

# Parallel Variable Neighborhood Searches

**José A. Moreno-Pérez\***

Dpto. de Sistemas Informáticos y Computación.

Universidad de La Laguna. 38271 La Laguna. Spain

e-mail: jamoreno@ull.es

**Pierre Hansen,**

GERAD and HEC Montreal, Canada,

e-mail: Pierre.Hansen@gerad.ca

**Nenad Mladenović**

GERAD and Mathematical Institute, SANU, Belgrade

e-mail: nenad@mi.sanu.ac.yu

September 2004

## **Abstract**

Variable Neighborhood Search (VNS) is a recent and effective metaheuristic for solving combinatorial and global optimization problems. It is capable of escaping from the local optima by systematic changes of the neighborhood structures within the search. In this paper several parallelization strategies for VNS have been proposed and compared on the large instances of the  $p$ -median problem.

---

\*The research of this author has been partially supported by the Spanish Ministry of Science and Technology through the project TIC2002-04242-C03-01; 70% of which are FEDER funds.

# 1 Introduction

A combinatorial optimization problem consists of finding, minimum or maximum of a real valued function  $f$ . If it is a minimization problem, it can be formulated as follows

$$\min\{f(x) : x \in X\}. \quad (1)$$

Here  $X$  is called *solution space*,  $x$  represents an alternative *feasible solution* and  $f$  an *objective function* of the problem. An *optimal solution*  $x^*$  (or a global minimum) of the problem is a feasible solution where the minimum of (1) is reached. That is,  $x^* \in X$  has a property that  $f(x^*) \leq f(x)$ ,  $\forall x \in X$ . A *local minimum*  $x'$  of the problem (1), with respect to (w.r.t. for short) the neighborhood structure  $N$  is a feasible solution  $x' \in X$  that satisfies the following:  $f(x') \leq f(x)$ ,  $\forall x \in N(x)$ . Therefore any *local* or neighborhood *search* method (i.e., method that only moves to a better neighbor of the current solution) is trapped when it reaches a local minimum.

Several metaheuristics, or frameworks for building heuristics, extend this scheme to avoid being trapped in a local optimum. The most known of them are Genetic Search, Simulated Annealing and Tabu Search (for discussion of the best-known metaheuristics the reader is referred to the books of surveys edited by Reeves (1993) and Glover and Kochenberger (2003)). Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997, Hansen and Mladenović 1999, 2001c, 2003) is a recent metaheuristic which exploits systematically the idea of neighborhood change, both in the descent to local minima and in the escape from the valleys which contain them.

The application of the parallelism to a metaheuristic can and must allow to reduce the computational time (the partition of the sequential program) or to increase the exploration in the search space (the application of independent search threads). In this survey, several strategies for parallelizing a VNS are considered. We analyze and test them with large instances of the  $p$ -Median problem. Next section describes the basics of the VNS metaheuristic. Details on the  $p$ -median problem and the application of VNS to solve it are given in section 3. The parallelization strategies for VNS applied to the  $p$ -median in the literature are analyzed in section 4. The computational experiences are summarized in sections 5 and, finally, the conclusions are outlined in section 6.

## 2 The VNS metaheuristic

The basic idea of VNS is a change of neighborhoods in search for a better solution. VNS proceeds by a descent method to a local minimum, then explores, systematically or at random, increasingly distant neighborhoods of this solution. Each time, one or several points within the current neighborhood are used as initial solution for a local descent. One jumps from the current solution to a new one if and only if a better solution has been found. So VNS is not a trajectory method (as Simulated Annealing or Tabu Search) and does not specify forbidden moves. Despite its simplicity it proves to be effective. VNS exploits systematically the following observations:

**Fact 1** *A local minimum with respect to one neighborhood structure is not necessary so for another;*

**Fact 2** *A global minimum is a local minimum with respect to all possible neighborhood structures.*

**Fact 3** *For many problems local minima with respect to one or several neighborhoods are relatively close to each other.*

This last observation, which is empirical, implies that a local optimum often provides some information about the global one. This may for instance be several variables with the same value in both. However, it is usually not known which ones are such. An organized study of the neighborhood of this local optimum is therefore in order, until a better one is found.

Unlike many other metaheuristics, the basic schemes of VNS and its extensions are simple and require few, and sometimes no parameters. Therefore in addition to providing very good solutions, often in simpler ways than other methods, VNS gives insight into the reasons for such a performance, which in turn can lead to more efficient and sophisticated implementations.

Variable neighborhood descent (VND) is deterministic version of VNS. It is based on *Fact 1* above, i.e., *a local optimum for a first type of move  $x \leftarrow x'$  (within a neighborhood  $N_1(x)$ ) is not necessary one for another type of move  $x \leftarrow \tilde{x}$  (within another neighborhood  $N_2(x)$ ).* It may thus be advantageous to combine descent heuristics. This leads to the basic VND scheme presented:

### VND method

1. Find an initial solution  $x$ .

2. Repeat the following sequence until no improvement is obtained:

(i) Set  $\ell \leftarrow 1$ ;

(ii) Repeat the following steps until  $\ell = \ell_{max}$ :

(a) Find the best neighbor  $x'$  of  $x$  ( $x' \in N_\ell(x)$ );

(b) If the solution  $x'$  thus obtained is better than  $x$ , set  $x \leftarrow x'$  and  $\ell \leftarrow 1$ ; otherwise, set  $\ell \leftarrow \ell + 1$ ;

Another simple application of the VNS principle is the reduced VNS. It is a pure stochastic search method: solutions from the pre-selected neighborhoods are chosen at random. Its efficiency is mostly based on Fact 3 from above. A set of neighborhoods  $N_1(x), N_2(x), \dots, N_{k_{max}}(x)$  will be considered around the current point  $x$  (which may be or not a local optimum). Usually, these neighborhoods will be nested, i.e., each one contains the previous. Then a point is chosen at random in the first neighborhood. If its value is better than that of the incumbent (i.e.,  $f(x') < f(x)$ ), the search is recentered there ( $x \leftarrow x'$ ). Otherwise, one proceeds to the next neighborhood. After all neighborhoods have been considered, one begins again with the first, until a stopping condition is satisfied (usually it will be maximum computing time since the last improvement, or maximum number of iterations). The description of the steps of the Reduced VNS is as follows:

### **RVNS method**

1. Find an initial solution  $x$ ; choose a stopping condition;

2. Repeat the following until a stopping condition is met:

(i)  $k \leftarrow 1$ ;

(ii) Repeat the following steps until  $k = k_{max}$ :

(a) *Shake*. Take (at random) a solution  $x'$  from  $\mathcal{N}_k(x)$ .

(b) If this point is better than the incumbent, move there ( $x \leftarrow x'$ ), and continue the search with  $\mathcal{N}_1$  ( $k \leftarrow 1$ ); otherwise, set  $k \leftarrow k + 1$ .

In the previous two methods, we examined how to use variable neighborhoods in descent to a local optimum and in finding promising regions for near-optimal solutions. Merging the tools for both

tasks leads to the General Variable Neighborhood Search scheme. We first discuss how to combine a local search with systematic changes of neighborhoods around the local optimum found. We then obtain the Basic VNS scheme.

### **BVNS method**

1. Find an initial solution  $x$ ; choose a stopping condition;
2. Repeat until the stopping condition is met:
  - (1) Set  $k \leftarrow 1$ ;
  - (2) Repeat the following steps until  $k = k_{max}$ :
    - (a) *Shaking*. Generate a point  $x'$  at random from the  $k^{th}$  neighborhood of  $x$  ( $x' \in \mathcal{N}_k(x)$ );
    - (b) *Local search*. Apply some local search method with  $x'$  as initial solution; denote with  $x''$  the so obtained local optimum;
    - (c) *Move or not*. If the local optimum  $x''$  is better than the incumbent  $x$ , move there ( $x \leftarrow x''$ ), and continue the search with  $\mathcal{N}_1$  ( $k \leftarrow 1$ ); otherwise, set  $k \leftarrow k + 1$ ;

If instead of simple local search, one uses Variable Neighborhood Descent and if one improves the initial solution found by Reduced VNS, one obtains the General Variable Neighborhood Search scheme (GVNS).

### **GVNS algorithm**

1. *Initialization*.  
 Select the set of neighborhood structures  $\mathcal{N}_k$ , for  $k = 1, \dots, k_{max}$ , that will be used in the shaking phase, and the set of neighborhood structures  $N_\ell$  for  $\ell = 1, \dots, \ell_{max}$  that will be used in the local search; find an initial solution  $x$  and improve it by using RVNS; choose a stopping condition;
2. *Main step*.  
 Repeat the following sequence until the stopping condition is met:
  - (1) Set  $k \leftarrow 1$ ;
  - (2) Repeat the following steps until  $k = k_{max}$ :

(a) *Shaking.*

Generate a point  $x'$  at random from the  $k^{\text{th}}$  neighborhood  $\mathcal{N}_k(x)$  of  $x$ ;

(b) *Local search by VND.*

(b1) Set  $\ell \leftarrow 1$ ;

(b2) Repeat the following steps until  $\ell = \ell_{\max}$ ;

· Find the best neighbor  $x''$  of  $x'$  in  $N_\ell(x')$ ;

· If  $f(x'') < f(x')$  set  $x' \leftarrow x''$  and  $\ell \leftarrow 1$ ; otherwise set  $\ell \leftarrow \ell + 1$ ;

(c) *Move or not.* If this local optimum is better than the incumbent, move there ( $x \leftarrow x''$ ), and continue the search with  $\mathcal{N}_1(k \leftarrow 1)$ ; otherwise, set  $k \leftarrow k + 1$ ;

### 3 VNS for the $p$ -median

The  $p$ -median problem has been chosen as test problems in a wide set of basic VNS algorithms and extensions.

#### 3.1 The $p$ -Median Problem

The  $p$ -median problem is a location/allocation problem consisting of selecting the  $p$  locations for the facilities that minimize the sum of the distances from a set of users to the set of facility points. It belongs to a wide class of hard combinatorial problems where the solutions consists in the selection of  $p$  items from an universe. The evaluation of the objective function of the location/allocation problems ranges from the simplest one to that needing to solve another hard problem or to perform a simulation process. The standard moves for this class of problems are the interchange moves. An interchange move consists of replacing an item in the solution by another one out of the solution.

Consider a space  $S$  that includes a set of potential location points for facilities or facility centers and a given set of users with their corresponding demand for the facility. Consider also a real-valued function  $D : S \times S \rightarrow \Re$  whose values  $D(s, t)$  represents,  $\forall s, t \in S$ , the distance traveled (or costs incurred) for satisfying a unit of demand of a user located at  $s$  from a facility located at  $t$ . The distance from a finite set of facility points  $X \subset S$  to a user located at  $s$  is:

$$D(X, s) = \min_{x \in X} D(x, s).$$

The total transportation cost for satisfying the demand of all the users located at a finite set of points  $U \subseteq S$  from the facilities located at the points of  $X \subset S$  is:

$$T(X, U) = \sum_{u \in U} D(X, u) \cdot w(u),$$

where  $w(u)$  is the total amount of demand of all the users located at  $u$ . The  $p$ -median problem consists of locating  $p$  facility centers (or medians) in  $S$  in order to minimize the total transportation cost for satisfying the demand of all the users. Several formulations and extensions of this optimization problem are useful to model many real world situations, such as the location of industrial plants, warehouses and public facilities. When the set of potential locations and the set of users are finite, the problem has a combinatorial formulation. The formulation of the combinatorial  $p$ -median problem is as follows.

Let  $L = \{v_1, v_2, \dots, v_m\}$  be the finite set of potential location for the  $p$  medians, and  $U = \{u_1, u_2, \dots, u_n\}$  be the set of demand points for the facility. Consider also a weight vector  $w = [w_i : i = 1, \dots, n]$  representing the amount of demand of the users located at demand point  $u_i$ . Let  $D$  be the  $n \times m$  matrix whose entries contain the distances  $dist(u_i, v_j) = d_{ij}$  between the demand point  $u_i$  and the potential location  $v_j$ , for  $i = 1, \dots, n, j = 1, \dots, m$ ; i.e.,

$$D = [d_{ij} : i = 1, \dots, n, j = 1, \dots, m] = [Dist(u_i, v_j) : i = 1, \dots, n, j = 1, \dots, m].$$

The  $p$ -median problem consists of choosing the location of  $p$  medians from  $L$  minimizing the total transportation cost for satisfying the whole demand. The objective of the combinatorial problem is to minimize the sum of the weighted distances (or transportation costs), i.e.,

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

where  $X \subseteq L$  and  $|X| = p$ . The capacitated version [Díaz and Fernández, 2004], [Lorena and Senne, 2004] includes a fixed capacity for the facility center located at each point of  $L$  that bound the amount of demand served by it, but usually the problem is uncapacitated and each customer is supplied from its closest facility.

Beside this combinatorial formulation, the  $p$ -median problem has a formulation in terms of integer programming with matrix  $D$  and vector  $w$  as data. The formulation includes two sets of decision variables: the location variables  $y = [y_j : j = 1, \dots, m]$  and the allocation variables  $x = [x_{ij} : i = 1, \dots, n, j = 1, \dots, m]$ . The meaning of the variable is as follows:

- $y_j = 1$  if a facility is located at  $v_j$  and  $y_j = 0$  otherwise.
- $x_{ij} = 1$  if all the users at demand point  $u_i$  are served from a facility located at  $v_j$  and  $x_{ij} = 0$  otherwise.

The integer linear programming formulation of the  $p$ -median problem is then:

$$\begin{aligned}
& \text{Minimize} && \sum_{i=1}^n \sum_{j=1}^m w_i d_{ij} x_{ij} \\
& \text{Subject to} && \\
& && \sum_{j=1}^m x_{ij} = 1, \quad i = 1, \dots, n \\
& && x_{ij} \leq y_j, \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& && x_{ij} \leq y_j, \quad i = 1, \dots, n, \quad j = 1, \dots, m \\
& && \sum_{j=1}^m y_j = p \\
& && x_{ij}, y_j \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m.
\end{aligned}$$

However, the most common version of the  $p$ -median problem is the unweighted case where all the weights are equal and can be eliminated from the formulation.

The unweighted and uncapacitated  $p$ -median problem is  $\mathcal{NP}$ -hard [Kariv and Hakimi, 1969]. Extensive references to works on this and related problems are contained in the main books, surveys and reviews in Location like [Cornuejols, Fisher and Nemhauser, 1977], [Brandeau and Chiu, 1989], [Mirchandani and Francis, 1990], [Drezner, 1995] and [Labb and Louveaux, 1997].

Many heuristics and exact methods have been proposed for solving it. Exact algorithms were developed by [Beasley, 1985] and [Hanjoul and Peeters, 1985] and others. [Avella and Sassano, 2001] and [Farias, 2001] have studied the  $p$ -median polytope.

Classical heuristics for the  $p$ -median problem often cited in the literature are *Greedy* [Kuehn and Hamburger, 1963], *Alternate* [Maranzana, 1964] and *Interchange* [Teitz and Bart, 1968]. The basic Greedy heuristics starts with an empty set and repeat the following greedy step; the facility point that most minimize the objective of the resulting set is added. The Alternate heuristics, from an arbitrary set of  $p$  locations “alternate” the following allocation and allocation steps. In an allocation step, all the users are allocated to the nearest facility point. In an location step, the best facility

point for the set of users allocated to a single facility point is chosen. These steps are iterated while some improvement is obtained. Finally, the Interchange heuristic, from an arbitrary initial set of  $p$  facility points chosen as median, iteratively interchanges a facility points in the median set with another facility point out of the median set.

Several hybrids of the classical heuristics have been suggested in literature and have been most often used for comparisons with other newly proposed methods. One of the most used is the combination of the greedy heuristic with alternate or interchange; where the greedy solution is chosen as the initial one for alternate or interchange heuristics. They have also been combined with general strategies or metaheuristics. For instance, the *GreedyG* heuristic proposed in [Captivo, 1991] performs the *Alternate* procedure at each step of the Greedy heuristic. A hybrid between the Alternate and the Interchange was proposed in [Pizzolato, 1994]. [Moreno, Rodríguez and Jiménez (1991)] proposed a variant of a reverse Greedy, a combination of Greedy and Alternate and a Multistart Alternate.

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 [Erlenkotter, 1978]. Heuristic of this type have been suggested in [Galvão (1980)] and Captivo (1991)]. Lagrangean-based methods have also been proposed by from a “primal” perspective, either as part of exact methods or as stand-alone heuristics (see [Beasley, 1993]). Combining these results with polyhedral results, an exact algorithm have been proposed in [Avella, Sassano and Vasilev, 2003].

Several Tabu Search ([Glover, 1989, 1990]) procedures have recently been proposed for solving the  $p$ -median problem. In [Mladenović, Moreno and Moreno-Vega, 1996] a interchange move is extended into a so-called chain-substitution move. Another Tabu Search heuristic is suggested by [Voss, 1996], where a few variants of the so-called reverse elimination method are discussed. In [Rolland, Schilling and Current, 1996] the interchange step is separated into add and drop moves that do not necessary follow each other and so feasibility is not necessary maintained during the search. In [Taillard, 2003] a tabu search framework that uses decomposition principles is considered.

The VNS heuristic, and its variants, have also been applied to the  $p$ -median problems (see [Hansen and Mladenovic, 1997], [Hansen, Mladenovic and Pérez-Brito, 2001], [Hansen and Mladenovic, 2000]).

### 3.2 Application of VNS to $p$ -Median problem

In this section we describe some details of the application of VNS to the standard  $p$ -median problem. In the standard instances of  $p$ -median problem, there is not weights associated to the users and the set of potential locations for the facilities and the set of locations of the users coincide. Then  $m = n$  and  $L = U$  is the universe of points and  $w_i = 1$ , for  $i = 1, \dots, n$ . A solution of the  $p$ -median problem consists of a set  $S$  of  $p$  points from the universe  $U$  to hold the facilities. The objective function is usually named cost function due to the economical origin of the formulation of the problem.

Therefore, the solutions are evaluated by the cost function computed as the sum of the distances to the points in the solution.

$$Cost(S) = \sum_{v \in U} Dist(v, S) = \sum_{v_i \in U} \min_{v_j \in S} Dist(v_i, v_j)$$

Most of the heuristic searches use a constructive method and then apply a good strategy to select base moves to apply. Basic constructive heuristics are used to generate an initial solutions for the searches. They consist of adding elements to an empty solution until a set of  $p$  points is obtained. The good strategies for adding elements to the solutions are to use simple, random and good rules to select the point to be added to the solution. The base moves for this problem are the **interchange moves**. They are used in most of the heuristic searches. Given a solution  $S$ , an element  $v_i$  in the solution and an element  $v_j$  not in the solution, an interchange move consists of dropping  $v_i$  from  $S$  and adding  $v_j$  to  $S$ .

The selection of a solution coding that provides an efficient way of implementing the moves and evaluating the solutions is essential for the success of any search method. For this purpose the following coding of the solutions are often applied. A solution  $S$  is represented by an array  $S = [v_i : i = 1, \dots, n]$  where  $v_i$  is the  $i$ -th element of the solution, for  $i = 1, 2, \dots, p$ , and the  $(i - p)$ -th element outside the solution, for  $i = p + 1, \dots, n$ .

The computation of the cost of the new solution after each move can be simplified by storing the best allocation costs in a vector named  $Cost1[.]$ , defined as:

$$Cost1[i] = \min_{j=1..p} Dist[v_i, v_j], i = 1, \dots, n.$$

and the second best allocation cost of  $v_i$ ,  $i = 1, \dots, n$ , in

$$Cost2[i] = \min_{j=1..p, j \neq r(i)} Dist[v_i, v_j].$$

where  $r(i)$  is such that  $Cost1[i] = Dist[v_i, v_{r(i)}]$ . The first and second best allocation costs have been used in a Variable Neighborhood Decomposition Search (VNDS) by [Hansen, Mladenovic, and Pérez-Brito, 2001].

For  $1 < i \leq p$  and  $p < j \leq n$ , let  $S_{ij}$  be the new solution consisting in interchanging  $v_i$  and  $v_j$ . Then the cost of the new solution is given by:

$$Cost(S_{ij}) = \min\{Dist[v_i, v_j], \min_{l=1, \dots, p, l \neq i} Dist[v_i, v_l]\} \\ + \sum_{k=p+1}^n \min\{Dist[v_k, v_j], \min_{l=1, \dots, p, l \neq i} Dist[v_k, v_l]\},$$

which would imply  $O(pn)$  operations. However, using the values in  $Cost1$  and  $Cost2$ , an improved procedure takes  $O(pn_i + n)$  time,  $n_i$  being the number of points assigned to point  $v_i$ . Note that if  $p$  is large then  $n_i$  must be short and the difference between  $pn$  and  $p n_i + n$  is important in shaking and local searches.

The Shake procedure consists of, given the size  $k$  for the shake, choosing  $k$  times two points at random,  $v_i$  and  $v_j$ ;  $v_i$  in the solution and  $v_j$  outside the solution, and performing the corresponding interchange move.

### Shake

Repeat  $k$  times

    Choose  $1 < i \leq p$  and  $p < j \leq n$  at random

    Let  $S_{ij} \leftarrow S - \{v_i\} + \{v_j\}$

    Do  $S \leftarrow S_{ij}$

The usual greedy local search is implemented by choosing iteratively the best possible move among all interchange moves. Let  $S_{ij}$  denote the solution obtained from  $S$  by interchanging  $v_i$  and  $v_j$ , for  $i = 1, \dots, p$  and  $j = p + 1, \dots, n$ . The pseudocodes of the local searches are:

### Local Search

Initialize  $S'$

Repeat

$$S \leftarrow S'$$

$$S' \leftarrow \arg \min_{i=1, \dots, p, j=p+1, \dots, n} Cost(S_{ij})$$

Until  $Cost(S') = Cost(S)$

A simple C code for a simple version of sequential Variable Neighborhood Search is as follows.

### Algorithm Sequential VNS

```

1: initialize(best_sol);
2: k = 0 ;
3: while (k < k_max) {
4:   k++ ;
5:   cur_sol = shake(best_sol,k) ;
6:   local_search(cur_sol) ;
7:   if improved(cur_sol,best_sol)
8:     best_sol = cur_sol ;
9:   k = 0 ;
10: } /* if */
11:} /* while */

```

This code of the VNS can be applied to any problem if the user provides the initialization procedure `initialize`, the shake `shake`, the local search `local_search` and the function `improved` to test if the solution is improved or not.

For those problem consisting on selecting a fixed number  $p$  of items from an universe  $U = \{u_1, \dots, u_n\}$ , the local search and the shake based on the interchange moves can also be implemented using a function `exchange` also provided for the user for his problem.

The code C for the sequential local search (SLS) is as follows:

### Algorithm LS

```

void local_search(sol cur_sol)
{
1: init_sol = cur_sol ;

```

```

2: while improved(cur_sol,init_sol)) {
3:   for (i=p;i<n;i++)
4:     for (j=0;j<p;j++) {
5:       exchange(init_sol,new_sol,i,j) ;
6:       if improved(new_sol,cur_sol)
7:         cur_sol = new_sol
8:     } /* for */
9:   } /* while */
} /* local_search */

```

Analogously the C code for the shake procedure is as follows:

#### Shake Procedure

```

void shake(sol cur_sol,k) ;
{
1: init_sol = cur_sol ;
2: for (r=0;r<k;r++) {
3:   i = rnd % p ;
4:   j = p + rnd % (n-p) ;
5:   exchange(cur_sol,new_sol,i,j) ;
6:   cur_sol = new_sol ;
7: } /* for */
8: } /* shake */

```

## 4 The parallelizations

The application of the parallelism to a metaheuristic can and must allow to reduce the computational time (the partition of the sequential program) or to increase the exploration in the search space (the application of independent search threads). In order to do it, we need to know the parts of the code of an appropriate size and that can be partitioned to be solved simultaneously. Several strategies for parallelizing a VNS algorithm have been proposed in the literature and they have been analyzed and

tested with large known instances of the  $p$ -Median Problem [García López et al., 2002] and [Crainic et al. 2004]. The parallel VNS heuristics reported were coded in C (using the OpenMP, a model for parallel programming portable across shared memory architectures) in [García López et al., 2002] and in Fortran 90 (using MPL) in [Crainic et al. 2004].

Using the OpenMp, pseudocodes very similar to a C programs were obtained as adaptation of codes originally written for serial machines that implement the sequential VNS. The OpenMP is based on a combination of compiler directives, library routines and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs (see The OpenMP architecture review Board. *OpenMP: A proposed Industry Standard API for Shared Memory Programming*. White Paper, October 1997 (<http://www.openmp.org/openmp/mp-documents/paper/paper.html>). Only few lines had to be replaced by specific directives of the OpenMP compiler in the pseudocode to get the code of the programs used in the computational experience.

Four different parallelization strategies have been reported in the literature; two are simple parallelization and other two are more complex strategies. The two simple parallelizations of the VNS consist of parallelizing the local search and of replicate the whole VNS in the processors. The other two additional parallelization strategies are proposed in [García López et al., 2002] and [Crainic et al., 2004], and tested for the  $p$ -median.

The first of the two simple parallelization strategies analyzed in [García López et al., 2002] attempts to reduce computation time by parallelizing the local search in the sequential VNS and is denoted SPVNS (*Synchronous Parallel VNS*). The second one implements an independent search strategy that runs an independent VNS procedure on each processor and is denoted RPVNS (*Replicated Parallel VNS*).

The parallel local search is implemented trying to get a balanced load among the processors. The procedure divide the set of  $p(n - p)$  solutions of the neighborhood of the current solution among the available *num\_proc* processors to look for the best one. The code of the procedure `par_local_search` that implements parallel local search is as follows.

#### Algorithm PLS

```
void par_local_search(sol cur_sol)
{
1: init_sol = cur_sol
```

```

2: while (improved()) {
3:   load = (n-p) div (num_proc) ;
4:   parallel (pr = 0; pr symbol< num_proc; pr++) {
5:     tmp_sol(pr) = init_sol;
6:     low = pr * load ; high = low + load ;
7:     for (i = low; i symbol< high; i++)
8:       for (j = 0; j symbol< p; j++) {
9:         exchange(init_sol,new_sol,i,j)
10:        if improve(new_sol,tmp_sol(pr))
11:          tmp_sol(pr) = new_sol ;
12:        } /* for */
13:    critical
14:    if improve(tmp_sol(pr), cur_sol)
15:      cur_sol = tmp_sol(pr);
16:  } /* parallel */
17:} /* while */
} /* par_local_search */

```

Then the pseudocode of the *Synchronous Parallel Variable Neighborhood Search* SPVNS is as follows.

#### Algorithm SPVNS

```

1: initialize(best_sol);
2: k = 0 ;
3: while (k symbol< k_max) {
4:   k++ ;
5:   cur_sol = shake(best_sol, k);
6:   par_local_search(cur_sol);
7:   if improved(cur_sol,best_sol) {
8:     best_sol = cur_sol ;
9:     k = 0 ;

```

```

10: } /* if */
11:} /* while */

```

The second simple parallelization of the VNS is the Replicated Parallel VNS (RPVNS) that tries to search for a better solution by means the exploration of a wider zone of the solution space by using the multistart strategies. It is done by increasing the number of neighbor solution to start a local search (several starting solutions in the same neighborhood or in different ones). This method is like a multistart procedure where each local searches is replaced by the VNS. The pseudocode of the RPVNS is described by the following pseudocode.

### Algorithm RPVNS

```

1: initialize(joint_best_sol);
2: parallel pr = 0, num_proc-1 {
3:   Initialize(best_sol(pr));
4:   k(pr) = 0 ;
5:   while (k(pr) symbol< k_max) {
6:     k(pr)++;
7:     cur_sol(pr) = shake(best_sol(pr), k(pr));
8:     local_search(cur_sol(pr));
9:     if improved(cur_sol(pr), best_sol(pr)) {
10:      best_sol(pr) = cur_sol(pr);
11:      k(pr) = 0;
12:    } /* if */
13:  } /* while */
14:critical
15:  if improve(best_sol(pr), joint_best_sol)
16:    joint_best_sol = best_sol(pr);
17:} /* parallel */

```

The two additional parallelization strategies use cooperation mechanism to improve the performance. The Replicated-Shaking VNS parallelization (RSVNS) of the VNS proposed in [García López et al., 2002] applies a synchronous cooperation mechanism through a classical master-slave

approach. The *Cooperative Neighbourhood VNS* (CNVNS) parallelization proposed in [Crainic et al., 2004] applies a cooperative multi-search method based on a central-memory mechanism.

In the Replicated-Shaking VNS (RSVNS), the master processor runs a sequential VNS but the current solution is sent to each slave processor that shakes it to obtain an initial solution from which the local search is started. The solutions obtained by the slaves are passed on to the master that selects the best and continues the algorithm. The independence between the local searches in the VNS allows their execution in independent processors and updating the information about the joint best solution found. This information must be available for all the processors in order to improve the intensification of the search. The Replicated-Shaking VNS (RSVNS) is described by the following pseudocode.

#### Algorithm RSVNS

```
1: initialize(joint_best_sol);
2: k = 0 ;
3: while (k symbol< k_max) {
4:   k++;
5:   joint_cur_sol = joint_best_sol ;
6:   parallel pr = 0, num_proc-1 {
7:     cur_sol(pr) = shake(joint_best_sol, k);
8:     local_search(cur_sol(pr));
9:     critical
10:    if improved(cur_sol(pr), joint_cur_sol)
11:      joint_cur_sol = cur_sol(pr);
12:    barrier ;
13:    master ;
14:    if improved(joint_cur_sol, joint_best_sol) {
15:      joint_best_sol = joint_cur_sol;
16:      k = 0;
17:    } /* if */
18:    barrier
19: } /* parallel */
```

```
20:} /* while */
```

The *Cooperative Neighbourhood VNS* (CNVNS) proposed by [Crainic et al., 2004] is obtained by applying the cooperative multi-search method to the VNS metaheuristic. This parallelization method is based on the central-memory mechanism that has been successfully applied to a number of different combinatorial problems. In this approach, several independent VNSs cooperate by asynchronously exchanging information about the best solution identified so far, thus conserving the simplicity of the original, sequential VNS ideas. The asynchronous cooperative multi-search parallel VNS proposed allows a broader exploration of the solution space by several VNS searches.

The controlled random search nature of the shaking in the VNS and its efficiency is altered significantly by the cooperation mechanism that implements frequent solution exchanges. However, the CNVNS implements a cooperation mechanism that allows each individual access to the current overall best solution without disturbing its normal proceedings. Individual VNS processes communicate exclusively with a *central memory* or master. There are no communications among individual VNS processes. The master keeps, updates, and communicates the current overall best solution. Solution updates and communications are performed following messages from the individual VNS processes. The master initiates the algorithms by executing a parallel RVNS (without local search) and terminates the whole search by applying a stopping rule.

Each processor implements the same VNS algorithm. It proceeds with the VNS exploration for as long as it improves the solution. When the solution is not improved it is communicated to the master if better than the last communication, and the overall best solution is requested from the master. The search is then continued starting from the best overall solution in the current neighborhood. The CNVNS procedure is summarized as follows:

### **Cooperative Neighbourhood VNS**

- Master process:
  - Executes parallel RVNS;
  - Sends initial solutions to the individual VNS processes;
  - After each communication from an individual VNS process, updates the best overall and communicates it back to the requesting VNS process.

- Verify the stopping condition
- Each VNS process
  - Receives the initial solution, select randomly a neighborhood and explore it by shaking and local search;
  - If the solution is improved, the search proceeds from the first neighborhood: shake and local search
  - If the solution cannot be improved, the process
    - \* Communicates its solution to the master;
    - \* Requests the overall best solution from the master;
    - \* Continuous the search from the current neighborhood.

A pseudo-code similar to those of the above parallelizations is as follows:

### Algorithm CNVNS

#### Master process

```

1: parallel_RVNS(init_sol(pr): pr=1..num_proc);
2: initialize(joint_best_sol);
3: for (pr=1,pr<num_proc,pr++) {
4:   send(init_sol(pr): pr=1..num_proc)
5: } /* for */
6: while (not_stopping_criterion) {
7:   get(improved_sol(pr));
8:   if improved(improved_sol(pr), joint_best_sol)
9:     joint_best_sol = improved_sol(pr);
10:  init_sol(pr) = joint_best_sol ;
11:  send(init_sol(pr): pr=1..num_proc) ;
12: } /* while */

```

### Worker(pr) process

```
1: get(best_sol_pr));
2: k = 0 ;
3: while (k < k_max) {
4:   k++ ;
5:   cur_sol = shake(best_sol_pr,k) ;
6:   local search(cur_sol) ;
7:   if improved(cur_sol,best_sol_pr)
8:     best_sol_pr = cur_sol ;
9:     k = 0 ;
10: } /* if */
11: } /* while */
12: Return best_sol_pr
```

## 5 Computational experience

The algorithms of [García et al., 2002] were coded in C and tested with instance of the  $p$ -median problem where the distance matrix was taken from TSPLIB RL1400 that includes 1400 points. The values for  $p$  where from 10 to 100 with step 10; i.e., 10, 20, ..., 100. The algorithms run on the *Origin 2000* (64 processors R10000 at 250-Mhz, with 8 Gbytes and O.S. IRIX 6.5) of the *European Center for Parallelism of Barcelona*. The algorithm runs four times with four different number of processors(1, 2, 4, and 8 respectively).

Their results show that the algorithm SPVNS finds the same objective value using different number of processors. The objective values obtained with SPVNS were worse than the best known objective values. The comparison between the results of the four methods (the sequential VNS and the three parallelizations) showed that the speed up increased with the number of processors. However, the linearity was not reached due to the concurrent access to the data in the shared memory. Some computer results on this instance of the problem have been reported in [Hansen and Mladenovic, 1997], where several heuristics (including a basic VNS) were compared.

They reported results of the algorithms RPVNS and RSVNS using 2, 4, and 8 processors. The

best results were obtained for the algorithm RPVNS which gets CPU times near to that obtained by the sequential algorithm and in the most of cases it gets better objective values. Also they are better when the number of processors increases. The algorithm RSVNS provides worse results than RPVNS both in CPU time and objective values.

The work by [Crainic et al., 2004] extends to the VNS the success of the cooperative multi search method that had been applied to a number of difficult combinatorial optimization problems [Crainic and Gendreau, 2002]. They carried out extensive experimentations on the classical TSPLIB benchmark problem instances with up to 11948 demand points and 1000 medians. Their results indicate that the cooperative strategy yield, compared with the sequential VNS, significant gains in terms of computation time without a loss in solution quality.

The CNVNS and the sequential VNS for comparisons where coded in Fortran77. The cooperative parallel strategy was implemented using MPI. Computational experiments were performed on a 64-processor SUN Enterprise 1000 with 400 MHz clock and 64 gigabytes of RAM. Tests were run using 5, 10, 15 and 1 for the sequential version.

Note that, since VNS includes a random element int shake, solving the same problem repeatedly may yield different solutions. Therefore, the comparisons have to be based on average values taking int account the standard deviations. However standard deviations where extremely low or very low in all the case. This facts indicate that both the sequential and the cooperative multi-search parallel are robust with respect to the random move in the shake step.

VNS is based on the idea of aggressive exploration of neighborhoods, that is, on generation and evaluation many different solutions. Consequently, when the evaluation of the moves is not expensive in computing time (as is in the  $p$ -median instances using the fast interchange), the communication overhead associated to parallel computation result in less search time and generally somewhat lower quality solutions for the same total search time.

## 6 Conclusions

The combination of the Variable Neighborhood Search and parallelism provides a useful tool to solve hard problems. The VNS, as a combination of series of random and local searches, is parallelizable in several ways. Two simple parallelization strategies are the Synchronous Parallel VNS (SPVNS)

that is obtained by parallelizing the local search and the Replicated Parallel VNS (RPVNS) that is obtaining by parallelizing the whole procedure so that each processor runs in parallel a VNS. These parallelizations provide with the basic advantages of the parallel procedures. However using cooperative mechanisms, the performance is improved by the Replicated-Shaking VNS (RSVNS) proposed in [García López et al., 2002] that applies a synchronous cooperation mechanism through a classical master-slave approach and additionally improved by the *Cooperative Neighbourhood VNS* (CNVNS) proposed in [Crainic et al., 2004] that applies a cooperative multi-search method based on a central-memory mechanism.

## References

- [1] J.E. Beasley. A note on solving large  $p$ -median problems, *European Journal of Operational Research*, 21(1985), 270-273.
- [2] T.G. Crainic, M. Gendreau, P. Hansen and N. Mladenović, Cooperative parallel variable neighbourhood search for the  $p$ -median. *Journal of Heuristics* 10(2004), 293-314.
- [3] T.G. Crainic, M. Gendreau, Cooperative Parallel Tabu Search for the Capacitated Network Design. *Journal of Heuristics* 8-6(2002), 601-627.
- [4] J.A. Díaz, E. Fernández, Scatter Search and Path Relinking for the capacitated  $p$ -Median problem, *European Journal of Operational Research* (2004), forthcoming.
- [5] L.A.N. Lorena, E.L.F. Senne, A column generation approach to capacitated  $p$ -median problems *Computers and Operations Research* 31-6(2004), 863-876.
- [6] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(1978), 992-1009.
- [7] F. García López, B. Meli'an Batista, J.A. Moreno Pérez and J.M. Moreno Vega, The parallel variable neighbourhood search for the  $p$ -median problem. *Journal of Heuristics*, 8(2002), 375-388.
- [8] F. Glover, M. Laguna. (1998) *Tabu Search*, Kluwer Academic Publishers, Norwell, MA.

- [9] P. Hanjoul, D. Peeters. A comparison of two dual-based procedures for solving the  $p$ -median problem. *European Journal of Operational Research*, 20(1985), 387-396.
- [10] P. Hansen, B. Jaumard. Cluster Analysis and mathematical programming. *Mathematical Programming*, 79(1997), 191-215.
- [11] P. Hansen and N. Mladenovic. Variable Neighborhood Search for the  $p$ -Median. *Les Cahiers du GERAD G-97-39*, Montreal, Canada (1997).
- [12] P. Hansen and N. Mladenovic. Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization. *Les Cahiers du GERAD G-2000-3*, Montreal, Canada (2000).
- [13] P. Hansen, N. Mladenovic and D. Pérez-Brito. Variable Neighborhood Decomposition Search. *Journal of Heuristics* 7-4(2001) 335-350.
- [14] P. Hansen, N. Mladenovic. An Introduction to variable neighborhood search. In S. Voss et al. (eds), *Proceedings of the 2nd International Conference on Metaheuristics - MIC97*, Kluwer, Dordrecht. (1999).
- [15] O.Kariv, S.L.Hakimi. An algorithmic approach to network location problems; part 2. The  $p$ -medians. *SIAM Journal on Applied Mathematics*, 37(1969), 539-560.
- [16] A.A. Kuehn, M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4) (1963), 643-666.
- [17] F.E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly*, 12(1964), 138-139.
- [18] N. Mladenovic. A variable neighborhood algorithm-A new metaheuristic for combinatorial optimization. Presented at Optimization Days, Montreal. (1995).
- [19] N. Mladenovic and P. Hansen. Variable Neighborhood Search. *Computers Oper. Res.* 24 (1997), 1097–1100.
- [20] N. Mladenovic, J.A. Moreno-Pérez, and J. Marcos Moreno-Vega. A Chain-Interchange Heuristic Method, *Yugoslav J. Oper. Res.* 6 (1996), 41-54.

- [21] M.B. Teitz, P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5) (1968), 955-961.
- [22] S. Voss. A reverse elimination approach for the p-median problem. *Studies in Locational Analysis*, 8(1996), 49-58.