the invention of a way to automatically determine the critical distance for the clustering method that is used. The new methods are empirically evaluated on a large number of highly nonlinear models.

2. Background

2.1. *Nonlinear Programming and Notation*

The general NLP problem is formulated as:

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \tag{1a}$$

$$s.t. \; g_i(\mathbf{x})\{\leq,=\}0, \; (i=1,...,m) \tag{1b}$$

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

where $\mathbf{x} = (x_1,...,x_n) \in \mathbb{R}^n$ is an n dimensional solution vector, $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the continuously differentiable objective function, $g_i(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ are the continuously differentiable constraints, and ℓ_j and u_j are the lower and upper bounds on the j^{th} element of vector $\mathbf{x}$. The search space $\mathcal{S}$ is defined as

$$\mathcal{S} \doteq \{\mathbf{x} \in \mathbb{R}^n \mid \forall j : \; \ell_j \leq x_j \leq u_j\} \tag{2}$$

and the feasible region $\mathcal{F}$ is defined as

$$\mathcal{F} \doteq \{\mathbf{x} \in \mathcal{S} \mid \forall i : \; g_i(\mathbf{x})\{\leq,=\}0\}\,. \tag{3}$$

An optimal solution vector, $\mathbf{x}^* \in \mathcal{F}$, is a solution in the feasible region with the lowest objective function value.

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

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

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

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

Solvers are algorithms that attempt to find an optimal solution vector. Alternatively, constraint satisfaction algorithms attempt to find any solution vector within the feasible region.

2.2. *Constraint Consensus Methods for Constraint Satisfaction*

A variety of iterative procedures move points towards locations where they satisfy the constraints. Variations on Newton's method [5, 7, 9, 12] are popular, but require the calculation of an inverse matrix. For example, the method of Chootinan and Chen [5] requires the calculation of a pseudo-inverse matrix at every iteration. The inverse calculation is not suitable for large models having many constraints and variables because it is very time consuming.

Constraint Consensus is an alternative approach that is very fast because it avoids time-consuming inverse calculations and line searches [3, 10]. The first step is to construct a *feasibility vector* for each violated constraint, given by

$$\mathbf{w} = \frac{-g(\mathbf{x})}{||\nabla g(\mathbf{x})^T||^2} \nabla g(\mathbf{x})^T, \tag{6}$$

where $\nabla g(\mathbf{x})$ is the gradient of the violated constraint. $\mathbf{w}$ is the estimated vector from the current point to the closest feasible point for that constraint. It is exact for linear constraints.

The next step combines the feasibility vectors into a single *consensus vector* that updates the current point. In the Basic CC method, the consensus vector is calculated by a component-wise averaging of the elements of the feasibility vectors. For each variable, the averaging calculation considers only the feasibility vectors that include that variable. There are a variety of other ways to combine the feasibility vectors to create a consensus vector.

Pseudocode for the Basic Constraint Consensus method [3] is given in Alg. 1. $NINF$ is the number of infeasibilities, w_{ij} is the value of the feasibility vector element corresponding to the j^{th} variable in the i^{th} constraint.

In the algorithms developed here, CC is used for concentrating points near regions of attraction. We use the Basic consensus [3] and the new Augmented feasibility vector [22] variants of CC in this paper. The augmented version is a predictor-corrector style algorithm that adjusts the length of the feasibility vectors.

2.3. *Multistart Heuristics for Global Optimization*

The most efficient possible multistart method for global optimization would launch a local solver exactly once in the vicinity of each local optimum. For this reason, a variety of multistart heuristics try to identify disjoint feasible regions so that

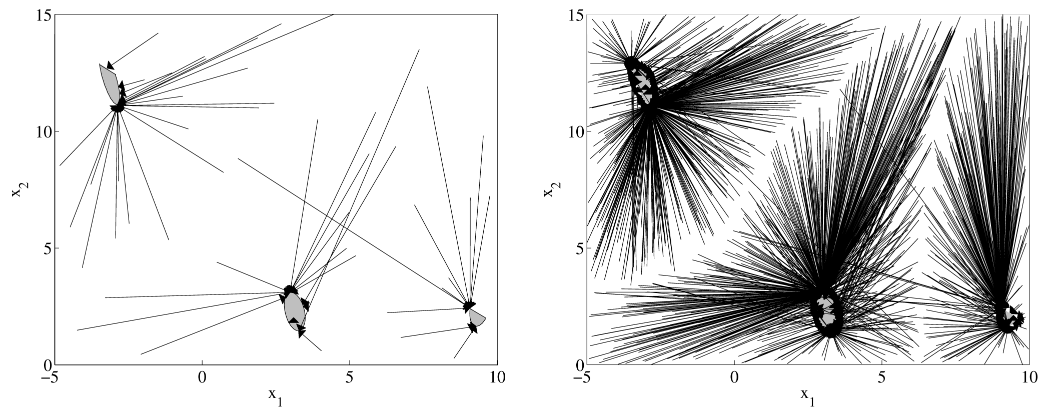
Algorithm 1 Basic Constraint Consensus [3]

```
**INPUT:** (i) a set of constraints, (ii) an initial point $\mathbf{x}$, (iii) feasibility distance tolerance $\alpha$, (iv) movement tolerance $\beta$, (v) maximum number of iterations $\mu$.
$k \leftarrow 0$
while $k < \mu$ do
    $NINF \leftarrow 0$; for all $j$: $n_j \leftarrow 0$, $s_j \leftarrow 0$
    for every constraint $g_i$ do
        if $g_i$ is violated then
            Calculate $\mathbf{w}_i$ and $||\mathbf{w}_i||$
            if $||\mathbf{w}_i|| > \alpha$ then
                $NINF \leftarrow NINF + 1$
                for every variable $x_j$ in $g_i$ do
                    $n_j \leftarrow n_j + 1$; $s_j \leftarrow s_j + w_{ij}$
                end for
            end if
        end if
    end for
    if $NINF = 0$ then
        Exit successfully
    end if
    for every variable $x_j$ do
        $t_j \leftarrow \frac{s_j}{n_j}$
    end for
    if $||\mathbf{t}|| \leq \beta$ then
        Exit unsuccessfully
    end if
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{t}$
    Reset $\mathbf{x}$ to respect any violated variable bounds
    $k \leftarrow k + 1$
end while
```

the local solver can be launched exactly once near each one, thereby reducing the number of redundant launches [4]. Two of the main concepts are concentration and acceptance/rejection.

Multistart-NLP (MSNLP) is a heuristic multistart algorithm composed of two phases [13]. The first phase generates a set of random candidate points, which are stored along with their calculated measure of merit. The second phase applies a *distance filter* and a *merit filter* that reject points according to their criteria. The distance filter tries to ensure that the starting points are diverse, i.e., that none of the starting points are within a basin of attraction of the same local optimum. MSNLP approximates the basins of attraction as spheres around the best known solutions. Additional proposed starting points within the approximated basins are rejected. The merit filter tries to ensure that starting points exceed a certain quality level. In MSNLP the merit filter is an L_1 exact penalty function. A candidate point is rejected if its merit function value fails to meet a certain threshold. A candidate point that passes both filters is passed to a local NLP solver for use as the launch point. If the solver returns a feasible and locally optimum solution, then its value is stored. Upon termination the best feasible solution yet discovered is returned as the global optimum point. An issue is that the hypersphere approximations of basins of attraction are not always accurate, as can be seen in Fig. 1.

GLOBALm is another two-phase multistart method [19]. It takes a uniform sample of points from the variable space in the first phase. These candidate points are



(a) 50 points.

(b) 1500 points.

Figure 1. Basin visualization via CC for the Branin1 model: (a) CC start-end pairs for 50 random start points. (b) CC start-end pairs for 1500 random start points.

assigned merit values with an L_1 exact penalty function. The initial points with the worst merit values are immediately rejected. In the second phase, points are selected from the remaining set of candidate points and run through a single linkage clustering algorithm. If a point cannot be assigned to an existing cluster, it is used to launch a local solver to begin the formation of a new cluster. This process is repeated until all of the candidate points are assigned to clusters.

The GLOBALm scheme makes no attempt to improve any of the points prior to the solver launch. Another issue is the way in which the critical distance is determined for the single linkage clustering step. GLOBALm uses a formula based on the number of samples and the dimension of the problem to calculate the critical distance (Eqn. 10 in [19]). This approach may not be dependable since problems and their respective characteristics vary greatly, regardless of the number of variables involved.

MacLeod [14] was the first to use Constraint Consensus to explore the variable space in an effort to identify good launch points for a local solver. His Multistart Constraint Consensus method uses the information obtained from an initial set of randomly started constraint consensus runs to assign votes to various zones in the variable space, where more votes correspond to a greater belief that a feasible point exists in the corresponding zone. This information is used to guide the placement of later CC initial points, whose information is used to update the vote totals. To avoid a combinatorial explosion in the number of zones, the voting information is projected onto the axes, so that each axis is individually subdivided into zones with associated vote totals. Promising experiments in finding feasible regions in difficult NLPs were carried out using this method in conjunction with the Knitro local solver [2]. A drawback of the voting method used is that it tends to make the feasible regions appear larger than they actually are. For the purposes of this paper, the major drawback is that the method is oriented towards finding a single launch point that leads the solver to feasibility, as opposed to identifying all of the disjoint feasible regions.

3. Constraint Consensus and Basins of Attraction

The central idea of the method developed in this paper is to use CC to quickly and inexpensively explore the variable space to identify promising regions in which to launch expensive local solvers. CC runs are started at random points in the variable space, and the CC end points concentrate in subsets near regions of attraction, typically near feasible regions. The trails of points from the starts of CC runs to their ends can be used to visualize approximate *basins of attraction* (regions in which a CC run will lead to the same feasible region) as shown in Fig. 1.

Consider the Branin1 model illustrated in Fig. 1, in which $g_1(x)$ is a variant of the well-known Branin function [11] used as a constraint:

$$find \;\; \mathbf{x} = \{x_1, x_2\} \tag{7a}$$

$$s.t. \;\; g_1(\mathbf{x}) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + \left(10 - \frac{10}{8\pi}\right)\cos(x_1) + 9 \leq 0 \tag{7b}$$

$$g_2(\mathbf{x}) = x_2 + \frac{x_1 - 12}{1.2} \leq 0 \tag{7c}$$

$$-5 \leq x_1 \leq 10 \tag{7d}$$

$$0 \leq x_2 \leq 15 \tag{7e}$$

The Branin1 model has both a linear and a nonlinear constraint. There are three feasible regions shown as gray areas in Fig. 1.

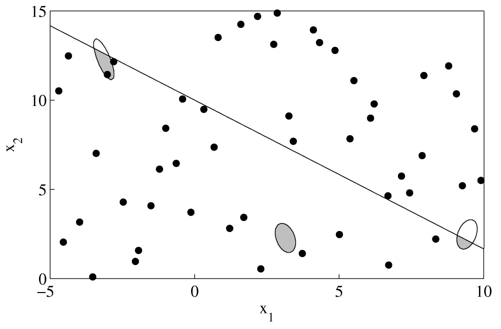
Fig. 1a illustrates 50 CC runs for the Branin1 model, where each line segment connects the start point and the end point for a CC run. The basins of attraction start to become obvious when CC is run from many start points as shown in Fig. 1b. The white space visualizes the boundaries between the basins of attraction. Some of the CC paths in Fig. 1 cross between basin approximations; this is because a basin may actually be composed of disjoint regions due to the nature of nonlinear functions and their interactions with solution algorithms (e.g., a solution algorithm may take a step that is too large). In these vicinities CC may take a path that does not lead to the closest (in terms of Euclidean distance) feasible region.

4. An Efficient Algorithm for Choosing Multi-Start Local Solver Launch Points

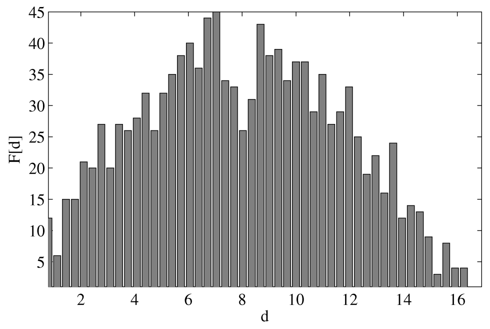
The main steps in the algorithm for selecting local solver launch points are:

(1) *Initial Sample* (Fig. 2a). Choose random points in the variable space.
(2) *Concentrate* (Fig. 2c). Launch CC from each of the random sample points. This concentrates the CC end points around various regions of attraction, typically feasible regions.
(3) *Choose the critical distance* (Fig. 2d). The concentrated CC end points are analyzed to choose a suitable critical distance for use in the subsequent clustering step.
(4) *Cluster* (Fig. 2e). Apply a single linkage clustering algorithm to identify clusters of CC end points using the critical distance found in the previous step.
(5) *Choose local solver launch points* (Fig. 2f). Evaluate the CC end points and launch the local solver from the most promising point in each cluster. The best solution value from all the local solver launches is kept as the optimum point.

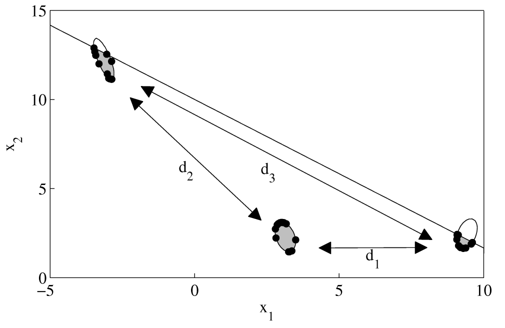
The details of these steps are explained below.



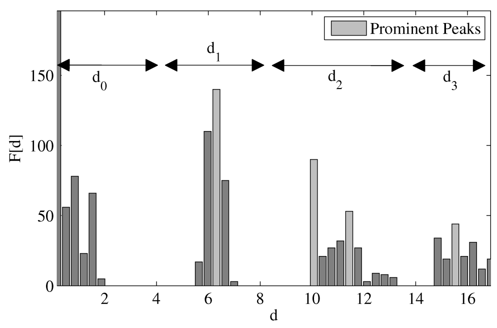
(a) Latin hypercube sample.



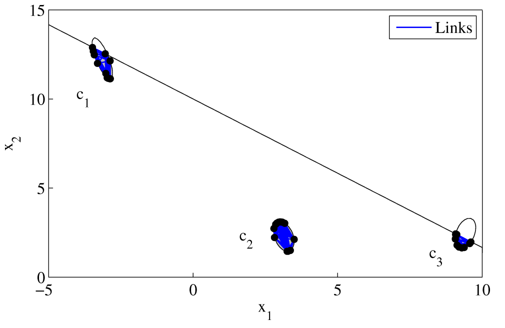
(b) Initial distribution.



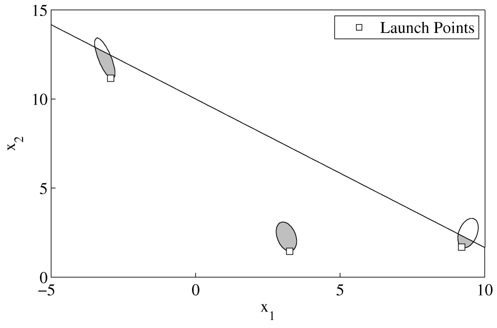
(c) Concentrated points.



(d) Distribution of concentrated points.



(e) Clusters formed by a single-linkage method.



(f) Launch points.

Figure 2. Concentration and clustering: (a) Fifty initial points chosen by Latin hypercube sampling. The gray areas represent feasible regions for the Branin1 model. (b) The initial inter-point distance frequency distribution. (c) The CC end points are concentrated near the feasible regions. (d) The frequency distribution of the inter-point distances for the concentrated points. The prominent peaks are calculated and used to determine the critical distances. (e) The concentrated final points are clustered using a single-linkage method. Three clusters are discovered. (f) The most promising points from each cluster are used as launch points for a local solver.

4.1. *Initial Sample*

Latin Hypercube sampling is used for the initial sample within the variable space defined by the variable bounds. This ensures that the points are distributed throughout the search space, and Latin Hypercube sampling is known to provide better coverage than simple random sampling [15]. Further, the number of samples required for Latin Hypercube sampling is independent of the number of dimensions of the model. This is important since our methods are intended for large-scale problems having many variables. The number of initial sample points is controlled by the parameter p.

4.2. *Concentration*

Running CC from a set of initial points produces a concentration of the CC end points near the various regions of attraction where the constraint violations are minimized. If feasible regions exist they will be near these regions. We use the Basic CC algorithm with augmented feasibility vectors, though other versions of CC could be substituted.

4.3. *Choosing the Critical Distance*

The single-linkage clustering method [21] that we use depends heavily on the accurate identification of an appropriate critical distance. Points that are closer together than the critical distance are merged into the same cluster. Any point in a given cluster is separated from any point in a different cluster by more than the critical distance. The critical distance thus determines the number of clusters that are identified. It is generally difficult to determine an appropriate critical distance a priori. We develop a new procedure for doing so automatically that depends on two parameters.

We first reduce the high-dimensional clustering data to a simple distribution of distances between data points [1]. We exploit the fact that the frequency distribution of the inter-point distances between the concentrated CC end points tends to have peaks that correspond to the distances within and between clusters. We observed through a large number of experiments that a set of randomly sampled points in equally weighted dimensions has an inter-point distance distribution that is most frequently unimodal and usually has a shape similar to a Rayleigh distribution (an example of this curve for the Branin1 variable space is shown in Fig. 2b). In contrast, points concentrated near the same region of attraction will be closer to each other than to points concentrated in other regions. Therefore, the frequency distribution of inter-point distances for the concentrated set will be multi-modal if there is more than one region of attraction. This is illustrated in Fig. 2d.

Critical distances that approximate the separation of clusters can be extracted from the multi-modal distribution. For instance, consider Fig. 2c showing post-CC concentrated points and Fig. 2d showing the frequency distribution of the inter-point distances. There are four distinct groups of inter-point distances: d_0, d_1, d_2, and d_3. The set of inter-point distances labeled d_0 represents the distances between points near the same region of attraction. The inter-point distances labeled d_1, d_2, and d_3 represent the distances between the points in different concentrated sets as depicted in Fig. 2c. We posit that the critical distance for a single-linkage clustering routine can be extracted from the inter-point frequency distribution, and used to estimate how many regions of attraction exist in a given model using a single-linkage clustering technique.

The inter-point frequency distribution, $F[d]$, of the concentrated points will have peaks because some subsets of points are close to each other and, in general, further away from other sets of points. This is shown in Fig. 2c and Fig. 2d. Distances, d, that correspond to minima in $F[d]$ are effective critical distances for the single-linkage algorithm because they are the distances that separate clusters. For instance, $d \simeq 4$ separates clusters c_1, c_2, and c_3 in Fig. 2e. Alternatively, $d \simeq 9$ separates the points in clusters c_2 and c_3 from those in c_1. The differences in frequencies between successive distances tend to be low in the regions near the minima, for instance, consider the region $2 < d < 6$ in Fig. 2d. For this reason we identify maxima in the inter-point distance distribution instead, and then approximate the minima using the midpoints between the maxima.

The main steps in the procedure for identifying the critical distance are:

(1) Calculate all of the inter-point distances among the CC end points. If there are p CC end points, then there will be $\frac{1}{2}p(p-1)$ inter-point distances.
(2) Construct the frequency distribution histogram for the inter-point distances by determining the histogram bin limits, and then assigning the inter-point distances to the correct bins.
(3) Identify the prominent peaks in the inter-point distance frequency distribution histogram.
(4) Use the prominent peaks to calculate a useful critical distance.

We determine the histogram bins representing the inter-point distance ranges based on the smallest and largest inter-point distances, d_{min} and d_{max}, respectively. The range of an individual bin, d_{width}, is determined by dividing the difference between d_{min} and d_{max} by the number of sample points, p. The nominal distance associated with a bin is given by its midpoint or bin center, where

$$d_i = d_{min} + (i + \frac{1}{2})d_{width}, i \in \{0, p-1\} \tag{8}$$

are the bin centers. All the inter-point distances are then sorted into their respective histogram bins. The number of inter-point distances in each bin constitutes the frequency distribution, $F[d_i]$.

The parameter ω is then used in the identification of *prominent peaks* in the frequency distribution, defined as bins with a population that is higher than the ω preceding bins and the subsequent ω bins. For example consider Fig. 3 in which $p = 7$, $d_{min} = 2$, $d_{max} = 16$, $d_{width} = \frac{16-2}{7} = 2$, and the nominal distance associated with the bin $[2, 4)$ is $d_0 = 3$. There are 21 inter-point distances among the 7 points. Fig. 3 illustrates the relation between frequencies and bin centers and the labels used to represent the distances and frequencies. Fig. 3 shows that there is only one prominent peak when $\omega = 2$. In this case $F[d_4] = 7$ is greater than the frequencies of the two preceding bins, $F[d_2] = 5$ and $F[d_3] = 3$, and the two subsequent bins, $F[d_5] = 2$ and $F[d_6] = 1$. If $\omega = 1$ then there are two prominent peaks: the bin centered at d_4 whose frequency is greater than the bins centered at d_3 and d_5, and the bin centered at d_2 whose frequency is greater than its preceding and subsequent bins.

Distances that correspond to minima in $F[d]$ tend to be effective critical distances for the single-linkage clustering algorithm. For example, consider a model that has two widely separated clusters of CC end points. There will be a peak in $F[d]$ at the smaller end of the scale that corresponds to the distances between points in the same cluster, and a peak in $F[d]$ at the larger end of the scale that corresponds to the distances between points in different clusters. An ideal critical distance will be halfway between these two peaks. Values smaller than this might break up the clusters into several parts, and values larger than this could cause the clusters to merge. For this reason we identify the critical distance by first identifying the prominent peaks, and then choosing the critical distances typically at the midpoint between the prominent peaks to approximate the minima in $F[d]$.

Since there may be multiple minima in $F[d]$, all of which may represent suitable values for the critical distance, a parameter τ is used to control which value is selected. We start with the smallest calculated critical distance, but if more than τ clusters are identified, then the next larger critical distance is used. This process repeats until the critical distance identifies τ or fewer clusters.

Consider the Branin1 model in Fig. 2d for example. The first peak occurs near $d \simeq 6$. The first minimum occurs somewhere in the range $2 < d < 5$. This minimum is approximated by averaging d_{min} and the distance associated with the first peak:

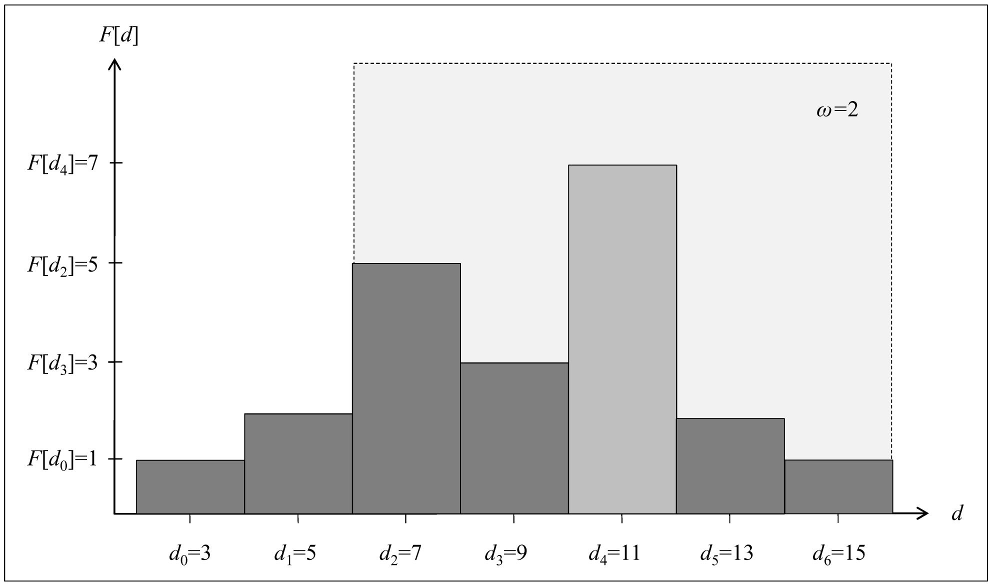


Figure 3. Finding prominent peaks. Only one prominent peak exists at $d_4 = 11$ when $\omega = 2$. The gray region highlights the bins involved in the prominent peak calculation for the bin centered at $d_4 = 11$. There are prominent peaks at $d_2 = 7$ and $d_4 = 11$ when $\omega = 1$. In this example $p = 7$, $d_{min} = 2$, $d_{max} = 16$, and $d_{width} = 2$.

$d_c \simeq \frac{0+6}{2} = 3$. If the resulting number of clusters falls below the threshold, τ, then this distance is accepted as the critical distance. If the number of clusters exceeds the threshold, then a distance between the first peak and the second peak is calculated and tried as the clustering distance. In the Branin1 example the second peak occurs at an inter-point distance $d \simeq 10$, hence the second critical distance candidate is $\frac{6+10}{2} = 8$. If the number of clusters again exceeds the threshold when $d_c = 8$ then the next clustering distance is tried. This pattern is repeated until the number of clusters falls below the threshold or a maximum number of clustering distances is tried. The method is summarized below.

(1) Determine the set of prominent peak distances $\{q_j | \, j \in \{1, ..., J\}\}$ using ω, where q_j is the bin center of the j^{th} prominent peak. Order the peaks from shortest to longest in terms of inter-point distances. If the number of peaks is zero (i.e., $J = 0$), reduce ω by one and repeat step 1 if $\omega \neq 0$, else exit unsuccessfully. We found that an initial value of $\omega = 3$ works well for many models.
(2) Cluster the concentrated points using the critical distance $d_c = (d_{min} + q_1)/2$. If the number of clusters is less than or equal to τ then exit successfully, else go to step 3.
(3) Try clustering the concentrated points with $d_c = (q_j + q_{j+1})/2$ for each element $j \in \{1, ..., J-1\}$. If the number of clusters found is ever less than τ then exit successfully, else reduce ω by one and go to step 1.

For the example in Fig. 3 the critical distance extracted is $\frac{2+11}{2} = 6.5$ if $\omega = 2$. If $\omega = 1$ the first critical distance is 4.5, and the second, if needed, is 9.0.

4.4. *Clustering*

We use a single-linkage clustering method [21] to group the concentrated CC end points into clusters, which typically correspond to a single region of attraction (generally a feasible region). The method begins by assuming that each point is its own cluster. The two clusters that are the closest are then merged into a single cluster if they are separated by a distance smaller than the critical distance. This process is repeated until either all the points are in one cluster or the closest clusters are further from each other than the critical distance.

4.5. *Choosing Local Solver Launch Points*

CC end points may be feasible or infeasible. If feasible they likely have various values of the objective function, and if infeasible likely have various values of the infeasibility measure. We must consider all of these factors in ranking points as potential solver launch points. We use the following hierarchical system to rank solution vectors in order of their *promise* as local solver launch points:

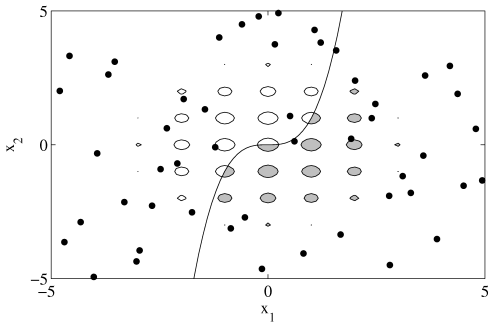
(1) All feasible solutions are more promising than infeasible solutions.
(2) If two solution vectors are feasible, the one with the lower objective function value is more promising.
(3) If two solutions vectors are infeasible the one with the lowest maximum constraint violation is more promising.

The most promising point in each cluster is selected and added to a short-list. The short-list of launch points is ordered from most to least promising. The local solver is launched from each of the points in the short-list, in order. The number of local solver launches is equal to the number of clusters identified, and is therefore τ or fewer, since τ limits the maximum number of clusters identified.

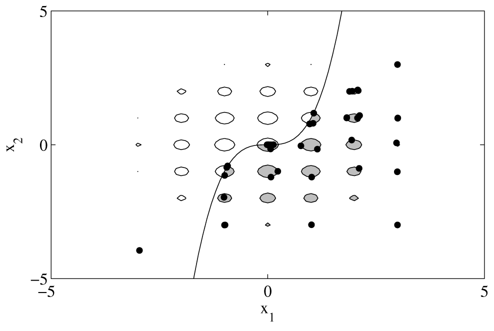
4.6. *Complete Algorithm*

Pseudocode for the complete multi-start with clustering (MS+C) algorithm follows:

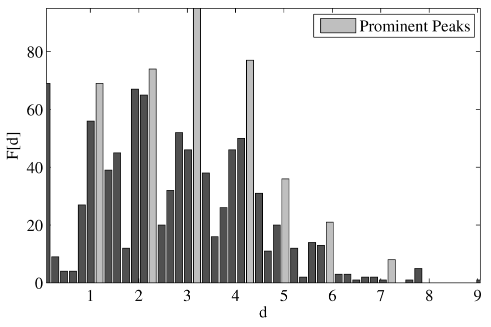
```
(1) Initialize: select values for $p$, $\omega$, and $\tau$, and CC parameters $\alpha$, $\beta$, and $\mu$.
(2) Choose random points in the variable space via Latin Hypercube sampling.
(3) Launch CC from each of the random sample points.
(4) Calculate all of the inter-point distances among the CC end points.
(5) Construct the frequency distribution histogram $F[d]$:
    a) Bin width is $\frac{1}{p}(d_{max} - d_{min})$.
    b) Nominal distance associated with each bin is
           $d_i = d_{min} + (i + \frac{1}{2})d_{width}, i \in \{0, p-1\}$.
    c) Assign each inter-point distance to the appropriate histogram bin.
(6) Determine the set of prominent peak distances $\{q_j | \, j \in \{1, ..., J\}\}$:
    a) Find the histogram bins whose populations are greater than the populations in the preceding and following $\omega$ bins. These are the prominent peaks.
    b) Sort the nominal distances $d_i$ associated with the prominent peaks into order from smallest to largest, and label these $q_1$ through $q_J$, in order.
    c) If the number of prominent peaks is zero (i.e., $J = 0$), reduce $\omega$ by one and repeat step (6) if $\omega \neq 0$, else exit unsuccessfully.
    d) Set the critical distance at $d_c = (d_{min} + q_1)/2$, and set $j = 1$.
(7) Cluster the points using the critical distance $d_c$.
(8) If the number of clusters is greater than $\tau$ then:
    a) $j \leftarrow j + 1$.
```



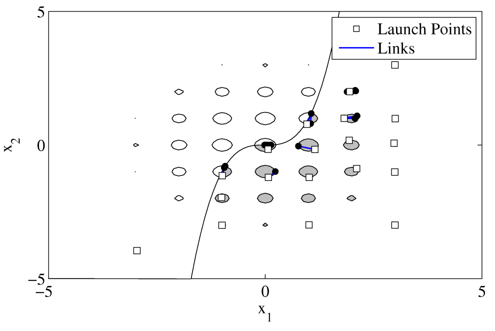
(a) Random points.



(b) Concentrated points.



(c) Inter-point frequency distributions.



(d) Clusters.

Figure 4. Rastrigin1 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 0.5945. 16 of the 20 feasible regions are identified correctly.

b) If $j = J$ then exit unsuccessfully.

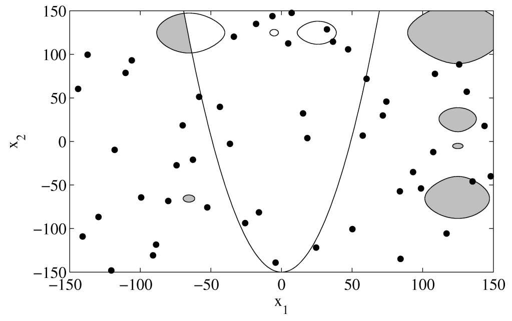
c) Reset the critical distance to $d_c = (q_j + q_{j+1})/2$. Go to step (7).

(9) Rank order the points in each cluster. Launch the local solver from the most promising point in each cluster.

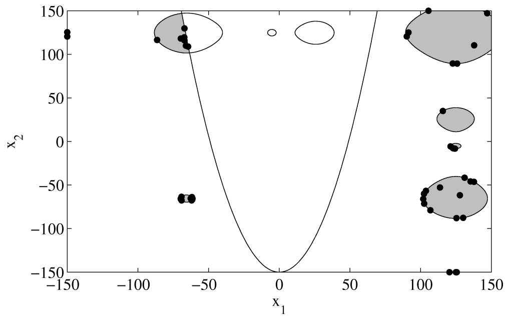
(10) Output: the best solution value found by any of the local solver launches.

5. Illustrated Experiments

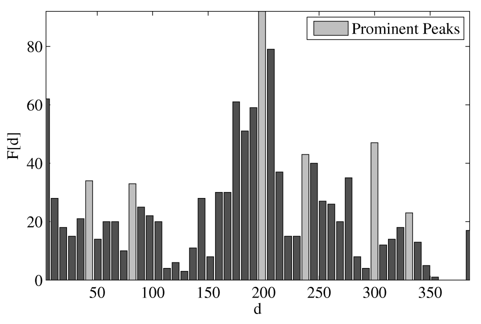
This section illustrates the method using models that have constraints based on variations of the Schwefel [18] and Rastrigin [17, 24] functions. The Latin hypercube sampling, CC concentration, and clustering steps from Section 4 are performed on each model with results as shown in Fig. 4 and Fig. 5. To focus on the performance of the concentration and clustering method, the solver launch in Step (9) from Section 4.6 is omitted. The parameter values are: $p = 50$, $\tau = 25$, and $\omega = 3$. Parameter selection is discussed in greater detail in Section 6. The augmented CC algorithm [22] is used for concentration with a maximum time limit of $0.05s$ per run.



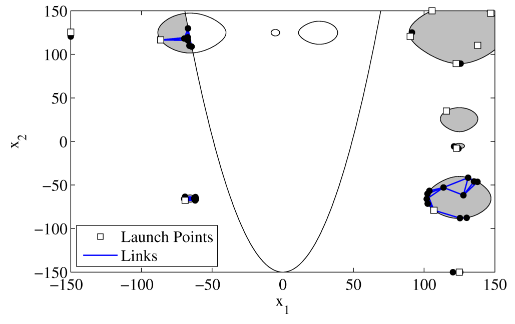
(a) Random points.



(b) Concentrated points.



(c) Inter-point frequency distributions.



(d) Clusters.

Figure 5. Schwefel1 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 21.42. All 6 of the feasible regions are identified.

5.1. *Dense Feasible Regions: The Rastrigin1 Model*

The Rastrigin1 model is as follows:

$$find \;\; \mathbf{x} = \{x_1, x_2\} \tag{9a}$$

$$s.t. \;\; g_1(\mathbf{x}) = x_1^2 + x_2^2 + 20 - 20(\cos 2\pi x_1 + \cos 2\pi x_2) \le 0 \tag{9b}$$

$$g_2(\mathbf{x}) = x_2 - x_1^3 \le 0 \tag{9c}$$

$$-5 \le x_1 \le 5 \tag{9d}$$

$$-5 \le x_2 \le 5. \tag{9e}$$

The Rastrigin1 model has many more feasible regions than the Branin1 model, and they are also closer together. This makes the task of finding all the feasible regions more difficult because they are not as widely separated and there are fewer sample points per feasible region. The Rastrigin1 model also has many infeasible regions of attraction.

The initial sample of 50 points and the concentrated data are shown in Fig. 4a and Fig. 4b, respectively. Several of the CC runs terminate near infeasible regions of attraction. Fig. 4c illustrates the inter-point distance frequency distribution and the prominent peaks. Note the smaller separation between the prominent peaks in this distribution compared to the distribution for the Branin1 model, because the clusters are closer together. The first critical distance, $d_c = 0.5945$, is accepted. Refer to Table 1 for further details.

5.2. *Irregular Feasible Regions: The Schwefel1 Model*

The Schwefel1 model is as follows:

$$\begin{aligned} find\ \ & \mathbf{x} = \{x_1, x_2\} && \text{(10a)} \\ s.t.\ \ & g_1(\mathbf{x}) = x_1 \sin(\sqrt{|x_1|}) + x_2 \sin(\sqrt{|x_2|}) + 125 \leq 0 && \text{(10b)} \\ & g_2(\mathbf{x}) = x_2 - \frac{1}{16}x_1^2 + 150 \leq 0 && \text{(10c)} \\ & -150 \leq x_1 \leq 150 && \text{(10d)} \\ & -150 \leq x_2 \leq 150. && \text{(10e)} \end{aligned}$$

This model has 6 feasible regions that vary greatly in relative size, as well as two regions of attraction near the boundaries of the search space at $(120, -150)$ and $(-150, 120)$. The distances between the feasible regions also vary.

Fig. 5b shows the concentrated points. The large feasible region centered near $(125, 125)$ attracts points to five distinct locations, resulting in five different clusters. The inter-point frequency distribution is illustrated in Fig. 5c. The prominent peak with the greatest frequency is around $d = 200$. This is the approximate length of the sides of a square with vertices located at the centers of the three largest feasible regions and the small one near $(-60, -60)$. The large frequency around $d = 200$ is due to the many clustered points located near the square's four vertices. The first critical distance of 21.42 is accepted.

5.3. *Discussion*

Table 1 summarizes the results for the Branin1, Rastrigin1, and Schwefel1 test cases. *Clusters calculated* is the number of clusters the single-linkage algorithm found in the concentrated data sets. *Feasible regions attracting exactly one cluster* is the number of feasible regions that were identified correctly by a single cluster. *Feasible regions attracting multiple clusters* is the number of feasible regions that were identified by two or more clusters. *Feasible regions attracting no clusters* is the number of feasible regions that were not identified by any clusters. Finally, *Infeasible regions attracting at least one cluster* represents the number of clusters that do not identify a feasible region.

Our method correctly identified all three of the feasible regions in the Branin1 model, each by a single cluster. It identified 19 clusters for the Rastrigin1 model: 16 of the 20 feasible regions were correctly identified by a single cluster. 3 clusters were located near infeasible regions of attraction, which is typical behavior for local NLP solution methods. The Rastrigin1 model has many infeasible regions of attraction.

All 6 of the feasible regions in the Schwefel1 model were identified, along with 2 infeasible regions of attraction. The largest feasible region was identified redundantly by 5 different clusters. A larger critical distance that might have avoided the redundant clusters may have amalgamated the cluster around the small feasible region near $(125, -10)$ into the larger clusters. This is less preferable.

The Branin1 and Rastrigin1 models illustrate easy and difficult cases, respectively. The Branin1 model has a few well separated feasible regions that the concentration and clustering methods easily identify. In contrast, the feasible regions of the Rastrigin1 model are numerous and dense because of the sinusoidal functions in the constraint. The method placed clusters near 16 of the 20 feasible regions, which is a good result for this difficult model: if used in a multistart algorithm for global optimization, 84% of the solver launches (16 of 19 identified clusters) would likely lead to different feasible solutions. The Schwefel1 model is also relatively difficult, having feasible regions that vary in size and separation distance. All 6 of the feasible regions are identified, though there would possibly be up to 4 redundant solver launches in the largest feasible region.

6. Experimental Setup

6.1. *Test Set*

The models used in the experiments are taken from libraries 1 and 2 of the COntinuous CONstraints - Updating the Technology (COCONUT) benchmark [20]. We have divided them into two sets based on the number of nonlinear constraints: Problem Set I contains the 148 models with 10 to 1000 nonlinear constraints, and Problem Set II contains the 75 models with more than 1000 nonlinear constraints. Sixteen possible models were omitted because they caused the solver software to return errors: *argauss, argtrig, bratu2d, bratu2dt, bratu3d, camcge, carenary, cbratu2d, dtoc6, ex8_3_13, ex8_3_14, ex6_6_1, grouping, oet7, pindyck, polak3*.

The models are listed in Appendix A. To eliminate bias, the two problem sets were further randomly subdivided into tuning and testing sets. The tuning subsets (denoted with an A) were used to determine the maximum time parameter for the CC runs in the algorithm, and consist of approximately 20% of the models. The testing subsets (denoted with a B) were used to test the algorithm. The statistics of the selected models are shown in Table 2.

The initial points provided with some of the models were ignored. This makes the problem set more difficult for the optimization algorithms because the provided points are often very close to optimal solutions. Additionally, unbounded variables (upper, lower, or both) were artificially bounded with the value 10^4. This is the same value Lasdon and Plummer chose to test the MSNLP solver [13].

6.2. *Hardware*

The experiments were run on a machine with an Intel®Core$^{\text{TM}}$2 Duo Processor E6600 (4M Cache, 2.40 GHz, 1066 MHz FSB) and 3GB of memory.

6.3. *Software*

The operating system was Linux Fedora Core 6. The compiler used was gcc 4.1.2. The C++ source code is available online at `https://github.com/LSmith4/LaunchPointGenerator`. All models were interpreted using A Modeling Language for Mathematical Programming (AMPL) [8].

The local solver used in our implementation is Ipopt, an open source interior-point solver for large-scale optimization [26]. Parameter settings for Ipopt were:

- *honor_original_bounds* = *yes*. Project final point back inside original bounds.
- *bound_relax_factor* = 0. Set to 0 to disable bounds relaxation.
- *max_iter* = 9999999. Set high so that maximum number of iterations is not a restriction.
- *constr_viol_tol* = 10^{-6}. This is the absolute tolerance by which a constraint can be violated and yet still be considered satisfied. In order to declare optimality, Ipopt requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

- *max_cpu_time*. For Problem Set I this is 60 seconds, and for Problem Set II it is 600 seconds.

All other parameters were left at their default settings.

6.4. *Parameter Settings*

A basic parameter for all nonlinear programming algorithms is the feasibility tolerance. Feasibility was measured as the maximum constraint violation, as shown in Eqns. 4 and 5. A feasible solution vector must not violate any constraint by more than 10^{-6}.

The parameters controlling the algorithm are (i) p, the number of initial CC start points, (ii) τ, the maximum number of local solver launches, (iii) ω, the number of preceding and following bins in the inter-point frequency distribution that is used to identify a prominent peak, (iv) the maximum time allowed for a single CC run, and (v) the maximum time allowed for a single run of the local solver.

p is 25 for Problem Set I and 50 for Problem Set II, values that work relatively well in our experience though there is no perfect setting that works well for all models. Since $\frac{1}{2}p(p-1)$ inter-point distances are calculated in determining the critical distance, we do not want p to be larger than necessary. We found that relatively small values of p work well for many models, even large models, but if the solution fails, it can always be run again with a different random seed if time is available. This is similar to determining the number of particles used for particle swarm optimization or the population size for a genetic algorithm.

τ was arbitrarily set to 25 for all algorithms. This is a reasonable choice for large NLPs. Different values for this parameter may change the relative performance of the algorithms compared in the experiments. The default setting for the number of launch points for Knitro's multistart procedure is $min\{200, 10n\}$, where n is the number of variables [27]. Given the average number of variables in the problem sets (listed in Table 2), the default setting for Knitro would run the solver 200 times for many models in Problem Set IB and all the problems in Problem Set IIB.

ω was set to 3, which worked well in our preliminary experiments.

The algorithm in Section 4.6 (with the exception of the local solver launches) was run on the tuning sets to choose the maximum time limit for a single CC run. The results are listed in Table 3. The independent variable in the experiments is the maximum time limit for a single CC run, ranging from 0.05s to 0.8s for Problem Set IA, and from 0.25s to 4s for Problem Set IIA. The table shows:

- **Median violation**: The violation (Eqns. 4 and 5) of the most promising launch point found by the clustering routine for a particular model. Median violation is a measure of how well the algorithm performed in terms of finding points close to a feasible region over the whole set of test problems.
- **Average clusters**: The average number of launch points calculated.
- **Average run time**: The average run time of the clustering method, including the time required to initially sample the space.
- **Feasible solutions**: The percentage of models for which feasible solutions were found by the clustering routine without running a local solver.

Using the tuning statistics, we set the maximum time for a single CC run to 0.05s for Problem Set I and to 1s for Problem Set II. The data in Table 3 indicate that longer times yield only relatively small and costly decreases in the median violation, and only a slight if any increase in the percentage of feasible solutions found by the CC method. The ratio of CC run time to an estimate of the run time for the local solver was also considered.

Note that there is no expectation that individual CC runs will reach feasibility. CC is only expected to find a point that is close to feasibility: the local solver is expected to proceed to feasibility and optimality. However CC finds a feasible point for a good fraction of the small models in Problem Set IA. It finds far fewer feasible solutions in Problem Set IIA because the models are on average much larger and more difficult than those in Problem Set IA.

The local solver was allowed 60s per launch for the smaller problems in set IB, and 600s for the larger problems in set IIB. The 600s time limit is likely overly restrictive for the larger models, but is needed for practical reasons to allow us to compare a number of alternative algorithms over a large number of models. Since the variations in results are mostly due to the solver launch points chosen, this restriction should not materially affect our conclusions.

The variant of CC used is Basic with Augmented feasibility vectors and a recurrence period of 3. The other CC parameters were set to the following values: $\alpha = 10^{-6}$, $\beta = 10^{-3}$, and $\mu = 100$.

6.5. *Performance Metrics*

Our experiments compare a variety of algorithms to determine which one most frequently finds the best solution the quickest given the same number of local solver launches. Note that this is not the same as imposing an upper time limit. For example, our algorithm may determine that a certain model has 3 distinct feasible regions and hence will launch the local solver 3 times and stop. In contrast, a naive multi-start algorithm will use all 25 local solver launches, and hence around 8 times as much computation time. For this reason we expect to see that our algorithm uses less time on average, especially for the larger models, and hope to see that it reaches a better solution more frequently due to more intelligent exploration of the variable space prior to launching the local solver.

The results are compared in terms of feasibility, optimality, and time, and are presented as performance profiles (see [6]). The success rate for the performance profiles in Fig. 6 and Fig. 7 is the percentage of models for which a particular algorithm found the best feasible solution, i.e., the feasible solution vector with the smallest objective function value. A tolerance of 5% is used so that solutions that are almost the same are both considered successes. For instance, if three algorithms found feasible solutions to a minimization problem with objective function values of 1.00, 1.03, and 1.74, then both 1.00 and 1.03 are considered successes, while 1.74 is not. The ratio to best time compares the relative times the algorithms took to find the best solutions. This type of performance profile measures how fast the algorithms find good solutions.

6.6. *Algorithms Compared*

Four different multistart algorithms are compared in this paper:

- **Knitro**: Knitro in multistart mode. Knitro is a state-of-the-art commercial solver that uses interior point and active-set methods for solving continuous, nonlinear optimization problems [27]. We used Knitro 6.0, which offers a multistart procedure that essentially restarts the Knitro solver from different initial points and returns the best solution found. All parameters were set at their default values, except for the following:
  - *ms_enable* = 1. Multistart is enabled.
  - *ms_maxsolves* = 25. The maximum number of solver launches in the multistart algorithm.

  - $feastol = 0.0$. The relative feasibility tolerance, set to 0 so that Knitro will only declare optimality when an absolute feasibility tolerance is satisfied.
  - $feastol\_abs = 10^{-6}$. The absolute feasibility tolerance.
  - $opttol = 0.0$. The relative optimality tolerance.
  - $opttol\_abs = 10^{-6}$. The absolute optimality tolerance.
  - $maxit = 9999999$. The maximun number of iterations per launch. Set high so that the number of launches is not a restriction.
  - $honorbnds = 1$. Honor variable bounds.
  - $outmode = 0$. Direct output to standard out.
  - $outlev = 1$. Printing output level.
  - *maxtime_cpu*. Maximum local solver CPU time is 60 seconds for Problem Set I and 600 seconds for Problem Set II.
  - *ms_maxtime_cpu*. Maximum total multistart time is 1,500 seconds for Problem Set I and 15,000 seconds for Problem Set II.
- **MS**: Basic Multistart with Ipopt. An initial random sample of 25 local solver launch points is generated by Latin Hypercube sampling. These points are then ordered by increasing value of the maximum violation, and the local solver is launched from each point in order. The local solver is Ipopt, with parameter settings as described earlier. The best solution found after the 25 local solver launches is returned.
- **MS+CC**: Multistart with Constraint Consensus and Ipopt. An initial random sample of $p = 25$ CC launch points is generated by Latin Hypercube sampling. Basic CC with Augmented feasibility vectors (with recurrence period of 3) is run from each of the initial points. The CC end points are then ordered by increasing value of the maximum violation, and the local solver is launched from each point in order. The local solver is Ipopt with parameter settings as described earlier. The best solution found after the 25 local solver launches is returned.
- **MS+C**: Multistart with clustering using Constraint Consensus and Ipopt. This is the algorithm given in Section 4.6. $p = 25$ for Problem Set I and $p = 50$ for Problem Set II. The maximum number of solver launches is $\tau = 25$. A single most promising point is identified for each cluster, and these are then ordered from most to least promising using the rules defined in Section 4.5. The local solver (Ipopt with parameter settings as described earlier) is launched from the first τ points in the list (or from the number of points in the list if less than τ).

When possible, the MS, MS+CC, and MS+C algorithms used the same initial start points, but Knitro used its own internal method to generate initial start points.

7. Numerical Experiments

The goal of this research is improved methods for multistart global optimization of large scale nonlinear programs. For this reason, the results for Problem Set II are of main interest. At the same time, the algorithm should not perform poorly on smaller models, hence the results for Problem Set I are also relevant. Results over all models are summarized in Table 4. The performance profiles in Figs. 6-8 concentrate on the results for time ratios under 10: longer times are much less preferred.

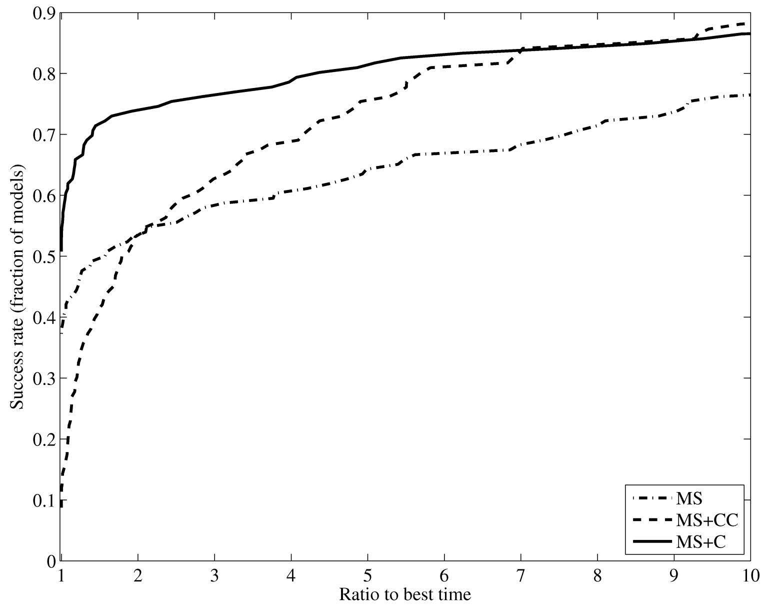


Figure 6. Performance Profile for Small Problems (Set IB)

7.1. *Results for Small Models*

For Problem Set IB, the best solutions returned by the three algorithms that used Ipopt as the local solver (MS, MS+CC, MS+C) are relatively similar in value. The performance profile in Fig. 6 shows that MS+C dominates all of the other methods over most time ratios. MS+C finds the best solution the fastest for 51% of the models and is also relatively robust, finding the best solution for 88% of the models.

Table 4 shows that both MS+CC and MS+C result in more feasible and best solutions than MS, which demonstrates the value of using CC prior to launching the local solver.

MS+C is the fastest on average. MS+CC is the most robust, successfully solving 93% of the models.

7.2. *Results for Large Models*

The large models in Problem Set IIB are much more difficult to solve because they are larger in scale and have many more nonlinear constraints. As for the small models, the objective values returned by the three algorithms are relatively similar. The performance profile shown in Fig. 7 shows that the MS+C algorithm dominates the other methods over all time ratios under 10. It finds the best solution the fastest for 37% of the models, and is the most robust, finding the best solution for 56% of the models tested.

A much smaller fraction of the large models are solved to optimality than for the small models, mostly because of the time restriction on the local solvers, as discussed earlier. Not being able to find a feasible point in the given maximum run time was the biggest challenge. Convergence of the local solver was also an issue. There are approximately 30 models in the large set for which finding feasible points is extremely difficult (perhaps impossible).

Table 4 shows that the numbers of feasible and best solutions for MS+C are slightly better for MS+CC and MS, but are obtained in much less time. MS+C uses 41% less time than MS+CC, its nearest competitor, while returning 4 additional

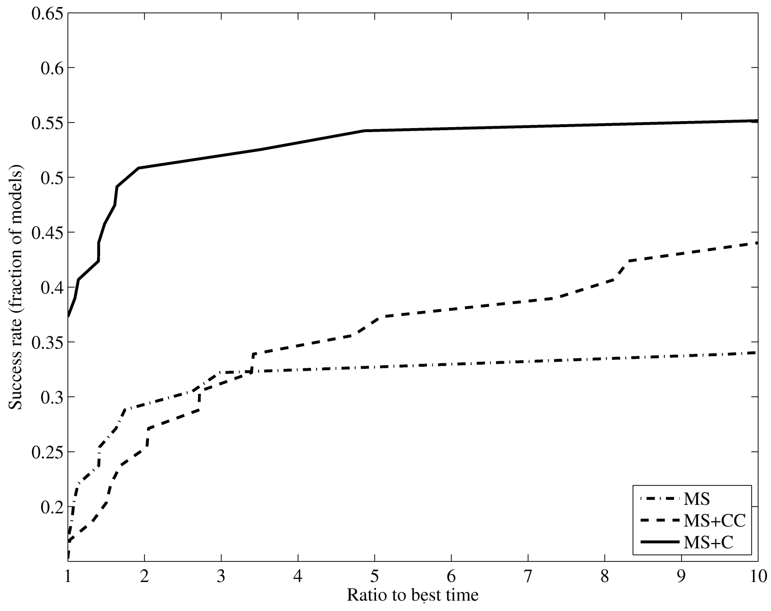


Figure 7. Performance Profile for Large Problems (Set IIB)

best solutions. MS+C is the fastest and most successful for large models.

Statistics over the complete set of 75 large models in Problem Set II were also collected. Collectively, the three (Ipopt-based) algorithms found feasible solutions for 46 of those models (61%). For those 46 successful models:

- The initial incumbent solution found by MS+CC was successful for 52% of the 46 models, while the initial incumbent found by MS+C was successful for 63%. The better exploration afforded by 50 vs. 25 CC runs pays off.
- The average time to the initial incumbent for MS+CC is 228s vs. 335s for MS+C. The extra initial exploration takes some time.
- MS+CC launched the local solver an average of 25 times, while MS+C launched the local solver an average of just 9 times. This demonstrates the advantage of the clustering method in eliminating redundant solver launches.
- The average times used by each individual local solver launch were close to identical for MS+CC and MS+C at 180s, indicating that both methods put the solver launch points at approximately the same distances from the local optimum point. In addition, the local solver failed an average of 7.2 times per model for MS+CC vs. an average of 2.6 times per model for MS+C, which is about the same fraction of failures per launch. These facts again underscore that the MS+C advantage derives mainly from eliminating redundant local solver launches.

7.3. *Comparison to Knitro*

Knitro was run on Problem Set IIB so that our methods could be compared directly to a complete commercial multistart algorithm. The parameters for Knitro were set so that the stopping conditions of the local solver were as similar as possible to those used by Ipopt and hence a fair comparison can be made. Our intention is not to compare the local solvers, but the multistart methods they use for global optimization (comparisons of Knitro and Ipopt as local solvers are found at [16] and in Section 5.1.3 of [25]).

Initial results data (not included) indicate that Knitro outperforms MS over Problem Set IIB. Both algorithms found feasible solutions to 35 (59%) of the

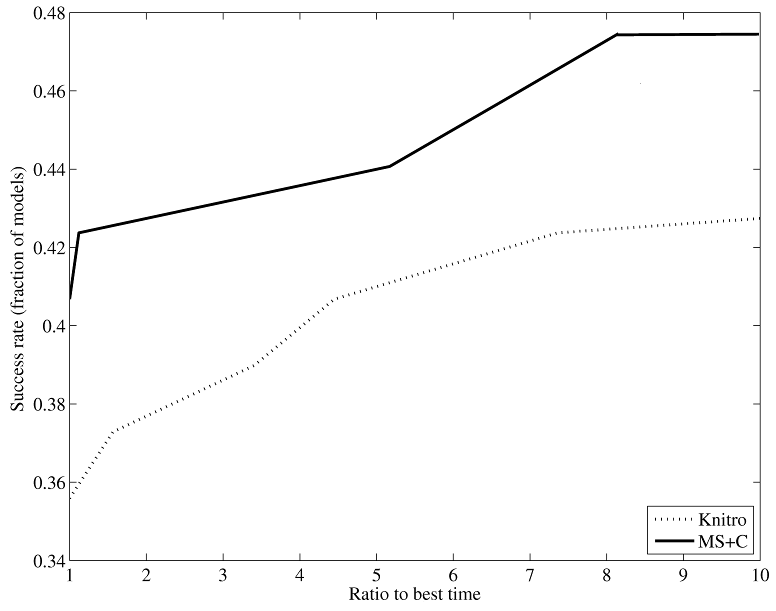


Figure 8. Performance Profile Including Knitro for Large Problems (Set IIB)

models. Knitro found the best objective function value for 31 of the models (vs. 24 for MS). Knitro is faster at finding the best solutions, however the total times indicate that Knitro is also slower when it does not find a solution. The average time per model for Knitro was 8859.9s compared to 6867.5s for MS. Overall, Knitro outperformed MS on Problem Set IIB.

Fig. 8 compares the Knitro and MS+C algorithms. The MS+C algorithm outperforms Knitro for all ratios to best time up to 10. Knitro found feasible solutions for 35 models (vs. 38 for MS+C), and the best solution for 31 models (vs. 28 for MS+C). The biggest difference between the algorithms is the average time used per model: MS+C used only 3737.95s, 58% less time than Knitro.

These results show that the MS+C algorithm compares well against a state of the art commercial multistart method. Solution times are much smaller, and feasible solutions are found for more models. MS+C found slightly fewer best solutions: this is the focus of ongoing research on how to use CC to move towards optimality as well as feasibility.

7.4. *Discussion*

An inherent problem with multistart methods for global optimization is knowing when to stop launching the local solver: it's always possible that a better solution will be returned by the next local solver launch. A major advantage of the MS+C algorithm is that it will stop when it thinks that the local solver has been launched once near each feasible region. That can also be a disadvantage since it is possible that a local solver launch from some other part of the same feasible region will produce a better solution, especially if the feasible region is quite large. However, the performance profiles in Fig. 6 and Fig. 7 show that MS+C is both faster and more robust than the other algorithms examined. The variable space exploration conducted by the MS+C method is clearly effective in producing recommendations for a small and efficient set of local solver launch points.

Table 3 shows that the concentration and clustering method identified an average of 14-16 clusters for the small models in tuning set IA and 10-14 clusters for large models in tuning set IIA. Assuming that testing sets IB and IIB are similar to their corresponding tuning sets, this shows that MS+C is likely to make far fewer than 25 solver launches on average, which accounts for much of its speed advantage. In fact, the MS+C algorithm averaged 14.1 and 11.9 solver launches for problem sets IB and IIB, respectively, while the MS, and MS+CC algorithms had no option but to use the full 25 solver launches for each problem. The relatively large numbers of clusters, on average, for both large and small models highlights the opportunity for efficiency improvement by avoiding redundant solver launches near the same regions of attraction. This is exactly the opportunity that is exploited by MS+C.

As shown in Table 4, MS+C does not do quite as well as MS+CC in terms of the number of feasible solutions and best solutions for the small models in Problem Set IB. This is not unexpected, given our experimental setup for Problem Set IB which specifies $p = \tau = 25$. This allows MS+CC to launch the local solver from all 25 points, where MS+C launches from only a subset of the same 25 points (i.e. only the most promising point for each cluster found). In a few cases, the most promising launch point for a cluster is not actually the best launch point for that cluster. However, the results also show that it frequently is the best choice. For better exploration of the variable space prior to a local solver launch we would normally set p to a much larger value than τ. The improved effect when $p = 50$ and $\tau = 25$ is seen in the results for the large models in Problem Set IIB.

The importance of using CC to improve the initial point prior to the local solver launch is demonstrated in Table 4: the MS+CC and MS+C algorithms both find the best solution more often and faster than the simple MS variant. CC comprises just 1.4% of the total solution time for MS+CC in Problem Set IB, and 0.4% of the total solution time in Problem Set IIB. Similarly, the CC plus clustering time comprises only a small fraction of the total solution time for the MS+C algorithm: just 1.6% for Problem Set IB and 1.5% for Problem Set IIB. These relatively small investments in exploration of the variable space prior to launching the local solver pay big dividends in terms of reduced total solution time and improved solution success.

Our major conclusion relates to the results for the large models in Problem Set IIB that are of main interest. As shown in Table 4, MS+C is not only the fastest by a significant margin, using 41% less time than the next fastest method on average, but it also solves the most models successfully. The performance profile in Fig. 8 shows that MS+C is a promising alternative to the Knitro multistart algorithm, which is representative of the state of current commercial solvers available. We thus conclude that the algorithm developed in this paper provides valuable improvements in both the effectiveness and efficiency of multistart methods for global optimization.

A variety of additional experiments further supporting these conclusions are reported by Smith [23].

8. Conclusions and Future Research

This paper presents a method for efficiently and effectively computing local solver launch points that are typically near all of the feasible regions in a constrained NLP, even when there are multiple feasible regions. This is done quickly and inexpensively by using a combination of Constraint Consensus and clustering, where an innovative way of determining the critical distance for the clustering algorithm is developed. The expensive local solver is then launched just once for each cluster to avoid redundant launches.

Extensive empirical results show that the MS+C algorithm typically finds the best reported optimum in significantly less time than competing algorithms. If further experiments were run with a total runtime limit imposed, this would translate into greater relative success for the MS+C algorithm.

Future research will develop the algorithm further. Some areas for exploration include:

- Detecting the size of feasible regions. This will help to determine whether more than one launch point should be used for each feasible region.
- Determining whether there are multiple optima within a single feasible region. This would again call for more than one local solver launch in a single feasible region.
- Search within clusters. Progress on the two preceding points will support work on how to quickly search within clusters to determine whether we can (i) find a point that is better than the most promising CC end point, or (ii) identify multiple very promising local solver launch points that are likely to lead to different solutions.
- Detecting infeasible regions of attraction. It would be helpful if it can be determined that a particular cluster is unlikely to provide a feasible solution if the local solver is launched. This will increase efficiency by allowing the local solver launch to be avoided.
- Dealing with a unimodal inter-point distance frequency distribution. This may be caused by a single feasible region, or it could have some other cause. It is difficult to set the critical distance in this case.
- Setting the parameter values. Although 25-50 sample points and 25 local solver launches worked relatively well for the problem sets examined in the paper, they may not work well for all models. Algorithms for automatic adjustment of these and other parameters could be very useful.

Two promising ideas for extensions to the existing algorithm include:

- Adding an aspiration level constraint to the model and updating its value as better incumbent solutions are found. An aspiration constraint specifies a minimum acceptable value of the objective function. This will help direct the CC algorithm towards feasible solutions that have better values of the objective function.
- Using the start, intermediate, and end points of the CC runs to determine the basins of attraction for identified clusters so that new random CC start points can be placed in unexplored regions of the variable space. This will provide better coverage of the search space.

References

[1] S. Brin, *Near Neighbor Search in Large Metric Spaces*, In Proc. 21st Conference on Very Large Databases (VLDB95), pp. 574-584, 1995.

[2] R.H. Byrd, J. Nocedal, and R.A. Waltz, *Knitro: An Integrated Package for Nonlinear Optimization*, in Large-Scale nonlinear Optimization, G. di Pillo and M. Roma, (eds.), Springer-Verlag, pp. 35-59, 2006.

[3] J.W. Chinneck, *The Constraint Consensus Method for Finding Approximately Feasible Points in Nonlinear Programs*, INFORMS Journal on Computing, 16 (3), pp. 255-265, 2004.

[4] J.W. Chinneck, *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, Vol. 118, International Series in Operations Research and Management Sciences, ISBn 978-0387749310, Springer.

[5] P. Chootinan and A. Chen, *Constraint Handling in Genetic Algorithms Using a Gradient-based Repair Method*, Computers and Operations Research, 33 (8), pp. 2263-2281, 2006.

[6] E.D. Dolan and J.J. More, *Benchmarking Optimization Software with Performance Profiles*, Mathematical Programming 91, (2002), 201–213.

[7] O.A. Elwakeil and J.S. Arora, *Methods for Finding Feasible Points in Constrained Optimization*, AIAA Journal, 33 (9), pp. 1715-1719, 1995.

[8] Fourer R, Gay DMM, Kernighan BW, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, 2002.

[9] M. Gertz, J. Nocedal, and A. Sartenaer, *A Starting-Point Strategy for Nonlinear Interior Methods*, Report OTC 2003/04, Optimization Technology Center, northwestern University, May 2003.

[10] W. Ibrahim and J.W. Chinneck, *Improving Solver Success in Reaching Feasibility for Sets of Nonlinear Constraints*, Computers and Operations Research, 35 (5), pp. 1394-1411, 2008.

[11] C. Jansson and O. Knueppel, *A Global Minimization Method: the Multi-Dimensional Case*, Technische Informatik III, TU Hamburg-Hamburg, p. 35 (problem "BR"), Jan. 1992.

[12] R. B. Kearfott, *Globsol: History, Composition, and Advice on Use*, In Global Optimization and Constraint Satisfaction, Lecture notes in Computer Science, Springer-Verlag, pp. 17-31, 2003.

[13] L. Lasdon and J. Plummer, *Multistart Algorithms for Seeking Feasibility*, Computers and Operations Research, 35 (5), pp. 1379-1393, May 2008.

[14] M. MacLeod, *Multistart Constraint Consensus for Seeking Feasibility in Nonlinear Programs*, MASc Thesis, Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2006.

[15] M.D. McKay, R.J. Beckman, and W.J. Conover, *A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code*, Technometrics, 42 (1), pp. 55-61, 2000.

[16] H. Mittelmann, *Benchmarks for Optimization Software: AMPL-NLP Benchmark*, Accessed Online: October 29, 2010. `http://plato.asu.edu/ftp/ampl-nlp.html`

[17] H. Mühlenbein, D. Schomisch and J. Born, *The Parallel Genetic Algorithm as Function Optimizer*, Parallel Computing, 17 (6-7), pp. 619-632, 1991.

[18] H.P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, Inc., 1981.

[19] J. H. Sendin, J. R. Banga, and T. Csendes, *Extensions of a Multistart Clustering Algorithm for Constrained Global Optimization Problems*, Industrial & Engineering Chemistry Research, 48 (6), pp. 3014-3023, 2009.

[20] O. Shcherbina, A. Neumaier, D. Sam-Haroud, X. Vum, and T. Nguyen, *Benchmarking Global Optimization and Constraint Satisfaction Codes*, pp.211–222 in: C.H. Bliek, C.H. Jermann and A. Neumaier (eds.), Global Optimization and Constraint Satisfaction, Lecture Notes in Computer Science, 2861, Springer-Berlin, 2003.

[21] R. Sibson, *SLInK: An Optimally Efficient Algorithm for the Single-link Cluster Method*, The Computer Journal, 16 (1), pp. 30-34, 1973.

[22] L.R. Smith, J.W. Chinneck, and V.C. Aitken, *Augmented and Quadratic Feasibility Vectors for Constraint Consensus*, Technical Report, Systems and Computer Engineering Department, Carleton University, Ottawa, Ontario, Canada, July 2010.

[23] L.R. Smith, *Improved Placement of Local Solver Launch Points for Large-scale Global Optimization*, PhD. Thesis, Systems and Computer Engineering Department, Carleton University, Ottawa, Ontario, Canada, April 2011.

[24] A. Törn and A. Zilinskas, *Global Optimization*, Lecture notes in Computer Science, 350, Springer-Verlag, 1989.

[25] A. Wachter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*, Phd Thesis, Carnegie Mellon University, 2002.

[26] A. Wachter and L.T. Biegler, *On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming*, Mathematical Programming, 106 (1), pp. 25-57, 2006

[27] R.A. Waltz and T.D. Plantenga, *KNITRO User's Manual: Version 6.0*, Zienna Optimization, Inc., March 2009.

Table 1. Clustering Results

Model:	Branin1	Rastrigin1	Schwefel1
d_{min}:	0	0	0
d_{max}:	17.1	9.15	389
d_{width}:	0.341	0.183	7.79
Peaks found:	4	7	6
Critical distances tried:	1	1	1
Critical distance:	3.16	0.595	21.4
Clusters calculated:	3	19	12
Feasible regions attracting exactly one cluster:	3	16	5
Feasible regions attracting multiple clusters:	0	0	1
Feasible regions attracting no clusters:	0	4	0
Infeasible regions attracting at least one cluster:	0	3	2
Actual number of feasible regions:	3	20	6

Table 2. Model Statistics

Problem Set (# Models)	IA (22)			IB (126)		
	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	259.0	3	2505	536.5	2	8997
Nonlinear Constraints	120.3	10	800	232.9	10	1000
Total Constraints	216.1	10	2495	481.1	10	7000
Nonzeros in Jacobian	1980.8	30	11425	3067.4	39	36185
Problem Set (# Models)	**IIA (16)**			**IIB (59)**		
	Avg.	Min.	Max.	Avg.	Min.	Max.
Variables	6055.7	200	11215	6282.5	3	20008
Nonlinear Constraints	4323.5	1024	10000	3897.5	1000	13798
Total Constraints	5083.3	1024	11192	5615.5	1000	14000
Nonzeros in Jacobian	33001.3	8000	128004	31567.8	2998	128004

Table 3. Clustering Times

Tuning Problem Set IA (22 models)					
Max CC Time (s)	0.05	0.1	0.2	0.4	0.8
Median Violation	15.15	1.07	0.14	0.15	0.11
Avg. Clusters	16.0	16.0	14.6	15.3	14.5
Avg. Run Time (s)	1.30	2.36	4.42	8.52	16.59
Feasible Solutions (%)	40.9	40.9	45.5	45.5	45.5

Tuning Problem Set IIA (16 models)					
Max CC Time (s)	0.25	0.5	1	2	4
Median Violation	4260.0	1310.0	912.8	455.9	345.5
Avg. Clusters	10.3	14.1	12.5	11.1	11.4
Avg. Run Time (s)	26.7	31.8	52.9	100.8	188.0
Feasible Solutions (%)	0.0	6.3	12.5	12.5	12.5

Table 4. Test Statistics

Problem Set IB (126 models)

Algorithm	MS	MS+CC	MS+C
Feasible Solutions	113	122	120
Best Solutions	105	118	111
Avg. Time (s)	115.6	106.6	94.4

Problem Set IIB (59 models)

Algorithm	MS	MS+CC	MS+C
Feasible Solutions	35	38	38
Best Solutions	28	29	33
Avg. Time (s)	6867.5	6385.4	3737.9

Table 5. Models (Testing subsets IB and IIB shown in bold font.)

Problem Set I				
airport	**dtoc1nd**	**ex8_3_2**	**hs085**	**orthrega**
britgas	**eg3**	**ex8_3_3**	**hs099**	**orthrege**
camshape100	**eigena2**	**ex8_3_4**	**hs108**	**otpop**
camshape200	**eigenaco**	**ex8_3_5**	**hs116**	pinene25
camshape400	**eigenb2**	**ex8_3_6**	**hs99exp**	**pinene50**
camshape800	eigenbco	**ex8_3_7**	**hvycrash**	**polygon25**
catena	**eigenc2**	**ex8_3_8**	integreq	**popdynm25**
catmix100	**eigencco**	ex8_3_9	**kissing**	**popdynm50**
catmix200	**eigmaxa**	**ex8_4_1**	**korcge**	**ramsey**
catmix400	**eigmaxb**	ex8_4_2	**lakes**	**reading3**
chakra	**eigmaxc**	ex8_4_3	launch	**robot100**
chandheq	**eigmina**	**ex8_4_4**	**lnts100**	**robot50**
chemrctb	**eigminb**	**ex8_4_5**	**lnts200**	**rocket100**
chenery	**eigminc**	**ex8_4_7**	**lnts50**	**s332**
circle	elec100	**ex8_4_8**	**madsschj**	**sawpath**
clnlbeam	**elec200**	**flowchan100_ed**	**makela3**	**semicon1**
core1	**elec25**	**flowchan100**	**manne**	**semicon2**
core2	**elec50**	**flowchan200**	**methanol100**	**sinrosnb**
corkscrw	**ex14_1_6**	**flowchan50**	**methanol50**	**smmpsf**
coshfun	**ex14_1_7**	**gasoil100**	**minc44**	ssebnln
cresc100	**ex5_2_5**	**gasoil50**	minmaxbd	**ssnlbeam**
cresc50	**ex5_3_3**	**glider50**	**mistake**	**swopf**
disc2	**ex7_2_1**	**gpp**	**optcdeg2**	**twirism1**
discs	**ex7_3_5**	**hadamard**	**optcdeg3**	**vanderm1**
dittert	**ex7_3_6**	**haifam**	**optcntrl**	**vanderm2**
dixchlnv	ex8_2_1	**haldmads**	**optctrl3**	**vanderm3**
dnieper	**ex8_3_10**	hanging	**optctrl6**	water
dtoc1na	**ex8_3_11**	**hatfldg**	**optmass**	**zigzag**
dtoc1nb	ex8_3_12	**himmel16**	optprloc	
dtoc1nc	**ex8_3_1**	**himmelbk**	**orthrds2**	
Problem Set II				
artif	**broydn3d**	ex8_2_5	**nuffield**	popdynm200
bdvalue	**broydnbd**	**flowchan400**	**nuffield2**	**porous1**
brainpc0	**catmix800**	**gasoil200**	**nuffield2_trap**	porous2
brainpc1	**cbratu3d**	**gasoil400**	**oet2**	reading1
brainpc2	**chemrcta**	**gausselm**	**orthrdm2**	**robot200**
brainpc3	**corkscrw**	**glider100**	orthregc	robot400
brainpc4	**cresc132**	**glider200**	**orthregd**	**rocket200**
brainpc5	**drcav1lq**	**glider400**	**orthrgdm**	**rocket400**
brainpc6	drcav2lq	lnts400	orthrgds	**semicon1**
brainpc7	**drcav3lq**	**methanol200**	**pinene100**	**semicon2**
brainpc8	**dtoc2**	**methanol400**	**pinene200**	**sreadin3**
brainpc9	**dtoc4**	**minperm**	polygon100	svanberg
bratu2d	**dtoc5**	msqrta	**polygon50**	**trainf**
bratu2dt	dtoc6	**msqrtb**	**polygon75**	**trainh**
bratu3d	**ex8_2_2**	**ngone**	popdynm100	**ubh5**

List of figure captions:

(1) Basin visualization via CC for the Branin1 model: (a) CC start-end pairs for 50 random start points. (b) CC start-end pairs for 1500 random start points.
(2) Concentration and clustering: (a) Fifty initial points chosen by Latin hypercube sampling. The gray areas represent feasible regions for the Branin1 model. (b) The initial inter-point distance frequency distribution. (c) The CC end points are concentrated near the feasible regions. (d) The frequency distribution of the inter-point distances for the concentrated points. The prominent peaks are calculated and used to determine the critical distances. (e) The concentrated final points are clustered using a single-linkage method. Three clusters are discovered. (f) The most promising points from each cluster are used as launch points for a local solver.
(3) Finding prominent peaks. Only one prominent peak exists at $d_4 = 11$ when $\omega = 2$. The gray region highlights the bins involved in the prominent peak calculation for the bin centered at $d_4 = 11$. There are prominent peaks at $d_2 = 7$ and $d_4 = 11$ when $\omega = 1$. In this example $p = 7$, $d_{min} = 2$, $d_{max} = 16$, and $d_{width} = 2$.
(4) Rastrigin1 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 0.5945. 16 of the 20 feasible regions are identified correctly.
(5) Schwefel1 Case Study: (a) The initial locations of the 50 sample points. (b) The new locations of the 50 points after concentration with CC. (c) Frequency distribution of the inter-point distances after concentration. (d) Links connect points that are closer than the critical distance of 21.42. All 6 of the feasible regions are identified.
(6) Performance Profile for Small Problems (Set IB).
(7) Performance Profile for Large Problems (Set IIB).
(8) Performance Profile Including Knitro for Large Problems (Set IIB).

Appendix A. Models

Table 5 lists the names of all the COCONUT models used in the numerical experiments.