Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization

Zsolt Ugray[1], Leon Lasdon[2], John Plummer[2], Fred Glover[3], James Kelly[4] and Rafael Marti[5]

[1]*The A. Gary Anderson Graduate School of Management, University of California, Riverside, CA, 92521-0203*

[2]*Management Science and Information Systems Department, McCombs School of Business, The University of Texas at Austin, Austin, TX, 78712, USA*

[3]*Graduate School of Business, University of Colorado, Boulder, CO, 80309*

[4]*OptTek Systems, Inc, 1919 7th St, Boulder CO, 80302*

[5]*Departament d'estadistica i Investigacio Operativa, University of Valencia, 46100 BURJASSOT, Valencia, Spain*

zsolt.ugray@ucr.edu, lasdon@mail.utexas.edu, jcplummer@mail.utexas.edu, fred.glover@colorado.edu, Kelly@OptTek.com, rafael.marti@uv.es

Submitted to INFORMS Journal on Computing, July 25, 2002.

Revised Version Submitted ????

The algorithm described here, called OptQuest/NLP or OQNLP, is a heuristic designed to find global optima for pure and mixed integer nonlinear problems with many constraints and variables, where all problem functions are differentiable with respect to the continuous variables. It uses OptQuest, a commercial implementation of scatter search developed by OptTek Systems, Inc., to provide starting points for any gradient-based local NLP solver. This solver seeks a local solution from a subset of these points, holding discrete variables fixed. The procedure is motivated by our desire to combine the superior accuracy and feasibility-seeking behavior of gradient-based local NLP solvers with the global optimization abilities of OptQuest. Computational results include 155 smooth NLP and MINLP problems due to Floudas et al., most with both linear and nonlinear constraints, coded in the GAMS modeling language. Some are quite large for global optimization, with over 100 variables and 100 constraints. Global solutions to almost all problems are found in a small number of local solver calls, often one or two.

(Global Optimization; Multistart Heuristic; Mixed Integer Nonlinear programming; Scatter Search; Gradient Methods)

---

1. Introduction

This paper describes OQNLP, a multistart heuristic algorithm designed to find global optima of smooth constrained nonlinear programs (NLPs) and mixed integer nonlinear programs (MINLPs). It uses the OptQuest Callable Library (OCL) implementation of Scatter Search [Laguna and Marti, 2000] to generate trial points, which are candidate starting points for a local NLP solver. These are filtered to provide a smaller subset from which the solver attempts to find a local optimum. Our GAMS implementation can use any GAMS NLP solver, and the stand-alone version uses the generalized reduced gradient NLP solver LSGRG2 [Smith and Lasdon, 1992].

The most general problem this algorithm can solve has the form

$$\text{minimize } f(x,y) \tag{1}$$

subject to the nonlinear constraints

$$gl \le G(x,y) \le gu \tag{2}$$

the linear constraints

$$l \le A_1 x + A_2 y \le u \tag{3}$$

$$x \in S \ , \ y \in Y \tag{4}$$

where x is an n-dimensional vector of continuous decision variables, y is a p-dimensional vector of discrete decision variables, and the vectors gl, gu, l, and u contain upper and lower bounds for the nonlinear and linear constraints respectively. The matrices A_1 and A_2 are m_2 by n and m_2 by p respectively, and contain the coefficients of any linear constraints. 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 OptQuest scatter search procedure. The set Y is assumed to be finite, and is often the set of all p-dimensional binary or integer vectors y which satisfy finite bounds. The objective function f and the m_1-dimensional vector of constraint functions G are assumed to have continuous first partial derivatives at all points in $S \times Y$. This is necessary so that a gradient-based local NLP solver can be applied to the relaxed NLP sub-problems formed from (1) - (4) by allowing the y variables to be continuous.

2. Multi-start algorithms for global optimization

In this section, which reviews past work on multi-start algorithms, we focus on unconstrained problems where there are no discrete variables, since to the best of our knowledge multi-start algorithms have been investigated theoretically only in this context. These problems have the form of (1)-(4) with no *y* variables and no constraints except the bounds $x \in S$ in (4).ε

All global minima of *f* are assumed to occur in the interior of *S.* By multi-start we mean any algorithm that attempts to find a global solution by starting a local NLP solver, denoted by *L,* from multiple starting points in *S*. The most basic multi-start method generates uniformly distributed points in *S,* and starts *L* from each of these. This converges to a global solution with probability one as the number of points approaches infinity--in fact, the best of the starting points converges as well. However, this procedure is very inefficient because the same local solution is located many times. A convergent procedure that largely overcomes this difficulty is called multi-level single linkage (MLSL) [Rinnooy Kan and Timmer, 1987]. MLSL uses a simple rule to exclude some potential starting points. A uniformly distributed sample of N points in *S* is generated, and the objective, *f,* is evaluated at each point. The points are sorted according to their *f* values, and the qN best points are retained, where q is an algorithm parameter between 0 and 1. *L* is started from each point of this reduced sample, except if there is another sample point within a certain critical distance that has a lower *f* value. *L* is also not started from sample points that are too near the boundary of *S*, or too close to a previously discovered local minimum. Then, N additional uniformly distributed points are generated, and the procedure is applied to the union of these points and those retained from previous iterations. The critical distance referred to above decreases each time a new set of sample points is added. The authors show that, if the sampling continues indefinitely, each local minimum of *f* will be located, but the total number of local searches is finite with probability one. They also develop Bayesian stopping rules, which incorporate assumptions about the costs and potential benefits of further function evaluations, to determine when to stop the procedure.

When the critical distance decreases, a point from which *L* was previously not started may become a starting point in the next cycle. Hence all sample points generated must be saved. This also makes the choice of the sample size, N, important, since too small a sample leads to many revised decisions, while too large a sample will cause *L* to be started many times. Random Linkage (RL) multi-start algorithms introduced by [Locatelli and Schoen, 1999] retain the good

convergence properties of MLSL, and do not require that past starting decisions be revised. Uniformly distributed points are generated one at a time, and *L* is started from each point with a probability given by a nondecreasing function $\phi(d)$, where d is the distance from the current sample point to the closest of the previous sample points with a better function value. Assumptions on this function that give RL methods the same theoretical properties as MLSL are derived in the above reference.

Recently, Fylstra et al. have implemented a version of MLSL that can solve constrained problems - see www.solver.com. Limited to problems with no discrete variables *y,* it uses the L_1 exact penalty function, defined as

$$P_1(x,w) = f(x) + \sum_{i=1}^{m} w_i viol(g_i(x)) \tag{5}$$

where the w_i are nonnegative penalty weights, $m = m_1 + m_2$, and the vector *g* has been extended to include the linear constraints (4). The function $viol(g_i(x))$ is equal to the absolute amount by which the *i*th constraint is violated at the point *x*. It is well known (see [Nash and Sofer, 1996]) that if x^* is a local optimum of (1)-(4), u^* is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at (x^*, u^*), and

$$w_i > abs(u_i^*) \tag{6}$$

then x^* is a local unconstrained minimum of P_1. If (1)-(4) has several local minima, and each w_i is larger than the maximum of all absolute multipliers for constraint *i* over all these optima, then P_1 has a local minimum at each of these local constrained minima. Even though P_1 is not a differentiable function of *x*, MLSL can be applied to it, and when a randomly generated trial point satisfies the MLSL criterion to be a starting point, any local solver for the smooth NLP problem can be started from that point. The local solver need not make any reference to the exact penalty function P_1, whose only role is to provide function values to MLSL. We will use P_1 in the same way in our OQNLP algorithm. We are not aware of any theoretical investigations of this extended MLSL procedure, so it must currently be regarded as a heuristic.

3. The OQNLP Algorithm

3.1 The Global Phase - Scatter Search

Scatter Search (ScS) is a population based meta-heuristic algorithm devised to intelligently perform a search on the problem domain [Glover, 1998]. It operates on a set of solutions called the *reference set* or *population*. Elements of the population are maintained and updated from iteration to iteration. Scatter Search differs from other population-based evolutionary heuristics like Genetic Algorithms (GAs) mainly in its emphasis on generating new elements of the population mostly by deterministic combinations of previous members of the population as opposed to the more extensive use of randomization. ScS was founded on strategies that were proposed as augmentations to GAs more than a decade after their debut in Scatter Search. It embodies principles and strategies that are still not emulated by other evolutionary methods and prove to be advantageous for solving a variety of complex optimization problems. For the most recent and complete description of ScS, see [Laguna and Marti, 2003]

A summary of the OptQuest [Laguna and Marti, 2002] and [Laguna and Marti, 2000] implementation of ScS follows. The problem to be solved has the form (1)-(4) but, to simplify the explanation, we assume there are no y variables.

3.1.1 Steps of Scatter Search

1. Initialize: size of reference set = b, initial point =$x0$, input upper and lower bounds on variables and constraint functions, and the coefficients of any linear constraints. Create an initial set of three points, $R0 \subset S$: all variables equal their lower bounds, all variables set to their upper bounds, and all variables equal the mid-point between their bounds. If an initial point has been determined, add it to *R0*.

2. Given *R0,* use a *diversification generation method* to augment it with additional points, creating an initial diverse reference set, $R \subset S$ of cardinality b. Optionally, map the elements of R into points that satisfy the linear constraints.

3. Evaluate the objective f and the nonlinear constraint functions G at each point in R, and evaluate a penalty function P_{OQ}, equal to the objective plus a penalty weight times the maximum percentage violation of the nonlinear constraints (the max of 100 times the absolute violation divided by 1 plus the absolute constraint value). P_{OQ} is used as the quality measure of a population point.

While (stopping criteria are not satisfied)

While (some distinct pair of points in *R* has not been processed)

4. Select a new pair of points in *R*

5. Use a s*olution combination method* to produce a small number of trial solutions from this pair of points. Optionally, map each trial point into the closest point that satisfies the linear constraints and variable bounds.

6. At each (mapped) trial solution, evaluate the objective *f* and nonlinear constraint functions *G,* and form the penalty function, P_{OQ} **.**

Endwhile

7. Update the reference set.

8. If the reference set has changed, return to step 4. Otherwise, restart the procedure by selecting a subset (typically the best half) of the best points in the reference set to be retained as the set *R0*, and return to step 2.

Endwhile

3.1.2 Description of the Scatter Search Steps

Step 1 generates the starting points to create the initial reference set *R0*. The 3 points always appearing in this set are the vectors *x* for which all element are set to the upper bounds, to the lower bound, and to the midpoints of the bounds. If there is an initial point recommended to the problem, it is also added to *R0* as a fourth point.

Step 2 generates the remaining points to the initial reference set *R*. The *diversification generation method* begins by generating $nr>b$ randomly generated points in *S*, using a stratified sampling procedure described in [Laguna and Marti, 2000]. It then creates the reference set, *R,* by adding to *R0* the random point farthest from its nearest neighbor in *R0*, and repeating this process until *R0* has cardinality *b.* If the problem has linear constraints and the points selected are infeasible for these linear constraints, they are first projected onto the convex polyhedron defined by the linear constraints and then added to *R0*. This is done by finding the point in this polyhedron that is closest (using the L1 norm) to the infeasible point by solving a linear program. The result of this step is a diverse reference set that satisfies the linear constraints of the problem.

The initial population resulting from this procedure for a reference set of size $b = 10$ is shown in Figure 1, which uses data from a 2 variable unconstrained problem due to [Dixon and Szegö, 1975] called the six-hump camelback function. The objective to be minimized is

$$F(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4 \tag{7}$$

This is the problem EX8_2_5 from a large set of problems described in [Floudas, et al., 1999]. Problems from this set are used as test problems for OQNLP, and will be discussed in detail later. The problem has upper bounds of 10.0 and lower bounds of –10.0 on both variables, and has 6 local minima, all lying well within these bounds (see Figure 2 for their location), plus a stationary point at the origin that is neither a local minimum nor a maximum. The initial set *R0* is the three points (0,0), (10,10), (-10,-10), where (0,0) is user-supplied and the other two are the vectors of upper and lower bounds respectively.

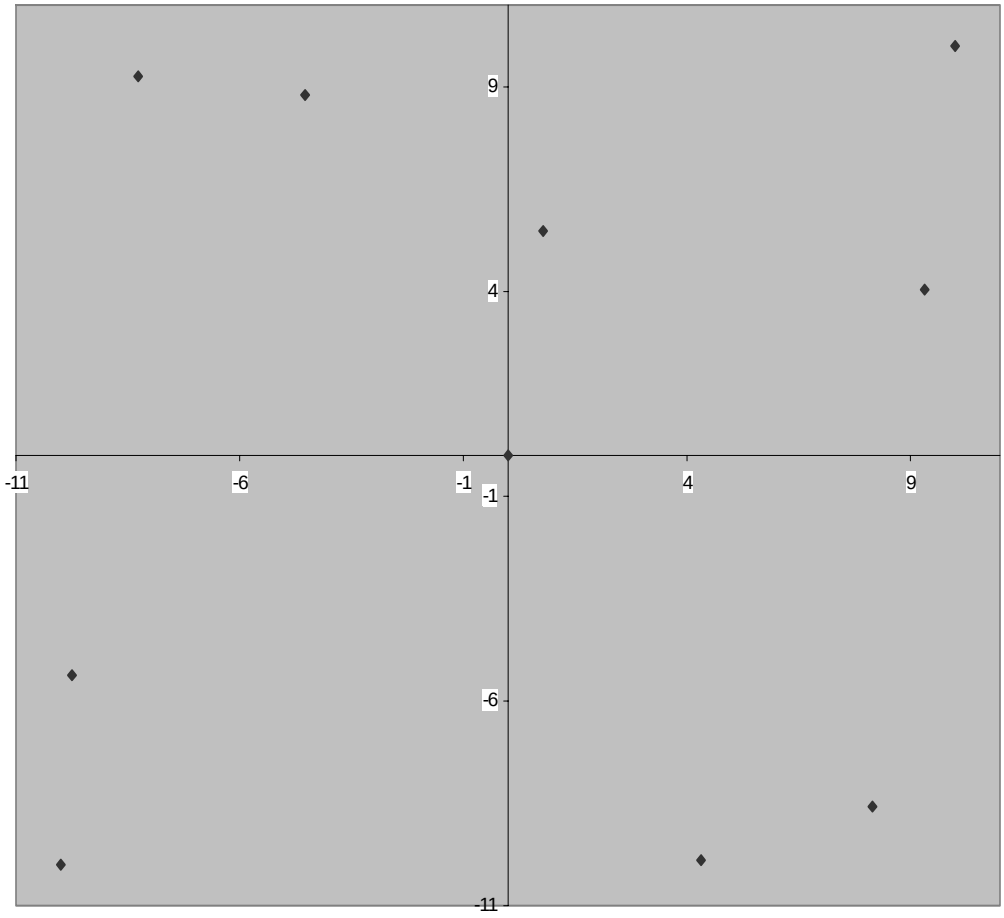

Figure 1: Initial Population for the Six-Hump Camelback Function

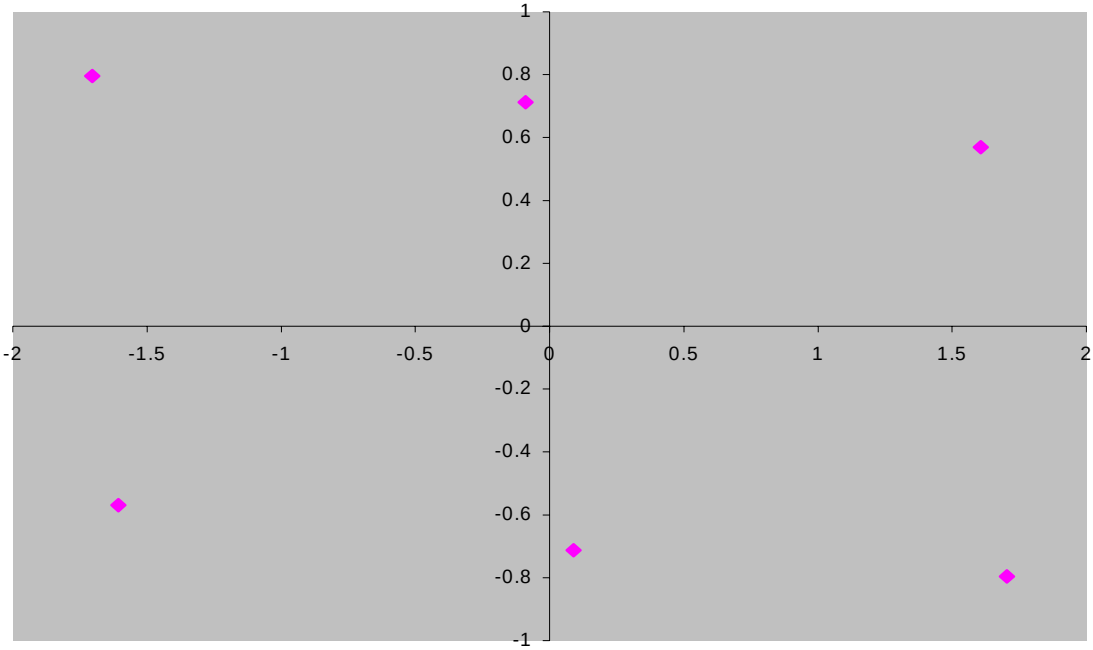

Figure 2: Locally Optimal Solutions for the Six-Hump Camelback Function

Step 3 ranks the points in the reference set based on their quality, measured by a penalty function P_{OQ} which is equal to the objective plus a penalty weight times the maximum percentage violation of the violated nonlinear constraints. The penalty function P_{OQ} is not the same as the exact penalty function P_1 described in (5), and is not exact. It is used because Lagrange multiplier information is assumed not to be available. Multipliers may not even exist if the problem is non-smooth or has discrete variables, and OptQuest is designed to solve such problems as well.

Steps 4 and 5 create new trial solutions by selecting 2 "parent" points from the reference set and performing the solution combination method on them. (If the option to always satisfy linear constraints is selected, the trial solutions are projected onto the linear constraints.) To illustrate how the combination method currently implemented in OptQuest works, Figure 3 demonstrates the generation of new trial points from the 3 best points of the initial population for the six-hump camel back function.

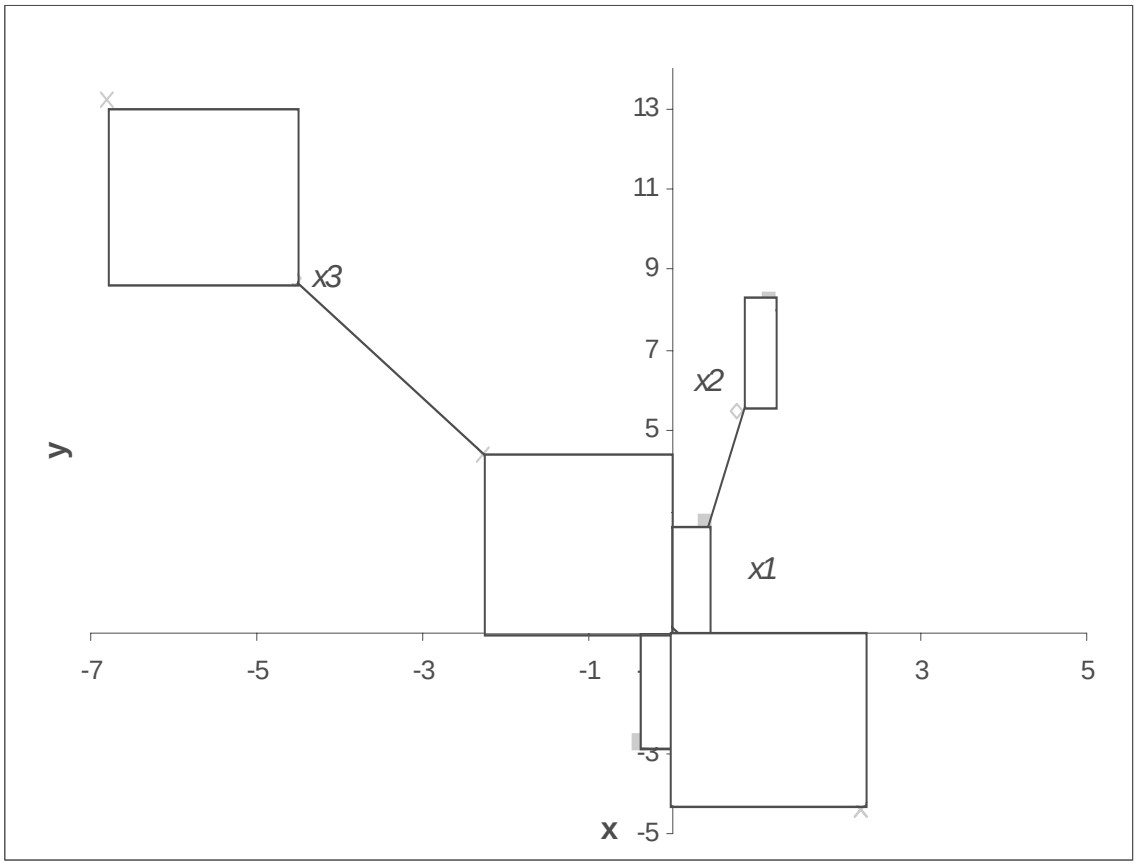

Figure 3: Trial Points for Six-Hump Camelback Function

The three points shown as diamonds and labeled *x1, x2, x3* (*x1* is the origin) have the lowest objective values in the initial population. The two lines in the figure are determined by the pairs of points *(x1, x2)* and *(x1, x3)*. Focusing on the line *(x1, x2)*, let

$$d = (x2-x1)/2$$

$$v1 = x1-d$$

$$v2 = x1$$

$$v3 = x1+d$$

$$v4 = x2$$

$$v5 = x2+d.$$

Thus *v3* is at the midpoint of this line, and *v1* and *v5* extend it beyond *x1* and *x2*. These points are shown as black or white squares in the figure. The points *v1* and *v2* can be used to define a hyper-rectangle whose 2^n vertices are the set

$$V = \{(z_1, z_2, ..., z_n) \mid z_i = (v1)_i \, or \, (v2)_i, i = 1, ..., n\} .$$

Thus the four pairs of points (*vi, v(i+1))* for i = 1,..,4 define 4 rectangles, three of which are shown on the line determined by *x1* and *x2*. OptQuest generates one randomly distributed trial point in each rectangle, and these points are shown as triangles. Three more trial points are generated in the same way starting with the points (*x1, x3).* These points lie "close" to the lines, but are not on them.

If there are discrete variables, the above process produces trial points whose values for these variables are not in the finite set of allowed values. These components are rounded to an allowable value using generalized rounding processes, i.e., processes where the rounding of each successive variable depends on the outcomes of previous roundings. For each discrete variable x_i, Optquest allows the definition of a lower bound (lo_i), an upper bound (up_i), and a step value (st_i). The step is the distance between two consecutive allowed values (it is usually equal to 1). Then, if old_x_i is the value to be rounded, the rounding process generates a first value new_x_i according to:

$$inc_i = \left\lfloor 0.5 + \frac{old\_x_i - lo_i}{st_i} \right\rfloor \quad , \quad new\_x_i = lo_i + inc_i \cdot st_i$$

If new_x_i is lower than up_i, then it is accepted as the rounded value. Otherwise, we compute it again, decreasing inc_i by one unit, which gives an allowed value.

The full set of 144 trial points generated from the initial population of this example is shown in Figure 4.

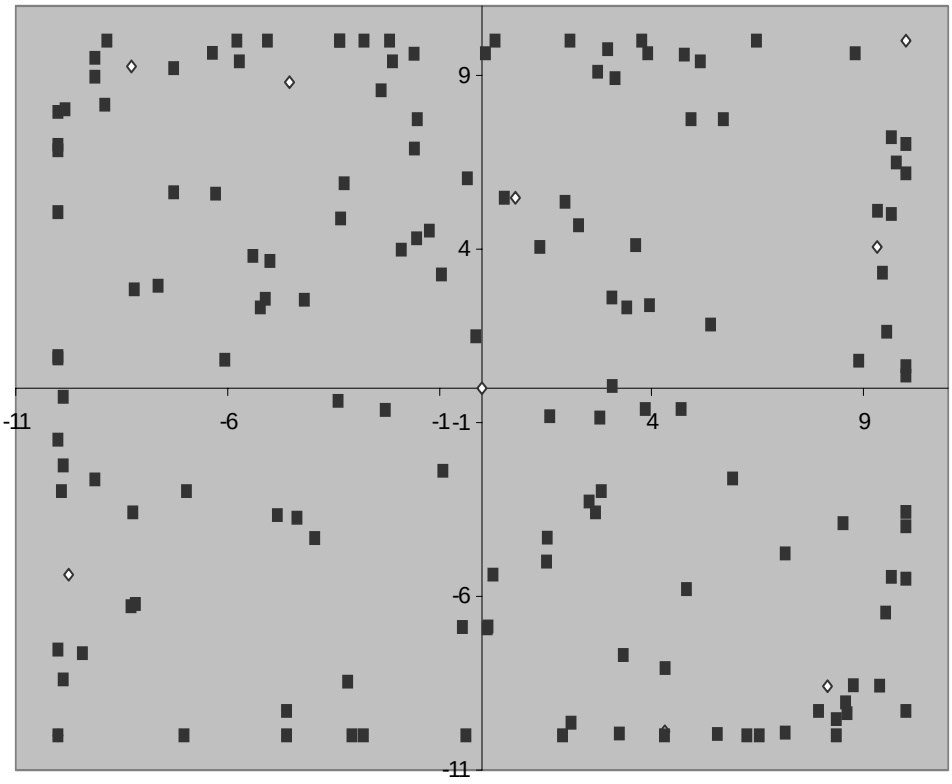

Figure 4: Initial Population and 144 Trial Points for Six-Hump Camelback Problem

The ten white diamond points are the members of the initial reference set, while the dark squares are the trial points, generated as described earlier. These are well scattered over the region defined by the bounds.

In Step 6 the objective f and non-linear constraints G are evaluated and the penalty function P_{OQ} is calculated. OptQuest considers f and G to be black boxes, and it is the responsibility of the user to provide the evaluation and return the corresponding values.

In step 7, after all trial points have been evaluated, the reference set is updated by replacing the population which generated the trial points by the best b points of the union of the trial points and the initial reference set, where best is determined by the OptQuest penalty function P_{OQ}. This is an aggressive update, emphasizing solution quality over diversity. This updated reference set, used to generate trial points from iteration 155 onward, is shown in Figure 5. The ten population points cluster in the region about the origin where the six local optima are located, so the next set of 144 trial points will lie within a slight expansion of this region. These trial points

thus have much better objective values than those generated by the initial population, as we illustrate later in section 6.

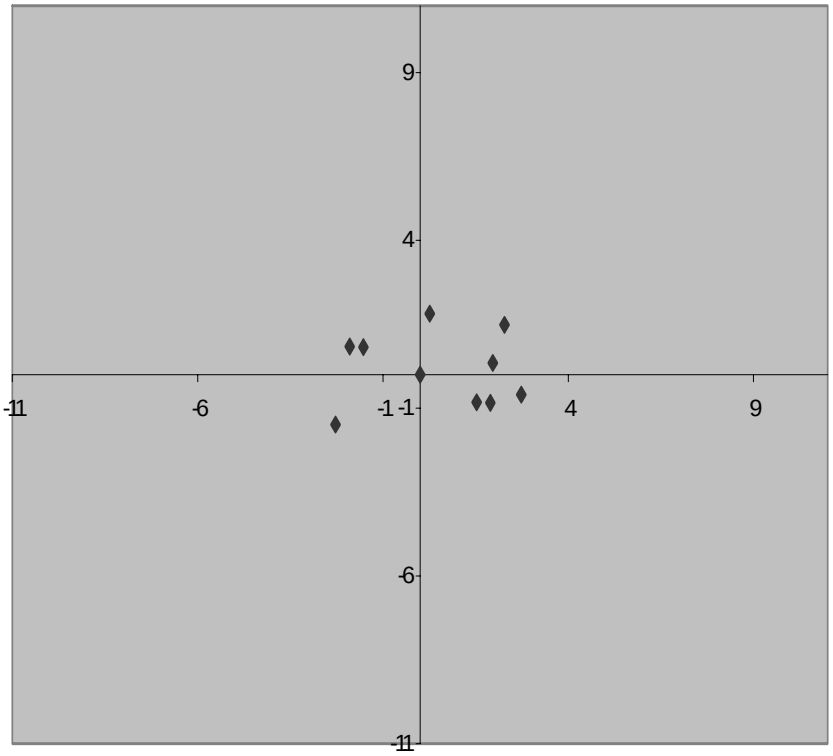

Figure 5: Second Reference Set for Six-Hump Camelback Function

When the diversity and quality of the reference are considered equally important, a different updating method is suggested in [Laguna and Marti, 2002]. In this variation, the reference set is split into 2 halves. The first half is created and maintained the same way as described earlier, focusing on the quality of the points. The other half contains diverse points. If a solution does not qualify to enter the first half of the reference set based on its quality, a test is performed to determine whether it fits the diversity criterion. That is, if the new point's minimum distance to any point in the second half of the reference set is larger than that of any points' already in the set, the new point will replace it. With this method the dynamic preservation of diversity is assured for the reference set.

If the updated reference set is unchanged, step 8 forces a return to Step 4 where a new diverse reference set is created. When this occurs, a number (typically $b/2$) of the best points from the current population are retained and newly generated points replace the rest.

3.2 The Local Phase – Gradient Based NLP Solvers

There are many papers and texts discussing gradient-based NLP solvers, e.g., [Nash and Sofer, 1996], [Nocedal and Wright, 1999], [Edgar, Himmelblau, and Lasdon, 2001]. These solve problems of the form (1)-(4), but with no discrete (y) variables. They require a starting point as input, and use values and gradients of the problem functions to generate a sequence of points which, under fairly general smoothness and regularity conditions, converges to a local optimum. The main classes of algorithms in widespread use today are Successive Quadratic Programming (SQP) and Generalized Reduced Gradient (GRG)-see [Edgar, Himmelblau, and Lasdon, 2001, Chapter 8.] The algorithm implemented in the widely used MINOS solver [Murtagh and Saunders, 1982] is similar to SQP. If there are nonlinear constraints, SQP and MINOS generate a sequence of points that usually violate the nonlinear constraints, with the violations decreasing to within a specified feasibility tolerance as the sequence converges to a local optimum. GRG algorithms have a simplex-like phase 1-phase 2 structure. Phase 1 begins with the given starting point and, if it is not feasible, attempts to find a feasible point by minimizing the sum of constraint violations. If this effort terminates with some constraints violated, the problem is assumed to be infeasible. However, this local optimum of the phase 1 objective may not be global, so a feasible point may exist. If a feasible point is found, phase 2 uses it as its starting point, and proceeds to minimize the true objective. Both phases consist of a sequence of line searches, each of which produces a feasible point with an objective value not worse (and usually better) than its predecessor.

Several good commercially available implementations of GRG and SQP solvers exist: see [Nash, 1998] for a review. As with any numerical analysis software, a local NLP solver can fail to find a local solution from a specified starting point. The problem may be too badly conditioned, badly scaled, or too large for the solver, causing it to terminate at a point (feasible or infeasible) which is not locally optimal. While the reliability of the best current NLP solvers is quite high, these difficulties occurred in our computational testing, and we discuss this in more detail later.

Let *L* be a local NLP solver capable of solving (1)-(4), and assume that *L* converges to a local optimum for any starting point $x_0 \in S$. Let $L(x_0)$ be the locally optimal solution found by *L* starting from x_0, and let x_i^*, *i = 1,2,...,nloc* be all the local optima of the problem. The basin of attraction of the *i*th local optimum relative to *L,* denoted by $B(x_i^*)$, is the set of all starting points in *S* from which the sequence of points generated by *L* converges to x_i^*. Formally:

$$B(x_i^*) = \{x_0 \mid x_0 \in S, L(x_0) = x_i^*\}$$

One measure of difficulty of a global optimization problem with unique global solution x_1^* is the volume of $B(x_1^*)$ divided by the volume of the rectangle, *S,* the *relative volume* of $B(x_1^*)$. The problem is trivial if this relative volume is 1, as it is for convex programs, and problem difficulty increases as this relative volume approaches zero.

3.3 Comparing Heuristic Search Methods and Gradient Based NLP Solvers

For smooth problems, the relative advantages of a heuristic search method like Scatter Search over a gradient-based NLP solver are its ability to locate an approximation to a good local solution (often the global optimum), and the fact that it can handle discrete variables. Gradient-based NLP solvers converge to the "nearest" local solution, and have no facilities for discrete variables, unless they are imbedded in a rounding heuristic or branch-and-bound method. Relative disadvantages of heuristic search methods are their limited accuracy, and their weak abilities to deal with equality constraints (more generally, narrow feasible regions). They find it difficult to satisfy many nonlinear constraints to high accuracy, but this is a strength of gradient-based NLP solvers. Search methods also require an excessive number of iterations to find approximations to local or global optima accurate to more than 2 or 3 significant figures, while gradient-based solvers usually achieve 4 to 8-digit accuracy rapidly.

The motivation for combining search and gradient-based solvers in a multi-start procedure is to achieve the advantages of both while avoiding the disadvantages of either. Surprisingly, we have been unable to locate any published efforts in this direction, besides the Frontline extended MLSL method discussed in Section 2.

3.4 The OQNLP Algorithm

A pseudo-code description of the simplest OQNLP algorithm follows:

INITIALIZATION

Read_Problem_Parameters (n, p, m_1, m_2, bounds, starting point);

Setup_OQNLP and OptQuest_Parameters and Options(problem size, stage 1 and 2 iteration limits, population size, accuracy, names and types of variables and constraints, bounds on variables and constraints);

Initialize_OptQuest_Population;

Stage 1 iterations = Stage 2 iterations = 0;

STAGE 1: INITIAL OPTQUEST ITERATIONS AND FIRST L CALL

WHILE (Stage 1 iterations < Stage 1 iteration limit)

DO {

Get (trial solution from OptQuest);

Evaluate (objective and nonlinear constraint values at trial solution,);

Put (trial solution , objective and constraint values to OptQuest database);

Stage 1 iterations = Stage 1 iterations + 1;

} ENDDO

Get_Best_Point_from_OptQuest_database (starting point);

Call_*L* (starting point, local solution);

Threshold = default value;

IF (local solution feasible) **THEN** **{**

Insert local solution in linked list;

Penalty weights = max(positive lower limit, absolute multiplier values from *L* call);

threshold = P_1 value of local solution; **}**

Penalty weights = max(positive lower limit, absolute multiplier values from *L* call)

STAGE 2: MAIN ITERATIVE LOOP

WHILE (Stage 2 iterations < Stage 2 iteration limit)

DO {

Get (trial solution from OptQuest);

Evaluate (objective and nonlinear constraint values at trial solution,);

Put (trial solution, objective and constraint values to OptQuest database);

Calculate_ Penalty_ Function (trial solution, Penalty weights, P_1);

IF (distance and merit filter criteria are satisfied) **THEN {**

Replace threshold with current P_1 value;

Call_*L* (trial solution, local solution);

IF (local solution feasible) **THEN {**

Insert local solution in linked list;

Penalty weights = max(positive lower limit, absolute multiplier values from *L* call); **}**

}

ELSE IF (P_1 > threshold for *waitcycle* consecutive iterations) {increase *threshold*}

Stage 2 iterations = Stage 2 iterations + 1;

} ENDDO

After initialization, there are two main stages. In the "initial OptQuest iterations" stage, the objective and constraint values at all trial points generated by the initial OptQuest population (including the population points themselves) are evaluated, and these values are returned to OptQuest, which computes its penalty function, P_{OQ}, at each point. The point with the best P_{OQ} value is selected, and *L* is started from this point. If there are any discrete variables, *y*, they are fixed at their current values during the *L* solution process. Figure 4 shows a graph of these trial points for a two variable unconstrained problem. In general, they are scattered within the rectangle defined by the bounds on the variables, so choosing the best corresponds to performing a coarse search over this rectangle. If the best point falls inside the basin of attraction of the global optimum relative to *L* (as it often does), then if the subsequent *L* call is successful, it will find a global optimum. This call also determines optimal Lagrange multiplier values, u^*, for the constraints. These are used to determine initial values for the penalty weights, w_i, satisfying (6), which are used in the exact penalty function, P_1, defined in (5). All local optima found are stored in a linked list, along with the associated Lagrange multipliers and objective values. Whenever a new local optimum is found, the penalty weights are updated so that (6) is satisfied over all known local optima.

The main iterative loop of stage 2 obtains trial points from OptQuest, and starts *L* from the subset of these points determined by two filters. The distance filter helps insure that these starting points are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent *L* from starting more than once within the basin of attraction of any local optimum, so it plays the same role as the rule in the MLSL algorithm of Section 2, which does not start at a point if it is within a critical distance of a better point. When the final point found by *L* is feasible, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, *maxdist,* is updated and stored. For each trial point, *t,* if the distance between *t* and any local solution already found is less than *distfactor*maxdist, L* is not started from the point, and we obtain the next trial solution from OptQuest.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least *maxdist.* The default value of *distfactor* is 0.75, and it can be set to any positive value. As *distfactor* approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it increases, the filtering effect increases until eventually *L* is never started in stage 2.

The merit filter helps insure that the *L* starting points have high quality, by not starting from candidate points whose exact penalty function value P_1 in (5) is greater than a threshold. This threshold is set initially to the P_1 value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than *waitcycle* consecutive iterations, the threshold is increased by the updating rule:

$$\text{threshold} \leftarrow \text{threshold} + \textit{threshfactor}*(1.0+\text{abs}(\text{threshold}))$$

where the default value of *threshfactor* is 0.2 and that for *waitcycle* is 20. The additive 1.0 term is included so that *threshold* increases by at least *threshfactor* when its current value is near zero. When a trial point is accepted by the merit filter, *threshold* is decreased by setting it to the P_1 value of that point.

The combined effect of these 2 filters is that *L* is started at only a few percent of the OptQuest trial points, yet global optimal solutions are found for a very high percentage of the test problems. Some insight is gained by examining Figure 6, which shows the stationary point at the

origin and the 6 local minima of the 2 variable six-hump camelback function defined in (7) as dark squares, labeled with their objective value. The ten points from which OQNLP starts the local solver are shown as nine white diamonds, plus the origin. The local minima occur in pairs with equal objective value, located symmetrically about the origin. There were 144 trial points generated in the "initial OptQuest iterations" stage, and these, plus the 10 points in the initial population, are shown in Figure 4. The best of these 154 points is the population point (0,0), so this becomes the first starting point for the local solver. This happens to be a stationary point of F, so it satisfies the optimality test (that the norm of the gradient of the objective be less than the optimality tolerance), and the local solver terminates there. The next local solver start is at iteration 201, and this locates the global optimum at (.0898, -.7127), which is located two times. The other global optimum at (-.0898, .7127) is found first at iteration 268, and is located 6 times.

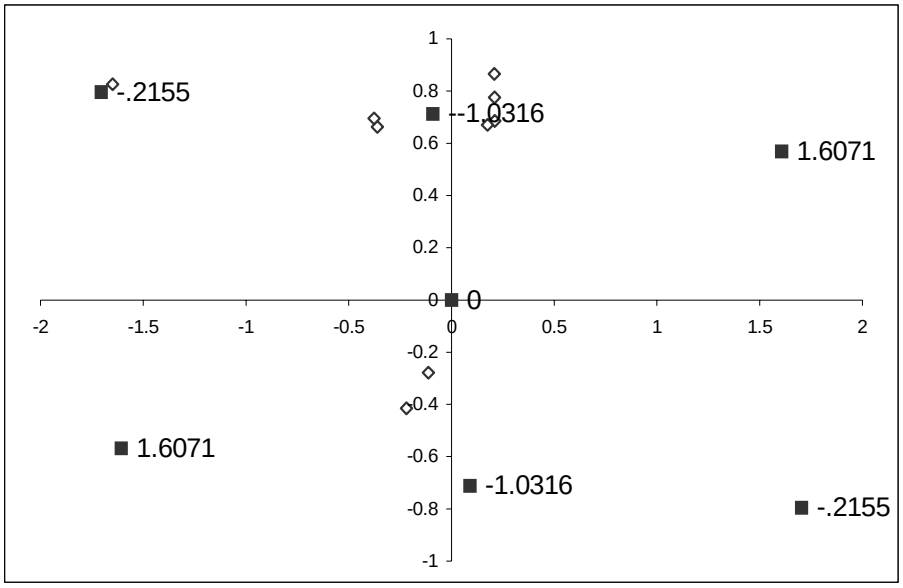

Figure 6: Local Optima and 10 L Starting Points for 6 hump camelback function

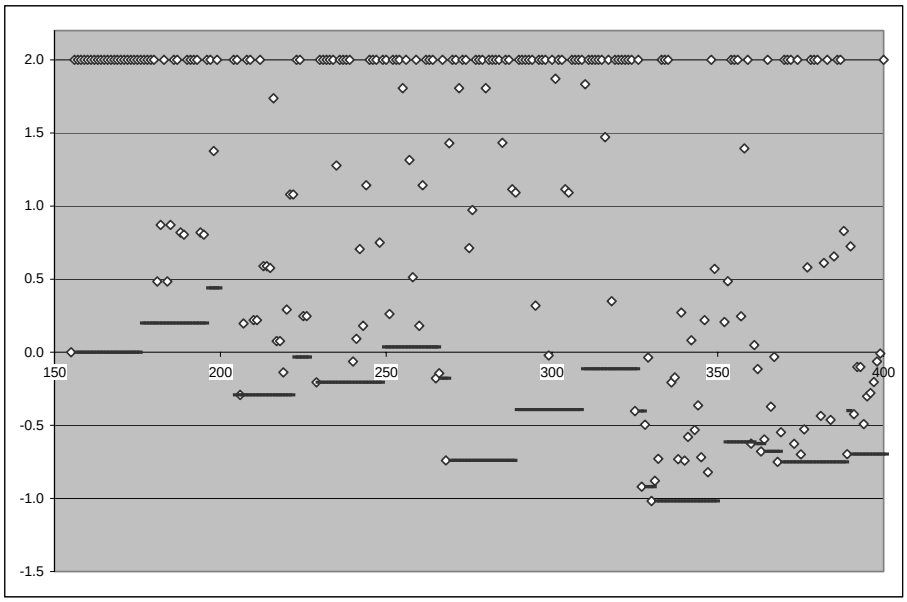

Figure 7: Objective and Threshold Values for Six-Hump Camelback Function for Iterations 155 to 407

The limit on total *OQNLP* iterations in this run was 1000. *L* was started at only 9 of the 846 OptQuest trial points generated in the main iterative loop of stage 2. All but 2 of the starting points are in the basin of attraction of one of the two global optima. This is mainly due to the merit filter. In particular, the threshold values are always less than 1.6071, so no starts are ever made in the basin of attraction of the two local optima with this objective value. The merit filter alone rejected 498 points, the distance filter alone 57, and both rejected 281.

Figure7 illustrates the dynamics of the merit filtering process for iterations 155 to 407 of this problem, displaying the objective values for the trial points as white diamonds, and the threshold values as dark lines. All objective values greater than 2.0 are set to 2.0.

The initial threshold value is zero, and it is raised twice to a level of 0.44 at iteration 201, where the trial point objective value of -0.29 falls below it. *L* is then started and locates the global optimum at (.0898, -.7127), and the threshold is reset to –0.29. This cycle then repeats. Nine of the ten *L* starts are made in the 252 iterations shown in the graph. In this span, there are 12 points where the merit filter allows a start and the threshold is decreased, but *L* is not started at three of these because the distance filter rejects them.

Figure 8 shows the same information for iterations 408 to 1000. There is only one *L* start in this span. This is not due to a lack of high quality trial points: there are more good points than previously, many with values near or equal to –1.0310 (the global minimum is –1.0316), and the merit threshold is usually –1.0310 as well. Every time this threshold is raised, the merit filter accepts one of the next trial points, but 51 of the 52 accepted points are too near one of the 2 global optima, and they are rejected by the distance filter.

This simple example illustrates a number of important points:

1. Setting the bounds on the continuous or discrete variables to be too large in magnitude is likely to slow the *OQNLP* algorithm (or any search algorithm) and may lead to a poorer final solution. In the above example, if the variable bounds had been [-2,2] rather than [10,10], the trial points generated by the initial population would have had much lower objective values. OptQuest can overcome this when the initial population is updated.
2. *L* found a highly accurate approximation to the global solution of this unconstrained problem at its second call. OptQuest alone would have taken many more iterations to achieve this accuracy.
3. The best trial point generated by the initial population may not have as good an objective value as those generated from the second or succeeding ones, especially if the variable bounds are too large. Using the best "first generation" point as the initial *L* starting point may not lead to as good a local solution as if some "second generation" points had been considered. For this reason our base case computational results use a first stage of 200 OptQuest trial points, which in this example would include all 144 first generation points and 56 from the second generation.

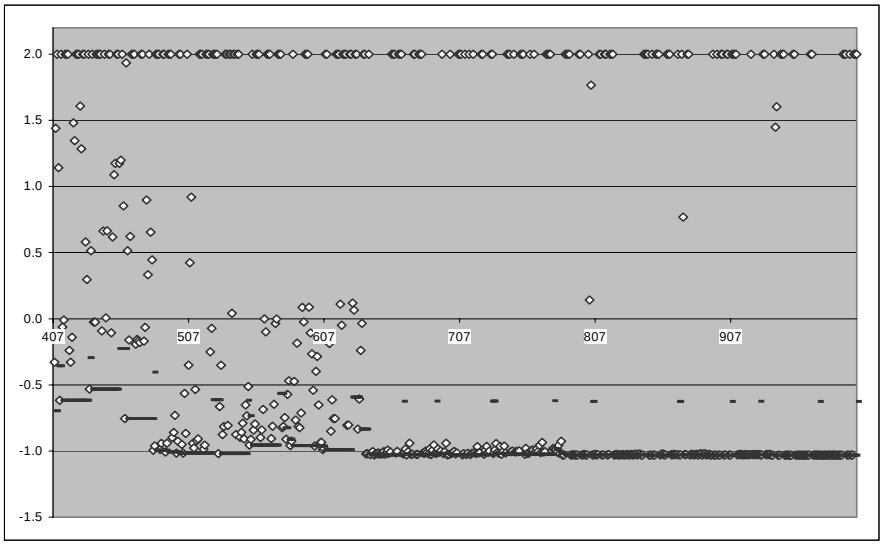

Figure 8: Objective and Threshold Values for Six-Hump Camelback Function: iterations 408 to 1000

3.5 Filtering Logic for Problems with Discrete Variables

The filtering logic described above must be extended when there are discrete variables (the *y* variables in the problem statement (1)-(4)). There are 2 distinct modes: (1) Optquest is aware of both the x and y variables and all problem constraints and (2) Optquest is aware only of the y variables and the constraints involving y only. Tests thus far do not show conclusively that one of these is preferred, so both modes are described and comparisons thus far are presented later.

In mode one, when a trial point (x_t, y_t) provided by OptQuest passes the two filtering tests and is passed to the local solver *L*, x_t acts as a starting point and is changed by *L*, but the y_t values are fixed and are not changed. Each new set of y_t values defines a different NLP for *L* to solve, say NLP(y_t), with its own set of local minima in *x* space, so both filters must be made specific to NLP(y_t). For the distance filter, it is irrelevant if x_t is close to any local minima (in *x* space) previously found which correspond to problems NLP(*y*) with *y* different from y_t. Hence the distance filter is based on the distance from *xt* to local minima of NLP(y_t) only. Similarly, the tests and threshold values in the merit filter must be specific to the problem NLP(y_t) currently being solved. However, the weights *w* in the exact penalty function $P_1(x,y,w)$ used in the merit filter are based on the maximum absolute multipliers over *all* local optima for *all* vectors y_t,

because these weights are large enough to ensure that this function is exact for all problems NLP(y).

Therefore, in stage 2 of the algorithm, the exact penalty function, $P_1(x_t,y_t,w)$, is calculated at each trial point (x_t,y_t), and L is started at (x_t,y_t) if P_1 is smaller than the current threshold for NLP(y_t). This threshold is initialized to plus infinity, so if the values y_t have not occurred in a previous stage 2 trial point, L will be called at this point. This leads to many more local solver calls in problems with discrete variables, as we show later in the computational results sections.

In mode two, Optquest presents candidate y vectors only to L, which are fixed while L finds corresponding (locally) optimal x values. The starting values for x can be chosen to minimize computational effort. We are experimenting with an option which obtains all possible trial points for the current population, sorts them in terms of their distance from each other, and calls L in that sorted order, starting each call of L from the previous optimum. It is expected that y's which are close to one another will have x's with that property, so the previous optimum will be a good starting point.

In mode two, there is no stage 1, and L must be called at each y vector that has not been produced previously. As a result, the local solver call where the best value is found typically comes later than with mode 1. On the other hand, OptQuest's effort is reduced since it processes a much smaller problem, and the information returned to it by the local solver (the optimal objective value over the continuous variables) is of much higher quality than in the base case (the penalized objective value at OptQuest's trial point).

An important option involves the return of information from the local solver to OptQuest, which is absent in the above procedure, i.e., local solutions found by the local solver are not returned to OptQuest. Such solutions are generally of very high quality, and might aid the search process if they were incorporated into the OptQuest population, because at least a subset would likely be retained there. However, this should be done so as to preserve the diversity of the population.

4. Computational Results

The algorithm described in the previous section has been implemented as a callable C-language function. In this form, the user supplies a C function that evaluates the objective and constraint functions, an optional routine that evaluates their first partial derivatives (finite difference

approximations are used otherwise), and a calling program that supplies problem size, bounds, an initial point, and invokes the algorithm. Algorithm parameters and options all are set initially to default values, and any changes are specified in an options file.. The local NLP solver is the LSGRG2 implementation of [Smith and Lasdon, 1992]. We have also developed an interface between our C implementation and the GAMS algebraic modeling language (see www.gams.com), using C library routines provided by GAMS Development Corporation. The user function routine is replaced by one that calls the GAMS interpreter, and a special derivative routine accesses and evaluates expressions developed by GAMS for first derivatives of all nonlinear problem functions. GAMS identifies all linear terms in each function, and supplies their coefficients separately, thus identifying all linear constraints. This enables us to invoke the OptQuest option which maps each trial point into a point which satisfies the linear constraints. The derivative information supplied by GAMS significantly enhances the performance of the local solver since only non-constant derivatives are re-evaluated, and these are always available to full machine precision. As mentioned earlier, this GAMS version can call any GAMS NLP solver.

For our computational experiments we used the large set of global optimization test problems coded in GAMS from [Floudas et al., 1999]. Table 1 shows the characteristics of 142 individual and 2 groups of problems.

Table 1: Floudas Test Problem Set Characteristics

Series	Problems	Max vars	Max discrete vars	Max linear constraints	Max nonlinear constraints	Problem Type
EX2_1_x	14	24	0	10	0	concave QP (min)
EX3_1_x	4	8	0	4	6	quadratic obj and constraints
EX4_1_x	9	2	0	0	2	obj or constraints polynomial
EX5_2_x	2	32	0	8	11	bilinear-pooling
EX5_3_x	2	62	0	19	34	distillation column sequencing
EX5_4_x	3	27	0	13	6	heat exchanger network
EX6_1_x	4	12	0	3	6	Gibbs free energy min
EX6_2_x	10	9	0	3	0	Gibbs free energy min
EX7_2_x	4	8	0	3	12	generalized geometric prog
EX7_3_x	6	17	0	10	11	robust stability analysis
EX8_1_x	8	6	0	0	5	small unconstrained, constrained
EX8_2_x	5	55	0	6	75	batch plant design-uncertainty
EX8_3_x	14	141	0	43	65	reactor network synthesis
EX8_4_x	8	62	0	0	40	constrained least squares
EX8_5_x	6	6	0	2	2	min tangent plane distance
EX8_6_1	N	3N	0	0	0	Lennard-Jones energy min
EX8_6_2	N	3N	0	0	0	Morse energy min
EX9_1_x	10	29	6	27	5	bilevel LP
EX9_2_x	9	16	3	11	6	bilevel QP
EX12_2_x	6	11	8	9	4	MINLP
EX14_1_x	9	10	0	4	17	infinity norm solution of equations
EX14_2_x	9	7	0	1	10	infinity norm solution of equations
Total:	142 + 2N					

Most problems arise from chemical engineering, but some are from other sources. Most are small, but a few have over 100 variables and comparable numbers of constraints, and 13 have both continuous and discrete variables. Almost all of the problems without discrete variables have local solutions distinct from the global solution, and the majority of problems have constraints. Sometimes all constraints are linear, as with the concave quadratic programs of series EX2_1_x, but many problems have nonlinear constraints, and these are often the source of the non-convexities. The best-known objective value and (in most cases) the corresponding

variable values are provided in [Floudas, et al., 1999]. The symbol N in the rows for the series EX8_6_1 and EX8_6_2 is the number of particles in a cluster whose equilibrium configuration is sought via potential energy minimization. Each particle has 3 coordinates, so there are 3N variables.

4.1 Continuous Variables-The Base Case

This section describes results obtained when OQNLP is applied to 128 of the problems in the Floudas et. al. test set with no discrete variables. A few problems for which no GAMS NLP solver can find a feasible solution in 800 solver calls are omitted. 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 options and main algorithm parameters used are shown in Table 2 (see Section 3.4 for definitions). The filter parameter values (*waitcycle, threshfactor, distfactor*) correspond to fairly tight filters, and these must be loosened to solve some problems. The OptQuest "use linear constraints" option, which projects trial points onto the linear constraints, is not used because it is very time consuming for larger problems. SNOPT, an SQP implementation, was used for the largest problems because many calls to the GRG solvers CONOPT and LSGRG2 terminate infeasible on these problems. The 8_3_x problems include many "pooling" constraints, which have bilinear terms. In almost all these terminations, the GRG solvers find a local minimum of the phase 1 objective. SNOPT has no phase 1, and never terminates infeasible.

Table 2: Base case OQNLP and OptQuestGRG Parameters and Options Used

Total iterations = 1000
Stage 1 iterations = 200
Waitcycle = 20
Threshfactor = 0.2
Distfactor = 0.75
Use linear constraints = no
OptQuest search type = *boundary*
Boundary search parameter = 0.5
NLP solver = LSGRG2 except SNOPT for 110 to 141 range

Table 3 shows outcomes and average effort statistics for 128 of the Floudas et. al. collection of test problems with continuous variables only, sorted into 6 groups by number of variables. Geometric rather than arithmetic means are used to reduce the effects of outliers: function calls, iterations, and times for the larger problem sets typically include a few problems with values much larger than all others. Computational effort is measured by OptQuest iterations, solver calls, function calls (each function call evaluates the objective and all constraint functions), and computation time. The three "to best" columns show the effort required to find the best OQNLP objective value. Function calls are not available for the largest problems because the SNOPT interface does not yet make them available.

Table 3: Results and Effort Statistics for 128 Continuous Variable Floudas Problems

Variable range	Problems	Variables	Constraints	Iterations to best	Solver calls to best	Solver calls	Locals found	Function calls to best	Function calls	Time to best	Total time	Failed	First *L call*	Second *L* call
1 to 4	32	2.5	1.7	206.3	1.1	7.5	1.9	263.9	2158.0	0.1	0.5	1	27	3
4 to 7	31	5.5	5.7	214.4	1.3	5.8	1.9	381.5	4766.7	0.2	0.6	0	22	6
8 to 12	21	9.4	8.1	238.2	1.5	13.2	3.0	575.6	19698.0	0.1	0.8	3	10	4
13 to 20	18	15.9	11.7	303.5	2.4	7.4	2.6	968.2	5211.9	0.3	0.7	4	8	1
22 to 78	13	37.9	27.6	259.6	2.1	14.1	3.1	1562.4	23077.9	0.6	2.5	1	5	3
110 to 141	13	116.4	80.1	305.0	2.7	23.7	22.7	NA	NA	6.6	64.1	0	7	2
Total/avg	128			251.3	1.8	10.5	3.5	537.9	7190.9	0.4	1.7	9	80	19

Since all problems have known solutions, we define "failures" as problems with a solution gap of more than 1%. This gap is the percentage difference between the best feasible OQNLP objective value, *fOQNLP*, and the best-known feasible objective value, *fbest*, defined for minimization problems as:

$$gap = \frac{100\ (\ foqnlp\ -\ fbest\ )}{1 + abs\ (\ fbest\ )}$$

and the negative of the above for maximization, so positive gaps indicate that the best known solution was not reached. Nine of the 128 problems failed to achieve gaps smaller than 1%, with gaps ranging from 2.2% to 80%. All these are solved with more iterations or by loosening the filters. Percentage gaps for almost all 119 "solved" problems are less than 1.e-4, and the largest gap among solved problems is 0.37%.

Computational effort needed to achieve these results is quite low, and increases slowly with problem size, except for the geometric mean solution time for the largest problems. The best OQNLP value is also found very early: in the first solver call in 80 of the 118 solved problems, and the second call in 19 more. This shows that, for these test problems, stage one of OQNLP is very effective in finding a point in the basin of attraction of the global optimum. The ratio of the "to best" effort to total effort is also small. For iterations, since there are always 200 stage 1 iterations, we subtract 200 before computing the ratio, giving 51.3/800 = 0.06. The solver call ratio is 0.17 and the time ratio 0.23. This implies that, for these problems, a criterion that stops OQNLP when the fractional change in the best feasible objective value found thus far is below a

small tolerance for some (reasonably large) number of successive iterations would rarely terminate the algorithm before the best solution was found. The ratio of total solver calls to locals found, a measure of filter efficiency, varies from 3 to 5, and is nearly one for the largest problems.

Table 5 shows results obtained in solving the 9 "failed" problems with looser filters and an OptQuest boundary search parameter of 1. Seven of these 9, the 2_1_x series, are quadratic programs (QP's) with concave objectives (to be minimized), so each has an optimal extreme point solution, and every extreme point is a local solution. The base case and new parameter values are in Table 4 below.

Table 4: Base Case and Loosened Filter Parameter Values

Parameter	**Base Case Value**	**Looser Value**
Waitcycle	20	10
Threshold_factor	0.2	1.0
Distance_factor	0.75	0.1
Boundary Search Parameter	0.5	1.0

The looser merit filter increases its threshold every 10 iterations, replacing the old value by *old value+1.0*(1+abs(old value)).* The looser distance filter rejects a trial solution if its distance from any previously found local solution is less than *0.1*maxdist*, where *maxdist* is the largest distance traveled to reach that solution. A search parameter of 1 causes more OptQuest trial points to have values on the boundary of the rectangle defined by the variable bounds, which helps solve the seven concave QP's.

Eight of the nine "unsolved" problems are solved with these new parameters, and the other, EX14_1_8, achieves a gap of 1.15%. It is solved by using 1000 stage one iterations and 5000 total, with all other parameters as in the base case.

Table 5: Solving 9 "failed" problems with looser filters and boundary parameter=1

Problem name	Variables	Constraints	Iterations to best	Solver calls to best	Total solver calls	Base case solver calls	Locals found	Function calls to best	Total function calls	Time to best	Total time	Gap
EX14_1_8	3	4	338	21	122	17	2	38141	1218680	0.49	2.25	1.15
EX9_2_5	8	7	344	2	6	45	3	690	2832	0.21	0.6	0
EX2_1_6	10	5	203	3	107	7	17	389	138749	0.2	1.38	0
EX2_1_9	10	1	201	1	111	66	39	252	342345	0.16	1.45	0
EX2_1_7_1	20	10	279	8	27	2	19	5051	60043	0.39	1.25	0
EX2_1_7_3	20	10	269	4	65	22	36	1321	286213	0.3	1.92	0
EX2_1_7_4	20	10	253	6	29	4	9	3676	83989	0.39	1.41	0
EX2_1_7_5	20	10	254	5	29	3	15	2620	71004	0.33	1.34	0
EX2_1_8	24	10	226	3	120	8	25	981	1071730	0.29	2.87	0
Means(geom)			258.6	4.1	48.7	10.5	12.8	1760.7	137876.8	0.3	1.5	

Comparing the "total solver calls" and "base case solver calls" columns shows that the new parameters represent a substantial loosening of both filters. The looser filters result in many more solver calls in all but problem 9_2_5, and the geometric mean solver calls is 48.7 with the loose filters versus 10.5 with the tighter ones. The behavior of 9_2_5 is surprising (6 solver calls with loose filters versus 45 with tighter ones), but the run with looser filters finds the global minimum at iteration 344, and after that its merit thresholds and set of local solutions differ from those of the base case run.

Table 6 below shows the geometric performance means and totals obtained from solving the 14 concave QP problems with base case parameters, with and without the OptQuest "use linear constraints" option, which maps each trial point into a nearest point feasible for the linear constraints. Since these are linearly constrained problems, invoking this option guarantees that all trial points are feasible.

Table 6: Solving Concave QP problems with and without "use linear constraints"

Case	Iterations to best	Solver calls to best	Total Solver calls	Locals found	fcn calls to best	Total fcn calls	Time to best	Total time	Failed
no use	284.8	2.3	6.6	3.7	643.8	3875.1	0.3	0.6	7
use	247.1	2.1	12.1	3.1	437.7	3827.6	6.9	19.0	2

Clearly this option helps: there are roughly twice as many solver calls on average when using it, and only 2 failures, versus 7 in the base case. The gaps for the 2 unsolved problems (2_1_7_5 and 2_1_9) are between 1% and 3.5% in both cases. However, this option increases run times here by about a factor of 30, so it is currently off by default.

4.2 The Lennard-Jones and Morse Energy Minimization Problems

The Floudas et. al. set of test problems includes two GAMS models that choose the locations of a cluster of N particles to minimize the potential energy of the cluster, using two different potential energy functions, called Lennard-Jones and Morse. The decision variables are the (x,y,z) coordinates of each particle. Particle 1 is located at the origin, and three position components of particles 2 and 3 are fixed, so each family of problems has 3N-6 variables. These problems have many local minima, and their number increases rapidly with problem size, so they constitute a good test for global optimization algorithms.

Results of applying OQNLP to 14 of these problems, using 200 stage 1 and 1000 total iterations, are shown in Tables 7 and 8. Each problem set was solved with LSGRG2 and CONOPT. These results use CONOPT for the Lennard-Jones problems and LSGRG2 for the Morse, because they provide slightly better results, illustrating the value of being able to call several solvers. Because of the many distinct local minima, the number of local minima found is equal to the number of solver calls for the 3 largest Lennard-Jones problems and for all the Morse problems,

Table 7: Solving 6 Lennard-Jones Problems Using CONOPT and Loose Filters

Problem name	Variables	Constraints	Solver calls to best	Total Solver calls	Locals found	Time to best	Total	Gap,%
EX8_6_1_5	9	10	1	152	39	1.09	21.83	0.00
EX8_6_1_10	24	45	21	130	114	18.56	68.34	0.00
EX8_6_1_15	39	105	6	104	100	13.63	165.09	0.00
EX8_6_1_20	54	190	67	118	118	257.62	396.21	1.12
EX8_6_1_25	69	300	42	94	94	325.82	730.68	1.84
EX8_6_1_30	84	435	16	59	59	134.35	434.56	0.88

The Lennard-Jones problems are the more difficult of the two. The 3 largest problems have gaps of roughly 1% to 2%, using the looser filter parameters in Table 4. The default filter

parameters led to positive gaps for the last 3 problems totaling 7.8%, while this sum in Table 7 is 3.8%. The objective approaches infinity as the distance between any 2 particles approaches zero, so its unconstrained minimization for N=20 leads to about 50,000 domain violations (either divide by zero or integer power overflow), and this number grows rapidly with N. Hence we added constraints lower bounding this distance by 0.1 for all distinct pairs of points, and the number of these constraints is shown in the table. None are active at the best solution found.

Table 8 shows the Morse potential results using the LSGRG2 solver and the default OQNLP parameters shown in Table 2. The objective here has no singularities, so there are no difficulties with domain violations, and the only constraints are variable bounds. All problems are solved to very small gaps except the largest (144 variables), which has a gap of .125%. The number of solver calls is much smaller than for the Lennard-Jones problems, because the filters are much tighter. Each call leads to a different local optimum. The largest problem is solved to a gap less than 1.e-4% with 5000 total and 1000 stage 1 iterations and the same filter parameters. This run terminated because the 3000 second time limit was exceeded, took 4083 iterations, and found 210 distinct local optima in 210 solver calls, compared to only 25 in the base case.

Table 8: Solving 8 Morse Problems Using Lsgrg2 and default Parameters

Problem name	Variables	Solver calls to best	Total Solver calls	Locals found	Time to best	Total time	Gap
EX8_6_2_5	9	1	5	5	0.23	0.61	0.0000
EX8_6_2_10	24	1	15	15	0.57	4.44	0.0000
EX8_6_2_15	39	1	6	6	1.41	6.43	0.0000
EX8_6_2_20	54	2	43	43	4.2	51.2	0.0000
EX8_6_2_25	69	4	20	20	13.44	58.38	0.0000
EX8_6_2_30	84	17	43	43	68.56	160.19	0.0000
EX8_6_2_40	114	7	33	33	66.29	273.91	0.0000
EX8_6_2_50	144	20	25	25	337.2	403.96	0.1251

4.3 Problems with Discrete Variables

There are 11 MINLP problems in the Floudas et. al. test set, with the total number of variables ranging from 3 to 29 and the number of binary variables ranging from 1 to 8. Two of these, EX12_2_3 and EX12_2_4, had been reformulated so that all binaries appeared linearly, and we restored them to their original state where the binaries appear nonlinearly. OQNLP allows such

representations, while the other GAMS MINLP solvers do not. The final test set contains 13 problems. These are far too small to yield meaningful inferences about the power of OQNLP on problems of practical size, but allow preliminary testing of the two MINLP modes described in Section 3.5. The geometric means of some measures of computational outcomes and effort for both modes are shown in Table 9, using the LSGRG2 NLP solver.

Table 9: Solution Statistics for 13 Problems with Discrete Variables

Discrete var mode	**Options and Parameters**	**Iterations to best**	***L* calls to best**	**Total *L* calls**	**Fcn calls to best**	**Total fcn calls**	**Time to best**	**Total time**	**Failures**
discrete only	default	10.2	10.2	20.1	2065.8	5794.1	0.8	1.5	0
all	default	261.7	4.0	32.1	1065.7	25022.2	0.3	0.8	7
all	(1000,5000)	1551.9	12	89.1	10196.8	224025.7	0.8	3.7	1
all	default,use	272.4	3.6	115.8	1178.0	88983.0	9.9	26.1	0

The first table row is for mode 2, where OptQuest manipulates only the discrete variables. Each NLP problem was "cold started" from the same initial point in these runs, so the number of function calls could be reduced substantially by warm starts. All runs are terminated by OptQuest after the small number of possible binary variable combinations have been completely enumerated. The optimal solution is found on average about midway through the solution process, but we expect that this will occur earlier as the number of discrete variables increases. The OptQuest logic requires that at least one population of binary solutions be evaluated before any learning can occur, and the average number of solver calls to find the best solution here is about equal to the population size of 10.

The last 3 rows of Table 9 show results for mode 1, where OptQuest manipulates both binary and continuous variables. In rows 2 and 3, we do not require trial points to satisfy linear constraints, while in row 4 we do. Without using linear constraints, the default number of stage 1 and total iterations of (200,1000), are not enough to find the best solution for about 7 of the 13 problems. This is because many OptQuest trial points have the same values for the binary

variables but different values for the continuous variables, so complete enumeration takes far longer than in mode 2. However, 1000 stage 1 and 5000 total iterations solve all but 1 problem (its gap is 9.4%), and the ratio (solver calls to best)/(total solver calls) of about 1/9 is favorably small. Row 4 shows that, if trial points are required to satisfy linear constraints, all problems are solved in 1000 total iterations. This is because these problems have mostly linear constraints (geometric mean of linear constraints is 9.1 and of total constraints is 9.9), so the projected trial points tend to contain an optimal set of binary variable values earlier, after only 3.6 solver calls on average. However, solving the MILP's which map trial points into nearest points which satisfy the linear constraints increases total solution time by about a factor of 30 (compare the times in rows 2 and 4).

Table 10 below shows that total solver calls increase quickly with the number of binary variables for the 2 discrete variable modes, especially the "all" mode. When more than one problem has the same number of binaries, averages over those problems are given. The values for the "Discretes Only" mode are the number of feasible binary vectors, averaged over the number of problems shown.

Table 10: Average Solver Calls Vs. Number of Binary Variables

Binaries	1	3	4	5	6	8
Problems	1	3	2	1	4	2
Discretes only	2	7	12.5	27	52.5	81
all	52	35.7	17.5	109	195.5	551.5

5. Summary and future research

The results of Section 4 show that OQNLP is a promising approach for smooth nonconvex NLP's with continuous variables. It solves all 142 of the test problems with no discrete variables with very reasonable solution effort. While there is no guarantee of optimality and no "gap" is available, it can be combined with other algorithms that provide this information, e.g., LGO or BARON. The lower bounds provided by these procedures can be used to estimate the gap for

the OQNLP solution, and the solution produced by any algorithm can be used as a warm start for any other.

Future research includes enhancing the filter logic. As described above, the filters needed to be loosened to solve 9 of the Floudas et. al. test problems, and this loosening could be done automatically. The merit filter parameter *threshfactor* (new threshold=*threshfactor*(1+old threshold)) could be calculated dynamically. Each time a penalty value is above the threshold, calculate the value of *threshfactor* that would cause the new threshold to equal the penalty value. If this happens for *waitcycle* consecutive iterations, set *threshfactor* to the smallest of these values, so the new threshold would have just accepted the lowest of the penalty values. Similar logic can be developed for the distance filter, reducing a basin radius *maxdist* if that basin's distance filter rejects trial points for *waitcycle* consecutive iterations.

Also, the current distance filter logic allows overlap of the spherical approximations to the attraction basins. The true basins can have no points in common, so we can impose this condition on the spheres. If the spherical model basins for any 2 local solutions *xi* and *xj* have radii *ri* and *rj,* these must satisfy

$$ri + rj \le d(i, j)$$

where *d(i,j)* is the Euclidean distance between *xi* and *xj*. If this inequality is violated, the radii *ri* and *rj* can be reduced by the same scale factor so that it holds as equality. We plan to test these options soon.

Another important aspect is the communication of locally optimal solutions back to OptQuest, to improve its search process. These solutions usually have substantially lower penalty values than typical OptQuest trial points, so they are likely to ultimately be included in OptQuest's population. However, their penalty values often become the merit filter thresholds, causing most other trial points to be rejected. Also, the local optima and their nearby "children" will be rejected by the distance filter. We have seen these effects in preliminary tests.

NLP algorithms can fail by failing to find a feasible point in cases where the problem instance is feasible. With GRG algorithms, this usually happens when Phase 1 terminates at a local optimum of the Phase 1 objective. OQNLP can be applied to such problems, if they are reformulated by dropping the true objective, adding deviation variables into all constraints, and minimizing the sum of these deviation variables. This approach could greatly improve the ability of existing NLP solvers to diagnose infeasibility. More generally, OQNLP can improve

NLP solver reliability by starting the solver from as many points as desired, while insuring that these points balance diversity and quality.

The performance of OQNLP in solving MINLP's is less clear, because the 13 MINLP test problems used here are so small. More extensive testing is needed, which should clarify the relative merits of the 2 MINLP "modes" discussed in Section 4.3. If OptQuest manipulates only the discrete variables, then all trial points generated by the current population may be generated at once, and the solver calls at these points may be done in any order. The points can be sorted by increasing distance from their nearest neighbor, and each NLP call can be started from the previous optimum. The NLP's can also be solved in parallel.

Finally, comparative studies of OQNLP and other global and MINLP solvers are needed. This testing is facilitated by the existing GAMS interfaces for BARON, LGO, DICOPT, and SBB. The "MINLP World" and "Global World" websites developed by GAMS Development Corporation (see www.gamsworld.org) provide solver information and test problems with known solutions.

References

Dixon, L., G.P. Szegö . 1975. Towards Global Optimization. *Proceedings of a Workshop at the University of Cagliari, Italy,* North Holland.

Drud, A. 1994. CONOPT—A Large-Scale GRG-Code. *ORSA Journal on Computing* **6** 2.

Edgar, T.F., D.M. Himmelblau, L.S. Lasdon. 2001. *Optimization of Chemical Processes.* McGraw-Hill Companies, Inc.

Floudas, C.A., et al. 1999. *Handbook of Test Problems in Local and Global Optimization.* Kluwer Academic Publishers.

Glover, F. 1998. A Template for Scatter Search and Path Relinking. J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, eds. *Artificial Evolution, Lecture Notes in Computer Science* 1363. Springer Verlag. 13-54.

Laguna, M., Rafael Marti. 2000. Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions. Working paper, Department D'Estadistica i Investigacio Operativa, Universitat de Valencia, Burjassot 46100, Spain.

Laguna, M., Rafael Marti. 2002. The OptQuest Callable Library. Stefan Voss and D. Woodruff, eds. *Optimization Software Class Libraries.* Kluwer Academic Publishers, Boston. 193-218.

Laguna, M., Rafael Marti. 2003. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers.

Locatelli, M., F. Schoen. 1999. Random Linkage: a Family of Acceptance/Rejection Algorithms for Global Optimization. *Mathematical. Programming* **85 2** 379-396.

Murtagh, B.A., M.A. Saunders. 1982. A Projected Lagrangian Algorithm and Its Implementation for Sparse Nonlinear Constraints. *Mathematical Programming Study* **16** 84-117.

Nash, S.G., A. Sofer. 1996. *Linear and Nonlinear Programming*. McGraw-Hill Companies, Inc.

Nash, S.G. 1998. Nonlinear Programming. *OR/MS Today* 36-45.

Nocedal, J., S. J. Wright. 1999. *Numerical Optimization*. Springer Series in Operations Research.

Pardalos, P.M., H.E. Romeijn, H. Tuy. 2000. Recent Developments and Trends in Global Optimization. *Journal of Computational and Applied Mathematics* **124 1-2** 209-228.

Pintér, J.D. 1996. *Global Optimization in Action*. Kluwer Academic Publishers.

Rinnooy Kan, A.H.G., G.T. Timmer. 1987. Stochastic Global Optimization Methods; part I: Clustering Methods. *Mathematical Programming* **37** 27-56.

Rinnooy Kan, A.H.G., G.T. Timmer, 1987. Stochastic Global Optimization Methods; part II: Multi Level Methods. *Mathematical Programming* **37** 57-78.

Smith, S., L. Lasdon. 1992. Solving Large Sparse Nonlinear Programs Using GRG. *ORSA Journal on Computing* **4 1** 3-15.