SCATTER SEARCH WITH MULTIPLE IMPROVEMENT METHODS FOR THE LINEAR ORDERING PROBLEM

Héctor Joaquín Fraire Huacuja[1], Guadalupe Castilla Valdez[1], Rodolfo A. Pazos Rangel[1], Javier González Barbosa[1], Laura Cruz Reyes[1], Juan Martín Carpio Valadez[2], Héctor José Puga Soberanes[2], David Terán Villanueva[2]

[1]Instituto Tecnológico de Ciudad Madero, Av. 1o. de Mayo and Sor Juana I. de la Cruz S/N C.P. 89440, Cd. Madero Tamaulipas, México. E-mail: hfraire@prodigy.net.mx,{ jjgonzalezbarbosa, r_pazos_r, lauracruzreyes, gpe_cas}@yahoo.com.mx

[2]Instituto Tecnológico de León, Av. Tecnológico S/N Fracc. Ind. Julián de Obregón. C.P. 37290 León Guanajuato, México. E-mail: jmcarpio61@hotmail.com, {pugahector, david_teran01}@yahoo.com.mx

ABSTRACT

In this work, the Linear Ordering Problem (LOP) is approached. This is an NP-hard problem which has been solved with different metaheuristic algorithms. Particularly, it has been solved with a Scatter Search algorithm that applies the traditional approach which incorporates a single improvement method. In this paper, we propose a Scatter Search algorithm which uses multiple improvement methods to achieve a better balance of intensification and diversification. To validate our approach, a statistically-supported experimental study of its performance was carried out using the most challenging standard instances. The overall performance of the proposed Scatter Search algorithm was compared with the state-of-the-art algorithm solution for LOP. The experimental evidence shows that our algorithm outperforms the best algorithm solution for LOP, improving 2.89% the number of best-known solutions obtained, and 71% the average percentage error. It is worth noticing that it obtains 53 new best-known solutions for the instances used. We claim that the combination of multiple improvement methods (local searches) can be applied to improve the balance between intensification and diversification in other metaheuristics to solve LOP and problems in other domain.

Keywords: *Metaheuristics, Scatter Search, Linear Ordering Problem, Local Search, Balancing of intensification and diversification.*

1.0 INTRODUCTION

The problem addressed in this work has important applications in areas such as scheduling, social sciences, electronics, archeology, and particularly in economy. In this context, the input-output model developed by Wassily Leontief [1] is used to represent the interactions of economic sectors in a country. The problem of the triangulation of an input-output table consists of finding a simultaneous permutation of its columns and rows, such that the sum of the values above the main diagonal is maximized. This problem is equivalent to the linear ordering problem (LOP) which is defined as follows:

Given a matrix C of weights of size $n \times n$, the problem consists of finding a permutation P of columns (and rows) such that $C_{LOP}(p) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{p_{(i)}\ p_{(j)}}$ is maximized. As we can see, the objective value of permutation P is the sum of the weights above the main diagonal of C. Permutation P provides the ordering of rows and columns and p_i is the index of column (and row) i [2].

This problem has been solved with a Scatter Search algorithm that applies the traditional structure, incorporating a single improvement method. In this paper, we propose a Scatter Search algorithm which uses multiple improvement methods to achieve a better balance of intensification and diversification.

2.0 RELATED WORK

LOP is NP-hard [3, 4], and several metaheuristic algorithms have been proposed to solve it. In this section, the most recent and relevant metaheuristic algorithm solutions for LOP are described.

A Scatter Search solution for LOP was proposed by Campos et al. [5], following the basic template proposed by Glover [6]. It includes five processes: 1) *Diversification Generator Method*, 2) *Improvement Method*, 3) *Reference Set Update Method*, 4) *Subset Generation Method* and 5) *Combination Method*. They evaluated ten strategies for the *Diversification Generator Method*: seven based on GRASP constructions, one based on a random generator, one

based on a diversified strategy proposed by Glover, and one method that uses a memory-based frequency, which maintains a record of the number of times that an element *i* occupies position *j* in the permutation, and penalizes the attractiveness of the element with respect to the position for subsequent constructions. The evaluation of attractiveness is given by the greedy function proposed by Becker [7]. The best performance was achieved by the method that uses a memory-based frequency. The *Improvement Method* is implemented using a single local search. The *Updating Method* consists of maintaining in the reference set, *b*, the best solutions, where *b* is a constant search parameter. The *Subset Generation Method* consists of generating different subsets of the reference set, which will be used by the combination method. Subsets with several sizes were generated: two-element subsets, three-element subsets which were obtained by including the best solution that is not in the two-element subsets, four-element subsets which are obtained from three-element subsets by incorporating the best solution that is not in the subset, and subsets that contain the best *i* elements, where $i = 5, ..., b$. In the work reported by Campos et. al. [5], the criterion of best solution is expanded to include in the *Reference Set*, solutions from *P* with the largest index of similarity, and the size of the *Reference Set* is maintained unchanged. The *Combination Method* applies a *min-max* construction based on voting.

A LOP solution based on the Tabu Search metaheuristic is proposed by Laguna et al. [2]. This solution includes an intensification phase using short-term memory based on a tabu criterion, a diversification process implemented by long-term memory based on frequency record, and an additional intensification process applying path relinking based on elite solutions. Two insertion neighborhoods and two selection strategies (*first* and *best*) were evaluated, the insertion neighborhood using consecutive swaps movements and the *first* selection criterion achieved the best results. In this evaluation, the *first* selection criterion achieved the best results. A first local search was used to explore the insertion neighborhood, and when this search ends, a *best* local search was applied to the current solution.

A Memetic algorithm is proposed by Schiavinotto and Stutzle [8]. They studied three neighborhoods: interchange, insertion, and another one based on the Chanas and Kobylanski heuristic [9]. It is noteworthy that the insertion neighborhood was implemented applying an exploration strategy based on the Dynasearch method proposed by Congram [10], which reduces the cost to evaluate the neighborhood from $O(n^3)$ to $O(n^2)$. Three different methods of local search were evaluated: two implemented by insertion movements, from which one uses a selection rule between the *best* and *first* criterion (LS_f), another uses a first random criterion, and the third local search is based on the Chanas and Kobylanski (LS_{CK}) algorithm. An iterative local search metaheuristic was implemented. This starts from a random initial solution, which is improved applying local search LS_f , followed by an iterative process that includes perturbation using an interchange movement and a local optimization. Finally, a memetic algorithm, that uses a single local search LS_f, was implemented. This algorithm currently is considered the best solution for LOP in the state of the art [11].

In [11], Martí and Reinelt report an assessment of the ten most relevant metaheuristics for LOP using a standard benchmark. They use two sets of standard instances, the OPT-I set contains instances whose optimal value is known, and the UB-I set, which comprises those instances for which only the best-known value is available. For each instance, the algorithms were executed for 10 and 600 CPU seconds. The experimental results show that for both time limits, the best solution is obtained by the Memetic algorithm, followed by the Tabu Search algorithm, while Scatter Search obtains the fifth position.

As we can see, the LOP solution based on Tabu Search is the only metaheuristic that uses multiple local searches. In this work, we propose to improve the Scatter Search performance using a modified structure that incorporates multiple local searches. Our main idea consists of achieving a better balance of intensification and diversification of the metaheuristic, using local search algorithms with different levels of intensification.

3.0 THEORETICAL FRAMEWORK

3.1 The Insertion Neighborhood

Given a permutation $P = (p_1, p_2, p_3, \ldots, p_n)$, an insertion move InsertMov(p_j, i), consists of: extracting from permutation *P* the element in position *j*, moving the items that are between position *j* and position *i* one position towards position *j*, and inserting the extracted element in position *i* [2]. An insertion movement produces the permutation *P*' given by

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

The insertion neighborhood of a permutation P is the set of all permutations that are obtained by applying to P an insertion movement. The cost of an insertion movement is the difference of the objective values of the permutation generated with the movement and that of the original permutation. It is given by the following expression:

$$Cost\big(InsertMov(\,p_j, i)\big) = C_{LOP}(P') - \,C_{LOP}(P)$$

Using the objective function definition, this cost can be expressed as:

$$Cost(InsertMov(\,p_j, i)) = \begin{cases} \sum_{k=i}^{j-1} \quad (e_{p_j p_k} - e_{p_k p_j}) & \text{for } i < j\,, \\ \sum_{k=j+1}^{i} (e_{p_k p_j} - e_{p_j p_k}) & \text{for } i > j\,. \end{cases}$$

In this expression, $e_{p_j p_k} - e_{p_k p_j}$ corresponds to the difference in the objective function value caused by the change of relative position of the elements in positions j and k, within the range of the insertion movement.

3.2 The Consecutive Insertion Movement

A consecutive insertion movement consists of the insertion of the element at position j to positions j+1, or j–1 [8]. Given permutations P and P' = InsertMov$\big(p_j, j \pm 1\big)$, then we have that:

$$P' = \begin{cases} p_1, .., p_j, p_i, p_{j+1}, \ldots, \, p_m & \text{for} \;\; i = j - 1 \\ p_1, .., p_{j-1}, p_i, p_j, \ldots, \, p_m & \text{for} \;\; i = j + 1 \end{cases}$$

The cost of a consecutive insertion movement is given by:

$$Cost(InsertMov(p_j, i)) = \begin{cases} e_{p_{j+1} p_j} - \, e_{p_j p_{j+1}} & \text{for } i = j + 1 \\ e_{p_j p_{j-1}} - \, e_{p_{j-1} p_j} & \text{for } i = j - 1 \end{cases}$$

Now we show how a general insertion movement can be carried out by a series of consecutive insertion movements.

Let P = (1, 2, 3, 4, 5, 6) and P' = InsertMov$(\,p_2, 5) = (1, 3, 4, 5, 2, 6)$, then the cost of the movement is given by:

$$\text{Cost(InsertMov}(\,p_2, 5)) = \sum_{k=3}^{5} \left(e_{p_k p_j} - e_{p_j p_k}\right) = \left(e_{p_3 p_2} - e_{p_2 p_3}\right) + \left(e_{p_4 p_2} - e_{p_2 p_4}\right) + \, (e_{p_5 p_2} - e_{p_2 p_5})$$

By applying consecutive insertion movements to permutation P = (1, 2, 3, 4, 5, 6) to obtain P'= InsertMov$(\,p_2, 5) = (1,3, 4, 5,2,6)$, we have that:

$P^1 = \text{InsertMov}(\,p_2, 3) = (1, 3, 2, 4, 5, 6)$

$Cost(InsertMov(\,p_2, 3)) = \, (e_{p_3 p_2} - e_{p_2 p_3})$

$P^2 = \text{InsertMov}(\,p_3^1, 4) = (1, 3, 4, 2, 5, 6)$; p_3^1 is the element in the position 3 in permutationP^1,

$Cost(InsertMov(\,p_3^1, 4)) = \left(e_{p_4^1 p_3^1} - e_{p_3^1 p_4^1}\right) = \, \left(e_{p_4 p_2} - e_{p_2 p_4}\right)$

$P^3 = \text{InsertMov}(\,p_4^2, 5) = (1, 3, 4, 5, 2, 6)$,

$Cost(InsertMov(\,p_4^2, 5)) \, = \left(e_{p_5^2 p_4^2} - e_{p_4^2 p_5^2}\right) = (e_{p_5 p_2} - e_{p_2 p_5})$

Let S be the sum of the costs corresponding to the consecutive movements that are required to transform P into P' = P^3, it is given by:

$S = Cost(InsertMov(p_2, 3)) + \, Cost(InsertMov(p_3^1, 4)) + Cost(InsertMov(\,p_4^2, 5))$
$S = \, \left(e_{p_3 p_2} - e_{p_2 p_3}\right) + \left(e_{p_4 p_2} - e_{p_2 p_4}\right) + \, (e_{p_5 p_2} - e_{p_2 p_5})$

$$S = \sum_{k=i+1}^{j} \left(e_{p_k p_j} - e_{p_j p_k}\right)$$

$$S = Cost(InsertMov(p_2, 5))$$

3.3 The Cost of the Insertion Neighborhood Exploration

As we can see in the previous section, the cost of an insertion movement is equal to the sum of the costs of consecutive insertion movements, and it can be calculated before the movement is performed. This property is very useful to efficiently determine the position where a given element must be inserted to produce an improvement in the objective value. For a permutation *P* of size *n*, the worst case occurs when we are looking for the position where to insert the first element. In Table 1, we can observe the process to determine the cost of inserting p_i in positions i = 2, 3, 4, …, *n*, without using consecutive movements. The columns contain the evaluated position, the insertion cost and the number of operations (subtractions) that have to be carried out respectively. As we can see, the computational complexity of the exploration process is $O(n^2)$.

Table 1. Cost of exploring the insertion neighborhood without using consecutive insertions.

Evaluated Position (*i*)	$\boldsymbol{Cost(InsertMov(p_1, i))}$	**# Subtraction Operations**
2	$(e_{p_2p_1} - e_{p_1p_2})$	1
3	$(e_{p_2p_1} - e_{p_1p_2}) + (e_{p_3p_1} - e_{p_1p_3})$	2
4	$(e_{p_2p_1} - e_{p_1p_2}) + (e_{p_3p_1} - e_{p_1p_3}) + (e_{p_4p_1} - e_{p_1p_4})$	3
….	…..	….
N	$(e_{p_2p_1} - e_{p_1p_2}) + (e_{p_3p_1} - e_{p_1p_3}) + (e_{p_4p_1} - e_{p_1p_4}) + (e_{p_np_1} - e_{p_1p_n})$	*n* -1
	Total	$\boldsymbol{\frac{1}{2}(n^2 - n)}$

On the other hand, calculating the same cost using consecutive insertions requires only one subtraction operation for each position. The number of operations performed in this case is shown in Table 2.

Table 2. Cost of exploring the insertion neighborhood using consecutive insertion movements.

Evaluated Position (*i*)	**Consecutive Insertion Cost**	**# Subtraction Operations**
2	$(e_{p_2p_1} - e_{p_1p_2})$	1
3	$(e_{p_3p_1} - e_{p_1p_3})$	1
4	$(e_{p_4p_1} - e_{p_1p_4})$	1
..	…..	..
n	$(e_{p_np_1} - e_{p_1p_n})$	1
	Total	$\boldsymbol{n - 1}$

As shown, the complexity of the exploration process is reduced from $O(n^2)$ to $O(n)$. In this work, the local searches used as improvement methods in the Scatter Search algorithm, apply consecutive insertion movements to explore the neighborhood of the current solution.

4.0 LOCAL SEARCH

As we have previously mentioned, we say that it is possible to achieve a better balance of intensification and diversification of the Scatter Search metaheuristic, using local search algorithms with different levels of intensification. The balance of intensification and diversification in the search algorithms depends on explicit structural elements, such as the exploration strategy and the criterion to select the neighbor to replace the current solution, or implicit elements such as the neighborhood structure and the problem representation. In this work, a set of local search algorithms with different intensification levels were implemented that try to reinsert the elements of

the current solution in new positions to improve its objective value. This process carried out a series of consecutive insertions, checking the positions from i+1 to n, and then from i–1 to 1. As was previously shown, this exploration process is performed in $O(n)$. The saving in time produced by this strategy is used for other processes of the algorithm that contribute to obtain the global optimum.

Five local searches that use an insertion neighborhood were implemented. LS_1 and LS_2, apply a selection rule which chooses the first neighbor that leads to an improvement, LS_2 includes a stagnation verification rule which ends the process after $n/2$ iterations without improvement. Fig. 1 and Fig. 2 show the algorithms for these local searches. LS_3 explores all the neighbors of the current solution and, it chooses one that produces the largest increase in the objective function. LS_4 performs an exhaustive search restarting the exploration of all the elements of the permutation whenever one improvement occurs during an iteration. The algorithms for these searches are shown in Fig. 3 and Fig. 4. Although LS_4 yields high quality solutions, it involves a higher computational cost than those of LS_1, LS_2, and LS_3 and could produce premature stagnation. Fig. 5 shows the random local search algorithm LS_5, which performs a random insertion movement for each element of the permutation. Clearly, this method is inefficient and is only used as a basis for comparison.

As we can see, the intensification level increases from LS_1 to LS_4, but also increases the time consumption. We believe that combining different local searches in the processes where an improvement is required in the Scatter Search algorithm, could avoid premature stagnation and improve the quality of the solutions. Summarizing, the combination of local searches with different intensification levels could contribute to improve the intensification and diversification balance of the Scatter Search metaheuristic.

```
Procedure LS1(P )
1.   Cost = Calculating_cost(P);
2.   without_improving = 0;
3.   for (i =1 to n)
4.       if(i >2) then Costmax= Cost(InsertMov(pi, i-1))
5.       else Costmax= Cost(InsertMov(pi, i+1))
6.       improving = 0; j = i-1;
7.       while (not improving and j ≥1)
8.           if (Cost(InsertMov(pi, j)) > Costmax) then
9.               Costmax = Cost(InsertMov(pi, j);
10.              jmax = j; improving =1; break;
11.          endif;
12.          j = j-1;
13.      endwhile;
14.      j = i+1;
15.      while (not improving and j ≤ n)
16.          if Cost(InsertMov (pi,j)) > Costmax) then
17.              Costmax = Cost(InsertMov(pi, j));
18.              jmax = j; improving =1;  break;
19.          end if;
20.          j= j+1;
21.      endwhile;
22.      if (improving )
23.          P' = Insert(pi, jmax);
24.          Cost = Cost + Costmax;
25.          P= P';
26.      endif;
27.  end_for
28.  return (P);
```

Fig. 1. LS_1 First Local Search Algorithm.

```
Procedure LS2(P )
1.   Cost = Calculating_cost(P);
2.   withoutimproving = 0;
3.   do
4.      i = getrandom(1, n);
5.      if(i >2) then Costmax= Cost(InsertMov(pi, i-1))
6.      else Costmax= Cost(InsertMov(pi, i+1))
7.      improving = 0; j = i-1;
8.      while (not improving and j ≥1)
9.         if (Cost(InsertMov(pi, j)) > Costmax) then
10.           Costmax = Cost(InsertMov(pi, j);
11.           jmax = j; improving =1; break;
12.        endif;
13.        j = j-1;
14.     endwhile;
15.     j = i+1;
16.     while (not improving and j ≤ n)
17.        if Cost(InsertMov (pi,j)) > Costmax) then
18.           Costmax = Cost(InsertMov(pi, j));
19.           jmax = j; improving =1;  break;
20.        end if;
21.        j= j+1;
22.     endwhile;
23.     if (improving )
24.        P' = Insert(pi, jmax);
25.        Cost = Cost + Costmax;
26.        withoutimproving = 0;
27.        P= P';
28.     endif;
29.     else
30.        ++withoutimproving ;
31.  while (whitoutimproving < (n/2))
32.  return (P);
```

Fig. 2. LS_2 First Local Search with stagnation detection Algorithm.

```
Procedure LS3(P )
1.  Cost = Calculating_cost(P)
2.  for(i =1  to n)
3.     if(i >2) then Costmax= Calculating_cost(InsertMov(pi, i-1))
4.     else Costmax= Calculating_cost(InsertMov(pi, i+1))
5.     j = i-1;
6.     improving = 0;
7.     while (not improving and j ≥1)
8.        if (Calculating_cost(InsertMov(pi, j)) > Costmax) then
9.           Costmax = Calculating_cost(InsertMov(pi, j);
10.          jmax = j; improving =1; break;
11.       endif;
12.       j = j-1;
13.    endwhile;
14.    j=i+1;
15.    while (not improving and j ≤ n)
16.       if Calculating_cost(InsertMov (pi,j)) > Costmax) then
17.          Costmax = Calculating_cost(InsertMov(pi, j));
18.          jmax = j; improving =1;  break;
19.       endif;
20.       j= j+1;
21.    endwhile;
22.    if (improving and Costmax > 0)
23.       P'= InsertMov(pi, jmax);
24.       Cost = Cost + Costmax;
25.       P = P';
26.    endif;
27. endfor
28. return (P);
```

Fig. 3. LS_3 Best Local Search Algorithm.

```
Procedure LS4 (P )
1.  Cost=Calculating_cost (P);
2.  do
3.   for (i= 1 to n)
4.       if(i>2) then Cost_max= Calculating_cost(InsertMov(p_i, i-1))
5.       else Cost_max= Calculating_cost (InsertMov(p_i, i+1))
6.       for(j=i-1 to 1)
7.           if (Calculating_cost(InsertMov(p_i, j)) > Cost_max) then
8.               Cost_max = Calculating_cost(InsertMov(p_i, j)) ;
9.               j_max = j;
10.          endif
11.      endfor
12.      for(j=i+1 to n)
13.          if (Calculating_cost(InsertMov(p_i, j)) > Cost_max) then
14.              Cost_max = Calculating_cost(InsertMov(p_i, j));
15.              j_max = j;
16.          endif
17.      endfor
18.      if(Cost_max > 0)
19.          Insert(p_i, j_max);
20.          Cost = Cost + Cost_max;
21.          P = P ';
22.          improving = 1;
23.      endif
24.  endfor
25. while (improving)
26. return (P )
```

Fig. 4. LS_4 Intensive Best Local Search Algorithm.

```
Procedure LS5(P )
1.  Cost = Calculating_cost(P);
2.  for(i =1 to n)
3.      do
4.          j = getrandom(1,n);
5.      while ( j = = i);
6.      Cost = Cost + Calculating_cost(InsertMov(p_i, j));
7.      P '= Insert(p_i, j);
8.      P = P ';
9.  endfor
10. return (P);
```

Fig. 5. LS_5 Random Local Search Algorithm.

5. 0 SCATTER SEARCH WITH MULTIPLE IMPROVEMENT METHODS

Scatter Search (SS) is an evolutionary method based on the classical methods of rules combination used to solve decision problems in the field of operations research. This metaheuristic developed by Fred Glover in the 70's, combines solutions of the Reference Set to create new improved solutions [12]. The main processes of the standard Scatter Search are described in the following lines and their structure can be reviewed in [13].

- *Diversification Generation Method*. Generate the set *U* of diverse solutions, from which the solutions to build the *Reference Set* (*RefSet)* will be extracted.

- *Improvemenet Method*. Typically, it is a local search method for improving the solutions of both the *RefSet* as well as those generated in the method of combination, before considering their inclusion in *RefSet*.

- *Generating and Updating Method*. Usually, the same criterion to initialize or to update the *RefSe*t is applied, which consist of selecting the best quality solutions from *U*, removing them from *U* and incorporating them into *RefSet*. The same is done to select the most diverse solutions from *U*, using some diversity metric.

- *Subset Generation Method*, this process consists of generating from *RefSet*, the subsets to which the combination method will be applied afterwards; the most common method for generating the subsets consists of forming all the possible solution pairs from *RefSet*.

- *Combination Method.* In this process, the solutions of the subsets generated in step 4 are combined. The most successful strategy for combining solutions is the weighted combination based on votes, which is suitable for most of the problems representations [14].

In this work, we propose a new approach to obtain a suitable balance of intensification and diversification in SS algorithms using different improvement methods throughout the entire global search process. Fig. 6 shows the Scatter Search structure used which includes four improving steps, and Fig. 7 shows the proposed Scatter Search algorithm (MI-SS). Most of the strategies in the MI-SS algorithm are taken from the Scatter Search standard structure. The processes in which the proposed strategies were incorporated are described in the following sections.

5.1 Diversification Method

The diversification method creates a set of |*U*| solutions, which will provide solutions to build the reference set (*RefSet*), |*U*|= 10 * |*RefSet*| = 10*10 =100. The solutions in *U* are randomly generated and ordered according to their objective value.

The process for generating and updating *RefSet* was implemented in the traditional way: selecting the best |*RefSet*|/2 solutions considering the quality, and the |*RefSet*|/2 most diverse solutions from *U*. The distance metric proposed by Pantrigo [15] is used to select the most diverse solutions. In this metric, the distance between two given solutions *R* and *S* is defined by:

$$d(R,S) = \min\left\{\sum_{i=1}^{n} | r_i - s_i |, \sum_{i=1}^{n} | r_i - s_{n-i+1} |\right\}$$

To select the most diverse solutions from *U*, we propose the following diversity indicator:

$$IDiv(R) = \left(\sum_{i=1}^{|\text{Re } fSet|} d(R,Q_i)\right) \Big/ n, \quad \text{where } R \in U \text{ and Ref } Set = \left\{Q_1, Q_2, ..., Q_{|\text{Re } fSet|}\right\}$$

For each candidate solution $R \in U$, *IDiv*(*R*) was calculated, and the solution in *U* with the largest *IDiv* value was included in *RefSet*.

If a stagnation condition in the average quality of the solutions in *RefSet* is detected, a diversification mechanism is triggered. It consists of updating *RefSet* by removing all its elements except the best.

5.2 Combination Method

The combination method implemented was inspired by the order based crossover [16]. It consists of combining two solutions randomly selected, followed by a random selection of a set of positions from the first solution (0.4 * instance size), then the values in the non-selected positions are copied directly to the corresponding positions of the new solution. In the next step, the values in the selected positions are ranked according to the order in the second solution, and they are copied to the vacant positions of the new solution. The current combination is recorded to avoid recombining the same solutions in the next iteration.

5.3 Improvement Method

Fig. 7 shows the MI-SS structure used in our algorithm proposal. It is worth noticing that this structure is different from the standard because it includes four places where different improvement methods can be applied (denoted by gray boxes). Using this structure, we assessed the algorithm performance with several combinations of the five local search algorithms described in the previous section: LS_1, LS_2, LS_3, LS_4, and LS_5.

6.0 EXPERIMENTAL RESULTS

The experimentation was carried out in two phases. In the first one, a set of improvement methods having different intensification levels at the four places of the scatter search were evaluated using the XLOLIB instances. In the second phase, the scatter search that incorporates the best combination of improvement methods was evaluated using a wide set of the hardest benchmark instances (whose description can be found in [11]). For both phases, we used ANSI C language for programming the solution algorithm, Visual Studio 6 for compiling, SPSS for the

statistical Wilcoxon test and a computer with an Intel XEON dual processor at 3.06 GHZ, 70 GB in hard disk and 4GB in RAM.

In the first experiment, the performance of the Scatter Search algorithm shown in Fig. 8 was evaluated. Different combinations of the five local search algorithms described in the previous section (LS_1, LS_2, LS_3, LS_4, and LS_5) were applied in the four improvement processes indicated. Table 3 shows the results for a subset of all the combinations tested, highlighting the combination that produces the best performance algorithm. In this table, the combination of local searches used in the algorithm is shown in the first column. The (LS_5, LS_5, LS_5, LS_5) combination indicates that LS_5 was applied in the four places of improvement, while the combination (LS_4, LS_3, LS_1, LS_4) indicates that LS_4 was applied in the first and last places, LS_3 in the second, and LS_1 in the third place. The second column contains the average deviation in percentage from the best-known solutions reached by the algorithm on the test instances. The last column shows the percentage deviation with respect to the objective function value of the reference combination (LS_5, LS_5, LS_5, LS_5). As we can observe, the combination (LS_4, LS_3, LS_4, LS_3) had the lowest error with respect to the best-known solutions of the test instances, and the largest improvement percentage with respect to the reference configuration; therefore the Scatter Search algorithm configured with this combination of local searches was used in the second experiment.

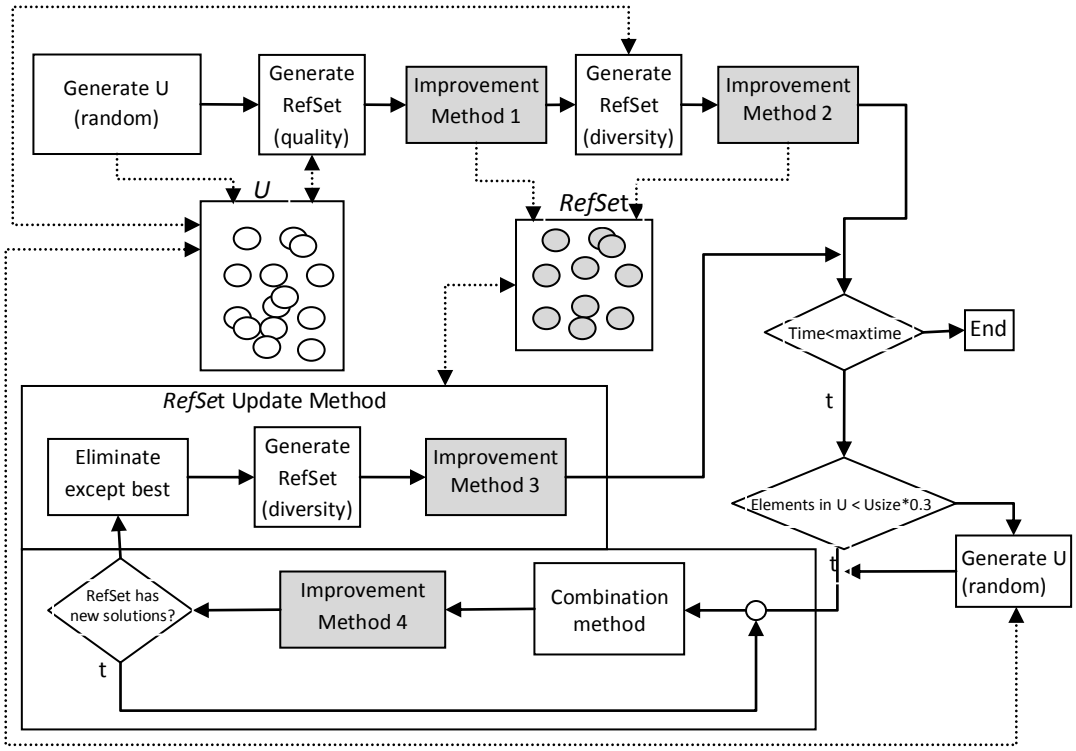

Fig. 6. Structure of the Scatter Search with Multiple Methods of Improvement (MI-SS).

```
Procedure MI-SS Algorithm
    Randomly_generate_P ( )
    LS4 (Generating_Ref_Set_quality( )) ----------( 1 )
    LS3 (Generating_Ref_Set_Diversity( ))----------( 2 )
    Update_Best_solutions( )
    while (not stop criterion)
        do
            if(stagnation condition)
                Randomly_generate_P ( )
            do
                LS4 (Combination_Method( )) ----------( 3 )
            while(stop criterion)
            Delete_except_best( )
            LS3 (Generating_Ref_Set_Diversity( )) -----( 4 )
        while(stop criterion)
    end_while
end MI-SS Algorithm
```

Fig. 7. Scatter Search with Multiple Methods of Improvement (MI-SS) Algorithm.

Table 3. Evaluation of different combinations of improvement methods in the MI-SS algorithm.

Combination of Improvement Methods	Average Deviation from Best Known (%)	Percentage of Improvement
LS_5, LS_5, LS_5, LS_5	6.36759175	0
LS_1, LS_1, LS_1, LS_1	6.84919255	-7.56331151
LS_2, LS_2, LS_2, LS_2	8.86974463	-39.295121
LS_3, LS_3, LS_3, LS_3	0.61858853	90.2853613
LS_4, LS_4, LS_4, LS_4	0.30895276	95.1480438
LS_3, LS_1, LS_3, LS_1	0.618427378	90.2878922
LS_3, LS_3, LS_1, LS_1	5.29794016	16.7983695
LS_1, LS_1, LS_3, LS_3	0.60104226	90.5609172
LS_4, LS_3, LS_3, LS_3	0.1869835	97.0635131
LS_4, LS_3, LS_4, LS_4	0.18266885	97.1312726
LS_3, LS_3, LS_4, LS_4	0.16042490	97.4806032
LS_4, LS_3, LS_4, LS_3	**0.12519809**	**98.0338235**
LS_4, LS_1, LS_4, LS_3	0.16419852	97.4213403

The second experiment consists of assessing the performance of the MI-SS algorithm with respect to the best algorithms of the state of the art. For this experiment, the set UB-I of the most challenging standard instances was used. This set includes 255 standard instances: 100 RandomAI, 50 RandomAII, 20 RandomB, 78 XLOLIB, and 7 Special [11]. For each instance a single run was carried out with a seed of 1471 and time limits of 10 and 600 CPU seconds. Tables 4 and 5 show the results for the proposed algorithm and the best algorithms of the state of the art reported in [11]. In these tables the average percentage error with respect to the best-known solutions, and the number of best-known solutions found is shown for each group of instances solved with the following algorithms: Scatter Search (SS), Tabu Search (TS), Memetic (MM) and Scatter Search with multiple methods of improvement (MI-SS). At the bottom of both tables the overall performance appears highlighted, which includes the average percentage error and the number of best known solutions found. In Tables 4 and 5, the average percentage error is calculated as follows:

$$\%Error(Avg) = \frac{1}{n}\sum_{i=1}^{n} 100\frac{(Z_i^{best} - Z_i^*)}{Z_i^{best}}$$

where Z_i^{best} represents the value of the objective function for the best-known solution for instance *i*, Z_i^* represents the value of the objective function for the solution found by the MI-SS algorithm for the same instance, and *n* is the number of instances.

Regarding overall performance, for both time limits, MI-SS clearly outperforms the best Scatter Search algorithm solution for LOP (SS) [5]. For the 10 seconds test, the average percentage error decreases from 0.272 to 0.11. The number of best-known solutions found increases from 15 to 89. For the 600 seconds test, the average percentage error decreases from 0.256 to 0.004. The number of best-known solutions found increases from 20 to 213. For all the instances sets, MI-SS improves both indicators (average percentage error and number of best-known solutions). Also, MI-SS clearly outperforms the best Tabu Search algorithm solution for LOP (TS) [2].

Table 4. Experimental results for UB-I instances (10 sec.)

Instances	Performance Indicators	TS [2]	MM [8]	SS [5]	MI-SS
RandA1	% Error (Avg)	0.13	0.05	0.27	0.14
	# Best	5	32	1	26
RandA2	% Error (Avg)	0	0	0.02	0
	# Best	3	39	0	39
RandB	% Error (Avg)	0	0	0.04	0
	# Best	20	20	11	20
XLOLIB	% Error (Avg)	0.63	0.13	0.69	0.13
	# Best	0	2	0	1
Spec	% Error (Avg)	0.46	0.06	0.34	0.07
	# Best	3	3	3	3
Average % Error		**0.4**	**0.08**	**0.272**	**0.11**
Total # Best		**31**	**96**	**15**	**89**

On the other hand, in the experiment with a time limit of 10 seconds, MI-SS has a lower overall performance than that of the state-of-the-art solution for LOP (MM); however in the experiment with 600 seconds of time limit, MI-SS clearly outperforms the MM algorithm. The average percentage error decreases from 0.014 to 0.004, which

constitutes a 71 % of improvement. The number of best-known solutions found increases from 207 to 213, indicating an improvement of 2.89 %. It is remarkable that in this experiment MI-SS obtains 53 new best-known solutions. An update of the best-known solutions for the UB-I instances set reported in [11] is presented in Table 8. The overall performance improvement of the MI-SS algorithm could be explained as a consequence of a better balance reached between intensification and diversification with the Scatter Search structure used in the MI-SS algorithm. The improved balance incorporated in the MI-SS algorithm seems to increase its performance.

As MM and MI-SS are randomized algorithms, a Wilcoxon non parametric hypothesis test was applied to determine if the observed differences, in the average error percentages, are statistically significant [17]. For this test the RandA1 and XLOLIB instances were distributed in groups of instances with the same size: RandA1 (100), RandA1 (150), RandA1 (200), RandA1 (500), XLOLIB (150), and XLOLIB (250). Tables 6 and 7 show the Wilcoxon test results using a significance level $\alpha = 0.05$. The tables contain the number of instances in each set (N), the names of the sets, the number of ranks after removing ties (n), the values for R^+ and R^- corresponding to the sum of positive and negative differences between the percentage errors found for each solved instance with MM and MI-SS, and the reference range taken from a Wilcoxon table (VC). Finally, the winner algorithm is shown in the last column.

In these tables, if R^- is larger than R^+, the algorithm with the best performance is MI-SS; otherwise the Memetic algorithm (MM) is the winner. When the highest value of R^+ and R^- is inside the VC range, the null hypothesis (both algorithms have the same performance) is accepted, and we can establish that there is not a significant difference in their performance; otherwise, the null hypothesis is rejected and the difference is statistically significant. As we can see in last column of Tables 6 and 7, the MI-SS has virtually the same performance as the Memetic algorithm.

Table 5. Experimental results for UB-I instances (600 sec.)

Instances	Performance Indicators	TS [2]	MM [8]	SS [5]	MI-SS
RandA1	% Error (Avg)	0.10	0.006	0.19	0.002
	# Best	19	74	2	84
RandA2	% Error (Avg)	0	0	0.01	0
	# Best	3	50	2	50
RandB	% Error (Avg)	0	0	0.02	0
	# Best	20	20	13	20
XLOLIB	% Error (Avg)	0.4	0.008	0.82	0.012
	# Best	0	59	0	52
Spec	% Error (Avg)	0.26	0.026	0.24	0
	# Best	3	4	3	7
Average % Error		**0.15**	**0.014**	**0.256**	**0.004**
Total # Best		**45**	**207**	**20**	**213**

Table 6. Wilcoxon test results for 10 seconds of time limit (Memetic algorithm versus MI-SS)

Execution time limit of 10 seconds (MM / MI-SS)						
N	**Instances**	***n***	R^+	R^-	***VC***	**Best Performance**
25	RandAI (100)	6	**21**	0	0-21	MM, MI-SS
25	RandAI(150)	17	**109**	44	34-119	MM, MI-SS
25	RandAI(200)	25	**194**	131	89-336	MM, MI-SS
25	RandAI(500)	25	**325**	0	89-236	MM
39	XLOLIB(150)	39	347	**433**	249-531	MM, MI-SS
39	XLOLIB(250)	39	**467**	313	249-531	MM, MI-SS

Table 7. Wilcoxon test results for 600 seconds of time limit (Memetic algorithm versus MI-SS)

Execution time limit of 600 seconds (MM / MI-SS)						
N	**Instances**	***n***	R^+	R^-	***VC***	**Best Performance**
25	RandAI (100)	1	1	0	NA	NA
25	RandAI(150)	2	2	1	NA	NA
25	RandAI(200)	14	51	**54**	21-84	MM, MI-SS
25	RandAI(500)	25	57	**268**	89-236	MI-SS
39	XLOLIB(150)	37	**591**	112	221-482	MM
39	XLOLIB(250)	39	**418**	362	249-531	MM, MI-SS

7.0 CONCLUSIONS AND FUTURE WORK

In this paper, the linear ordering problem is approached. This is an NP-hard relevant problem that has been solved using several metaheuristics. We propose to improve the best Scatter Search solution of the state of the art for LOP, using a modified structure that incorporates multiple local searches. The core idea of our approach consists of achieving a better balance of intensification and diversification of the metaheuristic, using local search algorithms with different levels of intensification.

An experimental study was carried out using the most challenging sets of instances. The performance of the proposed Scatter Search algorithm (MI-SS) and the best of the state-of-the-art algorithms for LOP (SS, TS, and MM) were compared. The experimental results show that MI-SS clearly outperforms the Scatter Search and Tabu Search algorithms.

Currently, the Memetic algorithm is considered the best of the state-of-the-art algorithm solution for LOP. It is worth noticing that, regarding overall performance, the proposed Scatter Search algorithm outperforms the Memetic algorithm when a time limit of 600 seconds is used. It achieves a reduction of 71 % in the average percentage error, an increase of 2.89% in the number of best-known-solutions found, and finds 53 new best-known solutions. A Wilcoxon statistical hypothesis test shows that MI-SS has virtually the same performance as the Memetic algorithm. This performance improvement is due to a larger diversification capacity that MI-SS seems to have as a result of the combination of multiple local searches incorporated into MI-SS.

We are currently applying the proposed approach to improve the balance of intensification and diversification of a GRASP solution for LOP.

Table 8. New *best-known* solutions obtained by the MI-SS algorithm in 600 seconds

	Instances	New Best
	RandA1 (100)	
1	N-t1d150.04	234510
	RandA1(200)	
2	N-t1d200.01	410992
3	N-t1d200.04	410105
4	N-t1d200.08	408883
5	N-t1d200.13	409270
6	N-t1d200.15	409073
7	N-t1d200.18	407728
8	N-t1d200.25	406476
	RandA1(500)	
9	N-t1d500.01	2404308
10	N-t1d500.04	2414801
11	N-t1d500.06	2400280
12	N-t1d500.08	2414152
13	N-t1d500.09	2407035
14	N-t1d500.10	2406593
15	N-t1d500.11	2416484
16	N-t1d500.12	2403299
17	N-t1d500.14	2410932
18	N-t1d500.15	2412056
19	N-t1d500.16	2416692
20	N-t1d500.17	2401928
21	N-t1d500.19	2404662
22	N-t1d500.20	2415076
23	N-t1d500.22	2408392
24	N-t1d500.23	2408978
25	N-t1d500.24	2403497
26	N-t1d500.25	2406618

	Instances	New Best
	XLOLIB (150)	
1	N-stabu2_150	4327571
2	N-t70d11xn_150	5825692
3	N-t75d11xx_150	9643994
4	N-tiw56r67_150	2057074
	XLOLIB (250)	
5	N-be75tot_250	30984685
6	N-stabu2_250	11509729
7	N-stabu3_250	11906623
8	N-t59d11xx_250	3842366
9	N-t59f11xx_250	3994038
10	N-t65f11xx_250	8410169
11	N-t70b11xx_250	25405187
12	N-t70d11xx_250	16043521
13	N-t70f11xx_250	13589177
14	N-t70l11xx_250	1113154
15	N-t75d11xx_250	25038262
16	N-t75k11xx_250	4094205
17	N-t75n11xx_250	4525472
18	N-tiw56n54_250	2099294
19	N-tiw56n62_250	4143436
20	N-tiw56n67_250	6326150
21	N-tiw56n72_250	11151289
22	N-tiw56r54_250	2387755
23	N-tiw56r67_250	5292693
24	N-tiw56r72_250	7452411
	Special	
1	N-atp134	1797
2	N-atp163	2075
3	N-atp452	2711

ACKNOWLEDGEMENTS

We would like to thank the following agencies of the Mexican Government: Consejo Nacional de Ciencia y Tecnología, Consejo Tamaulipeco de Ciencia y Tecnología and Dirección General de Educación Superior Tecnológica for the financial support received. We would also like to thank to Manuel Laguna and Abraham Duarte for their valuable technical support and to the anonymous reviewers for their valuable observations.

REFERENCES

[1] Leontief, Wassily W.W., *Input-Output Economics*, New York, Oxford University Press, ed. 2, 1986.

[2] Laguna, M., Martí, R., Campos, V. "Intensification and diversification with elite tabu search solutions for the linear ordering problem", *Computers and Operations Research*, Vol. 26, No. 12, 1999, pp. 1217-1230.

[3] Karp, R., "Reducibility among combinatorial problems", in *Complexity of Computer Computation*. Plenum Press, 1972, pp. 85-104.

[4] Garey, M. R., Johnson, D.S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", ed. W. H. Freeman and Co, 1975.

[5] Campos, V., Laguna, M. and Marti, R. "An experimental evaluation of a scatter search for the linear ordering problem", *Journal of Global Optimization*, Vol. 21, 2001, pp. 397–414.

[6] Glover F, "A Template for Scatter Search and Path Relinking", *Artificial Evolution*, 1997, pp. 1-51.

[7] Becker, O. "Das Helmstädtersche Reihenfolgeproblem — die Effizienz verschiedener Näherungsverfahren" in *Computer uses in the Social Sciences*, *Berichteiner Working Conference*, Wien, January 1967.

[8] Schiavinotto T. and Stutzle T. "Search Space Analysis of the Linear Ordering Problem"; *Applications of Evolutionary Computing: Lecture Notes in Computer Science*, Vol. 2611/2003, 2003, pp. 197-204.

[9] Chanas, S. and Kobylánski, P., "A New Heuristic Algorithm Solving the Linear Ordering Problem", *Computational Optimization and Applications*, Vol. 6, 1996, pp. 191-205.

[10] Congram Richard K., "Polynomially Searchable Exponential Neighborhoods for Sequencing Problems in Combinatorial Optimization", *Faculty of Mathematical Studies*, 2000.

[11] Martí R., Reinelt G. "The Linear Ordering Problem. Exact and heuristic methods in combinatorial optimization", *Springer, Heidelberg*, ISBN: 978-3-642-16728-7, 2011.

[12] Laguna M. and Marti R., "Scatter Search: Methodology and implementations in C", *Kluwer Academic Publisher*, Boston, 2003.

[13] Resende M., Ribeiro C., Glover F. and Martí R., "Scatter Search and Path Relinking: Fundamentals, advances and applications", *Handbook of Metaheuristics (2nd Edition) Michel Gendreau and Jean-Yves Potvin*, ed.Springer, 2009.

[14] Martí, R., Laguna, M., "Scater Search, Diseño Básico y Estrategias Avanzadas", *Revista Iberoaméricana de Inteligencia Artificial*, Vol. 2 No. 4, 2003, pp. 123-130.

[15] Pantrigo, J.J.; Martí, R.; Duarte A.; Pardo, E.G., "Scatter Search for the Cutwidth Minimization Problem", *Annals of Operations Research. Submitted* in 2010.

[16] Michalewicz, Z., Fogel, D.B. "How to solve it: Modern Heuristics", ed. Springer Verlag, 2000.

[17] García S., Molina D., Lozano M., and Herrera F., "An experimental study about the use of non-parametric tests for analysing the behavior of evolutionary algorithms in optimization problems", in *Proceeding of the III Congreso Español de Metaheuristicas, Algoritmos Evolutivos y Bioinspirados, MAEB 2007*, España, 2007.

BIOGRAPHY

Héctor J. Fraire H. received the B.S. Math and M.I.S degrees from the Universidad Autónoma de Nuevo León, México in 1976 and 1988, and the PhD degree in computer science from the Centro Nacional de Investigación y Desarrollo Tecnológico in 2005. Currently he is full professor at the Instituto Tecnológico de Cd. Madero, Mexico. His research interests include heuristic optimization and machine learning.

Guadalupe Castilla V. was born in Monterrey N.L. Mexico in 1959. She received the M.S. degree from the Instituto Tecnológico de León, Mexico. Currently she is a PhD in computer sciences student at the Instituto Tecnológico de Tijuana, Mexico. Her research interests include algorithmics and heuristic optimization.

Rodolfo A. Pazos R. was born in Tampico, Mexico in 1951. He received the B.S.E.E. and M.S.E.E. degrees from the Instituto Politécnico Nacional, Mexico in 1976 and 1978, and the PhD degree in computer science from U.C.L.A. in 1983. Currently he is full professor at the Instituto Tecnólogico de Cd. Madero, Mexico. His research interests include algorithmics and natural language processing.

Juan J. González B. received the PhD degree in computer science from the Centro Nacional de Investigación y Desarrollo Tecnológico in 2005. His scientific interests include metaheuristic optimization and natural language processing.

Laura Cruz-Reyes was born in Mexico in 1959. She received the PhD degree in computer science from the Centro Nacional de Investigación y Desarrollo Tecnológico, Mexico in 2004. She is a professor at Instituto Tecnológico de Cd. Madero, Mexico. Her research interests include optimization techniques, complex networks, autonomous agents and algorithm performance explanation.

Juan M. Carpio V. received his BS degree in Mathematics from the Universidad Autónoma de Nuevo León in 1985. He obtained his Master degree in Optics Sciences in 1990 and his PhD in Optics Sciences in 1995 from the Universidad de Guanajuato. He has publications in international journals with rigorous refereeing and works in national and international conferences. His research interests include heuristic optimization, optical metrology and mathematical modeling.

Héctor J. Puga S. was born in Ecatepec, Mexico in 1960. He received his B.S. degree in Physics and Mathematics degree from the Instituto Politécnico Nacional, Mexico in 1993. He obtained his Master and PhD degrees in Optics from the Universidad de Guanajuato in 1995 and 2002. Since 2002 he is full professor at the Instituto Tecnólogico de León, Mexico. His research interests include heuristic optimization and optic metrology.

Jesús D. Terán V. was born in Tampico, Mexico in 1979. He received the Bachelor in Computer System Engineering from the Instituto de Estudios Superiores de Tamaulipas in 2003 and M.S degree from the Instituto Tecnológico de Ciudad Madero, Mexico in 2005. Currently he is studying a PhD in Computer Sciences at the Instituto Tecnológico de Tijuana, Mexico. His research interests include algorithmics and heuristic optimization.