Context-Independent Scatter and Tabu Search for Permutation Problems

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

a Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, 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 Graduate School of Business and Administration, 419 UCB, University of Colorado, Boulder, CO 80309-0419, USA Laguna@Colorado.edu

Abstract

In this paper, we develop a general purpose heuristic for permutations problems. The procedure is based on the scatter search and tabu search methodologies and treats the objective function evaluation as a black box, making the search algorithm context-independent. We perform computational experiments with four well-known permutation problems to study the efficiency and effectiveness of the proposed method.

Keywords: Scatter search, tabu search, combinatorial optimization, permutation problems.

Last revision: July 2, 2001

1. Introduction

The purpose of this paper is to develop a context-independent method for solving an important class of combinatorial optimization problems. Specifically we tackle problems whose solutions can be represented with a permutation.

Our general-purpose heuristic is based on a model that treats the objective function evaluation as a black box, making the search algorithm context-independent. The procedure is a scatter-tabu search hybrid. The scatter search framework provides a means for diversifying the search throughout the exploration of the permutation solution space. Two improvement methods are used to intensify the search in promising regions of the solution space: a simple local search based on exchange moves and a short-term memory tabu search. Improved solutions are then used for combination purposes within the scatter search design.

Our method does not take advantage of the structure of each problem instance; therefore it may not directly improve upon solutions found with procedures designed for specific permutation problems. Nonetheless, we show that our results are competitive considering the general nature of the search procedure. We base our conclusions on experimental testing with four well-known problems: linear ordering, traveling salesman, matrix bandwidth reduction and a job-sequencing problem. Our procedure, however, is most useful when applied to problems for which specialized procedures are not available, for example, job-sequencing problems with nonlinear and/or stochastic objective functions.

Metaheuristics have provided a mechanism for considerably improving the performance of simple heuristic procedures. The search strategies proposed by metaheuristic methodologies result in iterative procedures that have the ability of escaping local optimal points. Genetic algorithms (GAs) and scatter search (SS), for example, are metaheuristics designed to operate on a set of solutions that is maintained from iteration to iteration. On the other hand, metaheuristics such as simulated annealing (SA) and tabu search (TS) typically maintain only one solution by applying mechanisms to change this solution from one iteration to the next. Metaheuristics have been developed to solve complex optimization problems in many areas, with combinatorial optimization being one of the most fruitful. Generally, the most efficient procedures achieve their efficiencies by relying on context information. The solution method can be viewed as the result of adapting metaheuristic strategies to specific optimization problems. In these cases, there is no separation between the solution procedure and the model that represents the system that produces the optimization problem to be solved.

Metaheuristics can also be used to create solution procedures that are context independent. The original genetic algorithmic designs were based on this model. The advantage of this design is that the same solver can be applied to a wide variety of problems. The obvious disadvantage is that the solutions found by context-independent solvers might be inferior to those of specialized procedures when both are allotted the same amount of computer effort (e.g., total search time). Context-independent solvers (also referred to as general-purpose optimizers) based on metaheuristics have found their home in commercial implementations. The Premium Solver Platform version 3.5 of Frontline Systems, Inc (www.frontsys.com), for instance, includes the Standard Evolutionary solver that is a context-independent GA implementation. Opttek Systems, Inc (www.opttek.com) commercializes OptQuest, a context-independent solver based on scatter search. Other GA-based commercial implementations of general-purpose optimizers are Evolver by Palisade Corporation (www.palisade.com) and Pointer by Synaps, Inc (www.synaps-inc.com).

One of the main design considerations when developing a general-purpose optimizer is the solution representation to be employed. The solution representation is used to establish the communication between the optimizer and the solution evaluator (which

generally is an abstraction of a complex system represented, for instance, by means of a computer simulation). Classical GA implementations used binary strings to represent solutions even in problems where this representation was clearly inadequate (like in the case of permutation problems). In our current development, we tackle problems whose solutions can be represented with permutations and use this representation within a solution procedure based on the SS methodology. It is interesting to point out that modern commercial solvers based on metaheuristics such as Frontline's Standard Evolutionary solver, OptQuest and Evolver support the permutation-based solution representation. (Pointer does not support this representation because it specializes in searching for optimal solutions to engineering design problems.)

2. Proposed Scatter Search Procedure

Scatter search is a methodology that operates on a reference set (or population) of solution vectors (which in our case are permutations). The SS process is organized to (1) capture information not contained separately in the original vectors, (2) take advantage of auxiliary heuristic solution methods (to evaluate the combinations produced and to actively generate new vectors), and (3) make dedicated use of strategy to carry out component steps. For a detailed description of the SS methodology see Glover, Laguna and Martí (2000).

We have adapted scatter search with the goal of developing a context-independent solver for permutation problems. The solver is designed in such a way that the user must specify whether the objective function evaluation is more sensitive to the "absolute" positioning of the elements in the permutation or to their "relative" positioning. Hence, we differentiate between two classes of problems:

> *A-permutation problems*—for which absolute positioning of the elements is more important
>
> *R-permutation problems*—for which relative positioning of the elements is more important

The procedure, summarized in Figure 1, 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. This contrasts with the typical GA approach of randomly generating a initial set of solutions from which to start the evolutionary search. In order to obtain a set of solutions of reasonable quality and diversity, we apply an improvement method to the solutions in *P*. The improvement method consists of two phases, a simple tabu search and a local search (LS), which are described in the next section. The TS is based on a short-term memory function and is applied only to the most promising solutions. In particular, given a solution *p* with an objective function value *value*(*p*), the TS is applied if the following condition holds:

$$value(p) - |value(p)| * (1 - threshold) \leq value(BestSol)$$

where *BestSol* is the best solution found so far and *threshold* is a given search parameter. The local search procedure is applied to all trial solutions, that is, those returned by the TS method as well as those not submitted to the tabu search. This is performed in steps 2, 6 and 9 in the outline of Figure 1. After step 2, *P* consists of the improved solutions obtained after the application of TS/LS or LS alone.

1. *Generate solutions* — Apply the diversification generation method to generate a set of *PopSize* solutions.
2. *Improve solutions* — Apply the TS and LS methods 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*) 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 methods 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 (G*lobalImprove* < *MaxIter*)

Figure 1. Scatter Search outline

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 $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ 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 described above, that is, with the application of the TS and/or LS procedures. 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.

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 *MaxIter* global iterations are performed without improving the value of the best solution.

The combination method is a key element in scatter search implementations. This method is 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. (2000). (This combination method 7 below.) In order to design a context-independent combination methodology that performs well across a wide collection of different problems, we propose a set of 10 combination methods from which one is probabilistically selected according to its performance in previous iterations.

In our implementation, 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 selections are made completely at random. A description of the ten combination methods follows.

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 point while the remaining elements are randomly selected from both reference points. For each position i (i = k+1, …, n) the method randomly selects one reference point and copy 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 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 permutations, and uses the rule that each reference permutation votes for its first element that is still not included in the combined permutation (referred to as the "incipient element"). The voting determines the next element to enter the first still unassigned position of the combined permutation. This is a min-max rule in the sense that if any element of the reference permutation is chosen other than the incipient element, then it would increase the deviation between the reference and the combined permutations. Similarly, if the incipient element were placed later in the combined permutation than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the combined solution from the reference solution under consideration, subject to the fact that other reference solution is also competing to contribute. A bias factor that gives more weight to the vote of the reference permutation with higher quality is also implemented for tie breaking. This rule is used when more than one element receives the same votes. Then the element with highest weighted vote is selected, where the weight of a vote is directly proportional to the objective function value of the corresponding reference solution.

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 combined permutation. 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 (1994).

3. Tabu Search and Local Search methods

Insertions are used as the primary mechanism to move from one solution to another in the local search 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 of constructing 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.

The tabu search (TS) method is also based on the insert neighborhood described above. At each iteration, an element p_j is probabilistically selected according to its *freq*(*j*) value and the move that leads to the best solution in N_j is selected. The move is executed even when the move does not improve upon the current solution, resulting in a possible deterioration of the current objective function value. The moved element p_j becomes tabu-active for *TabuTenure* iterations, and therefore, it cannot be selected for insertion during this time. The TS routine terminates after *TabuIter* consecutive iterations without improvement. The best solution found is returned as the output of the method.

4. Permutation Problems used for Testing

We have used four combinatorial optimization problems to test our procedure. Solutions to these problems are naturally represented as permutations:

- the bandwidth reduction
- 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 in nature and problem instances with known optimal 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 (1998) and Campos, et al (2000). 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) that consists of finding a tour (cyclic permutation) visiting a set of cities that minimizes the total travel distance. A tremendous 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 up 20 jobs and Laguna, Barnes and Glover (1993) developed a TS method that is tested in a set of instances that ranges from 20 to 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 from 54 to 685 rows with an average of 242.9 rows (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 from 44 to 60 rows with an average of 48.5 rows (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 from 51 to 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 have not been proved optimal, however they are the best upper

bounds ever found. These instances range in size from 20 to 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 sample set of problems in order to establish the relative importance of the positioning of elements in the permutation.

The solution procedure was 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).

In our first experiment we consider the SS method with LS but without the TS routine as a baseline to compare future experiments as well as to measure the contribution of the TS improvement method. Table 1 reports for each problem type (1) the average percentage deviation from the best known solution to each problem, (2) the average number of evaluations, (3) the average percentage of improvement obtained from the best solution in the initial reference set, (4) the average number of rebuilds of the reference set, (5) the average CPU seconds, (6) the average time spent in the local search routine and (7) the average percentage improvement achieved by the local search method. 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 (2000) and the best solutions for the SMS instances are due to Laguna et al (1993).

Since the best solution in the initial reference set is obtained with the application of the diversification generator and the local search method, the values in the column labeled "Improvement from initial best solution" in Table 1 provide a measure of the contribution of the combination methods to the algorithm's output.

	Deviation from best	Number of evaluations	Improvement from initial best solution	Number of rebuilds	Total seconds	LS seconds	LS improvement
BRP	178.8%	263339	10.3%	3.9	12.7	12.4	4.6%
LOP	0.003%	745904	0.1%	3.1	8.9	8.8	31.0%
TSP	23.4%	12362564	31.8%	17.0	98.2	97.1	19.2%
SMS	0.1%	1272044	0.5%	5.2	2.6	2.5	14.2%

Table 1. Scatter Search method

As anticipated, the procedure does not perform equally well across problem types. Table 1 shows that in the LOP and SMS problem the procedure yields excellent results with average percent deviations from the best solutions of 0.003% and 0.1%, respectively. These results are obtained with a computational effort that average less than 9 seconds. The performance on the TSP is less desirable, but we should keep in mind that these problems are on average larger than in the case of the LOP and SMS. The method has unacceptable results for the BRP. There are two reasons for this output: the instances are significantly larger and 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.

Our context-independent method is then unable to obtain information from the evaluation of this "flat landscape" function to direct the search.

It is interesting to note the effect of the value of *MaxIter* in the number of times that the reference set is rebuilt. In most problem instances this value is relatively small, meaning that most of the good solutions are found early in the search and that the method does not perform many global iterations. In the TSP instances, however, the method finds some of the good solutions later in the search and therefore performs several "restarts" before terminating. This termination criterion is typical in commercial software, where the user is always able to stop the process manually or allow the search to run to completion according to a predefined rule (such as the one we have used here).

One of the key elements in the way solutions are combined within our adaptation of scatter search is the self-adapting nature of the combination methods. For the experiment reported in Table 1, we recorded the number of times each combination method is used throughout the search. In Table 2 we report the relative frequency of use of each combination method by problem class. Note that in the BRP case the most frequently used combination method is 6, which is one that operates on a single solution and uses a fair amount of randomization over strategy. This is not surprising given the established fact that the change on the objective function value does not provide adequate search directions. In problems where our procedure performs well, combination methods that rely on strategic choices, such as number 7, are used more often. In general, there is a more uniform distribution of frequencies in problems where the method performs well (see the LOP row in Table 2).

	Combination Methods									
	1	2	3	4	5	6	7	8	9	10
BRP	2%	1%	5%	5%	10%	59%	9%	7%	1%	0%
LOP	20%	9%	16%	2%	4%	5%	34%	7%	2%	2%
TSP	3%	1%	3%	2%	1%	4%	29%	55%	1%	0%
SMS	21%	11%	34%	3%	4%	5%	12%	7%	2%	1%

Table 2. Relative Frequency

In our next set of experiments we tested the merit of the short-term memory TS adaptation as an additional improvement method within our SS design. We begin our experimentation with the tabu search improvement method by setting the *TabuIter* and *TabuTenure* parameters to 100 and 10, respectively. We then test 3 values for *threshold.* = 1.0, 0.8, 0.2. Tables 3, 4 and 5 show the results for each value of this parameter, where this time we replace the LS values with the TS values in the last two columns of the tables.

	Deviation from best	Number of evaluations	Improvement from initial best solution	Number of rebuilds	Total seconds	TS seconds	TS improvement
BRP	175.5%	832370	6.2%	4.5	42.4	30.6	6.1%
LOP	0.001%	1177229	0.01%	2.4	14.4	12.8	51.8%
TSP	25.3%	27663495	15.6%	9.6	297.4	289.3	62.6%
SMS	0.1%	2326553	0.3%	5.2	5.4	4.5	32.0%

Table 3. Average performance with *threshold* = 1.0

	Deviation from best	Number of evaluations	Improvement from initial best solution	Number of rebuilds	Total seconds	TS seconds	TS improvement
BRP	164.1%	1308025	10.8%	4.4	74.3	62.3	5.6%
LOP	0.001%	1340786	0.01%	2.3	16.5	15.3	40.2%
TSP	17.6%	25024919	20.6%	8.1	281.3	274.6	21.4%
SMS	0.1%	2793349	0.4%	4.5	5.8	5.2	14.2%

Table 4. Average performance with *threshold* = 0.8

	Deviation from best	Number of evaluations	Improvement from initial best solution	Number of rebuilds	Total seconds	TS seconds	TS improvement
BRP	156.3%	1479022	10.8%	4.3	75.2	64.4	6.7%
LOP	0.001%	1394947	0.01%	2.3	17.2	16.2	40.1%
TSP	15.8%	25587886	22.0%	7.6	264.7	258.9	23.1%
SMS	0.05%	2669407	0.4%	4.2	5.5	5.0	14.7%

Table 5. Average performance with *threshold* = 0.2

Tables 3, 4 and 5 show that as the value of *threshold* decreases the average deviation from the best solutions generally decreases. This trend is expected, because skipping the TS phase may result in a missing opportunity to improve upon the trial solution generated as a result of combining solutions or applying the diversification generator. However, as *threshold* decreases, the method employs more computational time. Note that the results for the LOP and SMS problem are not significantly affected by the change on the *threshold* parameter, since the scatter search is capable of finding high quality solutions to these instances without the application of the TS routine.

These tables also show that when the TS method is activated, it is responsible for a large contribution in the quality of the solutions as well as for the total CPU time. For example in the LOP the TS improves the trial solutions by at least 40%, on average. We have also observed that the contribution of the LS procedure is significantly reduced when the TS method is used. This indicates that the average performance of the procedure with both LS and TS active is not significantly better than the average performance when TS alone is used as the improvement method. However, since the LS does not significantly add to the total computational time when TS is active, having both methods switched on at the same time give the best opportunity for finding high quality solutions.

In our next experiment we compare the performance of the SS procedure with TS as the local optimizer when the *threshold* parameter is set to 0.2 and the *Tabulter* is set to 0, 100 and 250. Figure 2 shows the behavior of the procedure in the four problems under consideration. Since the magnitude of the relative deviation from the best solutions is fairly different for each problem, we have depicted TSP and BRP in the same line graph and the LOP and SMS problem in a separate one using logarithmic scale.

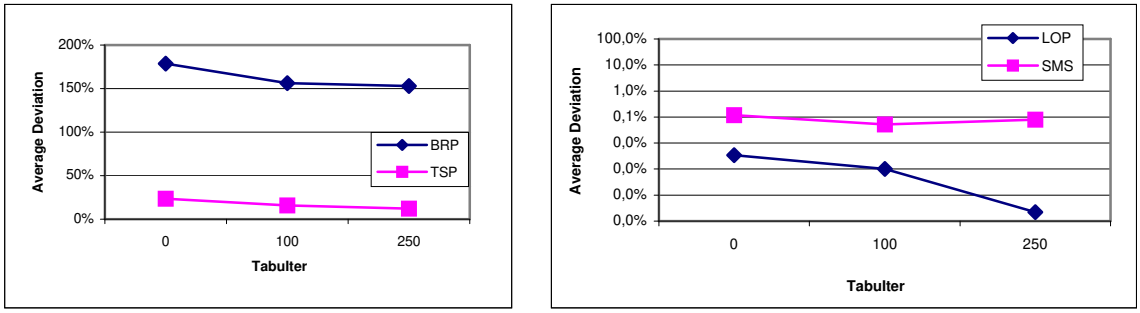

Figure 2. Summary of results with *TabuIter* = 0, 100, 250

As anticipated, the performance of the procedure improves as the number of TS iterations increases. Most of the gains, however, seem to be realized from *TabuIter* = 0 to *TabuIter* = 100. The additional computational effort of setting *TabuIter* = 250 does not seem justified in the set of problems that we tested.

In the introduction, we mentioned that there exist several commercial optimization packages that based on metaheuristic technology and that are capable of searching for solutions to permutation problems. In our next experiment, we compare the performance of our procedure with OptQuest and the Standard Evolutionary from Frontline Systems. The version that we use includes the tabu search improvement procedure with a threshold value of 0.2 and *TabuIter* = 100. Since the commercial optimizers include features such as graphical interfaces or other variable representations, they are significantly slower than our C++ implementation, which was specifically designed for permutation problems. To make a fair comparison, we have set a fixed number of objective function evaluations as a termination criterion for all procedures. The deviation from optimal or best solutions and the average CPU seconds for each procedure is reported in Table 6.

	Deviation from best			CPU seconds		
	OptQuest	Frontline	SS/TS	OptQuest	Frontline	SS/TS
BRP	252.2%	264.6%	172.4%	952	952	43
LOP	8.5%	16.1%	0.0%	301	300	25
TSP	311.4%	8.4%	5.7%	5,772	5,628	23
SMS	5.5%	0.8%	0.1%	237	300	5

Table 6. Comparison with commercial software

This table shows that the proposed scatter search with a tabu search improvement method yields higher quality solutions on the average when compared to two commercially available software packages. We include the execution time to show the advantage of using a specialized code that does not include additional costly routines, such as those associated with graphical output or databases to store all visited solutions. The main goal of our experiment was to compare the quality of the solutions and verify that our procedure represents a contribution to the solution of permutation problems with context-independent solvers.

6. Conclusions

We have developed a heuristic procedure based on the scatter search methodology for a class of combinatorial problems whose solutions can be represented as permutations. Our method uses a systematic diversification generator and 10 combination methods. We also tested two improvement methods: one based on a simple hill-climbing procedure and the other on a short-term memory tabu search. The method is context-independent in the sense that it treats the objective function evaluation as a black box. To allow for the use of key search strategies, the method requires that the problems be classified as either “absolute” or “relative” in terms of

the relevant factor for positioning elements in the permutation. We don't see this as a limiting feature of our procedure, since the speed of execution allows for experimenting with both settings before deciding whether two solve an instance or set of instances as A-permutation or R-permutation problems.

The performance of the procedure has been assessed using 157 instances of four different permutations problems. Solutions obtained with our method have been compared with the best-known solutions to each problem. The procedure has been shown competitive with methods specifically designed for problems the LOP and SMS problem. The method provides reasonable results although significantly inferior to those obtained with custom methods for the TSP. Finally, the method has been shown inadequate for the BRP due to the min-max nature of the objective function calculation associated with this class of problems. The method was shown superior to comparable procedures that are commercially available. Our experimentation allows us to conclude that context-independent methods can be useful in the context of permutation problems when the associated objective function is capable of discriminating among solutions in a given neighborhood.

References

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 (2000) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," Working paper, University of Colorado at Boulder.

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.

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

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.

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*, no. 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 (1998) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," to appear in *Computers and Operations Research*.

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. (2000) "Reducing the Bandwidth of a Sparse Matrix with Tabu Search", to appear in European Journal of Operational Research.

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.