Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions

MANUEL LAGUNA[1]

Graduate School of Business Administration, University of Colorado, Boulder, CO 80309-0419, USA
manuel.laguna@Colorado.edu

RAFAEL MARTÍ

Departament D'Estadística i Investigació Operativa, Universitat de València, Burjassot 46100, Spain
rafael.marti@uv.es

Latest version: May 22, 2000

Abstract — Scatter search is an evolutionary method that, unlike genetic algorithms, operates on a small set of solutions and makes only limited use of randomization as a proxy for diversification when searching for a globally optimal solution. The scatter search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication. In this paper, we test the merit of several scatter search designs in the context of global optimization of multimodal functions. We compare these designs among themselves and choose one to compare against a well-known genetic algorithm that has been specifically developed for this class of problems. The testing is performed on a set of benchmark multimodal functions with known global minima.

[1] Partially supported by the visiting professor fellowship program of the University of Valencia (Grant Ref. No. 42743).

1. Introduction

Scatter search is a solution procedure that falls within the evolutionary framework and consists of five methods:

1. A *Diversification Generation Method* to generate a collection of diverse trial solutions, using one or more arbitrary trial solutions (or seed solutions) as an input.

2. An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor the output solutions are required to be feasible, though the output solutions are typically feasible. If the input trial solution is not improved as a result of the application of this method, the "enhanced" solution is considered to be the same as the input solution.)

3. A *Reference Set Update Method* to build and maintain a *reference set* consisting of the *b* "best" solutions found (where the value of *b* is typically small, e.g., no more than 20), organized to provide efficient accessing by other parts of the solution procedure. Several alternative criteria may be used to add solutions to the reference set and delete solutions from the reference set.

4. A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions. The most common subset generation method is to generate all pairs of reference solutions (i.e., all subsets of size 2).

5. A *Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solutions. The combination method is analogous to the crossover operator in genetic algorithms but it must be capable of combining two or more solutions.

The scatter search methodology is very flexible, since each of its elements can be implemented in a variety of ways and degrees of sophistication. Details about the origin and applications of scatter search can be found in, Glover (1998), Glover, Laguna and Martí (1999) and Laguna (2000). In our current development, we study the merit of alternative scatter search designs in the context of nonlinear optimization. In particular, we use a benchmark set of unconstrained problems, for which global optima are known. That is, the class of problems that we use for testing can be characterized as:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{Subject to} & l \le x \le u \end{array}$$

Where $f(x)$ is a nonlinear function and x is a vector of continuous and bounded variables. We test several alternatives for generating diversification and updating the reference set in a procedure that does not use an improvement method and in which combinations are linear and limited to pairs of solutions. We also test the use of a two-phase intensification. These strategies are added to a basic procedure, which is outlined in form of pseudo-code in Figure 1.

The basic procedure in Figure 1 starts with the creation of an initial reference set of solutions (*RefSet*). The Diversification Generation Method is used to build a large set of diverse solutions *P*. The size of *P* (*PSize*) is typically 10 times the size of *RefSet*. Initially, the reference set *RefSet* consists of *b* distinct and maximally diverse solutions from *P*. The solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list. The search is then initiated by assigning the value of TRUE to the Boolean variable *NewSolutions*. In step 3, *NewSubsets* is constructed and *NewSolutions* is switched to FALSE. Since we are focusing our attention to subsets of size 2, the cardinality of *NewSubsets* corresponding to the initial reference set is given by $(b^2-b)/2$, which accounts for all pairs of solutions in *RefSet*. The pairs in *NewSubsets* are selected one at a time in lexicographical order and the Solution Combination Method is applied to generate one or more solutions in step 5. If a newly created solution improves upon the worst solution currently in *RefSet*, the new solution replaces the worst and *RefSet* is reordered in step 6. The *NewSolutions* flag is switched to TRUE and the subset s that was just combined is deleted from *NewSubsets* in steps 7 and 8, respectively.

1. Start with $P = \emptyset$. Use the Diversification Generation Method to construct a solution x. If $x \notin P$ then add x to P (i.e., $P = P \cup x$), otherwise, discard x. Repeat this step until $|P| = PSize$. Build $RefSet = \{ x^1, \ldots, x^b \}$ with b diverse solutions in P.
2. Evaluate the solutions in *RefSet* and order them according to their objective function value such that x^1 is the best solution and x^b the worst. Make *NewSolutions* = TRUE.

while (*NewSolutions*) **do**

    3. Generate *NewSubsets*, which consists of all pairs of solutions in *RefSet* that include at least one new solution. Make *NewSolutions* = FALSE.

    while ($NewSubsets \neq \emptyset$) **do**

        4. Select the next subset s in *NewSubSets*.
        5. Apply the Solution Combination Method to s to obtain one or more new solutions x.

        if (x is not in *RefSet* and $f(x) < f(x^b)$) **then**

            6. Make $x^b = x$ and reorder *RefSet*.
            7. Make *NewSolutions* = TRUE.

        end if

        8. Delete s from *NewSubsets*.

    end while

end while

Figure 1. Outline of basic scatter search.

Note that while this basic procedure is very aggressive in trying to improve upon the quality of the solutions in the current reference set, it those so by sacrificing search diversity. In fact, the Diversification Generation Method is used only once to generate *PSize* different solutions at the beginning of the search and it is never employed again. The initial *RefSet* is built selecting a solution from P and then making b-1 more selection in order to maximize the minimum distance between the candidate solution and the solutions currently in *RefSet*. That is, for each candidate solution x in *P-RefSet* and reference set solution y in *RefSet*, we calculate a measure of distance or dissimilarity $d(x,y)$. We then select the candidate solution that maximizes $d_{\min}(x)$, where $d_{\min}(x) = \min_{y \in RefSet} \{ d(x,y) \}$.

The updating of the reference set is based on improving the quality of the worst solution and the search terminates when no new solutions are admitted to *RefSet*. The procedure does not use an Improvement Method, which in scatter search implementations is typically applied after a new solution is constructed with either the Diversification Generation Method or the Combination Method. The Subset Generation Method is also very simple and consists of generating all pairs of solutions in *RefSet* that contain at least one new solution. This means that the procedure does not allow for two solutions to be subjected to the Combination Method more than once.

In the remainder of this paper, we develop and test scatter search designs that add to the basic procedure of Figure 1. All of these variations employ a method for creating new solutions that consists of linear combinations of two solutions in the reference set. We consider the following three types of linear combinations, where we assume that the reference solutions are x' and x'', $d = r\frac{x'' - x'}{2}$ and r is a random number in the range (0, 1):

C1: $x = x' - d$
C2: $x = x' + d$
C3: $x = x'' + d$

In the basic procedure of Figure 1, the Combination Method generates one solution of each type when combining two solutions in the reference set. Note that the combination mechanism includes a random element and therefore it is possible to combine the same pair of solutions more than once to generate new solutions. However the Subset Combination Method does not allow the same pair of solutions to be combined more than once.

2. Reference Set Update Method

In the basic procedure of Figure 1, the reference set is updated by replacing the reference solution with the worst objective function value with a new solution that has a better objective function value. Since the *RefSet* is always ordered, the best solution can be denoted by x^1 and worst solution by x^b. Then, when a new solution x is generated as a result of the application of the Combination Method, the objective function value of the new solution is used to determine whether the *RefSet* needs to be updated. That is, if:

$$x \notin RefSet \text{ and } f(x) < f\left(x^b\right),$$

we update *RefSet* by making $x^b = x$ and reorder the reference set. We have developed the following three variations of this basic update procedure (UP0), which we will refer to as UP1, UP2 and UP3.

RefSet Update UP1

This update adds a mechanism to partially rebuild the reference set when no new solutions can be generated with the Combination Method. The update is performed after the inner "while-loop" in Figure 1 fails and the *NewSolutions* variable is FALSE. (The inner while-loop consists of steps 4 to 8.) The *RefSet* is partially rebuilt with a diversification update, which works as follows. Solutions $x^{[b/2]+1}, \ldots, x^b$ are deleted from *RefSet*, where $[v]$ is the largest integer less than or equal to v. The frequency counts $freq(i,j)$ in the Diversification Generation Method are updated with the corresponding values from the solutions that remain in the reference set, that is, $x^1, \ldots, x^{[b/2]}$. Then, the generator is used to construct a set P of solutions. Solutions $x^{[b/2]+1}, \ldots, x^b$ in *RefSet* are sequentially selected from P with the max-min criterion used in step 1 of Figure 1 (and described in the previous section). The min-max criterion is applied against solutions $x^1, \ldots, x^{[b/2]}$ when selecting solution $x^{[b/2]+1}$, then against solutions $x^1, \ldots, x^{[b/2]+1}$ when selecting solution $x^{[b/2]+2}$, and so on.

The following modification of the pseudo-code in Figure 1 incorporates the UP1 update. The modification consists of adding the following instructions after the end of the inner while-loop:

if (*NewSolutions* = FALSE) **then**
9. Use the Diversification Generation Method to replace solutions $x^{[b/2]+1}, \ldots, x^b$ from *RefSet* and reorder *RefSet*.
10. Make *NewSolutions* = TRUE

end if

Note that when this update is added to the basic procedure, the resulting search does not have a built-in termination criterion. In our experimental testing we stop the search after a maximum number of objective function evaluations has been reached. The basic Combination Method is applied when using the UP1 update.

RefSet Update UP2

This update has the goal of dynamically preserving diversity in the reference set, instead of allowing it to become homogenous by only admitting high quality solutions that in many applications tend to be very similar to each other. Hence, in addition to updating the reference set when new solutions of high quality are generated, the reference set is also updated with highly diverse solutions. Specifically, the update consists of partitioning the reference into two subsets: $RefSet_1 = \{ x^1, \ldots, x^{[b/2]} \}$ and $RefSet_2 = \{ x^{[b/2]+1}, \ldots, x^b \}$. The first subset is referred to as the "quality" subset and the second is referred to as the "diverse" subset. The solutions in $RefSet_1$ are ordered according to their objective function value and the set is updated with the goal of increasing quality, using the UP0 criterion. That is, a new solution x replaces the reference solution $x^{[b/2]}$ if $f(x) < f\left(x^{[b/2]}\right)$. The solutions in $RefSet_2$ are ordered according to their diversity value and the update has the goal of increasing diversity. Therefore, a new solution x replaces the reference solution x^b if $d_{\min}(x) > d_{\min}\left(x^b\right)$.

We can modify the outline in Figure 1 to incorporate the UP2 update by replacing the "if" statement inside the inner while-loop with the following:

if (x is not in $RefSet_1$ and $f(x) < f\left(x^{[b/2]}\right)$) **then**
  6a. Make $x^{[b/2]} = x$ and reorder $RefSet_1$.
  7a. Make *NewSolutions* = TRUE.
else if (x is not in $RefSet_2$ and $d_{\min}(x) > d_{\min}\left(x^b\right)$) **then**
  6b. Make $x^{[b]} = x$ and reorder $RefSet_2$.
  7b. Make *NewSolutions* = TRUE.
end if

When adding this update to the basic procedure, the procedure exits the inner while-loop when no new solutions can be admitted to either $RefSet_1$ or $RefSet_2$ during the combination of all *NewSubsets*. The search, however, does not terminate, because *RefSet* is updated using the UP1 update. That is, $RefSet_1$ is kept and $RefSet_2$ is rebuilt with the Diversification Generation Method. The following rules are used during the application of the Combination Method when the UP2 update is active, assuming that x' and x'' are the reference solutions to be combined:

- If both x' and x'' are elements of $RefSet_1$, then C1 and C3 are applied once and C2 is applied twice to create 4 solutions.
- If only one of x' and x'' is a member of $RefSet_1$, then C1, C2 and C3 are applied once to create 3 solutions.
- If neither x' nor x'' is a member of $RefSet_1$, then C2 and either C1 or C3 is applied once to create 2 solutions, where the selection of C1 or C3 is random.

The rationale behind these rules is that we have experimentally determined that high-quality solutions tend to create other high-quality solutions and therefore we allow combinations of two high-quality solutions to create more new solutions than other types of combinations (Campos, et al. 1999).

RefSet Update UP3

This update is an extension of UP2. The reference set is divided into three subsets: $RefSet_1 = \{ x^1, \ldots, x^{[(b-2)/2]} \}$, $RefSet_2 = \{ x^{[(b-2)/2]+1}, \ldots, x^{b-2} \}$, and $RefSet_3 = \{ x^{b-1}, x^b \}$. $RefSet_1$ and $RefSet_2$ are updated using the same rules as in UP2. In order to update $RefSet_3$, we keep track of $g(x)$, the objective function value of the best solution ever created from a combination of $x \in RefSet_1$ and any other reference solution. $RefSet_3$ is ordered according to $g(x)$ in such a way that $g\left(x^{b-1}\right) < g\left(x^b\right)$. When solution $x^{[(b-2)/2]}$ in $RefSet_1$ is replaced with a newly created solution of higher quality, we compare $g\left(x^{[(b-2)/2]}\right)$ and $g\left(x^b\right)$ and update $RefSet_3$ accordingly.

We can modify the pseudo-code in Figure 1 to incorporate the UP3 update by replacing the "if" statement inside the inner while-loop with the following:

if (x is not in $RefSet_1$ and $f(x) < f\left(x^{[(b-2)/2]}\right)$) **then**
    6a. Make $y = x^{[(b-2)/2]}$ and $x^{[(b-2)/2]} = x$. Reorder $RefSet_1$.
    if (y is not in $RefSet_3$ and $g(y) < g\left(x^b\right)$) **then**
        6b. Make $x^b = y$ and reorder $RefSet_3$.
    end if
    7a. Make *NewSolutions* = TRUE.
else if (x is not in $RefSet_2$ and $d_{\min}(x) > d_{\min}\left(x^{b-2}\right)$) **then**
    6c. Make $x^{b-2} = x$ and reorder $RefSet_2$.
    7b. Make *NewSolutions* = TRUE.
end if

When adding this update to the basic procedure, the search exists the inner while-loop when no new solutions can be admitted to either $RefSet_1$ or $RefSet_2$ during the combination of all *NewSubsets*. As before, the search does not terminate, because *RefSet* is updated in a similar way as UP2. That is, $RefSet_1$ and $RefSet_3$ are kept and $RefSet_2$ is rebuilt with the Diversification Generation Method. Note that the update of $RefSet_3$ is dependent on the update of $RefSet_1$, in the sense that $RefSet_3$ can only be updated if a solution in $RefSet_1$ is replaced. The combination rules associated with the UP3 update are the same as the rules for UP2 when combining solutions from either $RefSet_1$ or $RefSet_2$. However, when x' is in $RefSet_1$ and x'' is in $RefSet_3$, then C1 and C2 are applied once with $d = r\frac{x'' - x'}{3}$. Note that we don't attempt combinations that consist of either one solution from $RefSet_2$ and one solution from $RefSet_3$ or both solutions from $RefSet_3$.

3. Diversification Generation Method

The basic Diversification Generation Method (DV0) uses controlled randomization and frequency memory to generate a set of diverse solutions. We accomplish this by dividing the range of each variable u_i - l_i into 4 sub-ranges of equal size. Then, a solution is constructed in two steps. First a sub-range is randomly selected. The probability of selecting a sub-range is inversely proportional to its frequency count. Then a value is randomly generated within the selected sub-range. The number of times sub-range j has been chosen to generate a value for variable i is accumulated in $freq(i, j)$.

In addition to DV0, we have developed two variants (DV1 and DV2), which are based on techniques from the area of statistics known as Design of Experiments. One of the most popular design of experiments is the factorial design k^n, where n is the number of factors (in our case variables) and k is the number of levels (in our case possible variable values). A full factorial design considers that all combinations of the factors and levels will be tested. Therefore, a full factorial design with 5 factors and 3 critical levels would require 3^5 = 243 experiments. Obviously, full factorial designs can quickly become impractical even for a small number of levels, because the number of experiments exponentially increases with the number of factors. A more practical alternative is to employ fractional factorial designs. These designs draw conclusions based on a fraction of experiments, which are strategically selected from the set of all possible experiments in the corresponding full factorial design. One of the most notable proponents of the use of fractional factorial designs is Genichi Taguchi (Roy, 1990). Taguchi proposed a special set of orthogonal arrays to lay out experiments associated with quality improvement in manufacturing. These orthogonal arrays are the result of combining orthogonal Latin squares in a unique manner. We use Taguchi's arrays as a mechanism for generating diversity. Table 1 shows the $L_9(3^4)$ orthogonal array that can be used to generate 9 solutions for a 4-variable problem.

Table 1. $L_9(3^4)$ orthogonal array.

Experiment	Factors 1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	2	3
8	3	2	1	3
9	3	3	2	1

The values in Table 1 represent the levels at which the factors are set in each experiment. For the purpose of creating DV1, a diversification generator based on Taguchi tables, we translate each level setting as follows:

1 = lower bound (l_i)

2 = midpoint $\left(\frac{l_i + u_i}{2}\right)$

3 = upper bound (u_i)

The generator DV1 uses the appropriate Taguchi table to draw solutions every time the diversification generator is called. The Taguchi table is selected according to the number of variables in the problem. When DV1 is employed and all solutions in the appropriate table have been tried, the generator returns solutions using DV0.

A second variant based on Taguchi tables (DV2) is obtained when level settings in each experiment are translated to variable values as follows:

1 = near lower bound $\left(l_i + r(u_i - l_i)\right)$, where r is randomly drawn from (0, 0.1)

2 = near midpoint $\left(\frac{l_i + u_i}{2} + r(u_i - l_i)\right)$, where r is randomly drawn from (-0.1, 0.1)

3 = near upper bound $\left(u_i - r(u_i - l_i)\right)$, where r is randomly drawn from (0, 0.1)

Although the same Taguchi experiment can result in more than one solution to the problem, we turn control to DV0 once all experiments have been used to generate solutions, because DV0 covers the entire feasible range of each variable, while DV2 focuses on generating solutions with variable values near the bounds and the midpoint.

4. Intensification Strategy

The Combination Method is the main intensification mechanism within scatter search. In section 2, we described the application of the Combination Method associated with each variant of the updating strategies for the reference set. In addition to these rules, we have developed and tested a two-phase intensification procedure.

The basic design of Figure 1 applies the Combination Method to all the subsets generated from a given reference set. Although the reference set is dynamically updated, the new solutions are not used for combination until all the subsets generated with the previous *RefSet* have been combined. In other words, the new solutions that become members of *RefSet* are not combined until all pairs in *NewSubsets* are subjected to the Combination Method. The goal of the first phase of our intensification strategy is to apply the Combination Method to new solutions in a manner that is faster than in the basic design. That is, if a new solution is admitted to the reference set the intensification goal is to allow this new solution to be subjected to the Combination Method as quickly as possible. This strategy can be easily implemented by modifying step 3 in Figure 1 as follows:

if (*Intensify* = TRUE) **then**
3a. Let *NewSubsets* consist of one pair of solutions, the two best solutions in $RefSet_1$ where at least one is a new solution. Make *NewSolutions* = TRUE.
if (*NewSubsets* = ∅) **then** *Intensify* = FALSE.
end if
if (*Intensify* = FALSE) **then**
3b. Generate *NewSubsets*, which consists of all pairs of solutions in *RefSet* that include at least one new solution. Make *NewSolutions* = FALSE.
end if

Note that by reducing *NewSubsets* to just one pair of solutions, the procedure returns to the generation of subsets immediately after one application of the Combination Method. The downside of this intensification phase is that some solutions in *RefSet* could be replaced before being combined. To illustrate this, suppose that the initial *RefSet* = { x^1, x^2, x^3, x^4 } and that the intensification strategy is active. Suppose also that the combination of the pair (x^1, x^2) results in a solution y for which:

$$f(y) < f(x^3)$$

Then, the reference set is changed in such a way that the updated *RefSet* = { x^1, x^2, y, x^3 }. The search continues with the combination of the pair (x^1, y). Clearly, the quick updating of the reference set eliminates the combination of the pair (x^1, x^4).

The second phase of our intensification strategy consists of selecting the two best solutions in $RefSet_1$ to create 8 solutions by applying C1 and C3 twice and C2 four times. Note that for this intensification, we do not require that at least one of the two best solutions be new. We do not test the two phases of our intensification strategies separately. That is, we either apply the entire two-phase strategy or we don't apply it at all. The application of the intensification strategy depends on two parameters: *IntPoint* and *IntLength*. *IntPoint* is the iteration number at which the first intensification phase is applied for *IntLength* iterations. The second intensification phase is automatically applied at iteration 2**IntPoint*. An iteration, in our context, is considered a single evaluation of the objective function.

5. Computational Experiments

Our computational testing consists of two main experiments. In the first experiment, we compare the performance of the scatter search variants that result from the application of the alternative strategies described in the previous sections. In the second experiment, we compare the performance of the most effective scatter search design against a well-known genetic algorithm. Table 2 shows summary information of 40 test problems that are based on a set of nonlinear objective functions, most of which can be found in the following web pages:

http://www.maths.adelaide.edu.au/Applied/llazausk/alife/realfopt.htm
http://solon.cma.univie.ac.at/~neum/glopt/my_problems.html
http://www-math.cudenver.edu/~rvan/phd/node32.html

The number between parentheses associated with some of the problems are the parameter values for the corresponding objective function. A typical parameter refers to the number of variables, since several of these functions expand to an arbitrary number of variables. Although the objective functions are built in a way that the optimal solutions are known, the optimization problems cannot be trivially solved by search procedures that do not exploit the special structure embedded in each function. A detailed description of the objective functions is provided in the Appendix.

Since the initial reference set includes the solution in which all the variables are set to their lower bound (i.e., $x = l$), the solution in which all the variables are set to their upper bounds (i.e., $x = u$) and the solution for which all the variables are set to the midpoint (i.e., $x = (u+l)/2$), we have modified the bounds in those cases where any of these solutions turns out to be the optimal. For example, the optimal solution to the "De Joung" function is $x = 0$ and the original bounds for all variables were l = -5.12 and u = 5.12. Since the midpoint of this range is the optimal solution, the problem is trivially solved by any of our scatter search variants. Therefore, we have modified the range and changed the lower bound to -2.56.

Table 2. Test problems

Number of variables	Problem number	Name and parameter values	x^*	$f(x^*)$
2	1	Branin	(9.42478, 2.475)†	0.397887
	2	B2	(0, 0)	0
	3	Easom	(π,π)	-1
	4	Goldstein and Price	(0, -1)	3
	5	Shubert	(0.0217, -0.9527)†	-186.7309
	6	Beale	(3, 0.5)	0
	7	Booth	(1, 3)	0
	8	Matyas	(0, 0)	0
	9	SixHumpCamelback	(0.089840, -0.712659) †	-1.0316285
	10	Schwefel(2)	(1, 1)	0
	11	Rosenbrock (2)	(1, 1)	0
	12	Zakharov(2)	(0, 0)	0
3	13	De Joung	(0, 0, 0)	0
	14	Hartmann(3,4)	(0.114614, 0.555649, 0.852547)	0
4	15	Colville	(1, 1, 1, 1)	0
	16	Shekel(5)	(4, 4, 4, 4)	-10.1532
	17	Shekel(7)	(4, 4, 4, 4)	-10.4029
	18	Shekel(10)	(4, 4, 4, 4)	-10.53641
	19	Perm(4,0.5)	(1, 2, 3, 4)	0
	20	Perm0(4,10)	(1, 1/2, 1/3, 1/4)	0
	21	Powersum (8,18,44,114)	(1, 2, 2, 3)	0
6	22	Hartmann(6,4)	(0.20169, 0.150011, 0.47687, 0.275332, 0.311652, 0.6573)	0
	23	Schwefel(6)	(1, …, 1)	0
	24	Trid(6)	$x_i = i*(7-i)$	-50
10	25	Trid(10)	$x_i = i*(11-i)$	-210
	26	Rastrigin(10)	(0, …, 0)	0
	27	Griewank(10)	(0, …, 0)	0
	28	Sum Squares(10)	(0, …, 0)	0
	29	Rosenbrock(10)	(1, …, 1)	0
	30	Zakharov(10)	(0, …, 0)	0
20	31	Rastrigin(20)	(0, …, 0)	0
	32	Griewank(20)	(0, …, 0)	0
	33	Sum Squares(20)	(0, …, 0)	0
	34	Rosenbrock(20)	(1, …, 1)	0
	35	Zakharov(20)	(0, …, 0)	0
> 20	36	Powell(24)	(3, -1, 0, 1, 3, …, 3, -1, 0, 1)	0
	37	Dixon and Price(25)	$x_i = 2^{-\left(\frac{z-1}{z}\right)},\ z = 2^{i-1}$	0
	38	Levy(30)	(1, …, 1)	0
	39	Sphere(30)	(0, …, 0)	0
	40	Ackley(30)	(0, …, 0)	0

† This is one of several multiple optimal solutions.

Our first experiment consists of trying all possible combinations of the designs resulting from the strategies described above. We consider the following settings:

Reference set update: UP1, UP2 and UP3
Diversification generation: DV0, DV1 and DV2
Intensification strategy: IN0 (deactivated) and IN1 (activated)

We use *IntPoint* = 3000 and *IntLength* = 200 for IN1. These settings result in 3*3*2 = 18 possible combinations and each combination is run up to a maximum of 10000 objective function evaluations. Note that we don't test settings with UP0 because this update strategy causes an early termination of the search and cannot compete with extended runs of the other variants. We execute 18*20 = 360 runs of our scatter search procedure, because we limit this experiment to the odd-numbered problems in

Table 2. We use half of the available problems to be able to confirm our findings when applying them to the entire set of problems. The results of our first experiment reveal that the best setting is UP1, DV0, and IN1. This setting achieves the least amount of deviation from optimality and the most number of optimal solutions.

In the second main experiment, we compare the performance of our scatter search with UP1, DV0 and IN1 against Genocop III (http://www.coe.uncc.edu/~zbyszek/gchome.html), an implementation of genetic algorithms that is customized for solving nonlinear optimization problems with continuous and bounded variables (Michalewicz 1993). For this experiment, we define the optimality gap as:

$$GAP = \left| f(x) - f(x^*) \right|$$

where x is a heuristic solution and x^* is the optimal solution. We then say that a heuristic solution x is optimal if:

$$GAP \leq \begin{cases} \varepsilon & f(x^*) = 0 \\ \varepsilon * \left| f(x^*) \right| & f(x^*) \neq 0 \end{cases}$$

In our experimentation we set $\varepsilon = 0.001$. Table 3 shows the average *GAP* value for our scatter search implementation and Genocop III. The average *GAP* is calculated over the set of 40 problems in Table 2 at several points during the search. Since Genocop III begins with a *GAP* on the order of 10^{27} for problem 23, we also provide the average *GAP* calculated over 39 problems, ignoring the *GAP* for problem 23.

Table 3. Average *GAP* values.

Evaluations	100	500	1000	5000	10000	20000	50000
Genocop[1]	5.37E+25	2.39E+17	1.13E+14	636.37	399.52	320.84	313.34
Genocop[2]	1335.45	611.30	379.03	335.81	328.66	324.72	321.20
Scatter Search	134.45	26.34	14.66	4.96	3.60	3.52	3.46

[1] Average values over all test problems.
[2] Average values ignoring problem 23.

Table 3 shows that the average *GAP* of scatter search is consistently lower than the average *GAP* associated with Genocop III. In fact, the average *GAP* of scatter search after 100 evaluations of the objective function is already better than the average *GAP* for Genocop III after 50000 objective function evaluations. The final *GAP* values for scatter search are all less than 1.0, except in problems 23, 26, 34, and 40, for which the *GAP* values are 118.4341, 9.9496, 2.2441, and 5.5033, respectively.

Counting the number of optimal solutions found with each method is an alternative view of the results from our second experiment. The plot in Figure 2 shows the number of ε-optimal solutions found by each procedure during a search with a stopping criterion of 50000 objective function evaluations. The plot shows, once again, the superior performance of our scatter search implementation, which solves 4 problems within 100 evaluations and 30 by 20000 evaluations. At the end of the search, Genocop III successfully solves a total of 23 problems.

We perform an additional secondary experiment to investigate whether any of the scatter search variant is capable of solving the 10 problems that the UP1, DV0 and IN1 setting did not solve. For each unsolved problem (15, 19, 23, 26, 29, 34, 36-38 and 40), we run a 50000-evaluation search using each of the 17 settings, excluding (UP1, DV0, IN1). The outcome of this experiment is that 5 additional problems are solved and the average *GAP* of the unsolved problems is reduced to 1.2322. These results are summarized in Tables 4 and 5.

Table 4 shows that both settings (UP1, DV1, IN0) and (UP1, DV1, IN1) are capable of solving problems 15, 19 and 26. However, the performance of (UP1, DV1, IN0) is considered better in regard to these two problems because it solves them in fewer evaluations than (UP1, DV1, IN1). This setting, on the other hand, is capable of solving the elusive problem 23 employing less than 10000 evaluations.

Also, a setting that uses UP1 and DV0 solves problem 36 regardless of the intensification strategy setting. Finally, note that from the unsolved problems in Table 5, only two have a *GAP* value larger than 1.

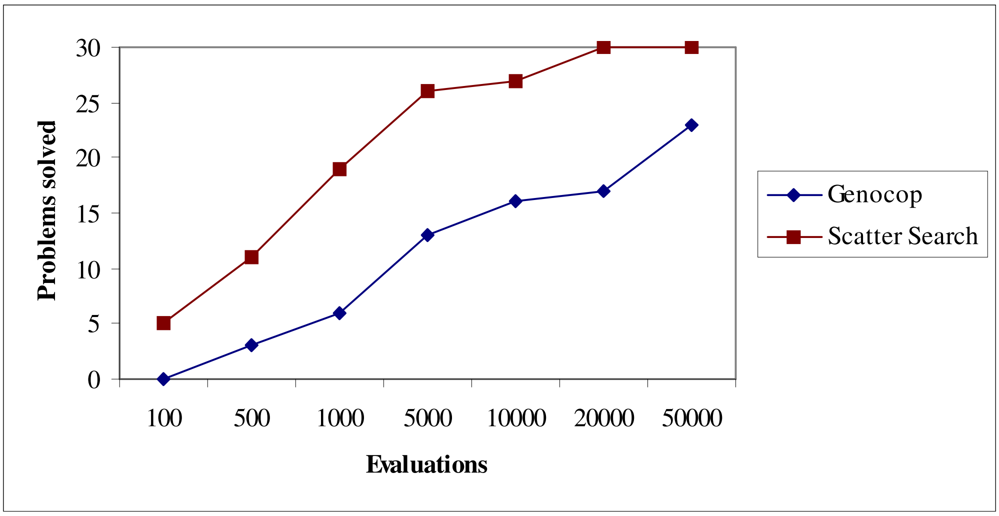

Figure 2. Number of problems solved.

Table 4. Additional solved problems.

Setting	Problem	Evaluations
(UP1, DV1, IN0)	15	13638
	19	13325
	26	27
(UP1, DV1, IN1)	15	16152
	19	37118
	23	9525
	26	27
(UP2, DV0, *)	36	119

* = all options.

Table 5. Unsolved problems.

Problem	Setting	GAP
29	(UP1, DV2, IN1)	0.327761
34	(UP1, DV1, IN1)	2.050292
37	(UP1, *, *)	0.666667
38	(UP1, DV0, IN1)	0.089528
40	(UP1, DV2, IN1)	3.026937
Average		1.232237

* = all options.

6. Conclusions

We have explored alternative mechanisms to perform key operations within the scatter search framework. In particular, we have focused on designing and testing strategies for updating the reference set, generating diversity and intensifying the search. We have gathered a set of 40 test problems with number of variables ranging from 2 to 30 to perform experiments with the goal of assessing the merit of each combination of the proposed strategies. For our initial testing, we chose 20 problems out of 40 and concluded that the best parameter setting was UP1, DV0 and IN1. We then used this setting to compare the performance of the resulting procedure against a well-known genetic algorithm. The computational tests show that our scatter search is robust, because it finds solutions of reasonable quality from the beginning of the search. This is an important feature in settings where the objective function evaluation is computational expensive (e.g., when optimizing simulations). The procedure is capable of finding ε-optimal solutions to 30 out of 40 problems within 20000 objective function evaluations. This compares favorable to the number of ε-optimal solutions found with Genocop III employing more than twice the number of evaluations.

References

Campos, V., F. Glover, M. Laguna and R. Martí (1999) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," Working paper, University of Colorado at Boulder.

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

Glover, F., M. Laguna and R. Martí (1999) "Scatter Search," to appear in *Theory and Applications of Evolutionary Computation: Recent Trends,* A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag.

Laguna, M. (2000) "Scatter Search," to appear in *Handbook of Applied Optimization,* P. M. Pardalos and M. G. C. Resende (Eds.), Oxford Academic Press.

Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs,* Second-Extended Edition, Springer-Verlag.

Roy, R.K. (1990) *A Primer on the Taguchi Method*, Van Nostrand Reinhold, New York.

Appendix

This appendix contains the description of the set of test functions in Table 2. The description consists of the objective function, parameter values and the bounds for each variable.

1. Branin

Minimize $f(x) = \left(x_2 - \left(\frac{5}{4\pi^2} \right) x_1^2 + \left(\frac{5}{\pi} \right) x_1 - 6 \right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_i) + 10$

Subject to $-5 \le x_1, x_2 \le 15$

2. B2

Minimize $f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$

Subject to $-50 \le x_i \le 100$ for i=1, 2

3. Easom

Minimize $f(x) = -\cos(x_1)\cos(x_2)\exp\left(-\left((x_1 - \pi)^2 + (x_2 - \pi)^2\right)\right)$

Subject to $-100 \le x_i \le 100$ for i=1, 2

4. Goldstein and Price

Minimize $f(x) = \left(1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right)$
$\left(30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right)$

Subject to $-2 \le x_i \le 2$ for i=1, 2

5. Shubert

Minimize $f(x) = \left(\sum_{j=1}^{5} j\cos((j+1)x_1 + j)\right)\left(\sum_{j=1}^{5} j\cos((j+1)x_2 + j)\right)$

Subject to $-10 \le x_i \le 10$ for i=1, 2

6. Beale

Minimize $f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$

Subject to $-4.5 \le x_1, x_2 \le 4.5$

7. Booth

Minimize $f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$

Subject to $-10 \le x_1, x_2 \le 10$

8. Matyas

Minimize $f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$

Subject to $-5 \le x_1, x_2 \le 10$

9. SixHumpCamelBack

Minimize $f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$

Subject to $-5 \le x_1, x_2 \le 5$

10, 23. Schwefel(n)

Minimize $f(x) = 418.9829n + \sum_{i=1}^{n}\left(-x_i \sin\sqrt{|x_i|}\right)$

Subject to $-500 \le x_i \le 500$ for i=1,..., n

11, 29, 34. Rosenbrock(n)

Minimize $f(x) = \sum_{i=1}^{n/2} 100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2$

Subject to $-10 \le x_i \le 10$ for i=1,..., n.

12, 30, 35. Zakharov(n)

Minimize $f(x) = \sum_{j=1}^{n} x_j^2 + \left(\sum_{j=1}^{n} 0.5jx_j\right)^2 + \left(\sum_{j=1}^{n} 0.5jx_j\right)^4$

Subject to $-5 \le x_i \le 10$ for i=1, …, n

13. De Joung

Minimize $f(x) = x_1^2 + x_2^2 + x_3^2$

Subject to -2.56 ≤ x_i ≤ 5.12 for i=1, 2, 3

14. Hartmann(3,4)

Minimize $f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij} \left(x_j - p_{ij}\right)^2 \right)$

Subject to $0 \le x_i \le 1$ for i=1, 2, 3

i		a_{ij}		c_i		p_{ij}	
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.0381	0.5743	0.8828

15. Colville

Minimize
$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$$

Subject to -10≤ x_i ≤ 10 for i=1,...,4.

16-18. Shekel(n)

Minimize $f(x) = -\sum_{i=1}^{n} \left((\boldsymbol{x} - \boldsymbol{a}_i)^T (\boldsymbol{x} - \boldsymbol{a}_i) + c_i \right)^{-1}$; $\boldsymbol{x} = (x_1, x_2, x_3, x_4)^T$; $\boldsymbol{a}_i = \left(a_i^1, a_i^2, a_i^3, a_i^4\right)^T$

Subject to $0 \le x_i \le 10$ for i=1, 2, 3, 4

i		a_i^T			c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

19. Perm(n, β)

Minimize $f(x) = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} \left(i^k + \beta\right) \left(\left(\frac{x_i}{i}\right)^k - 1 \right) \right)^2$

Subject to $-n \le x_i \le n$ for i=1,..., n.

20. Perm0(n, β)

Minimize $$f(x)=\sum_{k=1}^{n}\left(\sum_{i=1}^{n}(i+\beta)\left(x_i^k-\left(\frac{1}{i}\right)^k\right)\right)^2$$

Subject to $-n \le x_i \le n$ for i=1,..., n.

21. PowerSum(b_1,...,b_n)

Minimize $$f(x)=\sum_{k=1}^{n}\left(\left(\sum_{i=1}^{n}x_i^k\right)-b_k\right)^2$$

Subject to $0 \le x_i \le n$ for i=1,..., n.

22. Hartmann(6,4)

Minimize $$f(x)=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{6}a_{ij}\left(x_j-p_{ij}\right)^2\right)$$

Subject to $0 \le x_i \le 1$ for i=1, …, 6

i	a_{ij}						c_i	p_{ij}					
1	10.0	3.0	17.0	3.5	1.7	8.0	1.0	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10.0	17.0	0.10	8.0	14.0	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3.0	3.5	1.7	10.0	17.0	8.0	3.0	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	17.0	8.0	0.05	10.0	0.1	14.0	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

24-25. Trid(n)

Minimize $$f(x)=\left(\sum_{i=1}^{n}(x_i-1)^2\right)-\sum_{i=2}^{n}x_ix_j$$

Subject to $-n^2 \le x_i \le n^2$ for i=1,..., n.

26, 31. Rastrigin(n)

Minimize $$f(x)=10n+\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi x_i)\right)$$

Subject to $-2.56 \le x_i \le 5.12$ for i=1,..., n

27, 32. Griewank(n)

Minimize $$f(x)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$$

Subject to $-300 \le x_i \le 600$ for i=1,..., n

28, 33. Sum Squares (n)

Minimize $f(x) = \sum_{i=1}^{n} i x_i^2$

Subject to -5 ≤ x_i ≤ 10 for i=1,..., n

36. Powell(n)

Minimize $f(x) = \sum_{j=1}^{n/4} (x_{4j-3} + 10x_{4j-2})^2 + 5(x_{4j-1} - x_{4j})^2 + (x_{4j-2} - 2x_{4j-1})^4 + 10(x_{4j-3} - x_{4j})^4$

Subject to -4 ≤ x_i ≤ 5 for i=1,..., n

37. Dixon and Price(n)

Minimize $f(x) = \sum_{i=1}^{n} i(2x_i^2 - x_{i-1})^2 + (x_1 - 1)^2$

Subject to -10 ≤ x_i ≤ 10 for i=1,..., n

38. Levy(n)

Minimize $f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{k-1} (y_i - 1)^2 (1 + 10\sin^2(\pi y_i + 1)) + (y_k - 1)^2 (1 + \sin^2(2\pi x_k))$

Subject to $y_i = 1 + \frac{x_i - 1}{4}$ for i=1,..., n

-10 ≤ x_i ≤ 10 for i=1,..., n

39. Sphere(n)

Minimize $f(x) = \sum_{i=1}^{n} x_i^2$

Subject to -2.56 ≤ x_i ≤ 5.12 for i=1,..., n

40. Ackley(n)

Minimize $f(x) = 20 + e - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$

Subject to -15 ≤ x_i ≤ 30 for i=1,..., n