Scatter Search Vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems

Rafael Martí[a], Manuel Laguna[b] and Vicente Campos[a]

a Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de València, Dr. Moliner, 50, 46100 Burjassot (Valencia), Spain. Vicente.Campos@uv.es and Rafael.Marti@uv.es

Research partially supported by the Ministerio de Ciencia y Tecnología of Spain: TIC2000-1750-C06-01

b Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA Laguna@Colorado.edu

Abstract

The purpose of this work is to compare the performance of a scatter search (SS) implementation and an implementation of a genetic algorithm (GA) in the context of searching for optimal solutions to permutation problems. Scatter search and genetic algorithms are members of the evolutionary computation family. That is, they are both based on maintaining a population of solutions for the purpose of generating new trial solutions. We perform computational experiments with four well-known permutation problems to study and compare the performance of a SS and a GA implementation.

Keywords: Scatter search, genetic algorithms, combinatorial optimization, permutation problems.

Last revision: March 21, 2002

1. Introduction

Scatter search (SS) and genetic algorithms (GA) were both introduced in the seventies. While Holland (1975) introduced genetic algorithms and the notion of imitating nature and the "survival of the fittest" paradigm, Glover (1977) introduced scatter search as a heuristic for integer programming that expanded on the concept of surrogate constraints. Both methods fall in the category of evolutionary optimization procedures, from the point of view that they build, maintain and evolve a set (population) of solutions throughout the search. Although the population-based approach makes SS and GA part of the so-called evolutionary methods, there are fundamental differences between these two methodologies. One main difference is that genetic algorithms were initially proposed as a mechanism to perform hyperplane sampling rather than optimization. Over the years, however, GAs have morphed into a methodology whose primary concern is the solution of optimization problems (Glover 1994a).

In contrast, scatter search was conceived as an extension of a heuristic in the area of mathematical relaxation, which was designed for the solution of integer programming problems: surrogate constraint relaxation. The following three operations come from the area of mathematical relaxation and they are the core of most evolutionary optimization methods including SS and GAs:

1. Building, maintaining and working with a population of elements (coded as vectors)
2. Creating new elements by combining existing elements
3. Determining which elements are retained based on a measure of quality

Two of the best-known mathematical relaxation procedures are Lagrangean relaxation (Everett, 1963) and surrogate constraint relaxation (Glover 1965). While Lagrangean approaches absorb "difficult" constraints into the objective function by creating linear combinations of them, surrogate constraint relaxation generate new constraints to replace those considered problematic. The generation of surrogate constraints also involves the combination of existing constraints using a vector of weights. In both cases, these relaxation procedures search for the best combination in an iterative manner. In Lagrangean relaxation, for example, the goal is to find the "best" combination, which, for a minimization problem, is the one that results in the smallest underestimation of the true objective function value. Since there is no systematic way of finding such weights (or so-called Lagrangean multipliers) in order to produce the smallest (possibly zero) duality gap, Lagrangean heuristics iteratively change the weights according to the degree of violation of the constraints that have been "brought up" to the objective function.

Scatter search is more intimately related to surrogate relaxation procedures, because not only surrogate relaxation includes the three operations outlined above but also has the goal of generating information from the application of these operations. In the case of surrogate relaxation, the goal is to generate information that cannot be extracted from the "parent constraints". Scatter search takes on the same approach, by generating information through combination of two or more solutions.

Similarities and differences between SS and GAs have been previously discussed in the literature (Glover, 1994b and 1995). Hence, our goal is not to elaborate on those discussions and further analyze the fundamental differences of these two approaches. Instead, our main goal is to provide a direct performance comparison in the context of a class of combinatorial optimization problems. Specifically, we make our comparison employing four classes of problems whose solutions can be represented with a permutation. Our SS and GA implementations are based on a model that treats the objective function evaluation as a black box, making the search procedures context-independent. This means that neither implementation takes advantage of the structural details of the tests problems. We base our comparisons on experimental

testing with four well-known problems: linear ordering, traveling salesperson, matrix bandwidth reduction and a job-sequencing problem. The only information that both the SS solver and the GA solver have with respect to these problems is the nature of the objective function evaluation with regard to the "absolute" or "relative" positioning of the elements in the permutation. In other words, we differentiate between two classes of problems:

> *A-permutation problems*—for which absolute positioning of the elements is more important (e.g., linear ordering problem)
>
> *R-permutation problems*—for which relative positioning of the elements is more important (e.g., traveling salesperson problem)

We will see later that not all problems can be fully characterized as "absolute" or "relative", however, this does not render our implementations useless.

2. Scatter Search Implementation

Our scatter search implementation is summarized in Figure 1 and operates as follows. A generator of permutations, which focuses on diversification and not on the quality of the resulting solutions, is used at the beginning of the search to build a set P of $PopSize$ solutions (step 1). The generator, proposed by Glover (1998), uses a systematic approach to creating a diverse set of permutations. As is customary in scatter search, an improvement method is applied to the solutions in P in order to obtain a set of solutions of reasonable quality and diversity. The improvement method consists of the local search (LS) procedure described in the following section.

The reference set, $RefSet$, is a collection of b solutions that are used to generate new solutions by way of applying a solution combination method. The construction of the initial reference set in step 3 starts with the selection of the best $b/2$ solutions from P. These solutions are added to $RefSet$ and deleted from P. The minimum distance from each improved solution in P-$RefSet$ to the solutions in $RefSet$ is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to $RefSet$ and deleted from P and the minimum distances are updated. This process is repeated $b/2$ times. The resulting reference set has $b/2$ high-quality solutions and $b/2$ diverse solutions. The distance between two permutations $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ depends on the type of problem being solved. For A-permutation problems, the distance is given by:

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|.$$

The distance for R-permutation problems is defined as:

> $d(p,q)$ = number of times p_{i+1} does not immediately follow p_i in q, for $i = 1, \ldots, n\text{-}1$

The combination procedure is applied in step 5 to all pairs of solutions in the current $RefSet$. Since the reference set consists of b solutions, the number of trial solutions generated with the combination method is $b(b\text{-}1)/2$ when applied to the initial reference set. Note that only pairs with at least one new solution are combined in subsequent executions of this step and therefore the number of combinations varies after the initial reference set. The combined solutions are improved in the same way as mentioned above, that is, with the application of the LS procedure. The reference set is then updated by selecting the best b solutions from the union of $RefSet$ and the

improved trial solutions. Steps 5, 6 and 7 in the outline of Figure 1 are performed as long as at least one new trial solution is admitted in the reference set.

1. *Generate solutions* — Apply the diversification generation method to generate a set of *PopSize* solutions.
2. *Improve solutions* — Apply the LS method to improve solutions generated in Step 1.
3. *Build the reference set* — Choose the "best" b solutions to build the initial *RefSet*.
4. *Initialize* — Make *BestSol* the best solution in the current *RefSet* and *GlobalImprove* = 0

do {
  while (new solutions in *RefSet* and objective function evaluations < *MaxEval*) do {
    5. *Combine solutions* — Generate trial solutions from pairs of reference solutions where at least one solution in the pair is new.
    6. *Improve solutions* — Apply the local search method to improve the solutions generated in step 5.
    7. *Update reference set* — Choose the best b solutions from the union of the current *RefSet* and the set of improved trial solutions.
  }
  8. Update the best — Set *CurrentBest* as the best solution in the *RefSet*.
     if(*CurrentBest* improves *BestSol*)
       BestSol = *CurrentBest*
       GlobalImprove = 0
     else
       GlobalImprove = *GlobalImprove* + 1

  9. Rebuild RefSet — Remove the worst $b/2$ solutions from the *RefSet*. Generate *PopSize* improved solutions applying steps 1 and 2. Choose $b/2$ "diverse" solutions and add them to *RefSet*.
} while (objective function evaluations < *MaxEval*)

Figure 1. Scatter Search outline

When no new solutions qualify to be added to the *RefSet*, step 9 performs a partial rebuilding of the reference set. We keep the best $b/2$ solutions in the *RefSet* and delete the other $b/2$. As in step 1, a set P of *PopSize* improved solutions is generated and the $b/2$ with maximum diversity are added to complete the *RefSet*. The procedure stops when *MaxEval* objective function evaluations have been performed.

Since the SS and GA implementations will share the combination and improvement methods, we first provide the details for the GA procedure and then describe the mechanisms to combine solutions and the local search used for improving trial solutions.

3. GA Implementation

Just as in the case of the scatter search, we have implemented a standard genetic algorithm for the purpose of comparing performance. Our description of scatter search in the previous section is more detailed, because SS is not as known in the literature as GAs, even though they both date back to the mid seventies. The GA implementation follows the scheme of Michalewicz (1996) and is summarized in Figure 2.

1. *Generate solutions* — Build P by randomly generating *PopSize* permutations.
2. *Improve solutions* — Apply the LS method to improve solutions generated in Step 1.

while (objective function evaluations < *MaxEval*) do {

3. *Evaluate solutions* — Evaluate the solutions in P and update the best solution found if necessary.
4. *Survival of the fittest* — Calculate the probability of surviving based on solution quality. Evolve P by choosing *PopSize* solutions according to their probability of surviving.
5. *Combine solutions* — Select a fraction p_c of the solutions in P to be combined. Selection is at random with the same probability for each element of P. The selected elements are randomly paired for combination, with each pair generating two offspring that replace their parents in P.
6. *Mutate solutions* — A fraction p_m of the solutions in P is selected for mutation. The mutated solution replaces the original in P.

}

Figure 2. GA outline

Offspring generated with the combination procedure in step 5 and mutations generated in step 6 are subjected to the improvement method referred to in step 2. The improvement method is the same as the one used in the scatter search implementation and described in the following section.

3. Improvement Method

Insertions are used as the primary mechanism to move from one solution to another in our improvement method. We define $MOVE(p_j, i)$ to consist of deleting p_j from its current position j in p to be inserted in position i (i.e., between the current elements p_{i-1} and p_i). This operation results in the ordering p' as follows:

$$p' = \begin{cases} (p_1, \ldots, p_{i-1}, p_j, p_i, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n) & \text{for } i < j \\ (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_i, p_j, p_{i+1}, \ldots, p_n) & \text{for } i > j \end{cases}$$

Since the local search method is context independent, the only available mechanism for computing the move value is submitting p' for evaluation and comparing its value with the value of p. In order to reduce the computational effort associated with evaluating moves for possible selection and to increase the search efficiency, we define $INSERT(p_j)$ as the set of promising insert positions for p_j. We consider inserting p_j only in those positions in $INSERT(p_j)$. Then, the neighborhood N of the current solution is given as:

$$N = \{p' : MOVE(p_j, i), \text{ for } j = 1, \ldots, n \text{ and } i \in INSERT(p_j)\}$$

We partition N into n sub-neighborhoods Nj associated with each element p_j as:

$$N_j = \{p' : MOVE(p_j, i),\ i \in INSERT(p_j)\}$$

The set $INSERT(p_j)$ depends on whether the problem is an A-permutation or a R-permutation problem. In A-permutation problems we accumulate in $FreqIns(i,j)$ the number of times that element i has been inserted in position j improving the current solution. Then, given an element i, we compute $m(i)$ as the position j where the value of $FreqIns(i,j)$ is maximum. We consider that $m(i)$ and the positions around it are desirable positions for inserting element i. This information is used to assign $INSERT(p_j)$ the following values:

$$INSERT(p_j) = [\ m(p_j) - RANGE,\ m(p_j) + RANGE\ ]$$

The value of $RANGE$ is an additional search parameter. In R-permutation problems we accumulate in $FreqIns(i,j)$ the number of times that element i has been inserted in the position immediately preceding element j. Then we compute $m(i)$ as the element j with maximum $FreqIns(i,j)$ value. We define $pp(e)$ as the previous position of element e in the current solution. Then we can consider that $pp(m(i))$ is a desirable position for inserting element i. In this case, $INSERT(p_j)$ is assigned the following values:

$$INSERT(p_j) = \{\ pp(e)\ /\ FreqIns(p_j,e) \geq \alpha\ m(p_j)\ \}$$

where the value of α is dynamically adjusted to obtain a set with $2*RANGE$ elements. The implementation is such that we avoid ordering all the elements in $FreqIns(i,j)$, so we are able to construct the set $INSERT(p_j)$ with a low computational effort.

The rule for selecting an element for insertion is based on frequency information. Specifically, the number of times that element j has been moved resulting on an improved solution is accumulated in $freq(j)$. The probability of selecting element j is proportional to its frequency value $freq(j)$.

Starting from a trial solution constructed with either the diversification generator or any of the combination methods, the local search (LS) procedure chooses the best insertion associated with a given element. At each iteration, an element p_j in the current solution p is probabilistically selected according its $freq(j)$ value. The solution p' with the lowest value in N_j is selected. The LS procedure execute only improving moves. An improving move is one for which the objective function value of p' is better (strictly smaller for minimization problems or strictly larger for maximization problems) than the objective function value of p. The LS procedure terminates when no improving move is found after $NTrials$ elements are consecutively selected and the exploration of their neighborhood fails to find an improving move.

3. Combination Methods

The combination methods, as referred to in scatter search, or operators, as referred to in genetic algorithms, are key elements in the implementation of these optimization procedures. Combination methods are typically adapted to the problem context. For example, linear combinations of solution vectors have been shown to yield improved outcomes in the context of nonlinear optimization (Laguna and Martí 2000). An adaptive structured combination that focuses on absolute position of the elements in solutions to the linear ordering problem was shown effective in Campos, et al. (2001). (This combination method is labeled #7 below.) In order to design a context-independent combination methodology that performs well across a wide collection of different permutation problems, we propose a set of 10 combination methods from which one is probabilistically selected according to its performance in previous iterations during the search.

In our implementation of scatter search, solutions in the $RefSet$ are ordered according to their objective function value. So, the best solution is in the first one in $RefSet$ and the worst is the last one. When a solution obtained with combination method i (referred to as cm_i) qualifies to be the j^{th} member of the current $RefSet$, we add b-j+1 to $score(cm_i)$. Therefore, combination methods that generate good solutions accumulate higher scores and increase their probability of being selected. To avoid initial biases, this mechanism is activated after the first $InitIter$ combinations, and before this point the selection of the combination method is made completely at random.

In the GA implementation, when a solution obtained with combination method cm_i is better than its parent solutions, $score(cm_i)$ is increased by one. If the combination method is a mutation operator, then the score is increased by one when the mutated

solution is better than the original solution. Preliminary experiments showed that there was no significant difference between using this scheme from the beginning of the search and waiting *InitIter* combinations to activate it. Therefore, the probabilistic selection procedure is activated from the beginning of the GA search. A description of the ten combination methods follows, where we refer to the solutions being combined as "reference solutions" and to the resulting solution as the "trial solution" (although in the GA literature reference solutions are known as "parents" and trial solutions as "offspring").

Combination Method 1

This is 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 solution while the remaining elements are randomly selected from both reference solutions. For each position i ($i = k+1, \ldots, n$) the method randomly selects one reference solution and copies the first element that is still not included in the new trial solution.

Combination Method 2

This method is a special case of 1, where the crossing point k is always fixed to one.

Combination Method 3

This is 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.

Combination Method 4

This method is a case of what is referred to in the GA literature as a mutation operator. The method selects two random points 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.

Combination Method 5

This combination method also 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.

Combination Method 6

This is a special case of combination method 5 where the sublist always starts in position 1 and the length is randomly selected in the range $[2, n/2]$.

Combination Method 7

The method scans (from left to right) both reference solutions, and uses the rule that each reference solution votes for its first element that is still not included in the new trial solution (referred to as the "incipient element"). The voting determines the next element to enter the first still unassigned position of the trial solution. This is a min-max rule in the sense that if any element of the reference solution is chosen other than the incipient element, then it would increase the deviation between the reference and the trial solutions. Similarly, if the incipient element were placed later in the trial solution than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the trial solution from the reference solution under consideration, subject to the fact that other reference solution is also competing to contribute. A bias factor that gives more weight to the vote of the

reference solution 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.

Combination Method 8

In this method the two reference solutions vote for their incipient element to be included in the first still unassigned position of the trial solution. If both solutions vote for the same element, the element is assigned. If the reference solutions vote for different elements but 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.

Combination Method 9

Given two reference solutions p and q, this method probabilistically selects the first element from one of these solutions. The selection is biased by the objective function value corresponding to p and q. Let e be the last element added to the new trial solution. Then, p votes for the first unassigned element that is position after e in the permutation p. Similarly, q votes for the first unassigned element that is position after e in q. 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 probabilistically biased by the objective function values of p and q.

Combination Method 10

This is 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. Other wise, 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 p and q have objective function values of $value(p) = 40$ and $value(q) = 60$, than p should contribute with 40% of the elements in the new trial solution and q with the remaining 60% in a maximization problem. The scores are updated so 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 (1994b).

4. Test Problems

We have used four combinatorial optimization problems to compare the performance of the scatter search and GA implementations. Solutions to these problems are naturally represented as permutations:

- the bandwidth reduction problem
- the linear ordering problem
- the traveling salesman problem
- a single machine sequencing problem.

We target these problems because they are well known, they are different among themselves and problem instances with known optimal or high-quality solutions are readily available. Existing methods to solve these problems range from construction heuristics and metaheuristics to exact procedures. We now provide a brief description of each problem class.

The *bandwidth reduction problem* (BRP) refers to finding a configuration of a matrix that minimizes its bandwidth. The bandwidth of a matrix $A = \{a_{ij}\}$ is defined as the maximum absolute difference between i and j for which $a_{ij} \neq 0$. The BRP consists of finding a permutation of the rows and columns that keeps the nonzero elements in a band that is as close as possible to the main diagonal of the matrix, the objective is to minimize the bandwidth. This NP-hard problem can also be formulated as a labeling of vertices on a graph, where edges are the nonzero elements of the corresponding symmetrical matrix. Metaheuristics proposed for this problem include a simulating annealing implementation by Dueck and Jeffs (1996) and a tabu search approach by Martí, et al (2001).

The Linear Ordering Problem (LOP) is also a NP-hard problem that has a significant number of applications. This problem is equivalent to the so-called triangulation problem for input-output tables in economics and has generated a considerable amount of research interest over the years, as documented in Grötschel, Jünger and Reinelt (1984), Chanas and Kobylanski (1996), Laguna, Martí and Campos (1999) and Campos, et al (2001). Given a matrix of weights the LOP consists of finding a permutation of the columns (and simultaneously the rows) in order to maximize the sum of the weights in the upper triangle, the equivalent problem in graphs is that of finding, in a complete weighted graph, an acyclic tournament with a maximal sum of arc weights. For a complete description of this problem, its properties and applications see Reinelt (1985).

The Traveling Salesman Problem (TSP) consists of finding a tour (cyclic permutation) visiting a set of cities that minimizes the total travel distance. A incredibly large amount of research has been devoted to this problem, which would be impossible and impractical to summarize here. However, a couple of valuable references about the TSP are Lawler, et al (1985) and Reinelt (1994).

Finally, the fourth problem is a single machine-sequencing problem (SMS) with delay penalties and setup costs. At time zero, n jobs arrive at a continuously available machine. Each job requires a specified number of time units on the machine and a penalty (job dependent) is charged for each unit that job commencement is delayed after time zero. In addition, there is a setup cost s_{ij} charged for scheduling job j immediately after job i. The objective is to find the schedule that minimizes the sum of the delay and setup costs for all jobs. Note that if delay penalties are ignored, the problem becomes an asymmetric traveling salesman problem. Barnes and Vanston (1981) reported results on three branch and bound algorithms of instances with up 20 jobs and Laguna, Barnes and Glover (1993) developed a TS method that is tested in a set of instances whose size ranges between 20 and 35 jobs.

5. Computational Experiments

For our computational testing, we have employed the following problem instances:

- 37 BRP instances from the Harwell-Boeing Sparse Matrix Collection found in http://math.nist.gov/MatrixMarket/data/Harwell-Boeing. This collection consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of

scientific and engineering disciplines. The size of these instances ranges between 54 and 685 rows with an average of 242.9 rows (and columns).

- 49 LOP instances from the public-domain library LOLIB (1997) found in http://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB. These instances consist of input-output tables from economic sectors in the European community and their size ranges between 44 and 60 rows with an average of 48.5 rows (and columns).
- 31 TSP instances from the public-domain library TSPLIB (1995) found in http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib. These instances range in size between 51 and 575 cities with an average of 159.6 cities.
- 40 SMS instances from Laguna, Barnes and Glover (1993) available upon request from laguna@colorado.edu. The best solutions available for these instances are not proven optimal, however they are the best upper bounds ever found. These instances range in size between 20 and 35 jobs with an average of 26.0 jobs.

In order to apply the different strategies described above, we have classified the BRP and SMS as A-permutation problems and the LOP and TSP as R-permutation problems. Note that the objective function in the SMS problem is influenced by both the absolute position of the jobs (due to the delay penalties) and the relative position of the jobs (due to the setup costs). Our classification is based on the knowledge that the delay penalties in the tested instances are relatively larger when compare to the setup cost. In cases when this knowledge is not available, it would be recommended to run the procedure twice on a sample set of problems in order to establish the relative importance of the positioning of elements in the permutation.

The solution procedures were implemented in C++ and compiled with Microsoft Visual C++ 6.0, optimized for maximum speed. All experiments were performed on a Pentium III at 800 MHz. The scatter search parameters *PopSize*, *MaxIter, b* and *InitIter* were set to 100, 2, 10 and 50 respectively, as recommended in Campos et al. (1999). The parameters associated with the local search procedure were set after some preliminary experimentation (*RANGE* = 3 and *Ntrials* = 25). The GA parameters were set to *PopSize* = 100, p_c = 0.25 and p_m = 0.01, as recommended in Michalewicz (1996). Both procedures used the stopping criterion of 1 million objective function evaluations.

In our experiments we compare the SS implementation with two versions of the GA procedure: 1) without local search (GA) and 2) with local search (GALS). SS always uses local search by design. The two GA versions complement each other in that one uses part of its objective function evaluation "budget" to perform local searches and the other uses its entire budget to apply the combination operators and evolve the population of solutions. In our first experiment we restrict the use of combination methods in such a way that both GA versions use combination methods 1-6 and SS uses combination methods 7-10. Recall that combination methods 1-3 are crossover operators and combination methods 4-7 are mutation operators. Combination methods 7-10 are more strategic in nature and one is completely deterministic.

Table 1 shows the average percent deviation from the best-known solution to each problem . The procedures were executed once with a fixed random seed and the average is over all instances. It should be mentioned that in the case of LOP and TSP the best solutions considered are the optimal solutions as given in the public libraries. In the case of the BRP the best solutions are from Martí et al (2001) and the best solutions for the SMS instances are due to Laguna et al (1993).

Method	BRP	LOP	SMS	TSP
GA	59.124%	6.081%	4.895%	95.753%
GALS	55.500%	0.005%	0.832%	143.881%
SS	58.642%	0.000%	0.291%	43.275%

Table 1. Percent deviation from best

Table 1 shows that SS has a better average performance than GA in all problem types but is inferior to GALS when solving BRP instances. The three methods are able to obtain high quality results for the LOP and the SMS problem instances. The results for the TSP are less desirable, but it should be noted that these problems are on average larger than the LOP and SMS instances. Table 2 presents the average percent improvement of SS over the two GA versions. The largest improvement occurs when solving TSP instances and the negative improvement associated with GALS shows the better average performance of this method when compared to SS.

Method	BRP	LOP	SMS	TSP
GA	0.3%	6.5%	4.4%	26.8%
GALS	-2.0%	0.0%	0.5%	41.3%

Table 2. Percent improvement of SS over GA and GALS

Table 2 shows that in general the use of local search within the GA framework results in improved outcomes, with the TSP as a notable exception to this general rule. For the TSP, the performance of the GA implementation deteriorates when coupled with the local search procedure. The SS vs. GALS in the case of BRP instances deserves a closer examination, because it is the only case in which either GA implementation outperforms SS. Figure 3 shows the trajectory of the average percent deviation from the best-known solution as the SS and GALS searches progress.

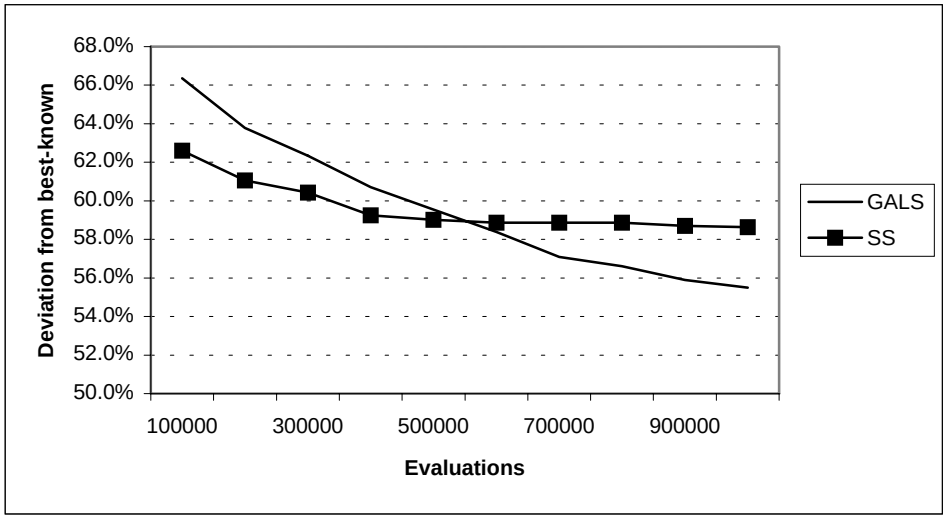

Figure 3. Objective function trajectory for SS and GALS solving BRP instances

The percent deviation value trajectory in Figure 3 shows that while SS results are better than or at least as good as GALS results before 600 thousand evaluations, GALS keeps improving while SS stagnates. Our explanation for this behavior is as follows. In the BRP, the change in the objective function value from one solution to another does not represent a meaningful guidance for the search. In particular, the objective is a min-max function that in many cases results in the same evaluation for all the solutions in the neighborhood of a given solution. Since SS relies on strategic

choices, it is unable to obtain information from the evaluation of this "flat landscape" function to direct the search. GALS, heavily relying on randomization, is able to more effectively probe the solution space presented by BRP instances.

Figure 4 depicts the trajectory of the percent deviation from the best-known solution to the SMS problem as the search progress. The average values of the three methods under consideration are shown.

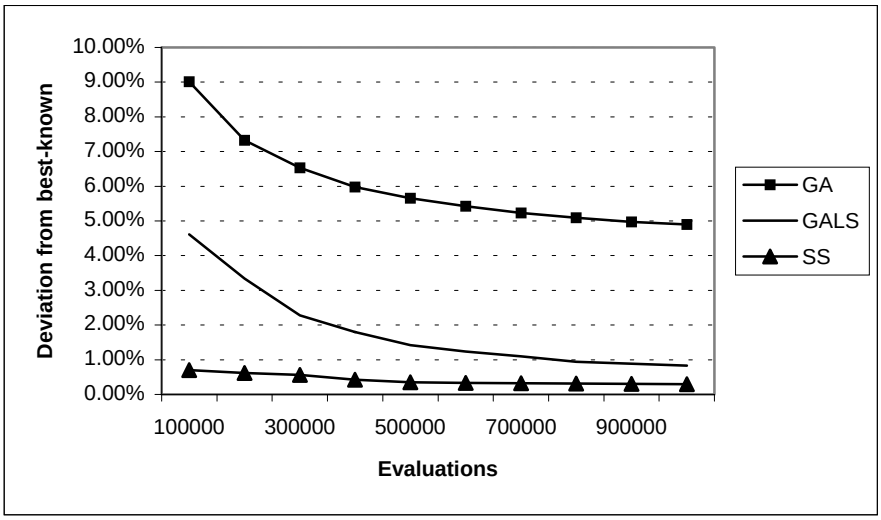

Figure 4. Percent deviation trajectory when solving SMS instances

Figure 4 shows that the SS procedure is able to obtain high quality solutions from the very beginning of the search. Specifically, after 100,000 objective function evaluations, the percent deviation from the best-known solutions for the SS method is 0.7 while after 1 million evaluations the GA and the GALS methods are still at 4.8 and 0.8, respectively.

In our second experiment, we compare SS, GA and GALS when allowed to use all the combination methods described in section 3. The results of these experiments are summarized in Tables 3 and 4. These tables are equivalent to Tables 1 and 2 in that they shows the percent deviation from the best known solution values and the percent improvement of SS over GA and GALS, respectively.

Method	BRP	LOP	SMS	TSP
GA	58.244%	6.722%	4.940%	101.689%
GALS	55.102%	0.004%	0.268%	133.792%
SS	52.587%	0.000%	0.207%	54.321%

Table 3. Percent deviation from best

Method	BRP	LOP	SMS	TSP
GA	3.57%	7.21%	4.51%	23.49%
GALS	1.62%	0.00%	0.06%	33.99%

Table 4. Percent improvement of SS over GA and GALS

A direct comparison of tables 1 and 3 shows the advantage of using all combination methods within both GA and SS. Tables 3 and 4 indicate that when all combination methods are used, SS has a superior average performance than GA and GALS. The performance is only marginally better in the case of LOP and SMS, but it continues to

be significantly better in the case of TSP. When using all operators, including those that incorporate a fair amount of randomization, SS is able to outperform GALS in the BRP instances, as shown in Figure 5.

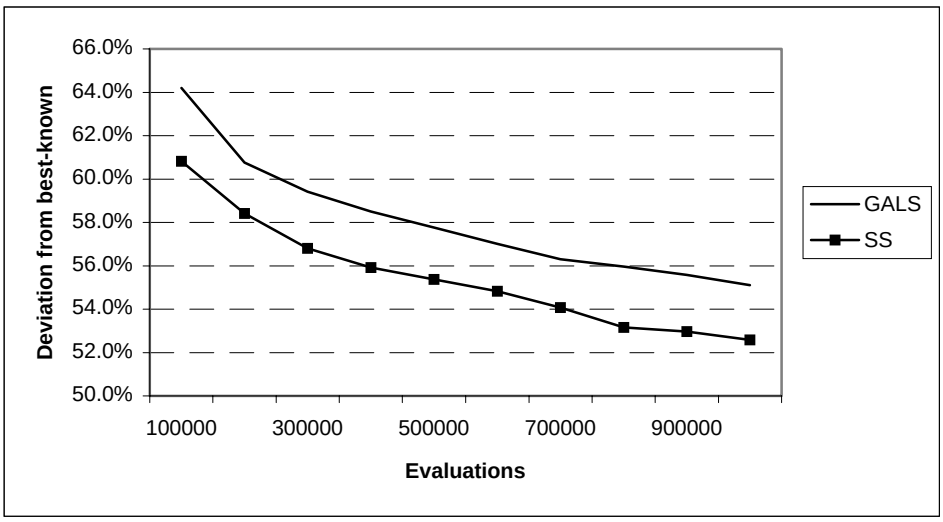

Figure 5. Objective function trajectory for SS and GALS solving BRP instances

Figure 6 shows the trajectory of the percent deviation from the best-known solutions to the SMS problem as the search progress. The average values shown in Figure 6 correspond to the second experiment, where all combination methods are made available to SS, GA and GALS.

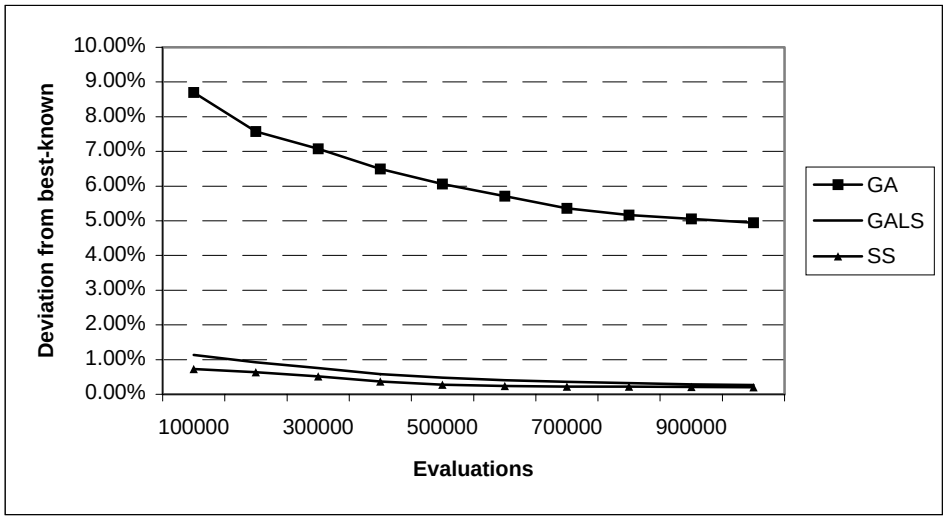

Figure 6. Percent deviation trajectory when solving SMS instances

A comparison of figures 4 and 6 reveals that while SS shows a minor improvement when solving SMS problems with using all combination methods, the improvement associated with GALS is significant. GALS achieves a deviation of 1.14% at the 100,000 evaluation mark when using the 10 combination methods, which compares quite favorably with a deviation of 4.8% at the same stage of the search when using a fraction of the available combination methods.

6. Conclusions

We have implemented a scatter search and a GA procedure for a class of combinatorial problems whose solutions can be represented as permutations with the purpose of comparing performance of these competing metaheuristics. The implementations are standard and share 10 combination methods. They also share the improvement method based on a simple hill-climbing procedure. The procedures are context-independent in the sense that they treat the objective function evaluation as a black box. To allow for the use of key search strategies, both implementations require that the problems being solved be classified as either "absolute" or "relative" in terms of the relevant factor for positioning elements in the permutation.

The performance of the procedures was assessed using 157 instances of four different permutations problems. SS can claim superiority when solving TSP instances but only a modest improvement over GAs with local search when solving LOP, BRP or SMS instances. We are certain that this won't be the last time SS will be compared against GAs and that in any given setting one could outperform the other. Our main finding is that both methodologies are capable of balancing search diversification and intensification when given the same tools for combining and improving solutions.

References

Barnes, J. W. and L. K. Vaston (1981) "Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs," *Operations Research,* vol. 29, pp. 146-160.

Campos, V., M. Laguna and R. Martí (1999) "Scatter Search for the Linear Ordering Problem," Corne, Dorigo and Glover (Eds.) New Ideas in Optimization, McGraw-Hill, UK.

Campos, V., F. Glover, M. Laguna and R. Martí (2001) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," *Journal of Global Optimization*, vol. 21, pp. 397-414.

Chanas, S. and P. Kobylanski (1996) "A New Heuristic Algorithm Solving the Linear Ordering Problem," *Computational Optimization and Applications*, vol. 6, pp. 191-205.

Dueck, G. H. and J. Jeffs (1995) "A Heuristic Bandwidth Reduction Algorithm," *J. of Combinatorial Math. And Comp.,* vol. 18, pp. 97-108.

Everett, H. (1963) "Generalized Lagrangean Multiplier Method for Solving Problems of Optimal Allocation of Resources," *Operations Research,* vol. 11, pp. 399-417.

Glover, F. (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research,* vol. 13, pp. 879-919.

Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences,* vol. 8, no. 7, pp. 156-166.

Glover, F. (1994a) "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing,* vol. 4, pp. 131-140.

Glover, F. (1994b) "Tabu Search for Nonlinear and Parametric Optimization with Links to Genetic Algorithms," *Discrete Applied Mathematics,* vol. 49, pp. 231-255.

Glover, F. (1995) "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," *OR Spektrum*, vol. 17, no. 2-3, pp. 125-138.

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

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

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

Grötschel, M., M. Jünger and G. Reinelt (1984), "A Cutting Plane Algorithm for the Linear Ordering Problem," *Operations Research*, vol. 32, no. 6, pp. 1195-1220.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

Laguna, M., Barnes, J.W. and Glover, F. (1993), "Intelligent Scheduling with Tabu Search: An Application to Jobs With Linear Delay Penalties and Sequence-Dependent Setup Costs and Times", *Journal of Applied Intelligent*, vol. 3, pp. 159-172.

Laguna, M., R. Martí (2000) "Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions," Technical Report TR11-2000, Dpto de Estadística e I.O., University of Valencia.

Laguna, M., R. Martí and V. Campos (1999) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," *Computers and Operations Research*, vol. 26, pp. 1217-1230.

Lawler, Lenstra, Rinnoy Kan and Shmoys (1985) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons.

Martí, R., Laguna, M., Glover, F. and Campos, V. (2001) "Reducing the Bandwidth of a Sparse Matrix with Tabu Search", *European Journal of Operational Research*, vol. 135, pp. 450-459.

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition, Springer-Verlag, Berlin.

Reinelt, G. (1985) "The Linear Ordering Problem: Algorithm and Applications," *Research and Exposition in Mathematics*, vol. 8, H. H. Hofman and R. Wille (eds.), Heldermann Verlag, Berlin.

Reinelt, G. (1994) "The Traveling Salesman: Computational Solutions for TSP applications," *Lecture Notes in Computer Science*, Springer Verlag, Berlin.