Les Cahiers du GERAD ISSN: 0711–2440

Parallel Variable Neighborhood Search

J.A. Moreno Pérez, P. Hansen
N. Mladenović

G–2004–92

December 2004

Parallel Variable Neighborhood Search

José A. Moreno Pérez
Universidad de La Laguna
Spain
jamoreno@ull.es

Pierre Hansen
GERAD and HEC Montreal
Canada
pierre.hansen@gerad.ca

Nenad Mladenović
GERAD and Mathematical Institute (SANU)
Belgrade
nenad@mi.sanu.ac.yu

December 2004

Les Cahiers du GERAD
G–2004–92

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.

Résumé

La Recherche à Voisinage Variable (RVV) est une métaheuristique récente et efficace pour la résolution de problèmes d'optimisation combinatoire ou globale. Elle permet de quitter un optimum local par des changements systématiques de structures de voisinage à l'intérieur de la recherche. Pour cet article plusieurs stratégies de parallélisation de RVV sont proposées et composées sur de grandes instances de problème de la p-médiane.

Acknowledgments: 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 founds.

1 Introduction

A combinatorial optimization problem consists of finding, the minimum or maximum of a real valued function f defined on a discrete or partially discrete set. If it is a minimization problem, it can be formulated as follows

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

Here X is called *solution space*, x represents a *feasible solution* and f the *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 best known of them are Genetic Search, Simulated Annealing and Tabu Search (for discussion of these metaheuristics and others, the reader is referred to the books of surveys edited by Reeves [26] and Glover and Kochenberger [9]). Variable Neighborhood Search (VNS) ([23, 24, 12, 13, 14, 15]) 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.

Hence, VNS proceeds by a descent method to a local minimum, then explore a series of different predefined neighborhoods of this solution. Each time, one or several points of the current neighborhood are used as starting point for a local descent method, that stops at a local minimum. The search jumps to the new local minimum if and only if it is better than the incumbent. In this sense, VNS is not a trajectory following method (that allows non-improving moves within the same neighborhood) as Simulated Annealing or Tabu Search.

The application of parallelism to a metaheuristic can and must allow to reduce the computational time (by a partition of the sequential program) or to increase the exploration in the search space (by 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. The next section describes the basics of the VNS metaheuristic. Several parallelization strategies for VNS are analyzed in Section 3. In Section 4 analyze the characteristics of the VNS applied to solve p-median problem. Computational experiments and conclusions are presented in Sections 5 and 6 respectively.

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 following 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:

- A local minimum with respect to one neighborhood structure is not necessary so for another;
- A global minimum is a local minimum with respect to all possible neighborhood structures.
- 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 a deterministic version of VNS. It is based on *Fact 1* above, i.e., *a local optimum for a first type of move* $x \leftarrow x'$ (or heuristic, or within the neighborhood $N_1(x)$) *is not necessary one for another type of move* $x \leftarrow \tilde{x}$ (within neighborhood $N_2(x)$)). It may thus be advantageous to combine descent heuristics. This leads to the basic VND scheme presented in Figure 1.

Another simple application of the VNS principle is 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 described above. A set of neighborhoods $N_1(x), N_2(x), \ldots, N_{kmax}(x)$ 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 the maximum computing time since the last improvement, or the maximum number of iterations). The description of the steps of Reduced VNS is as shown in Figure 2.

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

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 $\ell \leftarrow \ell + 1$;

Figure 1: Variable Neighborhood Descent Method

RVNS method

1. Find an initial solution x; choose a stopping condition;
2. Repeat the following until a stoping 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 $k \leftarrow k + 1$.

Figure 2: Reduced Variable Neighborhood Search Method

discuss how to combine a local search with systematic changes of neighborhoods around the local optimum found. We then obtain the Basic VNS scheme of Figure 3.

The simplest basic VNS is sometimes called Iterated Local Search [20]. The method gets by a perturbation a neighbor of the current solution, makes a local search from it to a local optimum, and moves to it if there has been an improvement. The steps of the simplest VNS are obtained taking only one neighborhood (see Figure 4).

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) shown in Figure 5.

Then a C code for the simple version of sequential Variable Neighborhood Search is shown in Figure 6.

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 $U = \{u_1, ..., u_n\}$, the local search and the shake based on the interchange moves can also

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 $k \leftarrow k + 1$;

Figure 3: Basic Variable Neighborhood Search Method

VNS algorithm

1. Initialization:
   Find an initial solution x. Set $x^* \rightarrow x$.
2. Repeat the following until a stoping condition is met.
   (a) Shake:
   Take (at random) a solution x' in $\mathcal{N}(x)$.
   (b) Local Search:
   Apply the local search method with x' as initial; denote x'' the so obtained local optimum.
   (c) Improve or not:
   If x'' is better than x^*, do $x^* \rightarrow x''$.

Figure 4: Simple Variable Neighborhood Search Algorithm

be implemented using a function `exchange` also provided for the user for his problem. A solution S is represented by an array $S = [u_i : i = 1, ..., n]$ where u_i is the i-th element of the solution, for $i = 1, 2, ..., p$, and the $(i - p)$-th element outside the solution, for $i = p + 1, ..., n$.

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 u_i and u_j, for $i = 1, ..., p$ and $j = p + 1, ..., n$. The pseudocode of the local search is in Figure 7.

The code C for the sequential local search (SLS) is shown in Figure 8.

GVNS algorithm

1. *Initialization.*
   Select the set of neighborhood structures $\mathcal{N}_k$, for $k = 1, \ldots, k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_ℓ for $\ell = 1, \ldots, \ell_{max}$ 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^{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 $k \leftarrow k + 1$;

Figure 5: General Variable Neighborhood Search Method

Sequential VNS Algorithm

```
initialize(best_sol);
k = 0 ;
while (k < k_max) {
  k++ ;
  cur_sol = shake(best_sol,k) ;
  local search(cur_sol) ;
  if improved(cur_sol,best_sol)
    best_sol = cur_sol ;
    k = 0 ;
 } /* if */
} /* while */
```

Figure 6: Sequential Variable Neighborhood Search Algorithm

Local Search

Initialize S'

Repeat

$S \leftarrow S'$

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

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

Figure 7: Local Search

Sequential Local Search

```
void seq_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 */
} /* seq_local_search */
```

Figure 8: Sequential Local Search Pseudocode

The Shake procedure consists of, given the size k for the shake, choosing k times two points at random, u_i and u_j; u_i in the solution and u_j outside the solution, and performing the corresponding interchange move (see Figure 9).

The C code for the shake procedure is in Figure 10.

3 The parallelizations

The application of parallelism to a metaheuristic can and must allow to reduce the computational time (by the partition of the sequential program) or to increase the exploration in the search space (by 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 and analyzed in the literature (see [8, 4]). The parallel VNS heuristics reported were coded in C (using the OpenMP, a model for parallel programming portable across shared memory architectures) in [8] and in Fortran 90 (using MPL) in [4].

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.

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 $S \leftarrow S_{ij}$

Figure 9: Shake Procedure

Sequential Shake

```
void seq_shake(sol cur_sol)
{
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 */
} /* seq_shake */
```

Figure 10: Sequential Shake Pseudocode

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 [35]). Only a few lines of the sequential code had to be replaced by specific directives of the OpenMP compiler in the pseudocode to get the code of the parallel programs used in the computational experiments.

Four different parallelization strategies have been reported in the literature; two are simple parallelization and the other two are more complex strategies. The two simple parallelizations of the VNS consist of parallelizing the local search and of replicating the whole VNS in the processors. The other two additional parallelization strategies are proposed in [8] and [4], and have been tested with known large instances of the p-Median Problem.

The first of the two simple parallelization strategies analyzed in [8] 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 divides 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. Figure 11 shows the code of the procedure `par_local_search` that implements parallel local search.

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 */
```

Figure 11: Parallel Local Search Pseudocode

Then the pseudocode of the *Synchronous Parallel Variable Neighborhood Search* SPVNS is shown in Figure 12.

The second simple parallelization of the VNS is the Replicated Parallel VNS (RPVNS) that tries to search for a better solution by means of the exploration of a wider zone of the solution space, using multistart strategies. It is done by increasing the number of neighbor solutions 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 search is replaced by the VNS. The pseudocode of the RPVNS is described in the Figure 13.

The two additional parallelization strategies use cooperation mechanisms to improve the performance. The Replicated-Shaking VNS parallelization (RSVNS) of the VNS proposed in [8] applies a synchronous cooperation mechanism through a classical master-slave approach. The *Cooperative Neighbourhood VNS* (CNVNS) parallelization proposed in [4] 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

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 */
```

Figure 12: Synchronous Parallel Variable Neighborhood Search 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 */
```

Figure 13: Replicated Parallel Variable Neighborhood Search Pseudocode

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) pseudocode is described in Figure 14.

The *Cooperative Neighbourhood VNS* (CNVNS) proposed by [4] is obtaining by applying the cooperative multi-search method to the VNS metaheuristic. This parallelization

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 */
```

Figure 14: Replicated-Shaking Parallel Variable Neighborhood Search Pseudocode

method is based on the central-memory mechanism that has been successfully applied to a number of different combinatorial problem. In this approach, several independent VNS's 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 implement 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 processors 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 any more

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 the continued starting from the best overall solution in the current neighborhood. The CNVNS procedure is summarized as follows:

Cooperative Neighborhood 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 communicate it back to the requesting VNS process.
  - Verifies the stopping condition
- Each VNS process
  - Receives the initial solution, selects randomly a neighborhood and explores 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.

The pseudo-codes, similar to the above parallelizations, of the master and workers procedures of the Cooperative Neighbourhood parallel Variable Neighborhood Search are shown in Figures 15 and 16.

4 Application of VNS for the p-median

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

4.1 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.

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 */
```

Figure 15: Master CNVNS Pseudocode

Algorithm CNVNS

Master process

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
```

Figure 16: Workers CNVNS Pseudocode

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 users. Several formulations and extensions of this optimization problem are useful to model many real word 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 admits a combinatorial formulation. This formulation is as follows.

Let $L = \{v_1, v_2, ..., v_m\}$ be the finite set of potential location for the p medians, and $U = \{u_1, u_2, ..., u_n\}$ be the set of demand points for the facility. Consider also a weight vector $w = [w_i : i = 1, ..., n]$ 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 $i = 1, ..., n$, $j = 1, ..., m$; i.e.,

$$D = [d_{ij} : i = 1, ..., n, j = 1, ..., m] = [Dist(u_i, v_j) : i1, ..., n, j = 1, ..., m].$$

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.,

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

where $X \subseteq L$ and $|X| = p$. The capacitated version (see [6, 19]) includes a fixed capacity for the facility center located at each point of L that bounds 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 $y = [y_j : j = 1, ..., m]$ and the allocation variables $x = [x_{ij} : i = 1, ..., n, j = 1, ..., n]$. The meaning of these variables 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:

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

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 [17]. Extensive references to works on this and related problems are contained in the main books, surveys and reviews in Location like [3, 2, 22, 7].

Many heuristics and exact methods have been proposed for solving it. Exact algorithms were developed in [1], [10] and others. Classical heuristics for the p-median problem often cited in the literature are *Greedy* [18], *Alternate* [21] and *Interchange* [33]. The basic Greedy heuristics starts with an empty set and repeat the following greedy step; the facility point that least increase 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 medians, iteratively interchanges a facility points in the median set with another facility point out of the median set.

Among other metaheuristics (see [29, 34, 25, 28, 30, 31, 32, 27]) the VNS and its variants have been applied to the p-median problems (see [11, 16, 13, 15]).

4.2 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 are no 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 $i = 1, ..., n$. 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 solution for the searches. They consist of adding elements to an empty solution until a set of p points is obtained. 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 $S = [v_i : i = 1, ..., n]$ where v_i is the i-th element of the solution, for $i = 1, 2, ..., p$, and the $(i - p)$-th element outside the solution, for $i = p + 1, ..., n$.

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:

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

and the second best allocation cost of v_i, $i = 1, ..., n$, 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 [16].

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:

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

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 small and the difference between pn and $pn_i + n$ is important in shaking and local searches.

5 Computational experience

The algorithms of [8] 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 numbers of processors(1, 2, 4, and 8 respectively).

The 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 [11], 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 those obtained by the sequential algorithm and in most 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 [4] extends to the VNS the success of the cooperative multi search method that had been applied to a number of difficult combinatorial optimization problems [5]. 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 yields, 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 comparison purposes 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 in the shake, solving the same problem repeatedly may yield different solutions. Therefore, the comparisons have to be based on average values taking into account the standard deviations. However standard deviations where extremely low or very low in all cases. This fact indicates 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 of many different solutions. Consequently, when the evaluation of the moves is not expensive in computing time (as is the case for the p-median instances using the fast interchange), the communication overhead associated to parallel computation results 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 the basic advantages of the parallel procedures. However using cooperative mechanisms, the performance is improved by the Replicated-Shaking VNS (RSVNS) proposed in [8] that applies a synchronous cooperation mechanism through a classical master-slave approach and additionally improved by the *Cooperative Neighborhood VNS* (CNVNS) proposed in [4] 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] M.L. Brandeau and S.S. Chiu. An overview of representative problems in location research. *Management Science* 35 (1989) 645–674.

[3] G. Cornuejols, M.L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science* 23 (1977) 789–810.

[4] 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.

[5] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for the capacitated network design. *Journal of Heuristics* 8 (2002) 601–627.

[6] J.A. Díaz and E. Fernández. Scatter search and path relinking for the capacitated p-median problem. *European Journal of Operational Research* (2004), forthcoming.

[7] Drezner, Z. (ed.) *Facility Location: A Survey of Applications and Methods*. Springer, 1995.

[8] 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.

[9] F. Glover and G. Kochenberger (eds.) *Handbook of Metaheuristics*. Kluwer, 2003.

[10] P. Hanjoul and D. Peeters. A comparison of two dual-based procedures for solving the p-median problem. *European Journal of Operational Research* 20 (1985) 387–396.

[11] P. Hansen and N. Mladenovic. Variable neighborhood search for the p-median. *Location Science* 5 (1997) 207–226.

[12] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In: S. Voss et al. eds., *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer (1999) 433–458.

[13] P. Hansen and N. Mladenović. Developments of variable neighborhood search. In: C. Ribeiro, P. Hansen (eds.), *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, Boston/Dordrecht/London (2001) 415–440.

[14] P. Hansen and N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research* 130 (2001) 449–467.

[15] P. Hansen and N. Mladenovic. Variable Neighborhood Search. In: F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, Kluwer (2003) 145–184.

[16] P. Hansen, N. Mladenovic, and D. Pérez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics* 7 (2001) 335–350.

[17] O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems; part 2. The p-medians. *SIAM Journal on Applied Mathematics* 37 (1969) 539–560.

[18] A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Science* 9 (1963) 643–666.

[19] L.A.N. Lorena and E.L.F. Senne. A column generation approach to capacitated p-median problems. *Computers and Operations Research* 31 (2004) 863–876.

[20] H.R. Lourenco, O. Martin, and T. Stuetzle. Iterated Local Search. In: F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*. Kluwer (2003) 321–353.

[21] F.E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly* 12 (1964) 138–139.

[22] P. Mirchandani and R. Francis (eds.) *Discrete location theory*. Wiley-Interscience, 1990.

[23] N. Mladenovic. A variable neighborhood algorithm-A new metaheuristic for combinatorial optimization. Presented at Optimization Days, Montreal (1995) pp. 112.

[24] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers Oper. Res.* 24 (1997) 1097–1100.

[25] N. Mladenovic, J.A. Moreno-Pérez, and J. Marcos Moreno-Vega. A chain-interchange heuristic method. *Yugoslav J. Oper. Res.* 6 (1996) 41–54.

[26] C.R. Reeves. *Modern heuristic techniques for combinatorial problems*. Blackwell Scientific Press, 1993.

[27] M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics* 10 (2004) 59–88.

[28] E. Rolland, D.A. Schilling, and J.R. Current. An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research* 96 (1996) 329–342.

[29] K.E. Rosing, C.S. ReVelle, and H. Rosing-Vogelaar. The p-median and its linear programming relaxation: An approach to large problems. *Journal of the Operational Research Society* 30 (1979) 815–823.

[30] K.E. Rosing and C.S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research* 97 (1997) 75–86.

[31] E.L.F. Senne and L.A.N. Lorena. Lagrangean/surrogate heuristics for p-median problems. In: M. Laguna and J. L. González-Velarde, eds., *Computing Tools for Modeling Optimization and Simulation: Interfaces in Computer Science and Operations Research*, Kluwer (2000) 115–130.

[32] E.L.F. Senne and L.A.N. Lorena. Stabilizing column generation using Lagrangean/surrogate relaxation: an application to p-median location problems. *European Journal of Operational Research* (2002), forthcoming.

[33] M.B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research* 16 (1968) 955–961.

[34] S. Voss. A reverse elimination approach for the p-median problem. *Studies in Locational Analysis* 8 (1996) 49–58.

[35] 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`).