Improved Filters and Randomized Drivers for Multi-start Global Optimization

Leon Lasdon
McCombs School of Business
The University of Texas at Austin
e-mail: lasdon@mail.utexas.edu

John Plummer
McCoy College of Business Administration
Texas State University

Zsolt Ugray
Business Information Systems Department
UTA State University

Michael Bussieck
GAMS Development Corporation

This paper also can be downloaded without charge from the Social Science Research Network Electronic Paper Collection: http://ssrn.com/abstract/886742
Improved Filters and Randomized Drivers
for Multi-start Global Optimization

Leon Lasdon
The University of Texas at Austin
McCombs School of Business
Management Science and Information Systems Dept.
1 University Station Stop B6500
Austin, TX 78712-0212
lasdon@mail.utexas.edu

John C. Plummer
Dept of CIS/QMST
McCoy College of Business Administration
Texas State University
San Marcos, TX

Zsolt Ugray
Business Information Systems Department
Utah State University
3515 Old Main Hill
Logan, UT 84322-3515

Michael Bussieck
GAMS Development Corporation
1217 Potomac Street, NW
Washington, D.C. 20007

Revised October 13, 2004

Key words: Multi-start, Search Methods, Heuristic, Computational Tests
Abstract. We describe and test 2 types of improvements to the multistart heuristic framework for smooth global optimization presented in [9]. This starts a gradient based local NLP solver from a filtered subset of points generated by a starting point generator or “driver,” and has been available as both a standalone and GAMS solver since 2003. Changes to the distance and merit filters make them theoretically sounder and more adaptive to the problem instance. Stochastic drivers are proposed whose performance is comparable to the OptQuest scatter search implementation, if reasonably tight bounds on all variables are imposed. Tests on two large and diverse sets of test problems (56 problems with over 100 variables, and 18 with over 1000) show that best known solutions to almost all problems can be obtained with a median of about 30 solver calls. The best solution is located in the first ten calls in about 93% of the problems, making the procedure attractive within a branch and bound framework. Further, allowing more time and iterations usually yields improved solutions to problems not solved with default limits.
1. Introduction

In an earlier paper [9], we described a multistart approach to smooth global optimization called OQNLP (OptQuest/NLP), which uses the OptQuest Scatter Search implementation [6] to create candidate starting points for a gradient-based local NLP solver. Stage 1 of the procedure performs n1 iterations of the search method, and chooses the best of these (as measured by an exact penalty function) as the initial solver starting point in stage 2. In the second stage, candidate starting points generated by the search method are filtered, and the solver is started at those points passing the filter tests. Promising computational results are presented using 155 of the test problems described by Floudas, et al. in [5].

This paper describes improvements and extensions to this algorithm, and demonstrates their value through experiments, both on the Floudas test set and on a much larger set of 339 problems, taken from the GAMS [1] globallib [www.gamsworld.org/global/globallib.htm]. Details on the sets of selected problems are given in sections 4 and 6. The new algorithm, called MSNLP (Multi- Start NLP), contains enhancements to the filter logic, and we show that these lead to a substantial improvement in MSNLP’s ability to obtain a global optimum, with some increase in the number of solver calls. Algorithm performance also depends strongly on the starting point generator or “driver,” and we describe and test two randomized drivers, one creating uniformly distributed points (“pure” random, or PR), the second using an initial coarse search to define a promising region within which random starting points are concentrated (“smart” random or SR). We discuss two SR variants, one using univariate normal distributions to generate these starting points, the other using triangular distributions.

Stochastic search procedures play a prominent role in global optimization. While the basic stochastic drivers we explore here probably take longer on average to find good solutions than the sophisticated scatter search strategies within OptQuest, they do converge in probability to a global solution under quite general smoothness assumptions, while no similar property has been demonstrated for scatter search. The computational results of Section 6 show that all drivers are about equally effective on a set of 339 test problems, if suitably tight bounds are imposed on all variables. Without these, OptQuest has an advantage because it is best at restricting its search to promising regions.
2. Problem Statement

This paper focuses on problems with continuous variables only, so we assume that there are no discrete variables in what follows. Then the problems to be solved have the form:

\[
\text{minimize } f(x) \tag{1}
\]

subject to the general constraints

\[
l \leq G(x) \leq u \tag{2}
\]

and the bound constraints

\[
x \in S \tag{3}
\]

where \(x\) is an \(n\)-dimensional vector of continuous decision variables, \(G\) is an \(m\)-dimensional vector of constraint functions, and the vectors \(u\) and \(l\) contain upper and lower bounds for these functions. The set \(S\) is defined by simple bounds on \(x\), and we assume that it is closed and bounded, i.e., that each component of \(x\) has a finite upper and lower bound. This is required by the starting point generators. The objective function \(f\) and the constraint functions \(G\) are assumed to have continuous first partial derivatives at all points in \(S\). This is necessary so that a gradient-based local NLP solver can be applied to the NLP (1) - (3). We also assume the existence of a vector of optimal Lagrange multipliers at each local minimum of (1)-(3).

The \(L1\) exact penalty function is used as a merit function for evaluating candidate starting points. For the problem (1) - (3) this function is

\[
P(x, w) = f(x) + \sum_{i=1}^{m} w_i \text{viol}(g_i(x)) \tag{4}
\]

where the \(w_i\) are positive penalty weights, \(g_i(x)\) is the \(i\)th component of \(G(x)\), and the function \(\text{viol}(g_i(x))\) equals the absolute violation of the \(i\)th constraint of (2) at the point \(x\).

It is well known ( [8]) that if \(x^*\) is a local optimum of (1) - (3), \(u^*\) is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at \((x^*, u^*)\), and

\[
w_i > \text{abs}(u_i^*) \tag{5}
\]

then \(x^*\) is a local unconstrained minimum of \(P\).
3. The MSNLP Multistart Algorithm

Let NEXT_CANDIDATE(\(x_t\)) denote the starting point generator or driver, where \(x_t\) is the candidate starting point produced by that procedure and \(x_t(i)\) refers to the \(i^{th}\) candidate point generated. We refer to the local NLP solver as \(L(xs,xf)\), where \(xs\) is the starting point and \(xf\) the final point. Pseudo-code for the algorithm, ignoring initializations, scalar input arguments, and other details, is given below. In it, the function UPDATE LOCALS(\(xs,xf,w\)) processes and stores solver output \(xf\), and produces updated penalty weights, \(w\).

After an initial call to \(L\) at the user-provided initial point, \(x_0\), stage 1 of the algorithm performs \(n_1\) iterations in which NEXT_CANDIDATE(\(x_t\)) is called, and the L1 exact penalty value \(P(x_t,w)\) is calculated. The point with the smallest of these \(P\) values, denoted \(x_t^*\) below, is chosen as the starting point for the next call to \(L\), which begins stage 2. In this stage, \(n_2\) iterations are performed in which candidate starting points are generated and \(L\) is started at any point which passes the distance and merit filter tests (detailed in section 3.3).

3.1 MSNLP Algorithm Pseudo-code

STAGE 1

\(x_0 = \) user initial point

Call \(L(x_0,xf)\) // attempt to generate initial local solution at user initial point

Call UPDATE LOCALS(\(x_0,xf,w\))

FOR i = 1, \(n_1\) DO // generate Stage 1 set of candidate points

\(x_t(i) = \) NEXT_CANDIDATE()

Evaluate \(P(x_t(i),w)\)

ENDDO

\(x_t^* = \arg\min_{1\leq i\leq n_1} P(x_t(i),w)\) // select candidate point with best penalty function value

call \(L(x_t^*,xf)\) // call local solver at best stage 1 point

Call UPDATE LOCALS(\(x_t^*,xf,w\))

threshold = \(P(x_t^*,w)\) // initialize merit filter threshold

STAGE 2
FOR i = 1, n2 DO  // iterate for n2 candidate points
    xt(n1+i) = NEXT_CANDIDATE()
    Evaluate P(xt(n1+i),w)
    // Perform merit and distance filter tests:
    dstatus = distance filter(xt(n1+i))  // result is accept or reject candidate point
    mstatus = merit filter(xt(n1+i), threshold)  // result is accept or reject
    IF (dstatus and mstatus = “accept”) THEN
        Call L(xt(n1+i),xf)
        Call UPDATE LOCALS(xt(n1+i),xf, w)
    ENDIF
ENDDO

In the logic above, points returned by the solver are inserted in the list of locals even if the point does not satisfy the Kuhn-Tucker conditions to within the solvers tolerances. Such points may still have the best objective value found. The penalty weight update insures that $w_i$ is always larger than the largest absolute multiplier for constraint i over all local optima.

3.2 Logic for UPDATE_LOCALS and Basin Overlap Exclusion

Pseudo-code for the UPDATE_LOCALS routine is given below. In it, $r(i)$ is the current radius of the spherical approximation to the basin of attraction of the ith local solution and $lbnd$ is a small positive lower bound, imposed to insure that all penalty weights are positive. The symbol $u^*_i$ in the update formula for $w_i$ is component i of the optimal Lagrange multiplier vector $u^*$ associated with the newly discovered local solution $xf$.

The “basin overlap exclusion,” not present in the previous OQNLP algorithm, insures that the following inequality holds for all distinct (i,j) pairs:

$$r(i) + r(j) \leq d(i, j)$$  (6)

so the spherical approximations to the attraction basins do not overlap. If this inequality is violated, $r(i)$ and $r(j)$ are multiplied by the factor $f$ which causes them to satisfy it as an equality. The outer loop eliminates overlap between the ith sphere and all others. The new radii may be further reduced as the outer loop proceeds. The final set of spheres will have no overlap, and some pairs of previously overlapping spheres will just touch.
The resetting of the counters drejctr(i) and drejctr(j) to 0 is a subtle interaction between this logic and that of the dynamic distance filter explained below. Drejctr(i) counts the number of times the distance filter for local optimum i rejects trial points. In that logic, when it reaches a threshold, the radius \( r(i) \) is reduced, and drejctr(i) is reset to 0. It is reset to zero here whenever \( r(i) \) is reduced, because its previous value corresponded to the previous \( r(i) \) value, and the current value has no rejection history.

**UPDATE LOCALS** \((xs,xf,w)\)

IF \( xf \) is feasible THEN

\[
d = \text{distance}(xs,xf)
\]

Check \( xf \) against all previously found locals and, if new, store \( xf \) in list of local solutions

IF \( xf \) is a new local

basin radius(\( xf \)) = \( d \)

ELSE

basin radius(\( xf \)) = \( \max(d, \text{previous radius}) \)

ENDIF

nlocals = number of distinct locals found thus far

\[
w_i \leftarrow \max(w_i, lbnd, 2 \times \text{abs}(u_i^*)) \quad // \text{Update penalty weights:}
\]

//Begin basin overlap exclusion:

FOR \( i = 1, \text{nlocals} \) DO

\( xl(i) = \text{ith local solution} \)

FOR \( j = 1, \text{nlocals}, j \neq i, \text{DO} \)

\[
d(i,j) = \text{distance}(xl(i),xl(j))
\]

IF \( d(i,j) < r(i)+r(j) \) THEN

\[
f = d(i,j)/(r(i)+r(j))
\]

\[
r(i) \leftarrow f \times r(i)
\]

\[
r(j) \leftarrow f \times r(j)
\]

drejctr(i) = 0

drejctr(j) = 0

ENDIF

ENDDO
3.3 Logic for the distance and merit filters

The MSNLP algorithm owes much of its efficiency to its merit and distance filters, whose effect is to start $L$ at a small fraction of the candidate starting points, while still finding the global solution to most problems.

3.3.1 Distance Filter

The distance filter rejects the input point $x_t$ if the distance from $x_t$ to any local solution $x_l(i)$ is smaller than a user supplied reduction factor $df_{user}<1$ (default value 0.8) times its basin radius, $r(i)$. The dynamic distance filter logic, not present in the algorithm described in [Ugray, Lasdon, et al., to appear], reduces $r(i)$ by a basin-specific factor $df(i) < 1$ each time $d_{wait}$ (default value 20) trial points (not necessarily consecutive) have been rejected by it. The reduction factor $df(i)$ is calculated dynamically each time a point is rejected, and the dynamic reduction factor for $r(i)$ is stored and updated as $d_{temp}(i)$, a quantity which is initialized to zero when the $i$th local solution is found. The expression $dist(i)/r(i)$ is the value of the reduction factor which would cause the filter test to place the rejected point on the borderline of acceptance. Thus the radius is reduced to a value which would place the farthest of the past waitcycle rejected points on the threshold of being accepted. The (conservative) lower bound of 0.5 insures that $d_{temp}(i)$ is not too small. This dynamic filter logic and the basin overlap exclusion provide mechanisms for decreasing the radii of some attraction basins, focusing on those which reject points most often and those which overlap. Without these, the radii can only increase. Since the true attraction basins may be far from spherical, allowing the radii to decrease seems desirable.

**Distance Filter**($x_t$, dstatus)

```plaintext
dstatus = accept
FOR i = 1, nlocals DO
    $x_l(i)$ = ith local solution
    dist(i) = distance($x_t$, $x_l(i)$)
```

// begin dynamic distance filter logic

IF (drejctr(i) = dwait) THEN
    df(i) = min(dfuser, dftemp(i))
    r(i) ← df(i) * r(i)
    drejctr(i) = 0, dftemp(i) = 0
ENDIF
// End dynamic distance filter logic

IF (dist(i) < dfuser * r(i)) THEN
    dstatus = reject
    drejctr(i) = drejctr(i) + 1
    dftemp(i) ← max(dftemp(i), dist(i) / r(i), 0.5)
ENDIF
ENDDO

3.3.2 Merit Filter Logic

The merit filter helps insure that starting points for $L((xs, xf)$ have high quality, by accepting only candidate points whose exact penalty function value $P$ is below a threshold. This threshold is set initially to the $P$ value of the best candidate point found in the first stage of the algorithm. When a trial point is accepted by the merit filter, the threshold is decreased by setting it to the $P$ value of that point. If trial points are rejected by this test for more than mwait (default value 20) consecutive iterations, the threshold is increased by the updating rule:

$$\text{threshold} \leftarrow \text{threshold} + \text{mf} \ast (1.0 + \text{abs(threshold)})$$

where mf is the merit threshold increase factor. The additive 1.0 term is included so that threshold increases by at least mf when its current value is near zero. In [9], mf was constant. Here it is computed dynamically. Given a trial point $x_t$, we calculate the value $(P(x_t, w) - \text{threshold})/(1 + \text{abs(threshold)})$. This is the value of the merit threshold increase factor, mftemp, which satisfies

$$P(x_t, w) = \text{threshold} + \text{mftemp} \ast (1 + \text{abs(threshold)})$$

i.e., which causes a rejected point $x_t$ to be on the borderline of acceptance. By initializing mftemp to plus infinity, and replacing mftemp by the minimum of the above quantity and
its previous value in the pseudocode below, we maintain mftemp as the smallest of these increase factors over all rejected trial points since mfctr was last reset to zero. This is the increase factor which would have caused the rejected point with smallest P value to be on the borderline of acceptance. The increase factor used is the larger of mftemp and the user supplied value, mfuser (default value 0.2). This dynamic logic leads to a looser or more greedy filter than previously, i.e., one which rejects fewer points.

**Merit Filter**\((xt,\text{threshold, mstatus})\)

mstatus = reject

IF \((P(xt, w) < \text{threshold})\) THEN

mstatus = accept

threshold = \(P(xt, w)\)

mrejctr = 0

mftemp = infinity

ELSE

mrejctr = mrejctr + 1

\(mftemp \leftarrow \min[mftemp, (P(xt, w) - \text{threshold})/(1 + abs(\text{threshold}))]\)

IF (mrejctr = mwait) THEN

mrejctr = 0

mf = max(mfuser, mftemp)

\(\text{threshold} \leftarrow \text{threshold} + mf*(1 + abs(\text{threshold}))\)

ENDIF

ENDIF

We show in the next section that the the two dynamic filters and the basin overlap exclusion lead to more solver calls, but also to a substantial increase in the number of problems solved to a close tolerance of their known solutions, using default parameters. Previously, the filters had to be loosened manually to solve some of these problems.

**4. Computational Experiments with the Dynamic Filters**

This section applies our C implementation of the MSNLP algorithm, now available as a GAMS global Solver, to a large set of smooth, nonconvex test problems, using CONOPT as the local solver, \(L\). The experiments show the effects of the dynamic filter logic
described in Section 3, the relative importance of the distance and merit filters, and what happens when more iterations are allowed. Computations were performed on a Dell OptiPlex PC with a 1.2 Ghz Pentium IV processor and 261 Mbytes of RAM, running under Windows 2000. The system was compiled using the Microsoft Visual C++ compiler version 6.0, using WIN32 release mode and optimization level set at “Maximum Speed.”

Table 1 shows the major algorithm parameter values used, all default values. The names in parentheses are those in the pseudocode of Section 3. The CONOPT [3] local NLP solver uses CONOPT default parameters. The artificial bound is imposed on variables with infinite upper or lower bounds. It (its negative) replaces the upper (lower) bound, as long as the modified bounds have upper bound > lower bound. If not, the artificial bound is subtracted from or added to the finite bound to give a nonempty interval. These modified bounds are communicated to the drivers, because they require a bounded rectangle to search in, but are not communicated to the NLP solvers, which are thus allowed to move to points outside this rectangle. As shown in Section 6, the artificial bound value has a strong effect on the randomized drivers, and must sometimes be reduced from its default value of 1.E4 to achieve acceptable results.

**Table 1.** Base case MSNLP Parameters and Options Used

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>time limit</td>
<td>6000 sec</td>
</tr>
<tr>
<td>Solver call limit</td>
<td>1000</td>
</tr>
<tr>
<td>total iterations</td>
<td>1000 (n2)</td>
</tr>
<tr>
<td>stage 1 iterations</td>
<td>200 (n1)</td>
</tr>
<tr>
<td>merit filter waitcycle</td>
<td>20 (mwaitcycle)</td>
</tr>
<tr>
<td>distance filter waitcycle</td>
<td>20 (dwaitcycle)</td>
</tr>
<tr>
<td>threshold increase factor</td>
<td>0.2 (mfuser)</td>
</tr>
<tr>
<td>basin radius decrease factor</td>
<td>0.8 (dfuser)</td>
</tr>
<tr>
<td>distance factor</td>
<td>1.0 for dynamic filters, 0.75 without (distfactor)</td>
</tr>
<tr>
<td>artificial bound</td>
<td>1.E4</td>
</tr>
<tr>
<td>NLP solver</td>
<td>Conopt (L in pseudocode)</td>
</tr>
<tr>
<td>Starting point generator</td>
<td>OptQuest (SP in pseudocode)</td>
</tr>
</tbody>
</table>
The test problems used here are 135 of those described in [5] with only continuous variables. These include the 128 continuous variable problems described and used in [9]. All are coded in the GAMS modeling language. The full set of these problems is available for download at http://titan.princeton.edu/TestProblems/index.html. Details and the GAMS model files for the 135 problems used here are available at www.gamsworld.org/global/apps/msnlp. A few of these problems were modified to (a) correct minor errors in gams syntax, (b) avoid domain violations in nonlinear functions, (c) enable the CONOPT local solver to solve the problem from the starting points generated by MSNLP by tightening some variable bounds, etc. The list of problems modified, what was changed, and the original and modified model files, are on the website mentioned immediately above. The best known objective values which we compare with here were obtained originally from [5]. In a few cases we found improved values, and modified these values accordingly. There are 14 problems with over 100 variables, the largest having 144. Five have over 100 constraints, with the largest having 435. However, in four of these five problems, the constraints are included to avoid domain violations (in the set of particle cluster energy minimizations EX8_6_1_n, where n is the number of particles), and none of these constraints are active at the final MSNLP solution.

As in [9], we use the relative percentage gap defined by

$$\text{gap} = \frac{100 \left( f_{\text{msnlp}} - f_{\text{best}} \right)}{1 + \text{abs} \left( f_{\text{best}} \right)}$$  \hspace{1cm} (7)

to measure the deviation between the final MSNLP objective value, $f_{\text{msnlp}}$, and the best known value, $f_{\text{best}}$. This differs slightly from the gap definition used by Mittleman and Pruessner in [7]; it yields a comparable result without conditional logic. We consider two cases, one using the dynamic filter logic described in Section 3, the other using both filters but not the dynamic filter logic. Both use the algorithm parameter values in Table 1. These are all default values, and are the same for both cases, except for distance factor. This is set to 0.75 for the runs without dynamic filters, because this is the value used in [9], and because results in this case are much worse using 1.0 (more on this below). All runs in this section terminated because they reached the iteration limit of 1000.
Table 2 below shows that only 3 of the 135 problems had gaps of more than 1% (“failures”) using dynamic filters, the sum of these gaps being 8.8%, versus 11 failures with a sum of 142.9% not using them. Results without dynamic filters correspond closely to those for the algorithm presented in [9], where there were 9 failures in a subset of 128 of these problems. Differences between the current and previous results are due to the slightly enlarged problem set and to numerous small changes in algorithm details and parameter values. In the case without dynamic filters, if distance factor =1.0, there are 22 failures with a gap sum of 977.5%. Using 1.0 in this situation creates filters which are much too tight. Almost all the gaps in the non-failed problems are less than .01%.

There is a significant increase in effort paid to achieve this increased reliability, but we feel it is justified. Dynamic filters cause total solver calls to increase from 2540 (median =11) to 4422 (median=32), and total time to increase from 1586 to 2484 seconds.

The best solution was found in solver call 1 (at the initial point) in 67 of 135 problems without dynamic filters and in 64 with. This indicates that many problems may have only one local solution-only one is found in 32 problems using dynamic filters. The small difference is caused by the dynamic filters locating improved solutions in calls after the first slightly more often. The best solution is found in the second solver call (at the final point found by stage 1) in 31 problems using dynamic filters and in 33 without. This shows that, using dynamic filters, even a coarse search of 200 stage 1 iterations provides a point within the basin of attraction of the global solution in 43.7% of the 71 problems whose best solution was not found in call 1. Using the dynamic filters, the median number of solver calls needed to first find the best solution is only 2, while the median number of total solver calls is 32.
Table 2. Results Solving 135 Floudas problems with and without dynamic Filters

<table>
<thead>
<tr>
<th>Case</th>
<th>Total failures</th>
<th>Sum of % gaps for failed problems</th>
<th>Solver Calls (total, median)</th>
<th>Total Time (sec)</th>
<th>Problems with best in call 1</th>
<th>Problems with best in call 2</th>
<th>Merit rejects (median)</th>
<th>Dist rejects (median)</th>
<th>Both reject (median)</th>
<th>Overlap fix (median)</th>
<th>Dynamic distance (median)</th>
<th>Dynamic merit (median)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Filters</td>
<td>3</td>
<td>8.8</td>
<td>(4422,32)</td>
<td>2484</td>
<td>64</td>
<td>638</td>
<td>15</td>
<td>121</td>
<td>19</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>No Dynamic Filters</td>
<td>11</td>
<td>142.9</td>
<td>(2540,11)</td>
<td>1586</td>
<td>67</td>
<td>469</td>
<td>17.5</td>
<td>288</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Columns 8 through 10 of Table 2 show the median number of points rejected by merit, distance, and both filters. The merit filter rejects far more points than the distance filter in both cases, especially with dynamic filters. Its strong overall activity is partially explained by the fact that OptQuest (and the other generators considered later) only occasionally generate points with a new best value and, as long as the merit threshold remains constant, any point with P value worse than that of a previously rejected point will also be rejected. Since two of the three dynamic filter mechanisms loosen the distance filters, it is expected that they will reject fewer points than they do without this logic.

The last three columns show that the basin overlap exclusion reduces a pair of basin radii a median of 20 times per problem, while the dynamic distance and merit filters reduce a basin radius or increase the merit threshold 10 and 9 times per problem respectively. Since the overlap fix can be applied at most each solver call, and the median number of solver calls using the dynamic logic is 32, the overlap fix is very active. Examination of the individual problem results show that the number of pairs of radii reduced increases strongly with both the number of solver calls and the number of different local solutions found. Problems with only one local found have no overlap fix reductions, and there are 32 of these when dynamic filters are used. The largest number of pairs of radii reductions is 479, in problem EX8_6_1_20, which corresponds to
Floudas problem EX8_6_1 with 20 particles. MSNLP with dynamic filters calls Conopt 64 times in this problem and finds 64 different local solutions.

The dynamic distance filter logic functions in a complementary way with the overlap fix. In problems where few locals are found, it reduces basin radii relatively often, because the few existing basins are each responsible for many rejections. In problems with many locals, any one basin is unlikely to reject 20 points in 800 stage 2 iterations, so it is applied few or no times. These effects are magnified by our resetting the dynamic distance filter rejection counters to zero whenever the overlap fix reduces a pair of radii. For example, in the 30 problems where only one local was found, the median number of dynamic basin radius reductions is 16 (there are no reductions by the overlap fix), while in the 13 problems where 40 or more locals were found the median number of dynamic basin reductions is 0 and the average is 1.5 (there are hundreds of reductions by the overlap fix).

**Using More iterations and Other Solvers**

In the above experiments, using Conopt as the local solver, there were eight problems which had gaps greater than 0.1%. Table 3 shows problem data and solution statistics for these problems.

**Table 3.** Data on 8 “Large Gap” Floudas problems using 1000 iterations

<table>
<thead>
<tr>
<th>Name</th>
<th>(vars,cons)</th>
<th>Type</th>
<th>Percent gap</th>
<th>Solver calls</th>
<th>Infeasible solver calls</th>
<th>Number of locals</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX2_1_9</td>
<td>(10,1)</td>
<td>Concave QP</td>
<td>3.03</td>
<td>58</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>EX3_1_3</td>
<td>(6,6)</td>
<td>quad obj and constr</td>
<td>3.86</td>
<td>82</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>EX8_6_1_25</td>
<td>(69,300)</td>
<td>Min Lennard-Jones energy</td>
<td>0.55</td>
<td>47</td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td>EX8_6_1_30</td>
<td>(84,435)</td>
<td>same</td>
<td>1.91</td>
<td>65</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>EX8_3_1</td>
<td>(116,77)</td>
<td>reactor network synthesis</td>
<td>0.79</td>
<td>54</td>
<td>45</td>
<td>9</td>
</tr>
<tr>
<td>EX8_3_3</td>
<td>(111,77)</td>
<td>same</td>
<td>0.44</td>
<td>36</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>EX8_3_7</td>
<td>(127,93)</td>
<td>same</td>
<td>0.9</td>
<td>51</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>EX8_3_11</td>
<td>(116,77)</td>
<td>same</td>
<td>0.45</td>
<td>54</td>
<td>44</td>
<td>10</td>
</tr>
</tbody>
</table>
The problems from the EX8_3_x series (x=1,…,14) are reactor network synthesis problems, and have many nonlinear equality constraints involving bilinear terms. In these 4 problems, from 47% to 83% of the Conopt calls terminate with Conopt declaring the problem infeasible, having found a local optimum in phase 1 of the GRG algorithm with a positive sum of infeasibilities (all 14 problems have feasible solutions). In fact, each of the 14 problems in this set had a high fraction of infeasible solver calls, illustrating a generic potential difficulty with multistart algorithms, namely they are completely dependent on the ability of the local solver to return a local solution, and so are adversely affected by solver failures. We show shortly how choosing SNOPT as the NLP solver for these four problems leads to better results.

The first four problems in the table include all three with gaps larger than 1%, and had no infeasible solver calls. Two of these are from the EX8_6_1_x series, which minimize the Lennard-Jones energy of a cluster of x particles. As found on the problem website, these are unconstrained, but the objective function involves reciprocals of the squared Euclidean distance between distinct pairs of particles. This leads to from hundreds to thousands of domain violations (attempts to divide by too small a quantity). We found that better results were obtained when we added a lower limit on these distances, chosen so that these were far from active at MSNLP’s best solution, and this is the source of the constraints listed in Table 3.

Table 4 shows the effects of increasing the number of stage 1 and stage 2 iterations on the first four problems of Table 3, with CONOPT as the local solver, and dynamic filters turned on.
Table 4. Final gaps and Computational Effort Using More Iterations for the first four problems in Table 3

<table>
<thead>
<tr>
<th>(Stage 1 itns, stage 2 itns)</th>
<th>Number of problems attempted</th>
<th>Number with gaps &gt; 0.1%</th>
<th>Number with gap&gt;1%</th>
<th>Sum of gaps (%)</th>
<th>Average solver calls per problem</th>
<th>Average runtime per problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>(200,800)</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>11.9</td>
<td>40</td>
<td>131</td>
</tr>
<tr>
<td>(400,1600)</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1.2</td>
<td>138.2</td>
<td>518.2</td>
</tr>
<tr>
<td>(800,3200)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.12</td>
<td>253</td>
<td>2164</td>
</tr>
</tbody>
</table>

There is a strong and steady improvement in accuracy, with all but one problem solved to within 0.1% in 4000 total iterations. Average solver calls and runtime increase roughly linearly with iterations.

Table 5 compares the GRG implementation CONOPT and the SQP implementation SNOPT on the last four problems in Table 3. SNOPT has far fewer infeasible calls than CONOPT, and is thus far more effective. We believe this is because CONOPT’s phase one objective is an indefinite quadratic form in these problems, and this evidently creates many local optima in phase one. The “elastic” variables used by SNOPT to deal with infeasibilities also contribute to its improved performance. See page 5 of the SNOPT user guide, included with the GAMS system, for a description. Doubling the MSNLP iterations using SNOPT approximately halves the sum of the gaps, but two problems still have small gaps. These problems have many distinct local solutions, many of which have nearly equal objective values.
Table 5. Solving 4 Reactor Network Problems with CONOPT and SNOPT

<table>
<thead>
<tr>
<th>Solver</th>
<th>(Stage 1 (its), Stage 2 (its))</th>
<th>Problems attempted</th>
<th>Number with gaps &gt;0.1%</th>
<th>Sum of gaps</th>
<th>Total solver calls</th>
<th>Total infeasible solver calls</th>
<th>Number of locals</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONOPT</td>
<td>(200,800)</td>
<td>4</td>
<td>4</td>
<td>2.58</td>
<td>195</td>
<td>132</td>
<td>63</td>
</tr>
<tr>
<td>SNOPT</td>
<td>(200,800)</td>
<td>4</td>
<td>2</td>
<td>0.74</td>
<td>191</td>
<td>8</td>
<td>183</td>
</tr>
<tr>
<td>SNOPT</td>
<td>(400,1600)</td>
<td>2</td>
<td>1</td>
<td>0.31</td>
<td>105</td>
<td>16</td>
<td>194</td>
</tr>
</tbody>
</table>

5. Pure and “Smart” Random Drivers

The “pure” random (PR) driver generates uniformly distributed points within the hyper-rectangle $S$ defined by the variable bounds. However, this rectangle is often very large, because users often set bounds to $(-\infty, +\infty)$, $(0, +\infty)$, or to large positive and/or negative numbers, particularly in problems with many variables. This usually has little adverse impact on a good local solver, as long as the starting point is chosen well inside the bounds. But the PR generator will often generate starting points with very large absolute component values when some bounds are very large, and this sharply degrades solver performance. Thus we were motivated to develop random generators which control the likelihood of generating candidate points with large components, and intensify the search by focusing points into promising regions. We present two variants, one using normal, the other triangular distributions. Pseudo-code for this “smart random” generator using normal distributions follows, where $w$ is the set of penalty weights determined by the “update locals” logic (see Section 3) after the first solver call at the user-specified initial point.

**Smart Random Generator with Normal Distributions, SRN(xt)**

IF (first call) THEN

Generate $k1$ (default 400) diverse points in $S$ and evaluate the exact penalty function $P(x, w)$ at each point.

B=subset of the $k1$ diverse points with $k2$ (default 10) best P values
FOR i = 1,nvars DO
    xmax(i) = max of component i of points in B
    xmin(i) = min of component i of points in B
    mu(i) = (xmax(i)+xmin(i))/2
    ratio(i) = (xmax(i)-xmin(i))/(1+buvar(i)-blvar(i))
    sigfactor = 2.0
    IF (ratio>0.7) sigfactor = sigAdjust(ratio)
    sigma(i) = (xmax(i)-xmin(i))/sigfactor
ENDDO
ENDIF
FOR i = 1,nvars DO
    Generate a normally distributed random variable rv(i) with mean mu(i) and standard deviation sigma(i)
    If rv(i) is between blvar(i) and buvar(i), xt(i) = rv(i)
    If rv(i)<blvar(i), generate xt(i) uniformly between blvar(i) and xmin(i)
    If rv(i)>buvar(i), generate xt(i) uniformly between xmax(i) and buvar(i)
ENDF
Return xt

This SRN generator attempts to find a subset, B, of k2 “good” points, and generates most of its trial points xt, within the smallest rectangle containing B. It first generates a set of k1 diverse points within the bounds using a stratified random sampling procedure with frequency-based memory, similar to that described in [6, pp 24-28]. For each variable x(i), this divides the interval [blvar(i), buvar(i)] into 4 equal segments, chooses a segment with probability inversely proportional to the frequency with which it has been chosen thus far, then generates a random point in this segment. We choose k2 of these points having the best P(x,w) penalty values, and use the smallest rectangle containing these, intersecting the ith axis at points [xmin(i), xmax(i)], to define n univariate normal distributions (driver SRN) or n univariate triangular distributions (driver SRT). The mean of the ith normal distribution, mu(i), is the midpoint of the interval [xmin(i), xmax(i)], and this point is also the mode of the ith triangular distribution, whose lower and upper limits are blvar(i) and buvar(i). The standard deviation of the ith normal
distribution is selected as described below. The trial point $x_t$ is generated by sampling $n$ times independently from these distributions. For the driver using normals, if the generated point lies within the bounds, it is accepted. Otherwise, we generate a uniformly distributed point between the violated bound and the start of the interval.

To determine the standard deviation of the normal distributions, we compute $\text{ratio}$, roughly the ratio of interval width to distance between bounds, where the factor 1.0 is included to avoid division by zero when the bounds are equal (fixed variables). If the interval width is small relative to the distance between bounds for variable $i$ ($\text{ratio} \leq 0.7$), then the standard deviation $\sigma(i)$ is half the interval width, so about 1/3 of the $x_t(i)$ values fall outside the interval, providing diversity when the interval does not contain an optimal value for $x(i)$. If the bounds are large, then ratio should be small, say less than 0.1, so $x_t(i)$ values near the bounds are very unlikely. If $\text{ratio} > 0.7$, the function $\text{sigAdjust}$ sets $\text{sigfactor}$ equal to 2.56 if ratio is between 0.7 and 0.8, increasing in steps to 6.2 if $\text{ratio} > 0.999$. Thus if $\text{ratio}$ is near 1.0, more than 99% of the values fall within the interval, and few have to be projected back within the bounds. The projecting back process avoids undesirable clustering of trial points at a bound, by generating points uniformly between the violated bound and the nearest edge of the interval $[\text{xmin}(i), \text{xmax}(i)]$.

When the interval $[\text{xmin}(i), \text{xmax}(i)]$ is sharply skewed toward one of the variable bounds and is much narrower than the distance between the bounds, a symmetric distribution like the normal, combined with our projection procedure, generates too many points between the interval and its nearest bound. A quick scan of the test results indicates that this happens rarely, but an asymmetric distribution like the triangular overcomes this difficulty, and needs no projection. We compare the performance of these 2 distributions in what follows.

Once a distribution is chosen, trial points governed by this distribution are generated by standard monte carlo sampling. It is possible that latin hypercube sampling would provide improved performance, and we plan to investigate its use in the future.
6. Computational Experiments Comparing Three Drivers

This section compares the performance of the MSNLP algorithm using four drivers: Optquest (OQ), pure random (PR), and smart random with normal and triangular distributions (SRN and SRT), on a large and diverse set of 339 test problems with all continuous variables, a subset of the 398 problems contained in the GAMS Globallib as of 2/04. The entire library may be downloaded from [www.gamsworld.org/global/globallib.htm]. We have excluded 59 of the largest problems (including 24 problems with between 2000 and 5000 variables and 23 with more than 5000 variables) in order to avoid run times of many hours. A few other problems were excluded as being redundant in some sense. The 339 problems selected include 80 with more than 50 variables, 56 with more than 100, and 18 with more than 1000. The largest, ARKI0023, has 8886 variables and 2 constraints. Others have hundreds or thousands of constraints. Details and the GAMS model files for the set of selected problems are available at www.gamsworld/global/apps/msnlp.

A “best known” feasible objective value was established for all problems by choosing the best of the final objective values obtained by the 3 drivers in 1000 total iterations and the 100 best known values provided by GAMS Development Company (these are on the Globallib website). To compare drivers, we continue to use the relative gap defined in (7). All runs use the three dynamic filtering mechanisms described and tested in Sections 3 and 4, with the default parameter values in Table 1 (except for the driver), unless otherwise stated. All runs terminated after reaching the iteration limit of 1000, unless otherwise indicated.

Table 6 below shows the number of problems with final objective value within 1% of the best known value and some effort and solver call statistics.
**Table 6.** Comparing 4 drivers on 339 globallib problems, artificial bound =1.E4

<table>
<thead>
<tr>
<th>Driver</th>
<th>Problems attempted</th>
<th>Number within 1% of best</th>
<th>Runtime (hrs)</th>
<th>CONOPT calls</th>
<th>Number of infeasible CONOPT calls</th>
<th>Problems with more than 50% infeasible CONOPT calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>OQ</td>
<td>339</td>
<td>331</td>
<td>4.2</td>
<td>9747</td>
<td>1680</td>
<td>30</td>
</tr>
<tr>
<td>SRN</td>
<td>339</td>
<td>321</td>
<td>2.6</td>
<td>11597</td>
<td>2750</td>
<td>52</td>
</tr>
<tr>
<td>SRT</td>
<td>339</td>
<td>321</td>
<td>3.7</td>
<td>13733</td>
<td>4795</td>
<td>76</td>
</tr>
<tr>
<td>PR</td>
<td>339</td>
<td>322</td>
<td>3.0</td>
<td>10919</td>
<td>3179</td>
<td>76</td>
</tr>
</tbody>
</table>

OQ solves all but 8 of the problems to within 1% of the best known value, while SRN, SRT, and PR solve all but 18, 18, and 17 respectively. OQ’s superior performance here is closely related to the fact that the three randomized drivers have many more CONOPT calls which terminate with infeasible solutions, and thus many more problems where more than 50% of these calls terminate infeasible. Reasons for termination in such calls include domain violations at the initial point (CONOPT abandons the problem in this case), and terminations at a local optimum in phase 1. OptQuest is much better than the randomized drivers at avoiding these difficulties. The apparently longer time for the OptQuest runs is addressed below in the discussion which follows Table 7. It uses a “Reference Set” of elite solutions to guide its search, and the OQ trial points are all within or slightly outside of the convex hull of this set-see [6]. By default, the Reference Set contains about 10 points, so stage 1 of MSNLP using OptQuest ends with a Reference Set which has been updated at least once, and contains roughly those stage 1 points with the 10 best \( P(x,w) \) values. Examination of this set on a few small problems showed that, when there are large bounds, the Reference Set points are well within them. We believe that this is true in general, so OptQuest provides a natural intensification of its search which makes it fairly insensitive to the presence of large bound values. This property is not shared by the PR driver, and is present to a lesser degree with the 2 smart random drivers.

As an example of the problems with infeasible solver calls using the randomized drivers, in the series ARKI00x.gms, with \( x = 1 \) and 19-24, all solver calls after the first
terminated without performing any iterations, with an error message that some derivatives could not be evaluated at the initial point. All these initial points were infeasible, so the distance and merit filters had much less effect than usual (only one local solution was found, starting from the initial point specified in the model), and from 101 to 279 failed solver calls were made, the number increasing with problem size. All variables in this problem series had infinite upper bounds, so the default artificial bound value of 1.e4 was imposed, leading to the domain violations. Replacing this bound by 1.e2 sharply reduced the domain violations, and most subsequent solver calls returned feasible solutions. The time for a successful solver call ranged from 5 to 15 minutes for the larger problems (8886 variables for ARKI0023), so we changed the time limit for any one problem to 900 seconds, and reran all problems using an artificial bound of 1e2, with the PR and both SR drivers. Results are shown in Table 7 below, with the previous OQ results repeated for ease of comparison. We also show the number of problems terminating with less than 1000 iterations performed, due to reaching the time limit. OQ has only 1 such problem, followed by 6 for the SRN driver, showing that their ability to avoid generating starting points with large components leads to fewer long solver calls.

Table 7. Comparing 4 drivers on 339 globallib problems, artificial bound =1.e2

<table>
<thead>
<tr>
<th>Driver</th>
<th>Problems attempted</th>
<th>Number within 1% of best</th>
<th>Runtime (hrs)</th>
<th>CONOPT calls</th>
<th>Number of infeasible CONOPT calls</th>
<th>Problems with more than 50% infeasible CONOPT calls</th>
<th>Number of problems terminating with less than 1000 iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OQ</td>
<td>339</td>
<td>331</td>
<td>4.2</td>
<td>9747</td>
<td>1680</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>SRN</td>
<td>339</td>
<td>332</td>
<td>4.7</td>
<td>10488</td>
<td>1442</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>SRT</td>
<td>339</td>
<td>327</td>
<td>5.8</td>
<td>11678</td>
<td>1732</td>
<td>34</td>
<td>12</td>
</tr>
<tr>
<td>PR</td>
<td>339</td>
<td>329</td>
<td>4.4</td>
<td>9842</td>
<td>1324</td>
<td>31</td>
<td>13</td>
</tr>
</tbody>
</table>

Reducing the artificial bounds improves the relative performance of the randomized drivers significantly. Now all 4 drivers solve about the same number of problems to within 1% of the best known solution, with SRN having a one problem advantage over OQ, followed by PR and SRT. The reduced search interval in problems with some
infinite bounds undoubtedly plays a role in the improved performance of the randomized drivers. However, we believe that the major factor causing this improvement is the sharp reduction in infeasible CONOPT calls for the randomized drivers, compared to their previous values in Table 6. With the reduced artificial bound, the SRN and PR drivers have fewer infeasible calls than OQ, while SRT is only slightly higher, due to the increased “spread” of the triangular distribution. However, the apparent run time advantage of the randomized drivers has now vanished, with all three taking longer than OQ. Runtime is now roughly proportional to CONOPT calls. The lower runtimes in Table 6 for the randomized drivers were caused by very short times for CONOPT calls which terminated infeasible, especially for those stopping at the initial point due to domain violations.

It is interesting that reducing the artificial bound to 1.e2 hurts rather than helps the OQ driver: there are 17 problems with gaps > 1% compared to eight with 1.e4. Some of these appear to be caused by the fact that the GAMS objective variable is a free variable, and it is thus communicated to OptQuest with bounds equal to plus and minus the artificial bound. Too small a bound apparently prevents OptQuest from generating points in the attraction basin of the best solution to these problems.

As in the experiments with the Floudas problems discussed in Section 4, the problems not solved to within 1% of the best known values in Table 7 are almost all solved to this tolerance with increased effort or by using different solvers. For example, using the SRN driver, 7 problems had gaps larger than 1%. Changing the (max time, total iterations, stage 1 iterations) from (900, 1000, 200) to (1800, 2000, 400) and re-solving these problems produces final objective values for 4 of the 7 which are equal to (in one) or better than the previous best values (in three). Increasing the artificial bound to 1000 solves another (Prolog) to the previous best value. In Prolog, 4 variables in the optimal solution have values ranging from 224 to 430, so an artificial bound of 100 evidently cannot produce starting points within the attraction basin of this global solution. This illustrates the importance of choosing bounds for variables which are at least the right order of magnitude—no single artificial bound value can be adequate for a wide range of problems. Using LSGRG2 as the solver achieves the best known value for problem
Bayes2_10, so 6 of the 7 problems are solved to very small or negative gaps by these measures.

The number of solver calls to first locate the best local solution found in an MSNLP run, when larger than 1, is a measure of efficiency of the driver used (the first solver call occurs at the initial point specified in the GAMS model). The fraction of the 339 problems with this value in the ranges shown is displayed in Table 8.

**Table 8. Fraction of problems with Solver calls to best in various intervals**

<table>
<thead>
<tr>
<th>Driver</th>
<th>Best on call 1</th>
<th>Best in calls 2 to 3</th>
<th>Best in calls 4 to 5</th>
<th>Best in calls 6 to 10</th>
<th>Best in calls 11 to 20</th>
<th>Best in calls 21 or more</th>
</tr>
</thead>
<tbody>
<tr>
<td>OQ</td>
<td>0.640</td>
<td>0.174</td>
<td>0.062</td>
<td>0.053</td>
<td>0.027</td>
<td>0.044</td>
</tr>
<tr>
<td>SRN</td>
<td>0.652</td>
<td>0.189</td>
<td>0.059</td>
<td>0.035</td>
<td>0.038</td>
<td>0.027</td>
</tr>
<tr>
<td>SRT</td>
<td>0.654</td>
<td>0.207</td>
<td>0.033</td>
<td>0.030</td>
<td>0.041</td>
<td>0.036</td>
</tr>
<tr>
<td>PR</td>
<td>0.664</td>
<td>0.204</td>
<td>0.021</td>
<td>0.038</td>
<td>0.032</td>
<td>0.041</td>
</tr>
</tbody>
</table>

All 3 drivers have a high percentage of problems with their best MSNLP solution found in the first Solver call. This ranges from 67% for PR to 64% for OQ. The driver affects these percentages because they have different behavior in finding better local solutions in solver calls beyond the first. This is why OQ has the lowest fraction of best values in call 1—it finds improved solutions later slightly more often, due to its more sophisticated search strategy. Many of the problems with best solution found in the first call may have only one local solution. In fact, there are 117 (out of 339) problems in which all 3 drivers find one distinct local solution, 34.5% of the total. The percent of problems with best solution found in calls 2 or 3 ranges from 17.4% for OQ to 20.7% for SRT, showing that stage 1 is effective in locating a point in the basin of attraction of the global solution. OQ finds 11.5% of its best solutions in Solver calls 4 to 10, compared to 9.4% for SRN, 6.3% with SRT, and 5.9% for PR, so it is best at finding improved solutions rapidly.

The best solution is found in the first 10 solver calls in about 93% of the 339 problems with all drivers, implying that it is fairly safe to limit the total number of solver calls to 10 or less in problems where solver calls are very expensive, or where MSNLP must be called often. The latter would occur when MSNLP is used within a branch and
bound algorithm for MINLP, to attempt to locate global solutions to the relaxed NLP subproblems. GAMS allows the use of MSNLP as the NLP solver within the SBB (Simple Branch and Bound) MINLP Solver [4], and we plan to test this option.

**Increasing the stage one iterations**

While we have not done a thorough investigation, there is evidence that increasing the number of stage 1 iterations, $n1$, can degrade MSNLP reliability. Using the OQ driver, we changed $n1$ from 200 to 1000, but left the stage 2 iterations, $n2$, at 800. Solving the 339 globallib problems, there were 12 problems with larger than 1% gaps, an increase of 4 from the base case. A longer stage 1 certainly produces at least as good a starting point for the first stage 2 solver call, since this is the stage 1 point with best penalty value. It should also lead to better OptQuest trial points throughout stage 2, because OptQuest remembers its past results through the Reference Set, and the penalty function values of points in this set improve as more iterations are performed. However, the GAMS log files show that, in all problems, the best OQ penalty value at the end of stage 1 is much larger (for min problems) than the objective value found by the first or second solver call. The relative differences are often an order of magnitude or more. OptQuest is not nearly as powerful in smooth, nonlinearly constrained NLP’s as a single gradient-based local solver call because it does not use derivatives, and thus cannot employ some variant of Newton’s method to achieve and maintain feasibility and to determine good search directions. This is even more so for the randomized drivers. Thus the stage 2 points produced are so far from optimal that small improvements do not produce better results in stage 2, and in fact produce worse outcomes in 4 problems, with no compensating improvements. Thus our strategy of performing only a coarse search in stage 1 seems sound.

**Performance profiles and increased time limits from the GAMS PAVER server**

In Dolan and More [2], performance profiles are introduced as a powerful graphical tool to compare the solution times and problem solving abilities of several solvers. These may be created and accessed online through the GAMS PAVER (Performance Analysis and Visualization for Efficient Reproducibility) server, available at [www.gamsworld.org/performance](http://www.gamsworld.org/performance). The following results were created using this facility to analyze results obtained solving 341 globallib models at GAMS Development
Company on a 3.4 GHz PC, much faster than the 1.2Ghz machine used to generate all previous results. This problem set includes the 339 models reported on above plus 2 others. All algorithm parameters and options were set to default values except when specified otherwise.

One objective in these experiments is to investigate more thoroughly the effect of allowing more MSNLP iterations. Hence we defined the 3 case studies defined in table 9 below:

**Table 9.** case studies designed to stop on time or iterations

<table>
<thead>
<tr>
<th>Driver</th>
<th>Time limit(sec)</th>
<th>Total iteration limit</th>
<th>Artificial bound</th>
<th>Case Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRN</td>
<td>300</td>
<td>Very large</td>
<td>100</td>
<td>SRN_t</td>
</tr>
<tr>
<td>SRN</td>
<td>300</td>
<td>1000</td>
<td>100</td>
<td>SRN</td>
</tr>
<tr>
<td>OQ</td>
<td>300</td>
<td>1000</td>
<td>1E4</td>
<td>OQ</td>
</tr>
</tbody>
</table>

In case SRN_t, the time limit per problem was set to 300 seconds, and the total iteration limit was set large enough so that it was never reached in that time. In the other 2 cases, almost all except the 2 or 3 largest problems stop after 1000 iterations. We used the artificial bound value best suited to the SRN and OQ drivers. Results of these cases are shown below, comparing each against the others. Case a is defined to have a better objective value than case b (for minimization) if the relative objective difference \((\text{obj}(b) - \text{obj}(a))/\text{abs}(\text{obj}(a))\) is greater than 1.00E-05. For objective values below 1e-1 we use absolute differences.

**Table 10.** Comparing the reliability of the 3 cases in Table 9

<table>
<thead>
<tr>
<th>Case1</th>
<th>Case2</th>
<th>Obj of case1 better</th>
<th>Both obj the same</th>
<th>Obj of case2 better</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRN_t</td>
<td>SRN</td>
<td>22</td>
<td>319</td>
<td>0</td>
</tr>
<tr>
<td>SRN_t</td>
<td>OQ</td>
<td>25</td>
<td>316</td>
<td>0</td>
</tr>
<tr>
<td>SRN</td>
<td>OQ</td>
<td>12</td>
<td>317</td>
<td>10</td>
</tr>
</tbody>
</table>

The first 2 rows of table 10 show that allowing the SRN driver to run for only 5 minutes achieves significant improvements over the values obtained with SRN and OQ drivers stopped after 1000 iterations. There are 22 problems with improved final
objective values comparing with SRN, and 25 comparing with OQ. With 1000 iterations, OQ and SRN drivers perform similarly, as observed earlier.

**Performance profiles for 5 cases**

Figure 1 shows performance profiles generated by the PAVER server for five cases using the SRN and OQ drivers. The problem set is the same 341 globallib problems used in tables 9 and 10. In the figure labels, “ms” refers to the SRN driver, 1e2 and 1e4 refer to the artificial bound value used, and “_t” denotes the conditions of case SRN_t (large iteration limit, 300 second time limit). Time factor is the ratio of run time for a problem divided by the smallest run time of all five cases being compared. The vertical axis shows the percent of the problems solved to the best of the five case objective values, using the tolerances described above, with time factor not exceeding the specified value.

All 4 cases with a 1000 iteration limit have very similar profiles, with small differences in the final percent of problems solved. These were discussed earlier. The curve labeled ms_1e2_t falls far below the others for time ratios <100, because the runs reflected there time out at 300 seconds on all problems. However, 1000 iterations for most of the problems take far less than 300 seconds. This curve ultimately reaches 100 percent. Since the other profiles solve almost as many problems far more quickly, the default iteration limit of 1000 seems a good compromise between speed and reliability.
7. Conclusions and Future Work

We have significantly improved and extended the multistart framework described in [9]. The dynamic filters allow the distance and merit filters to tune themselves to the problem instance, so they are now less sensitive to their parameters. More solver calls are typically made than previously, but almost all test problems are now solved to their best known solutions using default algorithm parameters and options. For the few that are not, increased iteration and time limits usually yield improved solutions.

The good performance of the randomized drivers makes them an attractive option within the MSNLP framework. A user can now try multiple drivers and choose the best result. Their main weakness is a tendency to generate trial points with large absolute components in problems where the bounds are large or infinite, leading to many NLP solver failures on some problems. This can be sharply reduced by imposing an artificial bound, but its best value is problem-specific, and too small a value can cut off the global
solution. It is best for users formulating problems for global solvers to pay careful attention to bound values, making them as tight as possible.

Hit and run stochastic search methods [10] offer intriguing possibilities as multistart drivers, and research has begun on adapting and testing them in this context. The “smart random” drivers might be improved by adding adaptive features to the coarse initial search used to determine their distributions. One possibility is to dynamically change the parameters of the normal or triangular distributions based on solutions found by the local NLP solver. Another is to include the correlations between different components of the vector of decision variables, rather than generating each component independently.

The need for tight bounds on the variables, discussed in section 6, implies that preprocessing algorithms which tighten these bounds will often improve the performance of any global (or local) NLP solver. The BARON global solver includes effective preprocessing methods—see http://archimedes.scs.uiuc.edu/baron.html. There are plans to implement these so that they can be used by other algorithms, and we will experiment with them when this occurs.

References
