Chapter 5
Scatter Search

Manuel Laguna

5.1 Introduction

Scatter search (SS) is an evolutionary approach for optimization. It has been applied to problems with continuous and discrete variables and with one or multiple objectives. The success of SS as an optimization technique is well documented in a constantly growing number of journal articles, book chapters (Glover et al. 2003a, 2003b, 2004; Laguna 2002) and a book (Laguna and Martí 2003). This chapter contains some of the material that can be found in the aforementioned publications but also contains some new ideas that are based on research that has been performed and reported in recent years. The chapter focuses on the basic SS framework, which is responsible for most of the outcomes reported in the literature. SS consists of five methods:

1. Diversification generation
2. Improvement
3. Reference set update
4. Subset generation
5. Solution combination.

The *diversification generation* method is used to generate a set of diverse solutions that are the basis for initializing the search. The most effective diversification methods are those capable of creating a set of solutions that balances diversification and quality. It has been shown that SS produces better results when the diversification generation method is not purely random and constructs solutions by reference to both a diversification measure and the objective function.

The *improvement* method transforms solutions with the goal of improving quality (typically measured by the objective-function value) or feasibility (typically

M. Laguna (✉)
Leeds School of Business, University of Colorado, Boulder, CO, USA
e-mail: laguna@Colorado.EDU

E.K. Burke and G. Kendall (eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, DOI 10.1007/978-1-4614-6940-7_5,

measured by some degree of constraint violation). The input to the improvement method is a single solution that may or may not be feasible. The output is a solution that may or may not be better (in terms of quality or feasibility) than the original solution. The typical improvement method is a local search with the usual rule of stopping as soon as no improvement is detected in the neighborhood of the current solution. There is the possibility of basing the improvement method on procedures that use a neighborhood search but that are able to escape local optimality. Tabu search, simulated annealing and variable neighborhood search qualify as candidates for such a design. This may seem as an attractive option as a general approach for an improvement method, however, these procedures do not have a natural stopping criterion. The end result is that choices need to be made to control the amount of computer time that is spent improving solutions (by running a metaheuristic-based procedure) versus the time spent outside the improvement method (e.g. combining solutions). In general, local search procedures seem to work well and most SS implementations do not include mechanisms to escape local optimality within the process of improving a solution.

The *reference set update* method refers to the process of building and maintaining a reference set of solutions that are used in the main iterative loop of any SS implementation. While there are several implementation options, this element of SS is fairly independent from the context of the problem. The first goal of the reference update method is to build the initial reference set of solutions from the population of solutions generated with the diversification method. Subsequent calls to the reference update method serve the purpose of maintaining the reference set. The typical design of this method builds the first reference set by blending high-quality solutions and diverse solutions. When choosing a diverse solution, reference needs to be made to a distance metric that typically depends on the solution representation. That is, if the problem context is such that continuous variables are used to represent solutions, then diversification may be measured with Euclidean distances. Other solution representations (e.g. binary variables or permutations) result in different ways of calculating distances and in turn diversification. The updating of the reference set during the SS iterations is customarily done on the basis of solution quality.

The *subset generation* method produces subsets of reference solutions which become the input to the combination method. The typical implementation of this method consists of generating all possible pairs of solutions. The SS framework considers also the generation of larger subsets of reference solutions; however, most SS implementations have been limited to operate on pairs of solutions. Clearly, no context information is needed to implement the subset generation method.

The *solution combination* method uses the output from the subset generation method to create new solutions. New trial solutions are the results of combining, typically two but possibly more, reference solutions. The combination of reference solutions is usually designed to exploit problem context information and solution representation. Linear combinations of solutions represented by continuous variables have been used often since suggested by Glover (1998) in connection with the solution of nonlinear programming problems. Several proposals for combining solutions represented by permutations have also been applied (Martí et al. 2005).

```
1.  Diversification generation and improvement methods
2.  while (stopping criteria not satisfied) {
3.      Reference set update method
4.      while(new reference solutions) {
5.          Subset generation method
6.          Combination method
7.          Improvement method
8.          Reference set update method
9.      }
10.     Rebuild reference set
11. }
```

Fig. 5.1 Scatter search framework

The strategy known as path relinking, originally proposed within the tabu search methodology (Glover and Laguna 1997), has also played a relevant role in designing combination methods for SS implementations.

The basic SS framework is outlined in Fig. 5.1. The search starts with the application of the diversification and improvement methods (step 1 in Fig. 5.1). The typical outcome consists of a set of about 100 solutions that is referred to as the population (denoted by P). In most implementations, the diversification generation method is applied first followed by the improvement method. If the application of the improvement method results in the shrinking of the population (due to more than one solution converging to the same local optimum) then the diversification method is applied again until the total number of improved solutions reaches the desired target. Other implementations construct and improve solutions, one by one, until reaching the desired population size.

The main SS loop is shown in lines 2–11 of Fig. 5.1. The input to the first execution of the reference set update method (step 3) is the population of solutions generated in step 1 and the output is a set of solutions known as the reference set (or *RefSet*). Typically, ten solutions are chosen from a population of 100. The first five solutions are chosen to be the best solutions (in terms of the objective-function value) in the population. The other five are chosen to be the most diverse with respect to the solutions in the reference set. If the diverse solutions are chosen sequentially, then the sixth solution is the most diverse with respect to the five best solutions that were chosen first. The seventh solution is the most diverse with respect to the first six and so on until the tenth one is added to the reference set.

The inner while-loop (lines 4–9) is executed as long as at least one reference solution is new in the *RefSet*. A solution is considered new if it has not been subjected to the subset generation (step 5) and combination (step 6) methods. If the reference set contains at least one new solution, the subset generation method builds a list of all the reference solution subsets that will become the input to the combination method. The subset generation method creates new subsets only. A subset is new if it contains at least one new reference solution. This avoids the application of the combination method to the same subset more than once, which is particularly wasteful

when the combination method is completely deterministic. Combination methods that contain random elements may be able to produce new trial solutions even when applied more than once to the same subset of reference solutions. However, this is generally discouraged in favor of introducing new solutions in the reference set by replacing some of the old ones in the rebuilding step (line 10).

The combination method (step 6) is applied to the subsets of reference solutions generated in the previous step. Most combination methods are designed to produce more than one trial solution from the combination of the solutions in a subset. These trial solutions are given to the improvement method (step 7) and the output forms a pool of improved trial solutions that will be considered for admission in the reference set (step 8).

If no new solutions are added to the reference set after the execution of the reference set update method, then the process exits the inner while-loop. The rebuilding step in line 10 is optional. That is, it is possible to implement a SS procedure that terminates the first time that the reference set does not change. However, most implementations extend the search beyond this point by executing a *RefSet* rebuilding step. The rebuilding of the reference set entails the elimination of some current reference solutions and the addition of diverse solutions. In most implementations, all solutions except the best are replaced in this step. The diverse solutions to be added may be either population solutions that have not been used or new solutions constructed with the generation diversification method. Note that only ten solutions out of 100 are used from the population to build the initial reference set and therefore the remaining 90 could be used for rebuilding purposes.

The process (i.e. the main while-loop in lines 2–11) continues as long as the stopping criteria are not satisfied. Possible stopping criteria include number of rebuilding steps or elapsed time. When SS is applied in the context of optimizing expensive black boxes, a limit on the number of calls to the objective-function evaluator (i.e. the black box) may also be used as a criterion for stopping. We now expand our description and provide examples of each of the SS methods.

5.2 Diversification Generation Method

The most effective form of diversification generation is one in which the solutions are not only diverse but their collective quality is better than the outcome of a purely random process. Campos et al. (2001) conducted an experimental study to determine the effectiveness of ten different diversification generation methods. They used two normalized measures to assess the quality and the diversity of the populations generated with each method. The methods varied from purely deterministic to purely random. Most methods were based on semi-greedy constructions, popularized by the first phase of GRASP (Feo and Resende 1995).

Figure 5.2 shows a summary of the results from this experiment. In this figure, Δ_C and Δ_d are values in the range from 0 to 1 that respectively represent the quality and diversity of the population generated by each method. Maximum quality

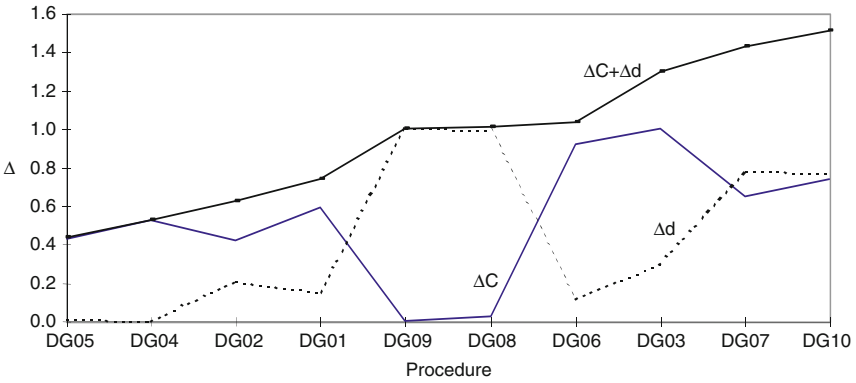

Fig. 5.2 Population diversity versus quality for ten diversification generation methods

or diversity is given by a value of 1 and therefore the maximum score for a diversification method is 2. DG08 is a purely random diversification generation method that achieves the highest level of diversity but also the lowest quality level. DG09 is a purely deterministic method that is designed to achieve maximum diversity without considering the objective-function value during the generation of solutions. The most effective methods, measured by the results achieved at the end of the search, are DG07 and DG10. These methods show an almost perfect balance between diversification and quality of the populations that they generate, as shown by their Δ_C and Δ_d values in Fig. 5.2. DG07 is a semi-greedy construction procedure that uses six different randomized and adaptive functions to select elements to add to a partial solution. DG10 also uses an adaptive function as well as frequency memory to bias the selection of elements during the construction process.

We focus on semi-greedy construction procedures as the basis for developing generation diversification methods. We assume that the problem to be solved consists of minimizing $f(x)$, where x is a solution to the problem. We do not restrict the representation of the problem to any particular form. That is, x may be a vector of real, integer or binary numbers, a permutation, or a set of edges in a graph. We assume, however, that the problem being solved is such that solutions can be constructed by selecting elements one at a time. Selecting an element may refer to choosing a value for a variable, adding an edge to a graph or a task to a sequence. Figure 5.3 outlines the semi-greedy procedure to construct solutions.

Because we assume that the objective of the problem is to minimize a function f, adding elements to a partial solution results in a non-negative change in the objective-function value. Therefore, the increase of the objective function value caused by all the candidate elements is evaluated as the initialization step for the construction process (see step 1 in Fig. 5.3). Following Resende and Ribeiro (2003), let $c(e)$ be the objective-function increase caused by adding element e to the partial solution. Also let $c^{\max}$ and $c^{\min}$ be the maximum and minimum c values, respectively. A purely greedy construction would select, at each iteration, the element e such that $c(e) = c^{\min}$.

1. Evaluate incremental f values for candidate elements
2. **while** (construction is partial) {
3. Build the restricted candidate list (RCL)
4. Randomly choose an element from RCL
5. Add the selected element to the partial solution
6. Re-evaluate incremental f values
7. }

Fig. 5.3 Semi-greedy construction

The procedure in Fig. 5.3 is semi-greedy because the next element to be added is randomly selected from a restricted candidate list (RCL). Candidate elements are assumed to be ordered in such a way that the top candidate has an incremental objective-function value of $c^{\min}$ and the last candidate's incremental objective-function value is $c^{\max}$. The RCL may be formed in two different ways:

- *Cardinality-based*: the top k candidate elements
- *Value-based*: all elements e such $c(e) \leq c^{\min} + \alpha(c^{\max} - c^{\min})$.

Both of these alternatives require a parameter value that controls the level of randomization. For instance, if $\alpha = 1$, the value-based procedure becomes totally random because all candidate elements have the same probability of being chosen. If $\alpha = 0$, the value-based procedure is totally greedy and results in the same construction every time that the procedure is executed.

A variation of the semi-greedy approach consists of adding a frequency memory structure that is typical to tabu search implementations (Glover and Laguna 1997). The main idea is to keep track of the number of times an element has been assigned a certain value (or occupied a certain position). The frequency counts are then used to modify the c values to discourage (in the probabilistic sense) an element to take on a value that has often been assigned to it in previous constructions. Let $q(e, v)$ be the number of times element e has been given the value v (or has been assigned to position v) in previous constructions, then the modified evaluation that measures the attractiveness of choosing element e is given by

$$c'(e) = c(e) + \beta \left(\frac{q(e, v)}{q^{\max}} \right).$$

The value of β should create a balance between the increase in the objective-function value and the importance given to moving the element away from values (or positions) that it has taken in past constructions. In this equation, $q^{\max}$ refers to the maximum frequency count and is used to normalize the individual frequency counts in order to facilitate the tuning of the β value. Note that we refer to an element taken on a particular value v only as a way of simplifying our discussion. However, this should be reinterpreted in each context. For instance, in nonlinear optimization with continuous variables, v could be interpreted as a particular range of values from which a variable should be moving away due to a frequency count that indicates that

```
1. Current solution is the trial solution being improved
2. while (current solution improves) {
3.         Identify the best neighbor of the current solution
4.         if (neighbor solution is better than current solution)
5.                 Make neighbor the current solution
6. }
```

Fig. 5.4 Local search

such a variable has often taken values in the given range. Similarly, v could represent a position in a permutation or a value that indicates whether or not an element (such as an arc in a graph) has been part of the final construction.

When using frequency memory, the selection of elements in each step of the construction does not require the RCL or random components. The first construction results in the pure greedy solution because all frequency counts are zero and $c'(e) = c(e)$. However, as the frequency counts grow, the constructions start diverting from the pure greedy solution. It is also possible, however, to include random components and use the $c'(e)$ to set probabilities proportional to the c' values, where the probability of choosing e increases as $c'(e)$ decreases.

5.3 Improvement Method

As discussed in the introduction to this chapter, most improvement methods within SS implementations consist of local search procedures. Problem context is relevant to the design of an effective local search. The design process starts with the definition of a neighborhood that depends on the moves that may be applied to a solution to transform it into another. Figure 5.4 shows the outline of a local search.

Identifying the *best* neighbor solution may have different meanings according to the problem context and the current solution. For instance, for an unconstrained optimization problem where all the solutions in the solution space generated by a particular solution representation are feasible, the best neighbor of the current solution may be the one that locally minimizes the objective-function value. If the optimization problem contains constraints, then the best neighbor may be either a feasible one that minimizes the objective function, if at least one feasible neighbor exists, or the one that minimizes a measure of infeasibility, if no feasible solutions can be found in the current neighborhood. The local search outlined in Fig. 5.4 results in either a better solution (either a feasible solution with a better objective-function value or an infeasible solution with a smaller infeasibility value) or the unchanged trial solution that was submitted to the improvement method. In either case, the method guarantees finding a locally optimal solution with respect to the defined neighborhood, because it stops only when the current solution cannot be improved (line 2 in Fig. 5.4).

```
1.  Current solution is the trial solution being improved
2.  while (current solution improves) {
3.      Set k to 1
4.      while k ≤ k_max {
5.          Identify the best neighbor of the current solution in
6.          neighborhood k
7.          if (neighbor is better than current solution)
8.              Make neighbor the current solution
9.              Set k to 1
10.         else
11.             Make k = k + 1
12.     }
13. }
```

Fig. 5.5 Variable neighborhood descent

Variable neighborhood descent (VND) represents an attractive alternative for an improvement method within SS. VND (Hansen and Mladenović 2003) operates on a finite set of neighborhood structures that are systematically changed until no improvement is obtained. Let k indicate the current neighborhood being explored and $k_{\max}$ the maximum number of neighborhoods. The VND steps are outlined in Fig. 5.5.

VND is the simplest form of the family of variable neighborhood search (VNS) procedures. The basic VNS employs deterministic and stochastic rules for changing the neighborhood that results in a randomized descent method with a first improvement strategy. Extensions transform the basic VNS into a descent–ascent method with first or best improvement strategies.

The appeal of VND as improvement method for SS is its deterministic orientation and its natural termination criterion. As mentioned in the introduction, allowing the improvement method to operate as a metaheuristic capable of escaping local optimality poses the problem of balancing the time spent improving solutions versus the time spent performing SS iterations. In other words, if the improvement method is a metaheuristic in its own right, then there is a risk of diminishing the role of the SS framework to a simple mechanism that provides starting points. Campos et al. (2005), for instance, test the use of a tabu search with a simple short-term memory that terminates after a number of iterations without improvement. This number is set to a relatively small value (e.g. less than or equal to 10) to balance the amount of computational effort spent in the improvement method and the other SS methods.

While the most effective improvement methods are designed for specific problems, it is also possible to develop generic or pseudo-generic local searches. Consider, for instance, the improvement method developed by Campos et al. (2005). In this work, SS is applied to a class of permutation problems. The improvement method is given limited information about the problem instance. In particular, the improvement method is provided only with a flag that indicates whether the objective function tends to be influenced more by the absolute positions of the elements in the permutation (e.g. as in the linear ordering problem) than by their relative

positions (e.g. as in the traveling salesman problem). The actual form of the objective function is not known and therefore the improvement method operates as a black-box optimizer, requiring a context-independent neighborhood search. For permutation problems, a generic local search could consist of either all swaps of two elements or the insertion of one element in every possible position in the permutation. A swap of elements i and j in a permutation with n elements may be represented as follows:

$$(1,\ldots,i-1,i,i+1,\ldots,j-1,j,j+1,\ldots,n) \\ \rightarrow (1,\ldots,i-1,j,i+1\ldots,j-1,i,j+1\ldots,n).$$

An insertion of element i before element j produces the following changes in the permutation:

$$(1,\ldots,i-1,i,i+1,\ldots,j-1,j,j+1,\ldots,n) \\ \rightarrow (1,\ldots,i-1,i+1,\ldots,j-1,i,j,j+1,\ldots,n).$$

Note that both neighborhoods are large, containing $O(n^2)$ moves, where n is the number of elements in the permutation. To avoid exploring such a large neighborhood (and in the SS spirit of using strategy instead of relying on randomness) frequency information is used to create a list of promising insertion positions for an element in the permutation. A promising position is one where historically (given by a frequency count) the objective function has improved when the element was placed there. The structure and updating of the frequency count depend on the general class of permutation problem. Thus, for an absolute-position permutation problem a frequency count $\text{freq}(i, p_j)$ may indicate the number of times that the objective function improved when moving (via a swap or an insert) element i to position p_j. Likewise, for a relative-position problem a frequency count $\text{freq}(i, j)$ may indicate the number of times that the objective function improved when moving (via a swap or an insert) element i immediately before element j. When exploring a neighborhood, the moves are limited to those where the frequency counts indicate that there may be merit in placing an element under consideration.

5.4 Reference Set Update

The execution of the diversification generation and improvement methods in line 1 of Fig. 5.1 results in a population P of solutions. The reference set update method is executed in two different parts of the SS procedure. The first time that the method is called, its goal is to produce the initial *RefSet,* consisting of a mix of b (typically ten) high-quality and diverse solutions drawn from P. The mix of high-quality and diverse solutions could be considered a tunable parameter; however, most implementations populate the initial reference set with half of the solutions chosen by quality and half chosen by diversity.

Choosing solutions by quality is straightforward. From the population P, the best (according to the objective-function value) $b/2$ solutions are chosen. These solutions are added to *RefSet* and deleted from P. To choose the remaining half, an appropriate measure of distance $d(r,p)$ is needed, where r is a reference solution and p is a solution in P. The distance measure depends on the solution representation, but must satisfy the usual conditions for a metric, that is

$d(r,p) = 0$ if and only if $r = p$
$d(r,p) = d(p,r) > 0$ if $r \neq p$
$d(r,q) + d(q,p) \geq d(r,p)$ triangle inequality.

For instance, the Euclidean distance is commonly used when solutions are represented by continuous variables:

$$d(r,p) = \left(\sum_{i=1}^{n} (r_i - p_i)^2\right)^{\frac{1}{2}}.$$

Likewise, the Hamming (1950) distance is appropriate for two strings of equal length. The distance is given by the number of positions for which the corresponding symbols are different. In other words, the distance is the number of changes required to transform one string into the other:

$$d(r,p) = \sum_{i=1}^{n} v_i \qquad v_i = \begin{cases} 0 & r_i = p_i \\ 1 & r_i \neq p_i. \end{cases}$$

This distance has been typically used for problems whose solution representation is given by binary vectors (e.g. the max-cut and knapsack problems) and permutation vectors (e.g. the traveling salesman, quadratic assignment and linear ordering problems).

The distance measure is used to calculate the minimum distance $d_{\min}(p)$ of a population solution and all the reference solutions r:

$$d_{\min}(p) = \min_{r \in RefSet} d(r,p).$$

Then, the next population solution p to be added to *RefSet* and deleted from P is the one with the maximum $d_{\min}$ value. That is, we want to choose the population solution p^* that has the maximum minimum distance between itself and all the solutions currently in *RefSet*:

$$p^* = \left\{ p : \max_{p \in P} d_{\min}(p) \right\}.$$

The process is repeated $b/2$ times to complete the construction of the initial *RefSet*. Note that after the first calculation of the $d_{\min}$ values, they can be updated as follows. Let p^* be the population solution most recently added to the *RefSet*. Then, the $d_{\min}$ value for a population solution p is given by

$$d_{\min}(p) = \min\left(d_{\min}(p), d(p,p^*)\right).$$

Alternatively, the diverse solutions for the initial reference set may be chosen by solving the maximum diversity problem (MDP). The MDP consists of finding, from a given set of elements and corresponding distances between elements, the most diverse subset of a given size. The diversity of the chosen subset is given by the sum of the distances between every pair of elements. The special version of the MDP that must be solved includes a set of elements that have already been chosen (i.e. the high-quality solutions). Mathematically, the problem may be formulated as follows:

$$\begin{aligned} &\text{Maximize } \textstyle\sum_{(p,p')\in P: p\neq p'} d\,(p,p')x_p x_{p'} \\ &\text{Subject to } \textstyle\sum_{p\in P} x_p = b \\ &\qquad\qquad x_p = 1 \quad \forall p \in RefSet \\ &\qquad\qquad x_p = \{0,1\} \quad \forall p \in P. \end{aligned}$$

The formulation assumes that a subset of high-quality solutions in P have already been chosen and added to *RefSet*. The binary variables indicate whether a population solution p is chosen ($x_p = 1$) or not ($x_p = 0$). The second set of constraints in the formulation force the high-quality solutions to be included in the set of b solutions that will become the initial *RefSet*. This nonlinear programming model has been translated into an integer program for the purpose of solving it as well as for showing that the MDP is NP-hard. Martí et al. (2009) embed the MDP in a SS procedure for the max-cut problem. Instead of solving the MDP exactly, they employ the GRASP_C_2 procedure developed by Duarte and Martí (2007). The procedure was modified to account for the high-quality solutions that are chosen before the subset of diverse solutions is added to the *RefSet*. The procedure is executed for 100 iterations and the most diverse *RefSet* is chosen to initiate the SS.

The reference set update method is also called in step 8 of Fig. 5.1. This step is performed after a set of one or more trial solutions has been generated by the sequential calls to the subset generation, combination and improvement methods (see steps 5–7 in Fig. 5.1). The most common update consists of the selection of the best (according to the objective function value) solutions from the union of the reference set and the set of trial solutions generated by steps 5–7 in Fig. 5.1. Other updates have been suggested in order to preserve a certain amount of diversity in the *RefSet*. These advanced updating mechanisms are beyond the scope of this tutorial chapter but the interested reader is referred to Laguna and Martí (2003, Chap. 5) for a detailed description and to Laguna and Martí (2005) for experimental results.

5.5 Subset Generation

This method is in charge of providing the input to the combination method. This input consists of a list of subsets of reference solutions. The most common subset generation consists of creating a list of all pairs (i.e. all 2-subsets) of reference solutions for which at least one of the solutions is new. A reference solution is new if it hasn't been used by the combination method. The first time that this method is called (step 5 in Fig. 5.1), all the reference solutions are new, given that the method

is operating on the initial *RefSet*. Therefore, the execution of the subset generation method results in the list of all 2-subsets of reference solutions, consisting of a total of $(b^2 - b)/2$ pairs. Because the subset generation method is not based on a sample but rather on the universe of all possible pairs, the size of the *RefSet* in SS implementation must be moderate (e.g. less than 20). As mentioned in the introduction, a typical value for b is 10, resulting in 45 pairs the first time that the subset generation method is executed.

When the inner while-loop (steps 5–8 in Fig. 5.1) is executing, the number of new reference solutions at the time that the subset generation method is called depends on the strategies implemented in the reference set update method. Nonetheless, the number of new solutions decreases with the number of iterations within the inner while-loop. Suppose that b is set to 10 and that, after the first iteration of the inner while-loop, six solutions are replaced in the reference set. This means that the reference set that will serve as the input to the subset generation method will consist of four old solutions and six new solutions. Then, the output of the subset generation method will be 39 2-subsets. In general, if the reference set contains n new solutions and m old ones, the number of 2-subsets that the subset generation method produces is given by

$$nm + \frac{n^2 - n}{2}.$$

The SS methodology also considers the generation of subsets with more than two elements for the purpose of combining reference solutions. As described in Laguna and Martí (2003), the procedure uses a strategy to expand pairs into subsets of larger size while controlling the total number of subsets to be generated. In other words, the mechanism does not attempt to create all 2-subsets, then all 3-subsets, and so on until reaching the $b - 1$-subsets and finally the entire *RefSet*. This approach would not be practical because there are 1,013 subsets in a reference set of size $b = 10$. Even for a smaller reference set, combining all possible subsets would not effective because many subsets will be very similar. For example, a subset of size four containing solutions 1–4 is almost the same as all the subsets with four solutions for which the first three solutions are solutions 1–3. And even if the combination of subset {1, 2, 3, 5}would generate a different solution than the combination of subset {1, 2, 3, 6}, these new trial solutions would likely reside in the same basin of attraction and therefore converge to the same local optimum after the application of the improvement method. Instead, the approach selects representative subsets of different sizes by creating subset types:

- *Subset Type* 1: all 2-element subsets.
- *Subset Type* 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
- *Subset Type* 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
- *Subset Type* 4: the subsets consisting of the best i elements, for $i = 5$ to b.

Campos et al. (2001) designed an experiment with the goal of assessing the contribution of combining subset types 1–4 in the context of the linear ordering problem. The experiment undertook to identify how often, across a set of benchmark

problems, the best solutions came from combinations of reference solution subsets of various sizes. The experimental results showed that most of the contribution (measured as the percentage of time that the best solutions came from a particular subset type) could be attributed to subset type 1. It was acknowledged, however, that the results could change if the subset types were generated in a different sequence. Nonetheless, the experiments indicate that the basic SS that employs only subsets of type 1 is quite effective and explains why most implementations do not use subset types of higher dimensions.

5.6 Solution Combination

The implementation of this method depends on the solution representation. Problem context can also be exploited by this method; however, it is also possible to create context-independent combination mechanisms. When implementing a context-independent procedure, it is beneficial to employ more than one combination method and track their individual performance. Consider, for instance, the context-independent SS implementations for permutation problems developed by Campos et al. (2005) and Martí et al. (2005). The following combination methods for permutation vectors with n elements were proposed:

1. An implementation of a classical GA crossover operator. The method randomly selects a position k to be the crossing point from the range $[1, n/2]$. The first k elements are copied from one reference point while the remaining elements are randomly selected from both reference points. For each position i ($i = k + 1, \ldots, n$) the method randomly selects one reference point and copies the first element that is still not included in the new trial solution.
2. A special case of 1, where the crossing point k is always fixed to one.
3. An implementation of what is known in the GA literature as the *partially matched crossover*. The method randomly chooses two crossover points in one reference solution and copies the partial permutation between them into the new trial solution. The remaining elements are copied from the other reference solution preserving their relative ordering (Michalewicz 1994).
4. A special case of what the GA literature refers to as a *mutation operator*, and it is applied to a single solution. The method selects two random positions in a chosen reference solution and inverts the partial permutation between them. The inverted partial permutation is copied into the new trial solution. The remaining elements are directly copied from the reference solution preserving their relative order.
5. A combination method that operates on a single reference solution. The method scrambles a sublist of elements randomly selected in the reference solution. The remaining elements are directly copied from the reference solution into the new trial solution.
6. A special case of combination method 5 where the sublist always starts in position 1 and the length is randomly selected in the range $[2, n/2]$.

7. A method that scans (from left to right) both reference permutations, and uses the rule that each reference permutation votes for its first element that is still not included in the combined permutation (referred to as the incipient element). The voting determines the next element to enter the first still-unassigned position of the combined permutation. This is a min–max rule in the sense that if any element of the reference permutation is chosen other than the incipient element, then it would increase the deviation between the reference and the combined permutations. Similarly, if the incipient element were placed later in the combined permutation than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the combined solution from the reference solution under consideration, subject to the fact that the other reference solution is also competing to contribute. A bias factor that gives more weight to the vote of the reference permutation with higher quality is also implemented for tie breaking. This rule is used when more than one element receives the same votes. Then the element with highest weighted vote is selected, where the weight of a vote is directly proportional to the objective-function value of the corresponding reference solution. Additional details about this combination method can be found in Campos et al. (2001).
8. A variant of combination method 7. As in the previous method, the two reference solutions vote for their incipient element to be included in the first still-unassigned position of the combined permutation. If both solutions vote for the same element, the element is assigned. But in this case, if the reference solutions vote for different elements and these elements occupy the same position in both reference permutations, then the element from the permutation with the better objective function is chosen. Finally, if the elements are different and occupy different positions, then the one in the lower position is selected.
9. Given two reference solutions r_1 and r_2, this method probabilistically selects the first element from one of these solutions. The selection is biased by the objective-function value corresponding to r_1 and r_2. Let e be the last element added to the new trial solution. Then, r_1 votes for the first unassigned element that is positioned after e in the permutation r_1. Similarly, r_2 votes for the first unassigned element that is positioned after e in r_2. If both reference solutions vote for the same element, the element is assigned to the next position in the new trial solution. If the elements are different then the selection is proportionally weighted by the objective-function values of r_1 and r_2.
10. A deterministic version of combination method 9. The first element is chosen from the reference solution with the better objective-function value. Then reference solutions vote for the first unassigned successor of the last element assigned to the new trial solution. If both solutions vote for the same element, then the element is assigned to the new trial solution. Otherwise, the "winner" element is determined with a score, which is updated separately for each reference solution in the combination. The score values attempt to keep the proportion of times that a reference solution "wins" close to its relative importance, where the importance is measured by the value of the objective function. The scores are calculated to minimize the deviation between the "winning rate" and the "relative importance". For example, if two reference solutions r_1 and r_2 have

objective-function values of value(r_1) = 40 and value(r_2) = 60, then r_1 should contribute with 40 % of the elements in the new trial solution and r_2 with the remaining 60 % in a maximization problem. The scores are updated so that after all the assignments are made the relative contribution from each reference solution approximates the target proportion. More details about this combination method can be found in Glover (1994).

The form of these combination methods show that SS provides great flexibility in terms of generating new trial solutions. That is, the methodology accepts fully deterministic combination methods or those containing random elements that are typically used in genetic algorithms (and labeled crossover or mutation operators). At each execution of step 6 of Fig. 5.1, a combination method is randomly selected. Initially, all combination methods have the same probability of being selected but, as the search progresses and after a specified number of iterations, the probability values increase for those combination methods that have been more successful. Success is measured by the quality of the solutions that the methods are able to produce. Suppose that the *RefSet* is ordered in such a way that the first solution is the best and the bth solution is the worst (according to the objective-function value). Then a *score* for each combination method is kept throughout the search. Initially, all the scores are zero. If a combination method generates a solution that is admitted as the jth reference solution, then a value of $b - j+1$ is added to the score of this combination method. The probability of selecting a combination method is proportional to its score and therefore combination methods that generate high-quality solutions accumulate higher scores and increase their probability of being chosen.

The same procedure is used by Gortazar et al. (2010) in the context of binary problems. They develop seven combination methods that are probabilistically selected according to the success score. One of these combination methods is based on the path relinking strategy. As described in Martí et al. (2006), the strategy of creating trajectories of moves passing through high-quality solutions was first proposed in connection with tabu search by Glover (1998). The approach was then elaborated in greater detail as a means of integrating intensification and diversification strategies, and given the name path relinking (PR), in the context of tabu search (Glover and Laguna 1997). PR generally operates by starting from an *initiating* solution, selected from a subset of high-quality solutions, and generating a path in the neighborhood space that leads toward the other solutions in the subset, which is called the *guiding* solution. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. PR variants consider the use of more than one guiding solution as well as reversing the roles of initiating and guiding solutions resulting in the approach known as *simultaneous relinking*.

Path relinking can be considered an extended form of the combination method. Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed.

Gortazar et al. (2010) apply path relinking to combine two binary vectors as follows. The procedure gradually transforms the initiating solution into the guiding solution by changing the value of the variables in the initiating solution with their value in the guiding solution. If the value is the same in both solutions, then no change is made and the procedure moves to the next variable. The procedure examines the variables in lexicographical order and in at most n steps (where n is the number of variables) it reaches the guiding solution. The procedure uses a first-improving strategy, meaning that if during the relinking process it finds an intermediate solution that is better than either the initiating or guiding solutions, then the procedure stops. If no better solution is found, the solution that is most distant from the initiating and guiding solutions is the combined solution resulting from the application of this method. In addition, the procedure reverses the roles of the initiating and guiding solutions and chooses the best solution found during both processes to be the outcome of the combination method.

5.7 Multiobjective Optimization

Evolutionary procedures have enjoyed a fair amount of success in multiobjective optimization, as documented by Coello et al. (2002). SS is starting to be applied to this area. Molina et al. (2007) developed a tabu/SS hybrid for approximating the efficient frontier of multiobjective nonlinear problems with continuous variables. The method consists of two major phases:

1. Generation of an initial set of efficient points through various tabu searches;
2. Combination of solutions and updating of the efficient frontier approximation via a SS.

As we have discussed above for single-objective problems, the *RefSet* contains a mixture of high-quality and diverse solutions, where quality is measured with reference to the single objective function and diversity is measured by an appropriate metric in the solution space. The role of the *RefSet* must be modified to deal with the special characteristics of multiobjective optimization. In particular, solution quality is measured considering multiple objective functions and solution diversity is measured in the objective-function space. The notion of diversity is related to the ability of finding solutions that cover the efficient frontier. Hence, measuring diversity in the objective-function space is an effective means to produce the desired results.

One of the key search mechanisms is the combination of solutions that are currently considered efficient and therefore belong to the best approximation of the efficient frontier. The *RefSet* is a subset of efficient solutions of a size larger than the number of objective functions in the problem and is initially constructed as follows:

- Select the best solution in the current approximation of the efficient froniter for each of the objective functions and add them to *RefSet*. (Note that it is possible, but unlikely, to select fewer solutions than the number of objective function if a solution happens to be best for more than one objective function.)

- Select additional solutions (up to a total of b) from the approximation of the efficient frontier in order to maximize the distance between them and those solutions already in the *RefSet*, where distance is measured in the objective-function space.

The construction of the initial *RefSet* reveals that in the multiobjective implementation of SS, the population P is the best approximation of the efficient frontier. This is an expanded role for P (when compared to single-objective optimization) because it not only supports the diversification of the *RefSet* but also acts as a repository of efficient solutions. A list of solutions that have been selected as reference points is kept to prevent the selection of those solutions in future iterations. Therefore, every solution that is added to the *RefSet* is also added to a *TabuRefSet*. The size of the *TabuRefSet* increases as the search progresses because this memory function is an explicit record of past reference solutions. The motivation for creating and maintaining the *TabuRefSet* is that the final approximation of the efficient frontier must have adequate density. To achieve this, the procedure must encourage a uniform generation of points in the efficient frontier and avoid gaps that may be the result of generating too many points in one region while neglecting other regions.

Combination methods operate in the solution space and therefore are similar to their single-objective counterpart. Improvement methods, however, must consider more than one objective and are typically based on concepts from compromise programming. A global criterion is used to guide the search and the goal is to minimize a function that measures the distance to an ideal point. Given that the ideal point consists of the optimal (or best known) values of the individual objective functions, compromise programming assumes that it is logical for the decision maker to prefer a point that is closer to the ideal point over one that is farther away.

The same framework has been adapted for multiobjective combinatorial optimization. In particular, Caballero et al. (2011) tackled partitioning problems for cluster analysis that requires the simultaneous optimization of more than one objective function. They considered two main classes of multiobjective partitioning problems: (1) partitioning of objects using one partitioning criterion but multiple dissimilarity matrices and (2) partitioning of objects using one dissimilarity matrix but more than one partitioning criteria. The adaptation consists of formulating an appropriate solution representation and employing the representation to develop combination and improvement methods. Other methods (particularly those that operate in the objective-function space) were applied with minor or no changes.

5.8 Tricks of the Trade

Tricks associated with implementing SS are extensively addressed in three tutorial Chaps. (2–4) in Laguna and Martí (2003). We summarize a few here:

1. Effective diversification generation methods employ controlled randomization and frequency-based memory to generate a set of diverse solutions. The use of frequency-based memory is typical in implementations of tabu search.

2. The diversification generation method is used at the beginning of the search to generate a set of diverse solutions. In most SS applications, the size of this population of solutions is generally set at $\max(100, 5 * b)$, where b is the size of the reference set.
3. While some solution generation is done without considering the objective function, in other words, some diversification generation methods focus on diversification and not on the quality of the resulting solutions, it is generally more effective to design a procedure that balances diversification and solution quality (such as those based on GRASP constructions).
4. Improvement methods must be capable of handling starting solutions that are either feasible or infeasible. When encountering an infeasible solution, an improvement method should search for a feasible solution first and then launch a search for improvement.
5. Whenever possible, the improvement method should be a known local search procedure. For example, when using SS for nonlinear optimization, the improvement method could be the well-known Nelder and Mead simplex procedure.
6. The reference set update method to try first is the so-called *static update*. Trial solutions that are constructed as combination of reference solutions are placed in a solution *pool*. After the application of both the combination method and the improvement method, the *pool* is full and the reference set is updated. The new reference set consists of the best b solutions (where b is the size of the reference set) from the solutions in the current reference set and the solutions in the *pool*, i.e. the updated reference set contains the best b solutions in the union of the reference set and the *pool*.
7. The subset generation method should be limited to generating all solution pairs first before trying more sophisticated designs to include more than two solutions in each subset. Even when the combination method includes stochastic elements, it is generally more effective to create only solution subsets that include at least one new reference solution (i.e. a solution that has been added to the reference set in the previous iteration).
8. The number of solutions created from the combination of two or more reference solutions should depend on the quality of the solutions being combined. For instance, the maximum number of solutions generated by the combination method should happen when the two best solutions in the reference set are being combined. Likewise, only one solution should be the result of combining the two reference solutions with the worst objective-function value.
9. If SS is implemented to exploit problem context, the combination method should take full advantage of the information associated with the problem being solved. For instance, a combination method for the linear ordering problem should take into consideration that the largest contribution to the objective-function value comes from the items that are placed in the first positions of the permutation.
10. Employing multiple combination methods has been shown to be an effective strategy. The combination methods are applied probabilistically, starting with an equal probability of selecting any of the available methods. The probability changes, with the success of each method, where success is typically defined

as creating trial solutions that are admitted to the reference set because of their quality. As the search progresses, the more effective (i.e. successful) methods are chosen more often. This strategy is particularly useful when SS is used as a black-box optimizer, where no context information is used to create combination methods that are known to be effective for certain classes of problems.

5.9 Conclusions

The goal of this chapter is to introduce the SS framework at a level that would make it possible for the reader to implement a basic but robust procedure. A number of extensions are possible and some of them have already been explored and reported in the literature. It is not possible within the limited scope of this chapter to detail completely many of the aspects of SS that warrant further investigation. Additional implementation considerations, including those associated with intensification and diversification processes, and the design of accompanying methods to improve solutions produced by combination strategies are found in several of the references listed below.

5.10 Promising Areas for Future Research

SS is at the core of OptQuest, a popular commercial software package for global optimization. OptQuest is implemented as a black-box optimizer that focuses on searching for high-quality solutions to problems with expensive objective-function evaluations (such as those characterized by a computer simulation). SS has shown merit in applications where the optimization horizon (represented by a number of objective function evaluations) is severely limited. This is a promising SS research avenue, considering the importance of optimizing black boxes in general and simulations in particular.

One way of creating effective heuristic search procedures with limited objective-function evaluations is through the use of rough set theory. Rough sets have been successfully adapted as a mechanism to combine solutions and to perform local optimization in the context of multiobjective optimization. They have been hybridized with search methods such as differential evolution (Hernández-Díaz et al. 2006), particle swarm optimization (Santana-Quintero et al. 2006) and metamodels based on radial basis functions (Santana-Quintero et al. 2007). Multiobjective optimization results are encouraging, showing the effectiveness of the hybrid implementations in obtaining dense approximations of the Pareto fronts. Additional details on the use of rough sets in multiobjective optimization can be found in Hernández-Díaz et al. (2008). These studies point to the merit of exploring the addition of mechanisms based on rough sets to SS implementations for multiobjective optimization.

While the merging of SS and rough sets for multiobjective optimization remains to be explored, Laguna et al. (2010) employed a rough-set theory procedure as a combination method within a SS for nonlinear optimization with a single multimodal objective function. The goal of this work was to find high-quality solutions to difficult multimodal functions while limiting the number of objective-function evaluations. The search process was divided into two stages, starting from a coarsely discretized solution space and ending at the original solution space represented by continuous variables. The authors adapted the standard SS methodology to provide the data that rough-set theory needs to identify promising areas in the solution space. The implementation departed from the traditional SS framework in that a sampling procedure was used to create subsets of solutions to which the rough-set combination method was applied. Similar mechanisms are worth exploring in order to create innovative ways of combining solutions for both single-objective and multiobjective optimization problems that are tackled with SS.

The experiments with 92 problems instances from the literature presented by Laguna et al. (2010) showed the merit of the SS/rough-set combination when compared to an existing method based on particle swarm optimization. A potential extension to this work is in the area of simulation optimization. As mentioned above, the evaluation of the objective function in this context typically requires a significant amount of computational effort and is noisy. The use of rough sets to reduce the number of evaluations required to identify promising regions in the solution space may be of great advantage, particularly in situations where the number of objective-function evaluations (i.e. calls to the simulation module) is limited to no more than 100 times the number of decision variables in the problem. Coupling rough sets, metamodels and ranking and selection may result in a highly effective approach for dealing with expensive and noisy objective functions. Also, given that rough sets have been shown to be effective in multiobjective optimization (see Hernández-Díaz et al. 2006, 2008), another promising research avenue is the development of a SS/rough-sets for multiobjective simulation optimization, an area that has generated a fair amount of interest in engineering and science (Lee et al. 1996; Mebarki and Castagna 2000; Yang and Chou 2005; Pasandideh and Niaki 2006; Rosen et al. 2007, 2008; Teng et al. 2007; Zhang 2008; Willis and Jones 2008).

Sources of Additional Information

The best source of information to get started with SS is the book by Laguna and Martí (2003). This book contains three tutorials, including searches in continuous spaces (nonlinear unconstrained optimization), constrained binary spaces (knapsack problems) and combinatorial optimization (linear ordering problem). Advanced strategies are also addressed and computer code is provided that can be easily modified to create basic and even advanced implementation of SS for other optimization problems.

Several SS tutorials have appeared in the literature and can be found at http://leeds-faculty.colorado.edu/laguna. Some of these chapters and tutorials include implementations that use path relinking as a mechanism to combine solutions within a SS framework.

The Optsicom Website (http://heur.uv.es/optsicom/) contains the description of several optimization procedures based on metaheuristics, including SS. Implementations of SS as black-box optimizer and for particular problem classes can be found in this website.

References

Caballero R, Laguna M, Martí R, Molina J (2011) Scatter tabu search for multiobjective clustering problems. J Oper Res Soc 62:2034–2046

Campos V, Glover F, Laguna M, Martí R (2001) An experimental evaluation of a scatter search for the linear ordering problem. J Glob Optim 21:397–414

Campos V, Laguna M, Martí R (2005) Context-independent scatter and tabu search for permutation problems. INFORMS J Comput 17:111–122

Coello CA, Van Veldhuizen DA, Lamont GB (2002) Evolutionary algorithms for solving multi-objective problems. Kluwer/Plenum, New York

Duarte A, Martí R (2007) Tabu search and GRASP for the maximum diversity problem. Eur J Oper Res 178:71–84

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133

Glover F (1994) Tabu search for nonlinear and parametric optimization with links to genetic algorithms. Discret Appl Math 49:231–255

Glover F (1998) A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D (eds) Artificial evolution. Lecture notes in computer science, vol. 1363. Springer, Berlin, pp 1–5

Glover F, Laguna M (1997) Tabu search. Kluwer, Boston

Glover F, Laguna M, Martí R (2003a) Scatter search. In: Ghosh A, Tsutsui S (eds) Advances in evolutionary computation: theory and applications. Springer, New York, pp 519–537

Glover F, Laguna M, Martí R (2003b) Scatter search and path relinking: advances and applications. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 1–35

Glover F, Laguna M, Martí R (2004) New ideas and applications of scatter search and path relinking. In: Onwubolu GC, Babu BV (eds) New optimization techniques in engineering. Springer, Berlin, pp 367–383

Gortazar F, Duarte A, Laguna M, Martí R (2010) Context-independent scatter search for binary problems. Comput Oper Res 37:1977–1986

Hamming RW (1950) Error detecting and error correcting codes. Bell Syst Tech J 26:147–160

Hansen P, Mladenović N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 145–184

Hernández-Díaz AG, Santana-Quintero LV, Coello Coello CA, Caballero R, Molina J (2006) A new proposal for multiobjective optimization using differential evolution and rough set theory. In: Proceedings of the genetic and evolutionary computation conference. ACM, New York, Seattle, Washington, pp 675–683

Hernández-Díaz AG, Santana-Quintero LV, Coello Coello CA, Caballero R, Molina J (2008) Improving multi-objective evolutionary algorithms by using rough sets. In: Ligeza A, Reich S, Schaefer R, Cotta C (eds) Knowledge-driven computing: knowledge engineering and intelligent computations. Studies in computational intelligence, vol 102. Springer-Verlag Berlin Heidelberg, pp 81–98

Laguna M (2002) Scatter search. In: Pardalos PM, Resende MGC (eds) Handbook of applied optimization. Oxford University Press, New York, pp 183–193

Laguna M, Martí R (2003) Scatter search: methodology and implementations in C. Kluwer, Boston

Laguna M, Martí R (2005) Experimental testing of advanced scatter search designs for global optimization of multimodal functions. J Glob Optim 33:235–255

Laguna M, Molina J, Pérez F, Caballero R, Hernández-Díaz A (2010) The challenge of optimizing expensive black boxes: a scatter search/rough set theory approach. J Oper Res Soc 61:53–67

Lee Y-H, Shin H-M, Yang B-H (1996) An approach for multiple criteria simulation optimization with application to turning operation. Comput Indust Eng 30:375–386

Martí R, Laguna M, Campos V (2005) Scatter search vs. genetic algorithms: an experimental evaluation with permutation problems. In: Rego C, Alidaee B (eds) Metaheuristic optimization via adaptive memory and evolution: tabu search and scatter search. Kluwer, Norwell, pp 263–282

Martí R, Laguna M, Glover F (2006) Principles of scatter search. Eur J Oper Res 169:359–372

Martí R, Duarte A, Laguna M (2009) Advanced scatter search for the max-cut problem. INFORMS J Comput 21:26–38

Mebarki N, Castagna P (2000) An approach based on Hotelling's test for multicriteria stochastic simulation-optimization. Simul Pract Theor 8:341–355

Michalewicz Z (1994) Genetic algorithms + data structures = evolution programs. Springer, Berlin

Molina J, Laguna M, Martí R, Caballero R (2007) SSPMO: a scatter search procedure for non-linear multiobjective optimization. INFORMS J Comput 19:91–100

Pasandideh SHR, Niaki STA (2006) Multi-response simulation optimization using genetic algorithm within desirability function framework. Appl Math Comput 175:366–382

Resende MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics. Kluwer Academic Publishers, pp 219–249

Rosen SL, Harmonosky CH, Traband MT (2007) A simulation optimization method that considers uncertainty and multiple performance measures. Eur J Oper Res 181:315–330

Rosen SL, Harmonosky CH, Traband MT (2008) Optimization of systems with multiple performance measures via simulation: survey and recommendations. Comput Ind Eng 54:327–339

Santana-Quintero LV, Ramírez-Santiago N, Coello Coello CA, Molina J, Hernández-Díaz AG (2006) In: Runarsson ThP, Beyer H-G, Burke E, Merelo-Guervós JJ, Whitley LD, Yao X (eds) Parallel problem solving from nature – PPSN IX. Lecture notes in computer science, vol 4193. Springer, Berlin/New York, pp. 483–492

Santana-Quintero LV, Serrano-Hernández VA, Coello Coello CA, Hernández-Díaz AG, Molina J (2007) Use of radial basis functions and rough sets for evolutionary multiobjective optimization. In: IEEE symposium on computational intelligence in multicriteria decision making (MCDM 07), Seattle, Washington, pp 107–114

Teng S, Lee JH, Chew EP (2007) Multi-objective ordinal optimization for simulation optimization problems. Automatica 43:1884–1895

Willis KO, Jones DF (2008) Multi-objective simulation optimization through search heuristics and relational database analysis. Decis Support Syst 46:277–286

Yang T, Chou P (2005) Solving a multiresponse simulation-optimization problem with discrete variables using a multi-attribute decision-making method. Math Comput Simul 68:9–21

Zhang H (2008) Multi-objective simulation optimization for earthmoving operations. Automat Constr 18:79–86