

# Parallelization of the Scatter Search

Félix García-López, Belén Melián-Batista,  
José A. Moreno-Pérez and J. Marcos Moreno-Vega,  
Dpto. de E.I.O. y Computación.  
Universidad de La Laguna

June 7, 2002

## Abstract

This article develops several strategies for the parallelization of the meta-heuristic called Scatter Search, which is a population-based method that constructs solutions by combining others. Three types of parallelization have been proposed to achieve either an increase of efficiency or an increase of exploration. The procedures have been coded in C using OpenMP and compared in a shared memory machine with large instances. The obtained algorithms are tested on the  $p$ -median problem.

## 1 Introduction

Scatter Search (SS) [8][16] is a population-based metaheuristic that constructs solutions by combining others in a set of solutions, named reference set. The basic SS combines solutions of the reference set and runs a local search procedure to reach a local optimum. A Local Search (LS) consists of selecting a better neighbor solution until no such solution exists. The reference set is updated depending on the results of the improvements.

In this paper we propose the parallelization of the SS metaheuristic to achieve either an increase of efficiency or an increase of exploration. In any parallelization some steps of the algorithm are distributed among the available processors. The increase of the efficiency is obtained by performing the same steps in less time than the sequential algorithm, whereas the increase of the exploration is obtained by performing more steps in the same time as the sequential algorithm. Several strategies for parallelizing a SS algorithm are analyzed. We test them with large instances of the  $p$ -median problem obtained from the TSPLIB [22]. The corresponding parallel algorithms are coded in C using OpenMP, a model for parallel programming portable across shared memory architectures. OpenMP [21] is based on a combination of compiler directives, library routines and environment variables that are used to specify shared memory parallelism in Fortran and C/C++ programs.

The  $p$ -median problem is a location/allocation problem, where  $p$  locations for the facilities are selected such that the sum of the distances from a set of users to the set of facility points is minimized. It is a prototype of a wide class of hard combinatorial problems, named  $p$ -selection problems, where the solutions are generated by selecting  $p$  items from a finite universe. The evaluation of the objective function in  $p$ -selection problems ranges from a simple calculation of a function value in a small constant time to solve another hard problem or even to perform a

simulation process. The standard moves for  $p$ -selection problems are the interchange or swap moves that exchange an item in the solution with another one out of the solution.

Next section describes the basics of the SS metaheuristic. In section 3 we provide a brief overview of logistic problems in general and of the  $p$ -median problem, in particular, as a prototype of the class of fixed-size location/allocation problems. In section 4 we discuss briefly the application of the SS to the  $p$ -median problem. Several parallelization strategies are analyzed in section 5. Computational results and concluding remarks are given in sections 6 and 7, respectively.

## 2 The Scatter Search metaheuristic

Scatter Search is a population-based metaheuristic that uses a reference set to combine its solutions and construct others. The method generates a reference set from a population of solutions. Then a subset is selected from this reference set. The selected solutions are combined to get starting solutions to run an improvement procedure. The result of the improvement can motivate the updating of the reference set and even the updating of the population of solutions.

The initial population must be a wide set of disperse solutions. However, it must also include good solutions. Several strategies can be applied to get a population with these properties. The solutions for the population can be created, for instance, by using a random procedure to achieve a certain level of diversity. Then a simple improvement heuristic procedure must be applied to these solutions in order to get better solutions. The initial population can also be obtained by a procedure that provides at the same time disperse and good solutions like GRASP procedures [4].

A set of good representative solutions of the population is chosen to generate the reference set. The good solutions are not limited to those with the best objective values. By good representative solutions we mean solutions with the best objective values as well as disperse solutions; in the sense that they would be improved to reach different local minima by a local search. Indeed, a solution may be added to the reference set if the diversity of the set improves. So the reference set must consist of a set of disperse and good solutions selected from the populations. The criteria for updating the reference set, when necessary, must be based on comparisons and measures of diversity between the new solutions and the existing solutions.

A subset of solutions from the reference set is selected to apply a combination method to get good starting solutions for an improvement procedure. In general, the method consists in selecting all the subsets of a fixed size. The combination procedure tries to combine good characteristics of the selected solutions to get new current solutions. The purpose is to get better solutions which are not similar to those already in the reference set.

The possible improvement solution methods applied to the solutions range from simplest local searches to a very specialized search. A very simple procedure is a local search based on basic moves consisting of selecting the best improving move or a first found improving move. The procedure must allow to use tools like recent or intermediate memory, variable neighborhoods, or hashing scanning methods of the neighborhood. Then the method applied could be a Tabu Search [6] [7], a Variable Neighborhood Search [10] [11] or any sophisticated hybrid heuristic search.

The metaheuristic strategy includes the decision on how to update the reference set having into account the state of the search. The algorithm must also realize when the reference set does not change and seek to diversify the search by generating a new set of solutions for the population. In

addition, the metaheuristic involves the stopping criterion for the whole search procedure. Then the best solution used in the reference set is provided by the method.

Usual stopping conditions are based on allowing a total maximum computational time or a maximum computational effort since the last improvement. The computational effort is measured by the number of iterations, number of local searches or real time.

A high level pseudocode of the Sequential Scatter Search (SSS) is showed in the figure 1.

```

Procedure Sequential Scatter Search
begin
repeat
  CreatePopulation(Pop,PopSize);
  repeat
    GenerateReferenceSet(RefSet,RefSetSize);
    repeat
      SelectSubset(SubSet,SubSetSize);
      CombineSolutions(SubSet, CurSol);
      ImproveSolution(CurSol, ImpSol);
    until (StoppingCriterion1);
    UpdateReferenceSet(RefSet);
  until (StoppingCriterion2);
until (StoppingCriterion3);

```

Figure 1: The Scatter Search Metaheuristic

Therefore, the Scatter Search strategy involves 6 procedures and 3 stopping criteria to solve an optimization problem. The procedures are the following:

1. **The Initial Population Creation Method.** The procedure *CreatePopulation* creates a random initial population *Pop* of good and disperse *PopSize* solutions.
2. **The Reference Set Generation Method.** The procedure *GenerateReferenceSet* selects *RefSetSize* of the best representative solutions in the population *Pop* to be included in the set *RefSet*.
3. **The Subset Generation Method.** The procedure *SelectSubset* generates subsets *SubSet*, which consists of *SubSetSize* good solutions in *RefSet*, to apply the next combination procedure.
4. **The Solution Combination Method.** The procedure *CombineSolutions*, provided with parameters used to modulate the intensification and/or diversification, combines the solutions in *SubSet* to get the new current solution *CurSol*.
5. **The Improvement Solution Method.** The procedure *ImproveSolution*, provided with parameters to modulate the specialization of the method, improves the current solution *CurSol* to get an improved solution *ImpSol*.

6. **The Reference Set Updating Method.** The procedure *UpdateReferenceSet* updates the reference set by deciding when and how the obtained improved solutions are included in the reference set replacing some solutions already in it.

In addition to these six procedures, the metaheuristic involves three stopping procedures that implement the criteria to decide when to go to an above step.

1. **New Reference Set Criterion.** The first criterion (*StoppingCriterion1*) decides when to generate a new reference set from the population.
2. **New Population Criterion.** The second criterion (*StoppingCriterion2*) decides when to generate a new initial population.
3. **Termination Criterion.** Finally, the third criterion (*StoppingCriterion3*) decides when to stop the whole search.

### 3 Selection problems

The selection problems are those problems that can be solved by choosing an optimal set of items from a universe. A generic selection problem consists in choosing the set of items that minimizes a cost function subject to some constraints. The objective functions for these problems range from the simplest one up to that needing to solve another hard problem or even to perform a simulation process. Also the set of constraints can be so restrictive such that only one selection is feasible or that all the selections of items are feasible. A fixed size, say  $p$ , selection problem, named a  $p$ -selection problem, consists in choosing the set of  $p$  items that minimizes a cost function subject to some constraints; i.e. a selection problem where all the feasible solutions have size  $p$ .

Several of the most relevant combinatorial optimization problems can be formulated as  $p$ -selection problems for an appropriate  $p$ . Among them are the most famous problems in Combinatorial Optimization: the Travelling Salesman Problem, Knapsack Problem,  $p$ -Median Problem, Spanning Tree Problem, Shortest Path Problem, Network Flow Problem, Matching Problem and Steiner Problem. Even most of the special versions of these problems can also be formulated as selection problems. Also the Linear Programming Problem consists in the optimal selection of a base, a set of points of which the size is known.

The  $p$ -facility location-allocation problems constitute a wide class of logistic problems. Consider a set  $L$  of  $m$  potential locations for  $p$  facilities and a set  $U$  of locations of  $n$  given users. A  $p$ -facility location-allocation problem consists of locating simultaneously  $p$  facilities at locations of  $L$  and allocate every user  $u$  to a facility point in order to minimize a cost function that represents the resource used for serving the demand of every user from the corresponding facility point. One of the most relevant  $p$ -facility location problems is the  $p$ -median problem. The  $p$ -median problem consists of locating  $p$  facilities in order to minimize the total distance between the users and its closest facility.

Given the set  $L = \{v_1, v_2, \dots, v_m\}$  of potential locations for the facilities (or location points), and the set  $U = \{u_1, u_2, \dots, u_n\}$  of users (or customers, or demand points), the entries of an  $n \times m$  matrix  $D = (d_{ij})_{n \times m} = (Dist(u_i, v_j))_{n \times m}$  give the distances travelled (or costs incurred) for satisfying the

demand of the user located at  $u_i$  from the facility located at  $v_j$ , for all  $v_j \in L$  and  $u_i \in U$ . The objective of the  $p$ -median problem is to minimize the sum of these distances (or transportation costs), i.e.,

$$\text{minimize } \sum_{u_i \in U} \min_{v_j \in X} \text{Dist}(u_i, v_j)$$

where  $X \subseteq L$  and  $|X| = p$ .

The  $p$ -median problem is  $\mathcal{NP}$ -hard [14]. Many heuristics and exact methods have been proposed to solve it. Exact algorithms are provided by Beasley [1] and Hanjoul and Peeters [13], among others. Classical heuristics for this problem often cited in the literature are Greedy [15], Alternate [19] and Interchange [23]. In addition, several hybrids of these heuristics have been suggested. Another type of heuristics suggested in the literature is based on the relaxed dual of the integer programming formulation of the  $p$ -median problem and uses the well-known Dual Ascent heuristic, DUALOC [3]. In [20] a 1-interchange move is extended into a so-called 1-chain-substitution move, which is applied to  $p$ -median problem. Another Tabu Search heuristic is suggested by Voss [24], where some variants of the so-called reverse elimination method are discussed. The Variable Neighborhood Search heuristic and its variants have also been applied to the  $p$ -median problems [9] [12]. In [5], several parallelization methods of the Variable Neighborhood Search are considered for the  $p$ -median problem.

## 4 Application of SS to the $p$ -median problem

For most instances, the set of potential locations for the facilities and the set of locations of the users coincide, so that  $L = U$  and  $m = n$ . In this case, a solution for the  $p$ -median problem consists of selecting a set  $X$  of  $p$  points from  $U$  to locate the facilities. The solution is evaluated by a cost function which is the sum of the distances from the users to the points in the solution. This cost function is given by:

$$\text{Cost}(X) = \sum_{u \in U} \min_{v \in X} \text{Dist}(u, v)$$

Regardless the search technique employed, it is necessary to specify a solution coding, which encodes alternative candidate solutions for manipulation. The choice of the coding that provides an efficient way of implementing the moves and evaluating the solutions is essential for the success of the heuristic. Every solution  $X$  is encoded by arranging all the points of  $U$  in an array  $[v_i : i = 1, \dots, n]$  where  $v_i$  is a point in  $X$  for  $i \leq p$ , and it is a point out of  $X$  for  $i > p$ .

### 4.1 Initial Population Creation Method

The initial population creation method must be designed in order to get a random set of disperse and good solutions. The method we use have several phases. The first phase starts by dividing the set  $L$  in several sets. A constructive method to get a good solution consists of, from an arbitrary initial point  $u$  of  $L$ , select  $p - 1$  times the farthest point to the already selected points. We perform this constructive method for each set of the partition of  $L$ . Then we get a solution for each set of the partition. However, since the constructed solution depends on the starting point, we apply it from several starting points to get different solutions. We also apply an improvement procedure to these solutions.

For evaluating the dispersion among solutions, we need to consider a distance between them. This distance is defined using the same objective function. Let  $f_Y(X)$  be the objective function for the set of users in  $Y$ :

$$f_Y(X) = \sum_{v \in Y} \min_{u \in X} Dist(u, v)$$

The distance between two solutions  $X$  and  $Y$  is  $Dist(X, Y) = f_Y(X) + f_X(Y)$ . Given a previously fixed size  $PopSize$  for the population  $Pop$  and a factor  $\alpha$ , we use the above method to get  $\lfloor \alpha PopSize \rfloor$  solutions for the population. The remainder solutions up to  $PopSize$  are obtained by an scoring procedure. For each solution  $X$  we define the score by:

$$h(X) = Cost(X) - \beta Dist(X, P)$$

where  $Dist(X, P) = \min_{Y \in Pop} Dist(X, Y)$  and  $\beta$  is a fixed factor. The  $\lfloor (1 - \alpha) PopSize \rfloor$  best solutions according  $h(\cdot)$  are included in the population  $Pop$ .

## 4.2 Reference Set Generation Method

The reference set consists of a set of good and disperse solutions selected from the population. The generation of a reference set is done by selecting  $RefSetSize_1$  solutions from the best solutions and  $RefSetSize_2$  disperse solutions taking into account the above  $RefSetSize_1$  solutions ( $RefSetSize = RefSetSize_1 + RefSetSize_2$ ). After including the best  $RefSetSize_1$  solutions in  $RefSet$ , we iteratively include in the  $RefSet$  the farthest solution from the solutions already in  $RefSet$ , repeating this procedure  $RefSetSize_2$  times.

## 4.3 Subset Generation Method

The selection of a subset for applying the combination usually consists in selecting all the subsets of fixed size  $r$ . For our computational experience we use  $r = 2$ . However, in order to avoid repetitions of combinations when some solutions of the reference set do not vary, we keep information on the combinations performed.

## 4.4 Combination Method

Given the subset of selected solutions from the reference set, the combination method tries to get a solution with the good characteristics of these solutions. First of all, we select the points that are in all these solutions; let  $X$  be the set of these points. For every user point  $u$  let

$$L(u) = \{v \in L : Dist(u, v) \leq \beta Dist_{max}\}$$

where

$$Dist_{max} = \max_{u, v \in L} Dist(u, v).$$

Choose the point  $u^* \in L$  such that  $Dist(X, u^*) = \max_{u \in L} Dist(X, u)$  and select at random a point  $v \in L(u^*)$  that is included in  $X$ . This step is iteratively applied until  $X$  has size  $p$ .

## 4.5 Improvement Solution Method

The improvement solution method is the a local search based on the base moves. A local search method for combinatorial optimization performs a series of moves in the solution space, which improve each time the value of the objective function until a local optimum is found. The base moves for local searches for the  $p$ -median problem are the swap or interchange moves. For every solution  $X$ , given an element  $v_i$  in the solution and an element  $v_j$  not in the solution, the interchange move consists of dropping  $v_i$  from  $X$  and adding  $v_j$  to  $X$ . Let  $X_{ij}$  denote the solution obtained from  $X$  by interchanging  $v_i$  by  $v_j$ , where  $v_i \in X$  and  $v_j \in L - X$ . Each solution  $X$  has a neighborhood associated  $\mathcal{N}(X)$ , that consists of the solutions  $X_{ij}$  that are reached from  $X$  by the base move. At each iteration, the local search procedure obtains an improved solution  $X'$  in the neighborhood  $\mathcal{N}(X)$  of the current solution  $X$ , until no further improvement is found. The local search is implemented by choosing each time the best possible move among all interchange moves.

The pseudocode of the local search is given in the figure 2.

```
Procedure Local Search ( $X$ )
  begin
    repeat
      Set  $X' \leftarrow X$ 
      For every  $i$  and  $j$  with  $1 < i \leq p$  and  $p < j \leq n$  do
        Set  $X'_{ij} \leftarrow X' - \{v_i\} + \{v_j\}$ 
        If  $Cost(X'_{ij}) < Cost(X)$  then  $X \leftarrow X'_{ij}$ 
      Until  $Cost(X) = Cost(X')$ 
  end.
```

Figure 2: Local Search

## 5 Parallelization methods

The main purpose of parallel processing is to perform computations faster than those with a single processor by using a number of processors concurrently. This pursuit has had a tremendous influence on almost all the activities related to computing. The need for faster solutions and for solving larger-size problems arises in a wide variety of applications. In general, the parallel computation can be used to increase the size of the problems that can be solved, to speed up the computations and to attempt a more thorough exploration of the solution space.

In this paper, we analyze three different parallelization strategies for the SS algorithm. The first two strategies reduces the running time of the procedure. The first parallelization reduces the running time of every local search algorithm and the second parallelization perform the local searches in parallel from the results of the combinations. The last parallelization increases the exploration in the solution space by parallel running the Sequential Scatter Search for several populations.

## 5.1 Synchronous Parallel Scatter Search

In the sequential Scatter Search algorithm, in every iteration, the most consuming time part is the local search. Then we propose a synchronous algorithm that enables solving, in parallel, the local searches. We denote the Synchronous Parallel Scatter Search algorithm that parallelizes the local search by SPSS. The following template (figure 3) shows the pseudocode of the SPSS with  $n_{pr}$  processors.

```

Procedure SPSS
begin
repeat
  CreatePopulation(Pop,PopSize);
  repeat
    GenerateReferenceSet(RefSet,RefSetSize);
    repeat
      SelectSubset(SubSet,SubSetSize);
      CombineSolutions(SubSet, CurSol);
      (* ImproveSolution(CurSol, ImpSol) *)
      Take  $I \leftarrow \{(i, j) : 1 \leq i \leq p, p < j \leq n\}$ .
      Divide  $I$  in  $n_{pr}$  subsets  $I_r, r = 1, \dots, n_{pr}$ .
      repeat
        Set  $ImpSol \leftarrow CurSol$ 
        For each processor  $r = 1, \dots, n_{pr}$ , do in parallel
          Set  $X_r \leftarrow CurSol$  ;
          For every  $(i, j) \in I_r$  do
            Set  $X_{ij} \leftarrow CurSol - \{v_i\} + \{v_j\}$  ;
            If  $Cost(X_{ij}) < Cost(X_r)$ 
              then  $X_r \leftarrow X_{ij}$ ;
          For  $r = 1, \dots, n_{pr}$  do
            If  $Cost(X_r) < Cost(CurSol)$ 
              then  $CurSol \leftarrow X_r$ ;
        Until  $Cost(ImpSol) = Cost(CurSol)$ ;
      until (StoppingCriterion1);
    UpdateReferenceSet(RefSet);
  until (StoppingCriterion2);
until (StoppingCriterion3);

```

Figure 3: Synchronous Parallel Scatter Search

Note that this pseudocode is obtained from the sequential SS by replacing the local search by the parallel version of the local search. In SPSS, the neighborhood,  $\{X_{ij} : 1 \leq i \leq p, p < j \leq n\}$ , is divided in  $n_{pr}$  subsets  $I_r$ . These subsets are assigned to the processors and each returns an improving neighbor in its subset of the neighborhood. The best solution among the neighbors provided by the processors is chosen as the current solution. This strategy is a low-level parallelism.

## 5.2 Replicated Combination Scatter Search

We propose another parallel algorithm that also reduces the running time of the procedure. The procedure is parallelized by selecting several subsets from the reference set that are combined and

improved by the processors. These procedures are replicated as many times as the number of available processors. The local optima found by the processors are used to update the reference set. This method is the Replicated-Combination Scatter Search (RCSS) that is described in the figure 4.

```

Procedure RCSS
begin
repeat
  CreatePopulation(Pop,PopSize);
  repeat
    GenerateReferenceSet(RefSet,RefSetSize);
    repeat
      For each processor  $r = 1, \dots, n_{pr}$ , do in parallel
        SelectSubset(SubSetr,SubSetSize);
        CombineSolutions(SubSetr, CurSolr);
        ImproveSolution(CurSolr, ImpSolr) }
    until (StoppingCriterion1);
    UpdateReferenceSet(RefSet);
  until (StoppingCriterion2);
until (StoppingCriterion3);

```

Figure 4: Replicated-Combination Parallel Scatter Search

### 5.3 Replicated Parallel Scatter Search

The Replicated Parallel Scatter Search (RPSS) consists of a multistart search where the local searches are replaced by SSs using different populations that run on the parallel processors (see figure 5). The RPSS parallelization method corresponds to a natural parallelization of the hybrid metaheuristic between a SS and a multistart search.

## 6 Computational results

The algorithms were coded in C and OdinMp (OdinMp is a free and portable implementation of OpenMP for ANSI C), and tested with large instances of the  $p$ -median problem. The distance matrix was taken from the instance TSPLIB RL1400 that includes 1400 points. The sets of instances are characterized with the number  $n$  of points (1400) and the number  $p$  of facility points or medians that is reported in first column of tables 1 2 and 3 going from 10 to 100. Some computational results on these instances of the problem have been reported in [12] [5], where several heuristics were compared. The algorithms run on the machine *TEIDE* (8 processors ALPHA at 466-Mhz, with 2 Gbytes and O.S. DIGITAL UNIX 4.0C) of the *University of La Laguna*.

In table 1, we report the results for the SPSS algorithm, which runs as many times as the number of available processors (1, 2, 4 and 8, respectively). The objective value, real time in seconds and speed-up are presented for each number of processors. The real time is used to obtain the time reduction provided by the parallel algorithm SPSS. Speed-up is the ratio of sequential

```

Procedure SSS
begin
repeat
  Foreach processor  $r = 1, \dots, n_{pr}$ , do in parallel
    CreatePopulation(Pop,PopSize);
  repeat
    GenerateReferenceSet(RefSetr,RefSetSize);
  repeat
    SelectSubset(SubSetr,SubSetSize);
    CombineSolutions(SubSetr, CurSol);
    ImproveSolution(CurSolr, ImpSolr);
  until (StoppingCriterion1);
  UpdateReferenceSet(RefSetr);
until (StoppingCriterion2);
until (StoppingCriterion3);

```

Figure 5: The Replicated Parallel Scatter Search

time to parallel time to solve a particular problem on a given machine. It is given by:

$$\text{speed-up} = \frac{\text{Time to solve a problem with the parallel code on one processor}}{\text{Time to solve the same problem with the parallel code on } n_{pr} \text{ processors}}$$

The results in table 1 show that, in general, the speed-up increases with the number of processors. For two processors, the speed-up is almost linear. For 4 processors and  $p = 80$ , we observed a detrimental anomaly [17] [18]; i.e., the real time is greater than the real time for 2 processors. We also observed that, in some cases, using 4 and 8 processing elements, an acceleration anomaly manifests itself in the form of a speed-up greater than 4 and 8, respectively.

Table 2 summarizes the results for the RCSS algorithm, where the headings are the same as in table 1. The speed-up reached is smaller than the speed-up for the SPSS algorithm, but anyway the speed-up increases with the number of processors for the RCSS algorithm. The objective values obtained with both algorithms are similar. The real times for the SPSS are better than the real times for the RCSS.

Table 3 shows the results for the RPSS algorithm. For each number of processors 2, 4 and 8, we report the best known objectives and the objective values and real times for the RPSS. The best objective values are found by this algorithm. The real times are similar for different number of processors. The real time reported in this table is the maximum of the times of the processors.

$p$	1 processor			2 processors			4 processors			8 processors		
	Objective	Time	S-u	Objective	Time	S-u	Objective	Time	S-u	Objective	Time	S-u
10	101249.47	380	1	101249.47	221	1.72	101249.47	137	2.77	101249.47	88	4.32
20	57857.55	682	1	57857.55	399	1.71	57857.55	260	2.62	57857.55	161	4.24
30	44057.55	1245	1	44057.55	718	1.73	44065.85	402	3.10	44084.10	271	4.59
40	35002.02	1858	1	35002.02	1047	1.77	35002.02	461	<b>4.03</b>	35002.02	283	6.57
50	29089.71	1443	1	29089.71	847	1.70	29089.71	612	2.36	29089.71	311	4.64
60	25185.79	1930	1	25185.79	1119	1.72	25186.24	634	3.04	25167.84	628	3.07
70	22125.46	1864	1	22125.46	1088	1.71	22125.46	649	2.87	22125.46	416	4.48
80	19884.51	1794	1	19884.51	1049	1.71	19870.51	1085	<i>1.65</i>	19870.51	471	3.81
90	17987.91	4168	1	17987.91	2322	1.80	18006.23	780	<b>5.34</b>	18002.35	488	<b>8.54</b>
100	16563.93	3341	1	16563.93	1912	1.75	16551.68	1266	2.64	16554.08	520	6.64

Table 1: Synchronous Parallel Scatter Search

$p$	1 processor			2 processors			4 processors			8 processors		
	Objective	Time	S-u	Objective	Time	S-u	Objective	Time	S-u	Objective	Time	S-u
10	101249.47	380	1	101249.47	237	1.60	101249.47	145	2.62	101249.47	103	3.69
20	57857.55	682	1	57857.55	461	1.48	57857.55	307	2.22	57857.55	193	3.53
30	44057.55	1245	1	44057.55	806	1.54	44023.23	667	1.87	44023.23	687	1.81
40	35002.02	1858	1	35002.02	1173	1.58	35002.02	578	3.21	35002.02	557	3.34
50	29089.71	1443	1	29089.71	965	1.50	29089.71	953	1.51	29089.71	410	3.52
60	25185.79	1930	1	25185.79	1240	1.56	25175.01	760	2.54	25167.82	433	4.46
70	22125.46	1864	1	22125.46	1263	1.48	22125.46	828	2.25	22125.46	486	3.84
80	19884.51	1794	1	19884.51	1214	1.48	19872.28	1015	1.77	19884.52	519	3.46
90	17987.91	4168	1	17987.91	2871	1.45	17987.91	1389	3.00	17987.91	537	7.76
100	16563.93	3341	1	16563.93	2285	1.46	16561.58	1113	3.00	16553.70	827	4.04

Table 2: Replicated Combination Scatter Search

## 7 Conclusions

The combination of Scatter Search and parallelism provides a useful tool to solve hard problems. Scatter Search is parallelized in several ways. The Synchronous Parallel Scatter Search (SPSS) is obtained by parallelizing the local search. By parallelizing the combinations of solutions we get the Replicated Combination Scatter Search (RCSS), where the set of possible combinations is divided among the available processors and solved in parallel. The Replicated Parallel Scatter Search (RPSS) parallelizes the whole procedure and each processor runs in parallel a Scatter Search.

The objective values found with these algorithms are comparable with the best obtained in the literature [5] [12]. The SPSS and RCSS algorithms reduced the computational time properly. RPSS increases the diversification. Note that it is possible to increase the intensification by sharing the best solution reached by the processors. Other possible strategy for increasing the diversification is to apply several combination methods in parallel.

		2 processors		4 processors		8 processors	
$p$	Best known	Objective	Time	Objective	Time	Objective	Time
10	101249.47	101249.47	428	101249.47	438	101249.47	424
20	57857.55	57857.55	787	57857.55	791	57857.55	779
30	44013.02	44013.02	1371	44013.02	1962	44013.02	1634
40	35002.02	35002.02	2048	35002.02	2058	35002.02	1750
50	29089.71	29089.71	1667	29089.71	1672	29089.71	1605
60	25160.40	25160.40	2067	25160.40	2124	25160.40	2382
70	22125.46	22125.46	2035	22125.46	2044	22125.46	3040
80	19870.29	19870.29	3557	19870.29	3591	19870.29	2796
90	17987.94	17987.91	<b>22359</b>	17987.91	4730	17987.91	<b>12742</b>
100	16551.20	16556.58	3477	16552.48	4076	16552.48	3838

Table 3: Replicated Parallel Scatter Search

## References

- [1] Beasley, J.E. A note on solving large  $p$ -median problems *European Journal of Operational Research* vol 21 pp. 270-273 (1985)
- [2] Brunschen, C., Brorsson, M. OdinMP/CCp-A Portable Implementation of OpenMP for C. *Proceedings of First European Workshop on OpenMP EWOMP99*, pp. 21-26 (1999).
- [3] Erlenkotter, D. A dual-based procedure for uncapacitated facility location *Operations Research* vol. 26 pp. 992-1009 (1978)
- [4] Feo, T. A., Resende, M.G.C. Greedy Randomized Adaptive Search Procedures *Journal of Global Optimization* vol. 6 pp. 109-133 (1995)
- [5] García-López, F., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M. The Parallel Variable Neighborhood Search for the  $p$ -Median Problem *Journal of Heuristics* vol. 8 pp. 375-388 (2002)
- [6] Glover, F. Tabu Search - Part I. *ORSA Journal on Computing* vol. 1 pp. 190-206 (1989)
- [7] Glover, F. Tabu Search - Part II. *ORSA Journal on Computing* vol. 2 pp. 4-32 (1990)
- [8] Glover, F., Laguna, M., Martí, R. Fundamentals of Scatter Search and Path Relinking *Control and Cybernetics*, vol. 39, no. 3 pp. 653-684 (2000)
- [9] Hansen, P., Mladenović, N. Variable Neighborhood Search for the  $p$ -Median *Location Science* vol. 5 pp. 207-226 (1997)
- [10] Hansen, P., Mladenović, N. An Introduction to Variable Neighborhood Search In *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, S. Voss et al. (editors) Kluwer, pp. 433-458 (1999)
- [11] Hansen, P., Mladenović, N. Variable Neighborhood Search: Principles and Applications *European Journal of Operational Research*, vol. 130 pp.449-467 (2001)

- [12] Hansen, P., Mladenović, N. Pérez-Brito, D. Variable Neighborhood Decomposition Search *Journal of Heuristics* vol. 7 pp. 335-350 (2001)
- [13] Hanjoul, P., Peeters, D. A comparison of two dual-based procedures for solving the  $p$ -median problem *European Journal of Operational Research* vol. 20 pp. 387-396 (1985)
- [14] Kariv, O., Hakimi, S.L. An algorithmic approach to network location problems, part 2. The  $p$ -medians. *SIAM Journal on Applied Mathematics*, vol. 37 pp. 539-560 (1969).
- [15] Kuehn, A.A., Hamburger, M.J. A heuristic program for locating warehouses. *Management Science* vol. 9 pp. 643-666 (1963)
- [16] Laguna, M. Scatter Search In *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 183-193 (2002)
- [17] Lai, T., Sahni, S. Anomalies in parallel branch-and-bound algorithms *Communications of the ACM* vol. 27 pp. 594-602 (1984)
- [18] Lai, T., Sprague, A, Performance of parallel branch-and-bound algorithms *IEEE Transactions on computers* vol. 34 pp. 962-964 (1985)
- [19] Maranzana, F.E. On the location of supply points to minimize transportation costs *Operations Research Quarterly* vol. 12 pp. 138-139 (1964)
- [20] Mladenović, N., Moreno-Pérez, J.A., Moreno-Vega, J. M. A Chain-Interchange Heuristic Method, *Yugoslav Journal of Operational Research* vol. 6 pp. 41-54 (1996)
- [21] *OpenMP: A proposed Industry Standard API for Shared Memory Programming*. White Paper, October 1997 (<http://www.openmp.org/openmp/mp-documents/paper/paper.html>).
- [22] Reinelt, G. TSP-Lib A Travelling Salesman library *ORSA J. Computing*, vol. 3 pp. 376-384 (1991) <http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/>.
- [23] Teitz, M.B., Bart, P. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research* vol. 16 pp. 955-961 (1968).
- [24] Voss, S. A reverse elimination approach for the  $p$ -median problem *Studies in Locational Analysis* vol. 8 pp. 49-58 (1996)