A Black-Box Scatter Search for Optimization Problems with Integer Variables

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

FRANCISCO GORTÁZAR, MICAEL GALLEGO, ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain
{francisco.gortazar, micael.gallego, abraham.duarte}@urjc.es

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universitat de València, Spain
Visiting the Leeds School of Business, University of Colorado at Boulder, USA
rafael.marti@uv.es

ABSTRACT

The goal of this work is the development of a black-box solver based on the scatter search methodology. In particular, we seek a solver capable of obtaining high quality outcomes to optimization problems for which solutions are represented as a vector of integer values. We refer to these problems as integer optimization problems. We assume that the decision variables are bounded and that there may be constraints that require that the black-box evaluator is called in order to know whether they are satisfied. Problems of this type are common in operational research areas of applications such as telecommunications, project management, engineering design and the like. Our experimental testing includes 171 instances within four classes of problems taken from the literature. The experiments compare the performance of the proposed method with both the best context-specific procedures designed for each class of problem as well as context-independent commercial software. The experiments show that the proposed solution method competes well against commercial software and that can be competitive with specialized procedures in some problem classes.

KEYWORDS: Black-box optimization, Metaheuristics, Hard optimization problems

June 16, 2012

1. Introduction

The black-box optimization framework is the OR (operations research) community response to the need for modeling flexibility and the separation between the modeling environment and the solution method. Achievements and progress in the solution of mathematical models of various types, including integer programs, are numerous and well-documented in the OR literature. Nonetheless, casting a problem as a mathematical model (integer or continuous, linear or nonlinear, or mixed) remains an exercise of imagination, abstraction and compromises. The chief compromise is associated with the set of real elements that the modeling framework may not support or that their inclusion makes the model unworkably complex. Uncertainty is a typical example of this situation, where clever but still limited modeling artifacts have been designed —for instance, those related to robust optimization techniques. The modeling boundaries broaden when the concerns of how the optimization problem will be solved are removed. That is precisely the role of black-box optimization. For example, for problems where uncertainty is not only real but relevant, tools such as computer simulation provide a rich environment to capture reality by liberating the analyst from the burden of maintaining a rigid structure that will conform to the norms dictated by the solution procedure. Thus, evaluating the merit of a solution is not restricted to a mathematical expression of a given type (e.g., a linear equation for linear programming).

Black-box optimization refers to the process in which there is a complete separation between the evaluation of the objective function —and perhaps other functions used to enforce constraints— and the solution procedure, as illustrated in Figure 1. In simulation-optimization, the black box model has been extremely popular in both research and practice. Commercial simulation software began adding black-box optimizers in the early 1990's. Nowadays, optimizers based on metaheuristic technology dominate the application space in all areas of computer simulation, including Monte Carlo, discrete-event, systems dynamics and agent based.

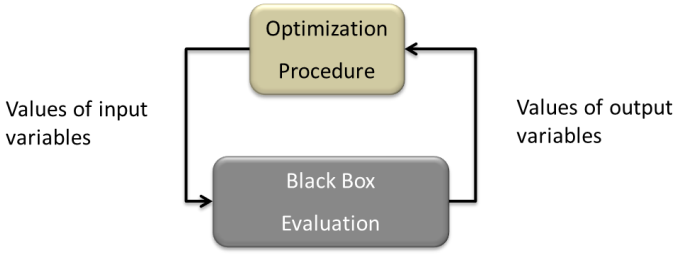

Figure 1. Schematic representation of the black-box optimization framework

This paradigm is not only useful in the case of optimizing simulations but also in situations where the real system is best represented by computational processes or algorithms that do not necessarily contain random elements but that are complex in nature. For instance, (Schittekat & Sörensen, 2009) describes the use of commercial software for vehicle routing as an evaluation box for a vendor selection problem. A search is conducted to find the best combination of 3PL (third party logistics) vendors, where each combination represents a solution to the problem and whose evaluation consists of solving a detailed distribution and delivery plan that the commercial software produces. In this case, the black-box evaluation is the result of solving a deterministic but complex optimization problem.

Engineering design has been a fertile ground for the application of black-box optimization. There is a simple explanation for this phenomenon. Over the course of many years, engineers have developed and tested rather complex computer codes to evaluate designs. Wrapping optimization routines around these codes became the simplest way to take advantage of this work, avoiding the arduous task of tweaking the internal logic and structure of the available computer codes in order to add optimization functionality. It is also an effective mechanism to bring together expertise from two different communities: the optimization gurus and the experts in the particular area of application. In the black-box environment, the optimization experts do not need to know the details of the system being optimized, in the same way that the context experts do not need to know the intricacies of optimization.

Consider, for example, the public-domain software called EPANET, developed and distributed by the Water Supply and Water Resources Division of the United States Environmental Protection Agency. This software models the hydraulics and water quality of water distribution piping systems. A design in this context is a selection of a set of pipes, nodes, pumps, valves and storage tanks or reservoirs, along with their properties and operational rules. The problem of evaluating a design is complex (Yates, Templeman, & Boffey, 1984) and this is why many person-hours have gone into developing EPANET. For a particular design, EPANET tracks the flow of water in each pipe, the pressure at each node, the height of the water in each tank, and the concentration of a chemical species throughout the network during a simulation period (Rossman, 2000). EPANET may be used to design a new network or expand an existing one. In a recent study (Vasan & Simonovic, 2010) a differential evolution algorithm was developed to optimize pipe network designs that are evaluated with EPANET. Designs are evaluated in terms of cost and resilience (defined as the design's capability to overcome sudden failures). Their experiments show the merits of the differential evolution algorithm when compared to similar approaches, such as (Baños, Gil, Agulleiro, & Reca, 2007) and (Lippai, Heaney, & Laguna, 1999).

As discussed in (Laguna, Molina, Pérez, Caballero, & Hernández-Díaz, 2010), "the challenge of optimizing black boxes is to develop methods that can produce outcomes of reasonable quality without taking advantage of problem structure and employing a computational effort that is adequate for the context." Equally challenging is the task of providing enough modeling flexibility to tackle optimization problems in a variety of environments. For instance, the differential evolution procedure designed in (Vasan & Simonovic, 2010) focuses on searching in a solution space defined by a vector of integer values. This was achieved by "codifying" the set of available pipe diameters into contiguous integers where zero represents the "no pipe" option. This solution representation is also popular in the optimization of business process simulations, where choosing the best mix of resources is critical. As shown in (April, Better, Glover, Kelly, & Laguna, 2005), the optimization of a computer simulation of a healthcare facility deals with choosing the right number of physicians, number of nurses, number of lab technicians, number of receptionists and number of examination rooms. Likewise, optimizing the staffing levels at a call center implies the selection of the right number of people at the right time in order to provide sales capacity and customer service for a number of products. In both cases, solutions are also represented as vectors of integers, where larger numbers result in increased resource levels.

The binary representation is a special case of the integer representation, where the variables are limited to take on only two values: zero or one. Optimization methods for binary problems have a long tradition, particularly in the area of evolutionary computation. Recall that the original genetic algorithm proposals favored binary encodings as a universal form of representing solutions. In some cases, binary vectors are indeed the natural representation of the relevant decisions. One example is the aforementioned work by Schittekat and Sörensen (Schittekat & Sörensen, 2009), where choosing vendors is best conceptualized as assigning a value of one to those that are chosen and a value of zero to those vendors that are not chosen. Since there are no restrictions in the number of vendors to choose, then every combination of ones and zeros (perhaps with the exception of all zeros, which results in not choosing any vendors at all) is a valid solution to the problem.

Commercial general-purpose metaheuristic optimizers, such as Opttek's OptQuest, Palisade's Evolver and Frontline's Premium Solver, provide solution-representation flexibility by allowing the modeling of optimization problems with a mix of continuous, integer, permutation, binary and other special types of variables. However, added flexibility translates into complex search spaces and increased difficulty in finding high-quality solutions. Previous studies have shown how even a small amount of specialization makes a significant difference in the performance of black-box optimizers. Campos, Laguna and Martí (Campos, Laguna, & Martí, 2005) developed a black-box optimization procedure for problems for which solutions are represented as permutations. Computational tests showed that their procedure outperformed the general-purpose optimizers that are not specialized to only tackle permutation problems. Similar results were obtained in (Gortázar, Duarte, Laguna, & Martí, 2010) for problems with binary variables.

Some solution methodologies have been designed with the black-box framework in mind, as in the case of random-key genetic algorithms (RKGA). This is an evolutionary computation procedure whose main search mechanisms are problem-independent crossover and mutation operators. They were originally introduced in (Bean, 1994) for the solution of combinatorial optimization problems related to sequencing. Solutions in RKGA implementations are represented as vectors of real numbers that are randomly generated between 0 and 1. Each vector corresponds to a feasible solution and mating is performed using the so-called parameterized uniform crossover (Spears & DeJong, 1991), which is designed to always produce a feasible offspring. The associated decoder is a deterministic algorithm that takes a random-key vector as input and returns a feasible solution of the optimization problem along with the objective function value. The decoder acts as a black box and is the only problem-dependent element of the solution procedure.

Additional examples of procedures that are partially based on the black-box paradigm are Bias Random Key (Gonçalves & Resende, 2011), Cross Entropy (Laguna, Duarte, & Martí, 2009) and Estimation of Distribution Algorithms (Larrañaga & Lozano, 2002). Because our design is fully based on a black-box structure, we don't compare performance against these procedures and limit our computational experiments to tests with commercial solvers. In addition, comparisons are made with specialized procedures for each class of problems in the test set. It is not expected for black-box methods to perform better than the state-of-the-art in any particular problem instance; however, the comparison is helpful in assessing the size of the gap between a general and a specialized procedure in each problem class.

This work may be viewed as the fourth in a series of general-purpose solvers created for problems with specific types of variables: continuous variables (Laguna & Martí, 2005), permutation variables (Campos, Laguna, & Martí, 2005) and binary variables (Gortázar, Duarte, Laguna, & Martí, 2010). Specifically, our interest is to develop a solver for an evaluation black box that takes as input a vector of integers. The process is such that a solver sends a vector of integers to the black box that calculates an objective function value and the left-hand-side of constraint values if any. These values are returned to the solver and are used to make decisions with respect to the next trial solution that is generated and turned in to the black box for evaluation. It is assumed that there is a computational budget that limits the number of times that the black box may be called. This is the criterion by which execution is stopped and computational effort is measured across competing solution procedures.

Within the continuum of solution procedures, our proposal is closer to the general-purpose commercial solvers than to the methods specially built for a particular class of problems. The specialization over the commercial solvers consists of focusing on a specific solution representation (i.e., a vector of integers). The results of our computational tests, discussed later, are consistent with where the procedure falls within the specialization scale.

2. Modeling Integer Problems

Let an optimization problem be such that all the decision variables x_j for $j = 1, \ldots, n$ must take on integer values. We refer to this class as integer problems. We assume that all variables are bounded in the interval $[l, u]$ and that the objective function to be maximized is denoted by $f(x)$. The problem may be written as:

$$
\begin{array}{lll}
\text{Maximize} & f(x) & \\
\text{Subject to} & g_i(x) \leq 0 & i = 1, \ldots, m \\
& l \leq x \leq u & \\
& x \in \mathbb{Z} &
\end{array}
$$

The solver enforces the box constraints on the variables but has knowledge neither of the functional form of $f(x)$ nor of any of the $g_i(x)$ functions. For each solution x, the black box calculates $f(x)$ and all $g(x)$ values and returns them to the solver. In order to establish search directions, the solver builds a penalized function of the form $f'(x) = f(x) - \pi(x)$, where $\pi(x)$ is the penalty term. (Yeniay, 2005) describes the merits of a fairly extensive family of penalty functions of the following form:

$$\pi(x) = \sum_i r_i \times \max\{0, g_i(x)\}^{\beta}$$

Clearly, if x is feasible then $f'(x) = f(x)$. The r_i constants are the factors that penalize infeasible solutions. In problems where there is knowledge of the relative importance of meeting the constraints, it would make sense to use a different penalty factor for each constraint. However, in the current context, we assume that there is no such knowledge and therefore a single penalty factor r is used for all constraints. Typically, β is given a value of 1 or 2. To ensure that the $f'(x)$

value of any infeasible solution is worse than the $f(x)$ value of any feaible solution, then $\pi(x)$ must be strictly greater than $f(x)$ for all infeasible solutions. In our procedure we set $r = 10^9$ and $\beta = 1$.

The large penalty factor has the effect of magnifying the infeasibility of the solutions explored during the search. Therefore, the identification of feasible solutions has a high priority and the search attempts to stay within the feasible region.

3. Scatter Search

The metaheuristic known as scatter search belongs to the evolutionary programming family. The procedure starts with the construction of a population of solutions from which a reference set (*RefSet*) is selected and evolved by means of combination and improvement mechanisms. The methodology consists of five methods:

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

As described in (Laguna & Martí, 2003), there are standard (problem independent) forms of implementing the reference set update and the subset generation methods. In our work, we adopted two of those standard forms, namely, the reference set is updated with the standard *static* method. This method consists of constructing the *RefSet* by selecting b solutions from a population P created by the diversification generator. Half of these solutions (i.e., $b/2$) are the best according to their objective function values and the other half are selected for diversity purposes. Diversity is measured by calculating the Hamming distance between solutions. After the initial construction, the *RefSet* is updated by selecting the solutions with the highest quality in the set of solutions consisting of the union of the current *RefSet* and the set of solutions that have been subjected to the improvement method after being generated by the solution combination method.

The subset generation method is also standard and consists of all pairs of reference solutions for which at least one of the two solutions is “new”. That is, pairs that have already been examined in previous iterations are not considered. Below, we describe the implementation details of the diversification generation, solution combination and the improvement methods, all of which are specific to the current context.

3.1 Diversification Generation Method

We generate diverse solutions by construction. The main characteristic of constructive methods is that solutions are typically built sequentially by adding one element at a time or setting the value of a variable one at a time. The process is guided by a function that indicates the attractiveness of adding an unselected element to the solution or of setting a variable to a particular value. In a black-box structure, we assume that it is not possible to evaluate partial solutions. In other words, we assume that the back box is designed to evaluate only full solutions. Therefore, we face a situation in which constructing a solution by choosing the values of the variables sequentially is either extremely convoluted or impossible because there may not be a simple mechanism to measure the

merit of assigning a specific value to a variable. In order to obtain a set of full solutions that is a representative sample of the solution space, we relied on a tool from the area of experimental design.

Within this field, a factorial design of experiments is a technique applied to a set of n factors, each of which can take on one of k levels. A full factorial design considers all combinations of factors and levels, resulting in k^n experiments. When applying this technique to our context, where factors are variables and levels are possible values of the variables (i.e., $k = u - l + 1$). A full factorial design corresponds to a complete enumeration of all possible (both feasible and infeasible) solutions. Clearly, this approach is only practical for small values of k^n.

In practice, the full factorial space is sampled instead of explored exhaustively. Latin Hypercube Sampling (LHS) —introduced in (McKay, Beckman, & Conover, 1979) and refined in (Iman & Conover, 1982)— is an effective technique that is also computationally efficient. Figure 2 shows a LHS design for 2 variables, x_1 and x_2, that can take on the integers from 1 to 4 (i.e., $n = 2$ and $k = 4$). The square grid in Figure 2 shows both the 16 possible solutions and the sample positions (indicated with X). The design is a *Latin square* because there is only one sample in each row and each column of the square. In multiple dimensions, the design is known as a *Latin hypercube*.

x_1 \ x_2	1	2	3	4
1			X	
2				X
3		X		
4	X			

Figure 2. Latin square for a problem with $n = 2$ and $k = 4$

Each LHS design results in $k = u - l + 1$ solutions and therefore a common practice consists of building several designs to generate a desired number of solutions. We start by creating n vectors of size k with elements $(l, l + 1, \dots, u)$ placed in the specified order. We then create a random permutation of $n - 1$ vectors, leaving one of the vectors in the order that it was created. A Latin hypercube sample is obtained by matching each of the elements of the individual vectors to create k samples. For instance, the Latin square of Figure 2 is the result of a process by which the values of the vector (1, 2, 3, 4), which correspond to x_1, are matched with the values of a random permutation of this vector, which correspond to x_2. The random permutation is (3, 4, 2, 1), resulting in the pairs (1, 3), (2, 4), (3, 2) and (4, 1). These pairs are the samples and the cells marked with X in the Latin square of Figure 2.

Procedure 1 shows how to employ this technique to generate a set with $DSize$ solutions. The procedure starts by initializing the repository of solutions D as an empty set and by setting the value of k. Then, the permutation p_1 is set to the ordered vector $(l, l + 1, \dots, u)$, where $p_1(i)$ is the value in position i. That is, $p_1(1) = l$ and $p_1(k) = u$. The main loop of the procedure is executed until the size of D reaches $DSize$. The loop starts by creating $n - 1$ permutations of p_1 (line 4). These permutations are labeled p_j, for $j = 2, \dots n$. The value in position i of the j^{th} permutation is $p_j(i)$. Samples are constructed (line 7) and added to set D (line 9), one by one. If D reaches the desired

size (line 10), the outer for-loop breaks and the procedure stops. We point out that, similar procedures based on design of experiments have been used in the literature to generate populations of diverse solutions. See for instance the implementations in (Laguna & Martí, 2005), (Duarte, Glover, Martí, & Gortázar, 2011) and (Duarte, Martí, & Gortázar, 2011).

```
LHS()
1.  Make D = ∅ and k = u − l + 1
2.  Make p_1 = (l, l + 1, …, u), where p_1(i) is the i^th value in the array
3.  WHILE |D| < DSize DO
4.      Construct n − 1 random permutations of p_1, where p_j is the j^th
        permutation (for j = 2, …, n)
5.      FOR i = 1 TO k DO
6.          FOR j = 1 TO n DO
7.              Set x_j = p_j(i)
8.          END FOR
9.          D = D ∪ x
10.         IF |D| = DSize BREAK
11.     END FOR
12. END WHILE
END
```

Procedure 1. Latin Hypercube Sampling

The solutions in D are evaluated by calling the black box. That is, the objective function value and feasibility of the solutions are known for each solution $x \in D$. Hence, D includes both feasible and infeasible solutions. These data are the starting point for acquiring knowledge about the characteristics that are embedded in high quality solutions. In the spirit of adaptive memory programming (Taillard, Gambardella, Gendreau, & Potvin, 2001), which embodies a broad framework that focuses on exploiting a collection of strategic memory components with the purpose of balancing search intensification and diversification (Glover, 1997), we employ a memory structure ($AvgOFV$) that stores the average value of the objective function corresponding to solutions for which the variables hold certain values. Specifically, $AvgOFV$ is a two dimensional array of size $k \times n$, where columns represent variables and rows represent variable values. The $AvgOFV(i,j)$ element of the matrix contains the average objective function value of solutions for which $x_j = i$. The objective function value information is stored every time a solution is evaluated with the black box and therefore it includes the objective function values of both feasible and infeasible solutions. The $AvgOFV$ matrix is constructed with the solutions in D.

The information contained in the $AvgOFV$ memory structure is used to construct empirical distribution functions that give the probability that a variable will be assigned a particular value. The empirical probability distribution function (PDF) for variable x_j is given by:

$$Prob(x_j = i) = \frac{AvgOFV(i,j)}{\sum_{i=1}^{k} AvgOFV(i,j)}$$

Instead of using the PDF directly, it is more convenient to work with its corresponding cumulative distribution function (CDF). As customary, the CDF is simply given by:

$$Prob(x_j \leq i) = \sum_{i'=1}^{i} Prob(x_j = i')$$

for $i = 1, \ldots, k$ and $Prob(x_j \leq k) = 1$.

To exploit the information contained in the CDFs, we configured a method to construct solutions based on quality information that is shown in Procedure 2 (and labeled QC). The method takes as input the CDFs and $QSize$, and returns a set Q of $QSize$ solutions. Procedure 2 simply constructs solutions for which the probability that a variable takes on a specific value is given by its corresponding CDF.

```
QC ()
1.  Q = ∅
2.  WHILE |Q| < QSize DO
3.      FOR j = 1 TO n DO
4.          r = rand(0,1)
5.          i = 1
6.          WHILE r < Prob(x_j = i) DO
7.              i = i + 1
8.          END WHILE
9.          x_j = i
10.     END FOR
11.     Q = Q ∪ x
12. END WHILE
END
```

Procedure 2. QC Constructive Method

In sum, the construction of solutions is done in two steps. In the first step, LHS (Procedure 1) is executed and $DSize$ solutions are obtained with Latin Hypercube Sampling. These solutions are evaluated with the black box and the quality of their elements (i.e., the assignments of values to the decision variables) is captured by the $AvgOFV$ memory structure. This structure is then used to create probability distribution functions, one for each variable. Then, $QSize$ additional solutions are constructed with the QC method (Procedure 2), which employs the information embedded in the CDFs that are associated with the PDFs constructed from the $AvgOFV$ memory structure. These new solutions are also evaluated with calls to the black box. Therefore, at the end of this process, the black box has been called a total of $DSize + QSize$ times.

3.2 Improvement Method

Improvement methods within the scatter search framework are typically based on neighborhood searches, and commonly limited to local searches. That is, the improvement method stops as soon as no neighbor is capable of improving upon the current solution. The improvement method employs two types of neighborhoods and is organized in a similar way as variable neighborhood descent (Hansen & Mladenovic, 1999). Neighborhoods are built from a solution representation and a move mechanism. In our context, we represent a solution x as a vector of n variables, i.e., $x = (x_1, x_2, \ldots, x_n)$ such that $l \leq x_j \leq u$ for all j. With this representation, we define the following moves:

- **Exchange of values.** Given a solution x, the exchange of the values of the variables x_i and x_j is denoted by $swap(i,j)$ and generates a new solution x' such that $x'_k = x_k \forall k \neq i,j$ and $x'_i = x_j$, $x'_j = x_i$.
- **Replacement of a single value.** Given a solution x, the replacement move denoted by $replace(j,v)$ creates a solution x' for which $x'_k = x_k \forall k \neq j$ and $x'_j = v$. The move is such that $x_j \neq v$.

We refer to the neighborhood generated by swap moves as the *swap neighborhood* and the neighborhood generated by the replacement moves as the *replace neighborhood*. The size of these neighborhoods is $n(n-1)/2$ and $n(u-l)$, respectively. A full exploration of both neighborhoods results in a computationally expensive local search that might not yield the desired results. Instead of attempting to find the best move (i.e., the swap or replace that results in the largest improvement of the objective function value), we adopt the first-improvement strategy. This strategy consists of traversing the neighborhoods and executing improving moves as soon as they are identified, as shown in Procedure 3.

```
FLS(x)
1.  Improve = true
2.  WHILE Improve DO
3.      Improve = false
4.      Make J = {1,2,…,n}
5.      WHILE J ≠ ∅ and Not Improve DO
6.          Randomly select j ∈ J and make J = J\j
7.          Make V = {l, l+1, …, u}\x_j
8.          WHILE V ≠ ∅ and Not Improve DO
9.              Randomly select v ∈ V and make V = V\v
10.             x' = replace(x_j, v)
11.             IF f(x') > f(x) THEN
12.                 x = x'
13.                 Improve = true
14.             END IF
15.         END WHILE
16.         IF Not Improve THEN
17.             Make I = {1, …, n}
18.             WHILE I ≠ ∅ and Not Improve DO
19.                 Randomly select i ∈ I and make I = I\i
20.                 x' = swap(x_i, x_j)
21.                 IF f(x') > f(x) THEN
22.                     x = x'
23.                     Improve = true
24.                 END IF
25.             END WHILE
26.         END IF
27.     END WHILE
28. END WHILE
END
```

Procedure 3. FLS Improvement Method

The first-improvement local search (FLS) in Procedure 3 starts from an initial solution x. The main loop (lines 2 to 28) is executed as long as the current solution keeps improving. All variables in the model are made available for the neighborhood search by creating the set J of variable indices (line 4). A variable index j is randomly selected and eliminated from J (line 6), and the set V of all

possible new values for variable j is constructed (line 7). Note that V does not include the current value of the chosen variable (i.e., it does not include de value of x_j). Lines 7 to 15 include the search for a value v whose replacement of the current value of x_j results in an improvement of the objective function. If such a value is found, the entire process is repeated from line 3. If not, then the swap neighborhood is explored, starting in line 17. The exploration consists of random selections of an index i (line 19) that is different from j in order to excute the move $swap(i, j)$, as shown in line 20. As soon as an improved moved is found (lines 21 to 23), the process restarts in line 3. The local search ends when all variables are explored and there is no replacement of swap capable of improving the current solution. This occurs when the Boolean variable `Improve` has the value of `false` and all variables in J have been explored (see line 5 in Procedure 3).

3.3 Combination Methods

The combination mechanisms employed in our implementation operate on two solutions, x' and x'', to generate a new solution x. We have created six of these procedures and their merit is assessed in the computational testing section:

1. **CM-A**. This procedure calculates the (rounded) midpoint between the two reference solutions. Therefore, each variable value in the new solution is $x_j = int\left(\frac{x'_j + x''_j}{2}\right)$.
2. **CM-AE**. This procedure generates three trial solutions and returns the best. The first trial solution is the same one as the one generated by CM-A. The second and third trial solutions are given by $x = int\left(\frac{l+\min(x',x'')}{2}\right)$ and $x = int\left(\frac{u+\max(x',x'')}{2}\right)$, respectively.
3. **CM-EP**. This is similar to CM-AE, with the first trial solution being the same as the one generated by CM-A. The second and third trial solutions, however, are generated by finding two points in the hyperplane $x = x' + t(x' - x'')$. One point is found with $t > 1$, such that it is between x'' and the point where the hyperplane reaches the (l, u) boundary. The other one is the midpoint on the segment of the hyperplane that is between x' and the (l, u) boundary, which is reached by making $t < 1$.
4. **CM-GPR**. This procedure is based on the notion of path relinking (Glover & Laguna, 1997) where a sequence of trial solutions are visited while transforming a so-called *initiating* solution into a so-called *guiding* solution. Assume that x' is the initiating solution and that x'' is the guiding solution. Then, at each step, the assignment $x'_j \leftarrow x''_j$ that improves $f(x')$ the most is made. The process stops when $x' = x''$. The best trial solution is returned.
5. **CM-R**. This procedure builds a new solution with a random selection, for each variable, of the values in the reference solutions. That is $x_j = x'_j$ or x''_j with $P(x_j = x'_j) = P(x_j = x''_j) = 0.5$ for all j. In the context of genetic algorithms, this combination method is known as the *fixed crossover* operator (Dolezal, Hofmeister, & Lefmann, 1999).
6. **CM-PR**. This is similar to CM-R, however, the selection probabilities are proportional to the quality of the reference solutions. Assuming that $f(x') > 0$ and $f(x'') > 0$, then $P(x_j = x'_j) = \frac{f(x')}{f(x')+f(x'')}$ and $P(x_j = x''_j) = 1 - P(x_j = x'_j)$.

Of all the solutions produced by the application of a combination method, only the top $b/2$ are chosen to be subjected to the improvement method.

4. Test Problems

We employed the following four sets of test problems:

- **Bandwidth Coloring Problem** (Martí, Gortázar, & Duarte, 2010). This is a special type of graph coloring problem for which colors have values. A valid coloring of the graph is one for which the difference of the values of the colors of two adjacent nodes (i.e., nodes that are connected) must exceed the value of the connecting edge. As in the classical coloring problem, the objective is to minimize the number of colors.
- **Capacitated Task Allocation Problem** (Lusa & Potts, 2008). This problem consists of assigning a set of tasks to a set of processors in order to minimize the total cost of the assignment. The total cost may include a fixed cost for using a processor and a communication cost associated with the assignment of related tasks to different processors.
- **Maximally Diverse Grouping Problem** (Gallego, Laguna, Martí, & Duarte, 2011). This is the problem of partitioning a set of elements into a given number of groups in order to maximize total diversity, measured as the sum of the individual group diversities.
- **Water Distribution Network Problem** (Vasan & Simonovic, 2010). The objective of this problem is to select the size and location of pipes in a water distribution network in order to meet water pressure requirements at a minimum cost.

These problems present different challenges according to their structure. However, they all have in common that their solution can be represented as a vector of integer values.

4.1 Bandwidth Coloring Problem

Let $G = (V, E)$ be a graph where V ($|V| = n$) is the set of vertices and E ($|E| = m$) is the set of edges. Associated with each edge between nodes i and j, there is a positive quantity d_{ij}. A k-coloring of G assigns the color $c(i) \in \{1, 2, \dots, k\}$ to vertex $i \in V$ in such a way that $|c(i) - c(j)| \geq d_{ij}$ for all $(i, j) \in E$. The objective of the bandwidth coloring problem (BCP) is to find a k-coloring such that k is minimized. Mathematically, the problem may be expressed as follows:

Minimize	k	
Subject to	$\lvert c(i) - c(j) \rvert \geq d_{ij}$	for all $(i, j) \in E$
	$c(i) \in \{1, 2, \dots, k\}$	for all $i \in V$

The vertex coloring problem is a special case of the BCP for which $d_{ij} = 1$ for all $(i, j) \in E$. Both problems are NP-Hard and the BCP has interesting practical applications in telecommunications, e.g., in the assignment of frequencies in wireless networks (Martí, Gortázar, & Duarte, 2010). The best solution method for the BCP (known as FCNS) belongs to (Prestwich, 2002).

For the computational experiments, we have employed the well-known GEOM instances. This set of 33 instances was created by Michel Trick and is available at http://mat.gsia.cmu.edu/COLOR02/. The set contains graphs of three different density levels and with number of vertices ranging from 20 to 120. In our solution representation, x_j ($1 \leq x_j \leq k$) is the index of the color assigned to vertex j.

4.2 Capacitated Task Allocation Problem

The capacitated task allocation problem (CTAP) consists of assigning a set of n tasks to a set of m processors. Each task i requires an amount a_i of resources and each processor k has a resource capacity of b_k. Each pair of tasks (i, j) has a communication cost $c_{ij} > 0$ that is incurred when the tasks are assigned to different processors. The cost of assigning task i to processor k is given by d_{ik}, and the cost of using processor k is given by s_k. The problem may be formulated as a quadratic binary program with one set of variables x_{ik} indicating whether task i is assigned to processor k and another set of variables y_k indicating whether processor k is being used. The mathematical model has the following form:

$$\text{Minimize} \quad \sum_{i=1}^{n-1}\sum_{j=1}^{n} c_{ij}\left(1 - \sum_{k=1}^{m} x_{ik}x_{jk}\right) + \sum_{i=1}^{n}\sum_{k=1}^{m} d_{ik}x_{ik} + \sum_{k=1}^{m} s_k y_k$$

$$\text{Subject to} \quad \sum_{k=1}^{m} x_{ik} = 1 \qquad i = 1, \dots, n$$

$$\sum_{i=1}^{n} a_i x_{ik} \leq b_k y_k \qquad k = 1, \dots, m$$

$$x_{ik} \in \{0,1\} \qquad \forall i, k$$

$$y_k \in \{0,1\} \qquad \forall k$$

The CTAP has practical applications in distributed computing environments and in the automobile industry, where micro-processing components must be placed and linked (Lusa & Potts, 2008). The best CTAP solution method in the literature is based on a variable neighborhood search (VNS) developed in (Lusa & Potts, 2008).

We use the 76 CTAP instances compiled in (Lusa & Potts, 2008), of which 8 belong to problems found in practice. The 68 instances that were artificially generated include three levels of difficulty, as related to the tightness of the capacity constraint. Also, some instances do not include the processor cost s_k and some do not include the assignment cost d_{ik}. The size of the instances ranges from 15 to 100 tasks and between 4 and 30 processors. In our solution representation, x_j ($1 \leq x_j \leq m$) is the index of the processor to which task j is assigned.

4.3 Maximally Diverse Grouping Problem

The maximally diverse grouping problem (MDGP) has the objective of dividing a set of M elements into G groups in order to maximize the diversity of all groups. There is a diversity measure between each pair of elements (i, j) that is given by d_{ij}. The diversity of a group is the sum of the diversity values corresponding to each pair of elements assigned to the group. The size of a group g must be between a_g and b_g. A special case of the problem arises when $a_g = b_g$ for all groups. The problem may be formulated as quadratic binary problem with a set of variables x_{ig} to indicate that element i is assigned to group g:

$$\text{Maximize} \quad \sum_{g=1}^{G}\sum_{i=1}^{M-1}\sum_{j=i+1}^{M} d_{ij}x_{ig}\,x_{jg}$$

$$\text{Subject to} \quad \sum_{g=1}^{G} x_{ig} = 1 \qquad i = 1, \dots, M$$

$$a_g \leq \sum_{i=1}^{M} x_{ig} \leq b_g \qquad g = 1, \ldots, G$$

$$x_{ig} \in \{0,1\} \qquad \forall i, g$$

The MDGP has practical applications in circuit design (Chen, 1986) (Feo & Khellaf, 1990), storage of large codes in paged memory (Kral, 1965) and the assignment of students to groups (Weitz & Jelassi, 1992). The best solution method in the literature is a tabu search with strategic oscillation (Gallego, Laguna, Martí, & Duarte, 2011).

A set of 60 Geo instances from the MDGPLIB (available in http://www.optsicom.es/mdgp) were employed for experimentation. The first 10 instances of each group with M = 30, 60, 120, 240, 480 and 960 were selected to compile the set of 60. The instances are of the DSG (different group size) type, i.e., $a_g < b_g$. In our solution representation, x_j ($1 \leq x_j \leq G$) is the index of the group to which element j is assigned.

4.4 Water Distribution Network Problem

A water distribution network problem (WDNP) that has been studied by the civil engineering community consists of expanding the capacity of a network of pipes to meet the needs of a water distribution system. In particular, the problem consists of determining the size of the pipe and the segment in an existing network where the new pipe will be added. The pipe network may be represented as a graph $G = (V, E)$ where E is the set of pipe segments and V is the set of nodes that connect the pipes. The objective is to add pipes of specified diameters to the segments in E in such a way that the total cost of the expansion is minimized and the water pressure at the nodes is satisfied. Let D_i be the diameter of the pipe added to segment $i \in E$ and let L_i be its length, then the problem may be stated as:

$$\text{Minimize} \qquad \sum_i 1.1 D_i^p L_i$$

$$\text{Subject to} \qquad u_k \geq J_k \qquad k \in V$$

where u_k is the pressure in node k associated with the pipes placed in the current solution and J_k is the required pressure at the node. The value of p depends on the problem and, in the literature, a value of $p = 1.24$ has been used for the New York City problem (Shaake & Lai, 1969), while a value of $p = 1.5$ has been used for the Hanoi network (Fujiwara & Khang, 1987). These are the two problems that we use for experimentation, with the values of p that are suggested in the literature. The pressure values are calculated with a hydraulic simulator known as EPANET 2.0 (Rossman, 2000). Therefore, the feasibility of a solution (i.e., a selection of pipe segments and their size) is not known until the hydraulic calculations are performed by the simulator. The most recent results associated with this problem are due to (Vasan & Simonovic, 2010). In our solution representation, x_j is the index of the diameter assigned to pipe-segment j.

5. Experimental Testing

The results reported in this section were obtained with an Intel i7 @ 2.7 GHz and 4GB of RAM computer running Windows XP. The code was implemented in Java SE6. The metrics that we use to measure algorithmic performance are:

- **Feas**: Fraction of a set of problems for which a method is capable of finding at least one feasible solution.
- **Score**: Let the experiment consist of comparing m procedures using n instances. Also, for a particular procedure, let p be the number of times that the other $m-1$ procedures produce a better result. The Score of a procedure is the fraction of times in which the other procedures "win" (i.e., they produce better solutions than the procedure being scored) and is calculated as $(n \times (m-1) - p)/(n \times (m-1))$. Therefore, the best score is 1 (meaning that no other procedure produced better solutions) and the worst is 0 (meaning that all other procedures produced better solutions). In this calculation, we do not distinguish among infeasible solutions, they are all assigned a large negative value (for a maximization problem) or a large positive value (for a minimization problem) to indicate that they are the worst.
- **Dev**: Average percent deviation from the best-known solution. Only feasible solutions are included in this calculation.
- **Best**: Fraction of instances in a set for which a procedure is able to find the best-known solution.

The collection of 171 instances (33 BCP, 76 CTAP, 60 MDGP and 2 WDNP) was divided into a training set of 51 instances (11 BCP, 26 CTAP, 14 MDGP and 0 WDNP) and a test set of 120 instances (22 BCP, 50 CTAP, 46 MDGP and 2 WDNP).

In our first experimentation, we are interested in determining the best values for $DSize$ and $QSize$. That is, we want to determine the right mix of LHS and QS solutions to be used as a departure point. We used the training set, ran several combinations, and quickly determined that, given a desired total number of solutions, the best results are obtained when half of the solutions are generated with the LHS method and half with the QC method.

As described in Section 3.2, the improvement method that we developed is based on the assumption that a full-neighborhood search is computationally too expensive to be effective. This is why the FLS improvement method (see Procedure 3) explores neighborhoods by making transitions from the current solution to the next as soon as an improvement opportunity is identified. To test our assumption and to assess the contribution of the improvement method, we perform an experiment that compares the performance of three forms of generating solutions: 1) the diversification generator (LHS.QC), 2) the diversification generator with the FLS improvement method (LHS.QC.FLS), and 3) the diversification generator with a best improvement strategy (LHS.QC.BLS). The best improvement strategy consists of a full neighborhood search to determine the next solution to visit. The three alternatives are given a time limit of 1 minute and performance is measured with *Feas* and *Score*.

The results in Table 1 show that the strategy of generating and improving solutions is better than the strategy of spending the entire computational budget (of 1 minute in this case) only generating solutions (as done by LHS.QC). The total *Score* of 0 associated with LHS.QC indicates that in all cases, the addition of either local search produced better results. The results also confirm that first-improvement is the better improvement strategy. Only in one instance (out of 51), the best-improvement strategy was a better choice, and hence the *Score* of 0.990 for LHS.QC.FLS.

Problem	Metric	LHS.QC	LHS.QC.BLS	LHS.QC.FLS
BCP	Feas	1	1	1
	Score	0	0.545	0.955
CTAP	Feas	0.885	1	1
	Score	0	0.615	1
MDGP	Feas	1	1	1
	Score	0	0.643	1
Total	Feas	0.941	1	1
	Score	0	0.608	0.990

Table 1. Diversification generation and improvement methods

To build the final configuration of our procedure, we test the efficacy of the combination methods described in Section 3.3. We once again employ the training set of instances and compare the performance of the six combination methods when embedded in a scatter search framework along with LHS.QC.FLS. All variants were executed a total of 1 minute per instance.

In Table 2 we compare the performance of the combination methods with the Score values only. Note that LHS.QC.FLS alone is already capable of finding at least one feasible solution in all instances (see Table 1). Therefore, *Feas* does not provide any discrimination power. The score values indicate that CM-EP does not perform well in any of the problems. The best methods are CM-AE for BCP, CM-A for CTAP and CM-RP for MDGP. Note that although CM-PR has a score of 0.914 on the MDGP instances, the method performs poorly on the other two types of problems. Overall, CM-A exhibits the most robust behavior with scores ranging from 0.709 to 0.846 and a total score of 0.780. This is the combination method that we merge with LHS.QC.FLS to build our scatter search (SS) for integer problems.

Problem	CM-A	CM-AE	CM-EP	CM-GPR	CM-R	CM-RP
BCP	0.709	0.818	0.455	0.745	0.509	0.473
CTAP	0.846	0.715	0.323	0.515	0.808	0.569
MDGP	0.714	0.643	0.371	0.657	0.814	0.914
Total	**0.780**	**0.718**	**0.365**	**0.604**	**0.745**	**0.643**

Table 2. *Score* values for 6 combination methods

We now compare the performance of the scatter search procedure with commercial black-box optimizers. For this experiment, we use all but 2 of the instances (the WDNP instances) in the test set for a total of 118. The comparison is made against the following three commercial software packages and the stopping criterion is set to 1 minute per problem instance:

- OptQuest by OptTek Systems (http://www.opttek.com/)
- Premium Solver by Frontline Systems (http://www.solver.com/)
- Evolver by Palisade Corporation (http://www.palisade.com/)

Figures 3, 4 and 5 show the evolution throughout the search of the average values of *Dev* and *Feas* for the 118 instances in the test set. Table 3 shows a summary of these measures for the entire set as well as the value of *Best* associated with each method.

The graphs of *Dev* show that SS has more search diversification power than the competing approaches. These approaches seem to stagnate, particularly in the CTAP and MDGP instances. The graphs also show that all the approaches have difficulties in producing high quality solutions for the BCP instances.

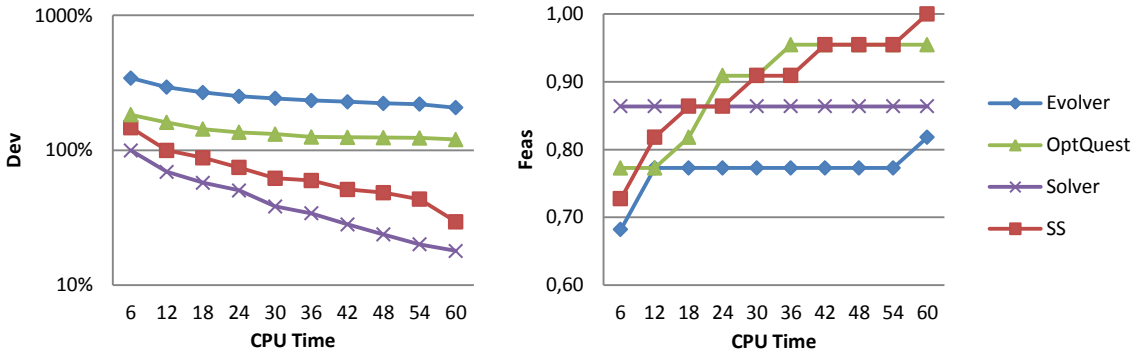

Figure3. Profile of *Dev* and *Feas* on BCP instances

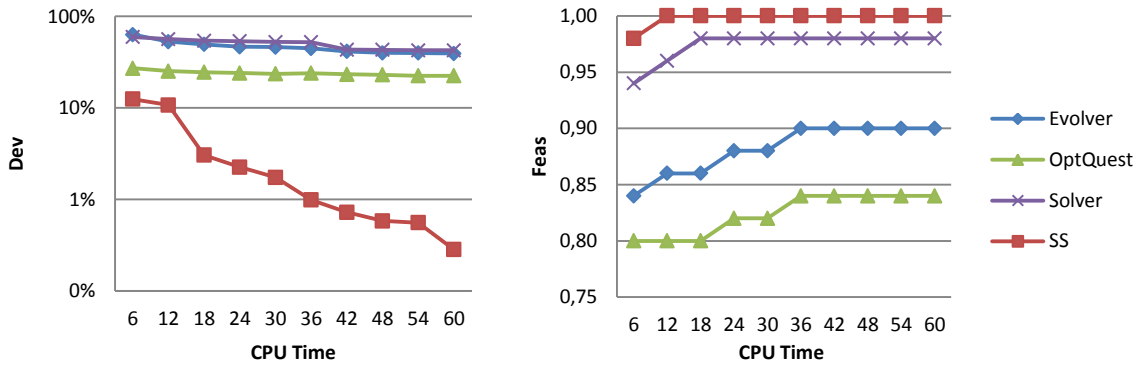

Figure4. Profile of *Dev* and *Feas* on CTAP instances

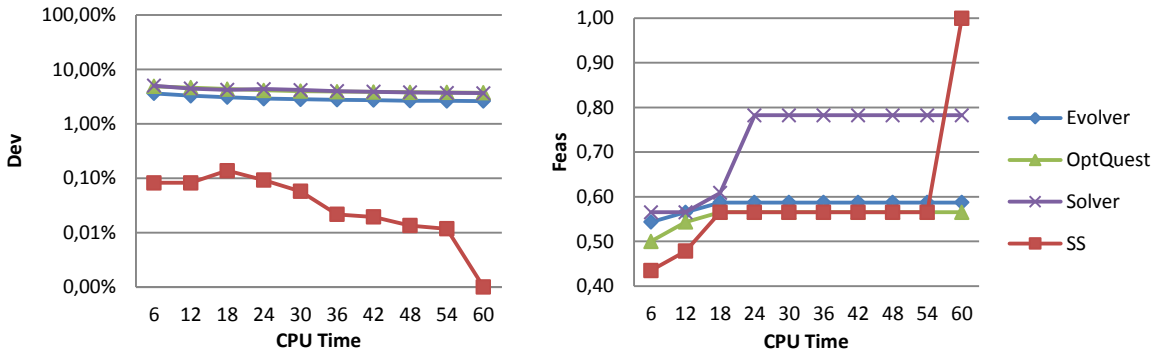

Figure5. Profile of *Dev* and *Feas* on MDGP instances

The summary results in Table 3 are all in favor of SS. The procedure is the only one capable of finding at least one feasible solution to all instances in the test set. It produces the smallest value of *Dev* and the largest values of *Best* and *Score*. The one case in which SS does not produce the best outcome is for the value of *Dev* in the BCP instances.

Problem	Metric	Evolver	OptQuest	Solver	SS
BCP	Feas	0.818	0.955	0.864	1
	Dev	7.168	6.822	2.330	3.968
	Best	0	0	0	0
	Score	0.167	0.379	0.788	0.758
CTAP	Feas	0.900	0.840	0.980	1
	Dev	0.513	0.328	0.530	0.081
	Best	0.020	0.020	0	0.320
	Score	0.347	0.472	0.293	0.987
MDGP	Feas	0.587	0.565	0.783	1
	Dev	0.027	0.038	0.039	0.003
	Best	0	0	0	0.109
	Score	0.522	0.238	0.478	1
Total	Feas	0.763	0.754	0.881	1
	Dev	1.698	1.776	0.689	0.775
	Best	0.008	0.008	0.000	0.178
	Score	0.381	0.362	0.458	0.949

Table 3. Comparison of SS and commercial software

The next experiment compares the performance of SS against the state-of-the-art for the three problem classes in the test set. In particular, the following procedures are used for comparison:

- FCNS for the BCP (Prestwich, 2002)
- VNS for the CTAP (Lusa & Potts, 2008)
- SO for the MDGP (Gallego, Laguna, Martí, & Duarte, 2011)

Once again, the test consists of 1-minute runs for each problem instance. The results are summarized in Table 4.

Problem	Metric	State-of-the-art	SS
BCP	Dev	0.025	3.848
	Best	0.318	0
	Score	1	0
CTAP	Dev	0.068	0.081
	Best	0.020	0.320
	Score	0.300	0.720
MDGP	Dev	0.002	0.003
	Best	0.174	0.109
	Score	0.848	0.261
Total	Dev	0.034	0.775
	Best	0.136	0.178
	Score	0.644	0.407

Table 4. Comparison of SS and the state-of-the-art

As expected, the state-of-the-art methods produce better results than the SS that has been designed under the black-box paradigm. However, the SS is able to find solutions of reasonably high quality for CTAP and MDGP instances. The experiment confirms that the structure and the objective-function landscape of the BCP are not amenable to the black-box paradigm.

Our final experiment deals with finding solutions to the 2 WNDP instances. Each instance is run for 5 minutes. The New York network consists of 21 segments and 7 possible pipe diameters. In the

Hanoi network there are 34 pipe segments and 16 possible diameters. Table 5 shows the cost (in millions) associated with the best solution found by each procedure, all of them treating the evaluation of the objective function as a black box. A penalized function is used to direct the search toward solutions that are feasible with respect to the required pressure values at the nodes.

Problem	Evolver	Solver	OptQuest	SS	Best-known
Hanoi	Infeasible	Infeasible	$6.42	$6.26	$6.19
New York	$38.64	$44.70	$38.64	$38.81	$38.64

Table 5. SS and commercial solvers applied to the 2 WNDP instances

Both, Evolver and Solver are not able to find a single feasible solution for the Hanoi problem. SS finds a solution that is slightly better than OptQuest's solution and that is 1.1% worse than the best known. On the other hand, SS is not able to match the best-known solution to the New York problem, while both Evolver and OptQuest do.

6. Conclusions

This work is part of a larger project that has produced black-box optimizers for problem represented by continuous variables, binary strings and permutation vectors. These representations cover a large spectrum of problems classes in the literature. Our goal in all of these projects has been to develop procedures that can be shown to be generally more effective than commercial black-box solvers and at least somewhat competitive to the individual state-of-the-art for each problem class. We recognize that our approach falls in between the specialized methods and the more general black-box approach used by commercial software. We believe, however, that the mechanisms that we have created can be (and in fact have been) embedded in commercial software packages in order to increase their effectiveness.

Acknowledgments

This research has been partially supported by the Government of Spain (Grant Refs. TIN2009-07516 and TIN2012-35632).

References

April, J., Better, M., Glover, F., Kelly, J. P., & Laguna, M. (2005, January). Enhancing Business Process Management with Simulation Optimization. *BPTrends*, 1-11.

Baños, R., Gil, C., Agulleiro, J. I., & Reca, J. (2007). A Memetic Algorithm for Water Distribution Network Design. In A. Saad, E. Avineri, K. Dahal, M. Sarfraz, & R. Roy, *Soft Computing in Industrial Applications: Recent and Emerging Methods and Techniques* (pp. 279–289). Berlin: Springer.

Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *INFORMS Journal on Computing, 6*(2), 154-160.

Campos, V., Laguna, M., & Martí, R. (2005). Context-Independent Scatter and Tabu Search for Permutation Problems. *INFORMS Journal on Computing, 17*(1), 111-122.

Chen, C.-C. (1986). *Placement and partitioning methods for integrated circuit layout.* Berkeley: University of California.

Dolezal, O., Hofmeister, T., & Lefmann, H. (1999). *A Comparison of Approximation Algorithms for the MaxCut-Problem.* Universität Dortmund.

Duarte, A., Glover, F., Martí, R., & Gortázar, F. (2011). Hybrid Scatter Tabu Search for Unconstrained Global Optimization. *Annals of Operations Research, 183*, 95-123.

Duarte, A., Martí, R., & Gortázar, F. (2011). Path Relinking for Large Scale Global Optimization. *Soft Computing, 15*, 2257-2273.

Feo, T. A., & Khellaf, M. (1990). A class of bounded approximation algorithms for graph partitioning. *Networks, 20*, 181-195.

Fujiwara, O., & Khang, D. B. (1987). A two-phase decomposition method for optimal design of looped water distribution networks. *Water Resources Research, 26*(4), 539–549.

Gallego, M., Laguna, M., Martí, R., & Duarte, A. (2011). Tabu search with Strategic Oscillation for the maximally diverse grouping problem. *Journal of the Operational Research Society*, 1-20.

Glover, F. (1997). Tabu Search and Adaptive Memory Programing – Advances, Applications and Challenges. In R. S. Barr, R. V. Helgason, & J. L. Kennington (Eds.), *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization and Stochastic Modeling Technologies* (pp. 1-75). Boston: Kluwer Academic Publishers.

Glover, F., & Laguna, M. (1997). *Tabu Search.* Boston: Kluwer Academic Publishers.

Gonçalves, J. F., & Resende, M. G. (2011). Biased Random-key Genetic Algorithms for Combinatorial Optimization. *Journal of Heuristics, 17*(5), 487-526.

Gortázar, F., Duarte, A., Laguna, M., & Martí, R. (2010). Black-box Scatter Search for General Classes of Binary Optimization Problems. *Computers and Operations Research, 37*(11), 1977-1986.

Hansen, P., & Mladenovic, N. (1999). An Introduction to Variable Neighborhood Search. In S. Voss, S. Martello, I. Ossman, & C. Roucairol (Eds.), *Meta-heuristics, Advances and Trends in Local Search Paradigms for Iotimization* (pp. 433-458). Boston: Kluwer Academic Publishers.

Iman, R. L., & Conover, W. J. (1982). A Distribution-free Approach to Inducing Rank Correlation Among Input Variables. *Communications in Statistics - Simulation and Computation, 11*, 311–334.

Kral, J. (1965). To the problem of segmentation of a program. *Information Processing Machines, 2*, 116-127.

Laguna, M., & Martí, R. (2003). *Scatter Search: Methodology and Implementations in C.* Boston: Kluwer Academic Publishers.

Laguna, M., & Martí, R. (2005). Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions. *Journal of Global Optimization, 33*, 235-255.

Laguna, M., Duarte, A., & Martí, R. (2009). Hybridizing the Cross Entropy Method: An application to the Max-Cut Problem. *Computers and Operations Research, 36*(2), 487-498.

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. *Journal of the Operational Research Society, 61*(1), 53-67.

Larrañaga, P., & Lozano, J. A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Boston: Kluwer Academic Publishers.

Lippai, I., Heaney, J. P., & Laguna, M. (1999). Robust Water System Design with Commercial Intelligent Search Optimizers. *Journal of Computing in Civil Engineering, 13*(3), 135-143.

Lusa, A., & Potts, C. N. (2008). A variable neighbourhood search algorithm for the constrained task allocation problem. *Journal of Operational Research Society, 59*(6), 812-822.

Martí, R., Gortázar, F., & Duarte, A. (2010). Heuristics for the bandwidth colouring problem. *International Journal of MetaHeuristics*, 11-29.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics, 21*, 239–245.

Prestwich, S. (2002). Constrained bandwidth multicoloration neighborhoods. *Computational symposium on graph coloring and its generalizations*, (pp. 126-133). Ithaca, NY.

Rossman, L. A. (2000). *EPANET: User's Manual.* Cincinnati: United States Environmental Protection Agency.

Schittekat, P., & Sörensen, K. (2009). Supporting 3PL Decisions in the Automotive Industry by Generating Diverse Solutions to a Large-Scale Location-Routing Problem. *Operations Research, 57*(5), 1058-1067.

Shaake, J. C., & Lai, D. (1969). *Linear programming and dynamic programming application to water distribution network design.* Cambridge: Massachusetts Institute of Technology.

Spears, W. M., & DeJong, K. A. (1991). On the Virtues of Parameterized Uniform Crossover. In R. K. Belew, & L. B. Booker (Ed.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 230-236). San Francisco: Morgan Kaufmann Publishers Inc.

Stein, M. (1987). Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics, 29*, 143–151.

Taillard, E. D., Gambardella, L. M., Gendreau, M., & Potvin, J.-Y. (2001). Adaptive Memory Programming: A Unified View of Metaheuristics. *European Journal of Operational Research, 135*, 1-16.

Vasan, A., & Simonovic, S. P. (2010). Optimization of Water Distribution Network Design Using Differential Evolution. *Journal of Water Resources Planning and Management, 136*(2), 279-287.

Weitz, R. R., & Jelassi, M. T. (1992). Assigning students to groups: a multi-criteria decision support system approach. *Decision Sciences, 23*(3), 746-757.

Yates, D. F., Templeman, A. B., & Boffey, T. B. (1984). The Computational Complexity of the Problem of Determining Least Capital Cost Designs for Water Supply Networks. *Engineering Optimization, 7*(2), 143-155.

Yeniay, Ö. (2005). Penalty Function Methods for Constrained Optimization with Genetic Algorithms. *Mathematical and Computational Applications, 10*(1), 45-56.