Journal of Heuristics, 8: 375–388, 2002

The Parallel Variable Neighborhood Search for the *p*-Median Problem*

FÉLIX GARCÍA-LÓPEZ, BELÉN MELIÁN-BATISTA, JOSÉ A. MORENO-PÉREZ
AND J. MARCOS MORENO-VEGA
Departamento de Estadística, Investigación Operativa y Computación, Universidad de La Laguna, 38271 La Laguna, Spain
email: fcgarcia@ull.es
email: mbmelian@ull.es
email: jamoreno@ull.es
email: jmmoreno@ull.es

Abstract

The Variable Neighborhood Search (VNS) is a recent metaheuristic that combines series of random and improving local searches based on systematically changed neighborhoods. When a local minimum is reached, a shake procedure performs a random search. This determines a new starting point for running an improving search. The use of interchange moves provides a simple implementation of the VNS algorithm for the *p*-Median Problem. Several strategies for the parallelization of the VNS are considered and coded in C using OpenMP. They are compared in a shared memory machine with large instances.

Key Words: parallel computing, VNS, *p*-median, OpenMP

1. Introduction

The Variable Neighborhood Search (VNS) metaheuristic is capable of escaping local optima by systematic changes of the neighborhood structure. A Local Search (LS) consists of selecting a better neighbor solution until no such solution exists. The basic VNS obtains a neighbor solution from the current solution and runs a local search procedure to reach a local optimum. If an improved solution is obtained, then it is the new current solution. Otherwise, the neighborhood structure is changed and the shake procedure looks for another starting solution to perform a new local search. We implement the local search and the shake procedures based on the interchange moves. The local search consists of applying the best improving interchange until no such move exists. The shake procedure consists of selecting randomly a solution from neighborhood.

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 belongs to a wide class of hard combinatorial problems, where the solutions are generated by selecting p items from a finite universe. The evaluation of the objective function in location/allocation problems ranges from the simple calculation of

*This research has been partially supported by Gobierno de Canarias through the project PI1999/116.

a function value to solve another hard problem or to perform a simulation process. The standard moves for location/allocation problems are the interchange or swap moves that exchange an item in the solution with another one out of the solution.

In this paper we propose the parallelization of the VNS 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. On one hand, the increase of the efficiency is obtained by performing the same steps in less time than the sequential algorithm. On the other hand, the increase of the exploration is obtained by performing more steps in the same time as the sequential algorithm. Several strategies for parallelizing a VNS algorithm are analyzed. We test them with large instances of the p-median problem obtained from the TSPLIB (1995). The corresponding parallel algorithms are coded in C using OpenMP, a model for parallel programming portable across shared memory architectures. The OpenMP (OpenMP, 1997) 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.

Next section describes the basics of the VNS metaheuristic. Section 3 summarizes the p-median problem, and the application of the VNS to it is considered in Section 4. Several parallelization strategies are analyzed in Section 5. Finally, the computational results and conclusions end the paper.

2. The VNS metaheuristic

The 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. Each solution S has a neighborhood associated $\mathcal{N}(S)$, that consists of the solutions that can be reached from S by a move. At each iteration, the local search procedure obtains an improved solution S' in the neighborhood $\mathcal{N}(S)$ of the current solution S, until no further improvement is found. Several metaheuristics extend this scheme to avoid being trapped in a local optimum. The most famous of these metaheuristics are Simulated Annealing (Kirkpatrick, Gelatt, and Vecchi, 1983), Genetic Algorithms (Holland, 1975) and Tabu Search (Glover, 1989, 1990). Variable Neighborhood Search (Hansen and Mladenovic, 1999; Mladenovic, 1995) is a promising new technique that tries to escape local optima by changing the neighborhood structure. The basic VNS gets a neighbor of the current solution, runs a local search from it to get a local optimum, and moves to it if there has been an improvement. Otherwise, the neighborhood structure is systematically changed. Usual stopping conditions are based on allowing a total maximum computational time or a maximum computational time since the last improvement. The computational effort is measured by the real time, the number of iterations, or the number of local searches.

The neighborhoods considered are nested and they are based on a set of standard or base moves. Given a base neighborhood structure $\mathcal{N}$, the series of nested neighborhood structures is defined as follows. The first neighborhood structure is the base one $\mathcal{N}_1(S) = \mathcal{N}(S)$, and the next neighborhood structures are recursively defined by:

$$\mathcal{N}_k(S) = \bigcup_{S' \in N_1(S)} \mathcal{N}_{k-1}(S').$$

Then the VNS comprises the following steps:

VNS algorithm

1. Initialization:
   Find an initial solution S^*. Set $k \leftarrow 1$.
2. Repeat the following steps until the stopping condition is met:
   (a) Shake Procedure:
   Set $S \leftarrow S^*$. Generate at random a starting solution S' in $\mathcal{N}_k(S)$.
   (b) Local Search:
   Apply a local search from the starting solution S' using the base neighborhood structure $\mathcal{N}(\cdot)$ until a local minimum S'' is found.
   (c) Improve or not:
   If S'' is better than S^*, do $S^* \leftarrow S''$ and $k \leftarrow 1$. Otherwise do $k \leftarrow k + 1$.

If the local search uses greedy strategy, then at Step 2(b) an iterative procedure tests all the base moves providing the best neighbor solution until a local minimum is obtained. The shake procedure selects randomly a solution from $\mathcal{N}_k$.

The base moves for the p-median problem are the interchange moves that swap a point in the solution by another one out of the solution. The large set of problems where each solution is given by a selection of a set of p elements from a universe shows the wide applicability of this basic version of VNS metaheuristic. In the next section the application of VNS to the p-median problem is described in more detail.

3. The p-median problem

Consider a set L of m potential locations for p facilities and a set U of locations for n given users. The p-median problem consists of locating simultaneously p facilities at locations of L in order to minimize the total transportation cost for satisfying the demand of the users. Each user is supplied from its closest facility.

The p-median problem and its extensions are useful to modelize many real world situations, such as the location of industrial plants, warehouses and public facilities. The p-median problem can be defined as a purely mathematical problem. Given an $n \times m$ matrix D, select p columns of D such that the sum of minimum coefficients in each line within these columns is as small as possible.

Given the set $L = \{v_1, v_2, \ldots, v_m\}$ of potential locations for the facilities (or location points, or medians), and the set $U = \{u_1, u_2, \ldots, u_n\}$ 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 is to minimize the sum of these distances (or transportation costs), i.e.,

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

where $S \subseteq L$ and $|S| = p$. Besides this combinatorial formulation, the p-median problem has a formulation in terms of integer programming. It is the problem:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij} x_{ij} \\ \text{subject to} \quad & \\ & \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, \ldots, n \\ & x_{ij} \leq y_j, \quad i = 1, \ldots, n, \; j = 1, \ldots, m \\ & \sum_{j=1}^{m} y_j = p, \\ & x_{ij}, y_j \in \{0, 1\}, \quad i = 1, \ldots, n, \; j = 1, \ldots, m \end{aligned}$$

where $y_j = 1$ means that a facility is located at v_j (0, otherwise) and $x_{ij} = 1$ means that the user located at u_i is assigned to facility v_j (0, otherwise).

The p-median problem is $\mathcal{NP}$-hard (Kariv and Hakimi, 1969). Many heuristics and exact methods have been proposed for solving it. Exact algorithms are provided by Beasley (1985) and by Hanjoul and Peeters (1985), among others. 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). 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 (Erlenkotter, 1978). In Mladenovic, Moreno-Pérez, and Moreno-Vega (1996) 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 (1996), where some variants of the so-called reverse elimination method are discussed. The VNS heuristic and its variants have also been applied to the p-median problems (Hansen and Mladenovic, 1997, 2000; Hansen, Mladenovic, and Pérez-Brito, 2001).

4. Application of VNS to p-median problem

For most instances, the set of potential locations for the facilities and the set of locations for 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 S 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:

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

Regardless the VNS 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 S is encoded by arranging all the points of U in an array $[v_i : i = 1, \ldots, n]$ where v_i is a point in S for $i \leq p$, and it is a point out of S for $i > p$.

The base moves for the p-median problem are the swap or interchange moves. For every solution S, 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 S and adding v_j to S.

Usually a constructive heuristic is used to generate an initial solution. The procedure consists of adding elements to an empty solution until a set of p points is obtained. The best strategies for adding points to the solution are the use of simple, random and good rules to select the point to be added.

The cost of a solution S is given by:

$$Cost(S) = \sum_{i=p+1}^{n} \min_{j=1..p} Dist(v_i, v_j).$$

An optimal solution S^* is a solution that minimizes $Cost(S)$.

The computation of the cost of the new solution after each move is simplified by storing the best allocation costs and the second best allocation costs in two vectors. The vector named $Cost1(\cdot)$ with the best allocation costs is defined by:

$$Cost1(i) = \min_{j=1..p} Dist(v_i, v_j), \quad \text{for } i = 1, \ldots, n.$$

The second best allocation cost of v_i is stored in the entry $Cost2(\cdot)$ given by:

$$Cost2(i) = \min_{\substack{j=1,\ldots,p \\ j \neq r(i)}} Dist(v_i, v_j), \quad \text{for } i = 1, \ldots, n.$$

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 the VNDS reported in Hansen, Mladenovic, and Pérez-Brito (2001).

Given the solution S, let S_{ij}, $1 \leq i \leq p$ and $p < j \leq n$, be the new solution obtained by interchanging v_i and v_j. The cost of the new solution is computed as follows:

$$Cost(S_{ij}) = \min \left\{ Dist(v_i, v_j), \min_{\substack{l=1,\ldots,p \\ l \neq i}} Dist(v_i, v_l) \right\} + \sum_{k=p+1}^{n} \min \left\{ Dist(v_k, v_j), \min_{\substack{l=1,\ldots,p \\ l \neq i}} Dist(v_k, v_l) \right\}.$$

The use of this formula implies $O(pn)$ operations. However, using the values in $Cost1$ and $Cost2$, the cost $Cost(S_{ij})$ is obtained in $O(pn_i + n)$ time, where n_i is the number of points assigned to point v_i (see Hansen, Mladenovic, and Pérez-Brito, 2001). Note that if p is large then n_i must be short and the difference between pn and $pn_i + n$ is important.

The shake procedure, already used in the VNS heuristic in Hansen and Mladenovic (1999), allows to escape local minima without destroying completely the good properties of the current solution. Given a size k for the shake procedure, a solution from $N_k(S)$ is randomly selected.

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 by the following template.

Local Search (S')

Repeat
  Set $S'' \leftarrow S'$.
  For every i and j with $1 < i \leq p$ and $p < j \leq n$ do:
    Set $S_{ij} \leftarrow S'' - \{v_i\} + \{v_j\}$.
    If $Cost(S_{ij}) < Cost(S')$ then $S' \leftarrow S_{ij}$.
Until $Cost(S') = Cost(S'')$.

The parameter k, that controls the shake procedure, is initialized to 1. Each time a local minimum is reached the shake procedure is applied. If the current local minimum is better than the best known solution, k is set again to 1. Otherwise k is augmented by one unit. As stopping criterion, the maximum value for k is fixed at $k_{\max}$.

The pseudocode of the sequential Variable Neighborhood Search (VNS) is as follows.

Algorithm VNS

Initialize S and set $k \leftarrow 1$.
Repeat
  Apply the Shake Procedure with size k to S to get S'.
  Apply the Local Search to S' to get S''.
  If $Cost(S'') < Cost(S)$ then $S \leftarrow S''$ and set $k \leftarrow 1$.
  Otherwise $(Cost(S'') \geq Cost(S))$ do $k \leftarrow k + 1$.
Until $k > k_{\max}$.

To implement this metaheuristic for another problem the user has to provide the initialization method, the shake procedure, the local search and the objective function.

5. The parallelization

The main purpose of parallel processing is to perform computations faster than can be done with a single processor by using a number of processors concurrently. The pursuit of this goal 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 parallelizations for the VNS algorithm. The first parallelization reduces the running time of the local search algorithm. The last two parallelizations increase the exploration in the solution space.

5.1. Synchronous parallel VNS (SPVNS)

In the sequential VNS 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 VNS algorithm that parallelizes the local search by SPVNS. The following template shows the pseudocode of the SPVNS with n_pr processors:

Algorithm SPVNS

Initialize S and set $k \leftarrow 1$.
Repeat
  Apply the Shake Procedure with size k to S to get S'.
  Take $I \leftarrow \{(i, j) : 1 \leq i \leq p,\ p < j \leq n\}$.
  Divide I in n_pr subsets $I_r, r = 1, \ldots, n\_pr$.
  Repeat
    Set $S'' \leftarrow S'$
    For each processor $r = 1, \ldots, n\_pr$, do in parallel
      Set $S_r \leftarrow S''$
      For every $(i, j) \in I_r$ do
        Set $S_{ij} \leftarrow S'' - \{v_i\} + \{v_j\}$.
        If $Cost(S_{ij}) < Cost(S_r)$ then $S_r \leftarrow S_{ij}$
    For $r = 1, \ldots, n\_pr$ do
      If $Cost(S_r) < Cost(S')$ then $S' \leftarrow S_r$.
  Until $Cost(S') = Cost(S'')$.
  If $Cost(S'') < Cost(S)$ then $S \leftarrow S''$ and set $k \leftarrow 1$.
  Otherwise $k \leftarrow k + 1$.
Until $k > k_{\max}$.

Note that this pseudocode can be obtained from the sequential VNS by replacing the local search by the parallel version of the local search. In SPVNS, the neighborhood, $\{(i, j) : 1 \leq i \leq p,\ p < j \leq n\}$, is divided in n_pr subsets I_r. The subsets are assigned to the processors and each returns an improving neighbor in its partition. The best neighbor is chosen as the current solution. This strategy is a low-level parallelism.

5.2. Replicated parallel VNS (RPVNS)

In order to explore a wider zone of the solution space, we propose the Replicated Parallel VNS algorithm, that runs several VNS procedures. It is a multistart procedure where each local search is replaced by a VNS. Each available processor performs a sequential VNS

algorithm. The parallel algorithm returns the best solution obtained by the processors. This parallelization is described by the following pseudocode:

Algorithm RPVNS

For each processor r, for $r = 1, \ldots, n\_pr$, do in parallel
  Initialize S_r.
  Set $k_r \leftarrow 1$.
  Repeat
    Apply the Shake Procedure with size k_r to S_r to get S'_r.
    Apply the Local Search to S'_r to get S''_r.
    If $Cost(S''_r) < Cost(S_r)$ then $S_r \leftarrow S''_r$ and set $k_r \leftarrow 1$.
    Otherwise $k_r \leftarrow k_r + 1$.
  Until $k_r > k_{\max}$.
Set $S'' \leftarrow S_1$.
For $r = 2, \ldots, n\_pr$ do
  If $Cost(S_r) < Cost(S'')$ then $S'' \leftarrow S_r$.

5.3. Replicated shaking VNS (RSVNS)

We propose another parallel algorithm that also increase the exploration in the search space. The shake and local search procedures are replicated as many times as the number of available processors. The best local optimum found by the processors is used to improve the intensification of the search. This method is the Replicated-Shaking VNS (RSVNS) that is described by the following pseudocode:

Algorithm RSVNS

Initialize S and set $k \leftarrow 1$.
Repeat
  Set $S'' \leftarrow S$.
  For each processor r, for $r = 1, \ldots, n\_pr$, do in parallel
    Apply the Shake Procedure with size k to S to get S'_r.
    Apply the Local Search to S'_r to get S''_r.
  For $r = 1, \ldots, n\_pr$ do
    If $Cost(S''_r) < Cost(S'')$ then $S'' \leftarrow S''_r$
  If $Cost(S'') < Cost(S)$ then $S \leftarrow S''$ and set $k \leftarrow 1$.
  Otherwise ($Cost(S'') \geq Cost(S)$) do $k \leftarrow k + 1$.
Until $k > k_{\max}$.

The RPVNS and the RSVNS parallelizations correspond to the natural parallelizations of the two hybrid metaheuristics between a VNS and a multistart search. The first one consists of a multistart search where the local searches are replaced by VNSs and the second one consists of a VNS where the single local search in the k-th neighborhood is replaced by a multistart search.

Since the objective of parallelism is time reduction then the performance of the parallelizations using measures of time (such as real time or speed-up) must be analyzed. Moreover, if the goal is to increase the exploration of the solution space and/or to compare several parallel algorithms, can be necessary other measures as work.

The real times reported in this paper are obtained by running the three parallel algorithms on a dedicated machine. The work is defined as the sum of real times needed for each processor.

We analyse the performance of the algorithms using real time and work functions. We now get theorical expressions for them and will show that the difference is mainly in the number of iterations, which will be used to compare the performance of the algorithms.

The real time functions for the parallelizations of the VNS algorithm depend on three parameters: the number of points n, the maximum shake size $k_{\max}$ and the number of available processors n_pr. The real time used in the computation of each algorithm is expressed in terms of: (1) the time used to obtain an initial solution T_{Ini}, (2) the number of iterations (number of local searches) performed by the algorithm *nIter*, (3) the time used to perform the shake procedure T_{Sh}, and (4) the time used to perform the local search T_{LS}. The initialization time and the local search time depend only on the size of the set of points n. The shake time depends on the shake size k, that is no greater than the time used for the maximum shake size $k_{\max}$. Then the real time of the sequential VNS algorithm is given by:

$$T_{VNS}(n, k_{\max}) = T_{Ini}(n) + nIter_{VNS} * (T_{LS}(n) + T_{Sh}(k_{\max})).$$

The parallel algorithm SPVNS tries to reduce the time for running the local search procedure. The computational effort of the local search is divided into the n_pr available processors. Then the time function for the SPVNS is given by:

$$\begin{aligned} T_{SPVNS}(n, k_{\max}, n\_pr) = {} & T_{Ini}(n) + nIter_{VNS} \\ & * (T_{LS}(\lceil n/n\_pr \rceil) + T_{Sh}(k_{\max}) + T_{Par}(n\_pr)) \end{aligned}$$

where $T_{Par}(n\_pr)$ is the time used to activate and synchronize the n_pr processors. We assume that all the processors are identical, so that the time needed to perform a parallel local search with n_pr processors is approximated by $T_{LS}(\lceil n/n\_pr \rceil)$, where $\lceil x \rceil$ represents the first integer number greater than or equal to x.

The only part of the parallel algorithm SPVNS that is perfomed by several processors is the local search. Since the factor $T_{Par}(n\_pr)$ is small compared to $T_{LS}(\lceil n/n\_pr \rceil) + T_{Sh}(k_{\max})$ and $T_{LS}(\lceil n/n\_pr \rceil) \simeq T_{LS(n)}/n\_pr$ then $W_{SPVNS}(n, k_{\max}, n\_pr)$ is similar to $T_{VNS}(n, k_{\max})$. This indicates that the parallel algorithm SPVNS must have almost linear reduction of the real time if the number of processors is small.

The Replicated Parallel VNS (RPVNS) performs as many VNS algorithms as the number of available processors. Each processor runs a sequential VNS procedure, and the RPVNS algorithm selects the best solution found among all the processors. The time used in the computation of the program is the maximum of the times needed for each processor. Therefore, it should be similar to that used in the computation of the sequential VNS procedure, except

the time used to activate and synchronize the processors, and the number of iterations; i.e.:

$$T_{RPVNS}(n, k_{\max}, n\_pr) = \max_{1 \le i \le n\_pr} \{T_{Ini}(n) + nIter_{RPVNS}(i) * (T_{LS}(n) + T_{Sh}(k_{\max}))\} + T_{Par}(n\_pr).$$

The work of the parallel algorithm RPVNS is given by:

$$W_{RPVNS}(n, k_{\max}, n\_pr) = \sum_{i=1}^{n\_pr} [T_{Ini}(n) + nIter_{RPVNS}(i) * (T_{LS}(n) + T_{Sh}(k_{\max}))] + T_{Par}(n\_pr).$$

Finally, the parallel algorithm RSVNS replicates the shake procedure as many times as the number of processors used and each performs also a local search. Therefore, the time should be similar to that used in the computation of the VNS procedure with one processor except the number of iterations; i.e.:

$$T_{RSVNS}(n, k_{\max}, n\_pr) = T_{Ini}(n) + nIter_{RSVNS} * (T_{LS}(n) + T_{Sh}(k_{\max}) + T_{Par}(n\_pr)).$$

The work of the parallel algorithm RSVNS is given by:

$$W_{RSVNS}(n, k_{\max}, n\_pr) = T_{Ini}(n) + n\_pr * nIter_{RSVNS} * (T_{LS}(n) + T_{Sh}(k_{\max}) + T_{Par}(n\_pr)).$$

Since the factor $T_{Par}(n\_pr)$ is again small compared to $T_{LS}(\lceil n/n\_pr \rceil) + T_{Sh}(k_{\max})$ then the work functions of the parallel algorithms RPVNS and RSVNS are compared attending to $\sum_{i=1}^{n\_pr} nIter_{RPVNS}(i)$ and $n\_pr * nIter_{RSVNS}$, respectively. These factors do not depend on the machine used to run the algorithms and allow to compare them. Therefore the number of iterations will be used in the next section to show the performance of both algorithms.

6. Computational results

The algorithms were coded in 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 1–4 going from 20 to 100. Some computational results on these instances of the problem have been reported in Hansen and Mladenovic (1997), where several heuristics (including a basic VNS) 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 real (wall clock) time in seconds obtained solving the instance set with the algorithm SPVNS, stopping it when $k_{\max} = 15$. The algorithm runs four times with four different numbers of processors (1, 2, 4, and 8 respectively). The second column

Table 1. Objective values and real time obtained for the algorithm SPVNS.

	VNS	SPVNS	$n\_pr = 1$	$n\_pr = 2$	$n\_pr = 4$	$n\_pr = 8$
$p = 20$	57857.55	57857.55	103	52	27	15
$p = 30$	44086.53	44083.31	200	102	53	30
$p = 40$	35005.82	35002.08	318	163	85	48
$p = 50$	29130.10	29395.25	165	84	44	25
$p = 60$	25176.47	25287.10	155	79	42	24
$p = 70$	22186.14	22167.46	152	78	41	23
$p = 80$	19900.66	19959.71	123	63	33	19
$p = 90$	18055.94	18002.35	472	243	130	73
$p = 100$	16551.20	16592.94	318	163	88	49

shows the best known objective values found by VNS in Hansen, Mladenovic, and Pérez-Brito (2001). The objective values obtained for the SPVNS are reported in the third column. The last four columns refer to the real time in seconds of the algorithm SPVNS using 1, 2, 4, and 8 processors, respectively. We use real time to show the time reduction provided by the parallel algorithm SPVNS.

Note that the algorithm SPVNS finds the same objective value using different number of processors because the parallelism is used only to reduce the time of the local search. Table 2 shows the *speed-up* for the algorithm SPVNS using 1, 2, 4, and 8 processors. Speed-up is the ratio of sequential time to parallel time to solve a particular problem on a given machine. The speed-up is given by:

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

Table 2. Speed-up obtained for the algorithm SPVNS.

Speed-up	$n\_pr = 1$	$n\_pr = 2$	$n\_pr = 4$	$n\_pr = 8$
$p = 20$	1.00	1.98	3.81	6.87
$p = 30$	1.00	1.96	3.77	6.67
$p = 40$	1.00	1.95	3.74	6.63
$p = 50$	1.00	1.96	3.75	6.60
$p = 60$	1.00	1.96	3.69	6.46
$p = 70$	1.00	1.95	3.71	6.60
$p = 80$	1.00	1.95	3.73	6.47
$p = 90$	1.00	1.94	3.63	6.47
$p = 100$	1.00	1.95	3.61	6.49

Table 3. Results obtained for the algorithm RSVNS.

nIter (*Time*) *Cost*	$n\_pr = 1$	$n\_pr = 2$	$n\_pr = 4$	$n\_pr = 8$
$p = 20$	14 (103) 57857.55	14 (110) 57857.55	14 (113) 57857.55	14 (116) 57857.55
$p = 30$	46 (200) 44083.31	30 (167) 44057.52	24 (147) 44057.52	24 (150) 44057.52
$p = 40$	47 (318) 35002.08	41 (329) 35003.80	44 (308) 35002.02	28 (210) 35002.08
$p = 50$	30 (165) 29395.25	48 (265) 29215.16	34 (192) 29089.71	52 (290) 29089.71
$p = 60$	26 (155) 25287.10	31 (183) 25242.99	44 (275) 25209.18	56 (340) 25160.40
$p = 70$	23 (152) 22167.46	34 (216) 22158.80	53 (302) 22125.46	36 (224) 22127.61
$p = 80$	14 (123) 19959.71	62 (330) 19896.33	58 (278) 19870.29	64 (341) 19870.29
$p = 90$	101 (472) 18002.35	128 (670) 17989.79	41 (228) 18030.59	82 (442) 17993.60
$p = 100$	52 (318) 16592.94	36 (253) 16583.20	118 (576) 16565.59	80 (476) 16568.01

Table 4. Results obtained for the algorithm RPVNS.

nIter (*Time*) *Cost*	$n\_pr = 1$	$n\_pr = 2$	$n\_pr = 4$	$n\_pr = 8$
$p = 20$	14 (103) 57857.55	39 (225) 57857.55	28 (182) 57904.09	41 (252) 57857.55
$p = 30$	46 (200) 44083.31	54 (263) 44084.22	46 (241) 44013.02	73 (369) 44023.23
$p = 40$	47 (318) 35002.08	60 (313) 35002.08	60 (324) 35002.08	40 (212) 35002.08
$p = 50$	30 (165) 29395.25	67 (289) 29130.04	47 (213) 29089.71	62 (274) 29089.71
$p = 60$	26 (155) 25287.10	86 (434) 25198.81	132 (665) 25198.81	75 (378) 25205.93
$p = 70$	23 (152) 22167.46	75 (371) 22127.61	74 (357) 22154.72	60 (293) 22125.46
$p = 80$	14 (123) 19959.71	35 (211) 19917.02	41 (238) 19902.71	85 (503) 19876.45
$p = 90$	101 (472) 18002.35	56 (279) 17993.60	97 (446) 18023.27	90 (431) 18010.01
$p = 100$	52 (318) 16592.94	29 (182) 16581.32	48 (296) 16572.48	66 (411) 16566.88

The comparison between the four columns shows that the speed-up increases with the number of processors. For small number of processors the speed-up is almost linear. The fact that the linearity is not reached is due to the factor $T_{Par}(n\_pr)$.

In Tables 3 and 4 we report the number of iterations (*nIter*), the real times in seconds (*Time*) and the best costs (*Cost*) obtained for the algorithms RPVNS and RSVNS using 1, 2, 4, and 8 processors. The number of iterations is reported because the performance analysis for RPVNS and RSVNS depends on this counter.

The RPVNS and RSVNS algorithms allow to increase the analyzed solution space when compared to the sequential VNS procedure. These two algorithms stop when $k_{\max} = 15$. Then, with the same number of iterations, we must expect that they obtain better solutions. This occurs in most cases in our computational experience. The RSVNS algorithm shows better number of iterations than RPVNS. The reason is that in the RPVNS, the solutions found during the searches for a processor are not known for other processors.

7. Conclusions

The combination of the Variable Neighborhood Search and the 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. The Synchronous Parallel VNS (SPVNS) is obtained by parallelizing the local search. By parallelizing the whole procedure we get the Replicated Parallel VNS (RPVNS) where each processor runs in parallel a VNS. The Replicated-Shake VNS (RSVNS) is obtained by performing in parallel a local search in each processor from a shook solution.

Our computational experience with instances of the p-median problem with 1400 points corroborate the theoretical analysis.

The advantages of parallelizing a metaheuristic algorithm are either the reduction of the computational time or the increase of the exploration in the search space. The SPVNS, which shows good speed-up, tries to reduce the computational time in the local search. Attending the results obtained we propose an approach that combines both RSVNS and SPVNS. RSVNS increases the diversification while keeping the intensification given by the VNS procedure and SPVNS allows to decrease the total computational time using several processors.

References

Beasley, J.E. (1985). "A Note on Solving Large p-Median Problems." *European Journal of Operational Research* 21, 270–273.

Erlenkotter, D. (1978). "A Dual-Based Procedure for Uncapacitated Facility Location." *Operations Research* 26, 992–1009.

Glover, F. (1989). "Tabu Search—Part I." *ORSA Journal on Computing* 1, 190–206.

Glover, F. (1990). "Tabu Search—Part II." *ORSA Journal on Computing* 2, 4–32.

Hanjoul, P. and D. Peeters. (1985). "A Comparison of Two Dual-Based Procedures for Solving the p-Median Problem." *European Journal of Operational Research* 20, 387–396.

Hansen, P. and N. Mladenovic. (1997). "Variable Neighborhood Search for the p-Median." In *Les Cahiers du GERAD* G-97-39. Montreal, Canada.

Hansen, P. and N. Mladenovic. (1999). "An Introduction to Variable Neighborhood Search." In S. Voss et al. (eds.), *Proceedings of the 2nd International Conference on Metaheuristics*—MIC97. Dordrecht: Kluwer.

Hansen, P. and N. Mladenovic. (2000). "Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization." In *Les Cahiers du GERAD* G-2000-3. Montreal, Canada.

Hansen, P., N. Mladenovic, and D. Pérez-Brito. (2001). "Variable Neighborhood Decomposition Search." *Journal of Heuristics* 7(3), 335–350.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: The University of Michigan Press.

Kariv, O. and S.L. Hakimi. (1969). "An Algorithmic Approach to Network Location Problems; Part 2. The *p*-Medians." *SIAM Journal on Applied Mathematics* 37, 539–560.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi. (1983). "Optimization by Simulated Annealing." *Science* 220, 671–666.

Kuehn, A.A. and M.J. Hamburger. (1963). "A Heuristic Program for Locating Warehouses." *Management Science* 9(4), 643–666.

Maranzana, F.E. (1964). "On the Location of Supply Points to Minimize Transportation Costs." *Operations Research Quarterly* 12, 138–139.

Mladenovic, N. (1995). "A Variable Neighborhood Algorithm—A New Metaheuristic for Combinatorial Optimization." *Presented at Optimization Days*, *Montreal*.

Mladenovic, N., J.A. Moreno-Pérez, and J. Marcos Moreno-Vega. (1996). "A Chain-Interchange Heuristic Method." *Yugoslav J. Oper. Res*. 6, 41–54.

OpenMP: A Proposed Industry Standard API for Shared Memory Programming. White Paper, Oct. 1997. (`http://www.openmp.org/openmp/mp-documents/paper/paper.html`).

Teitz, M.B. and P. Bart. (1968). "Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph." *Operations Research* 16(5), 955–961.

TSPLIB (1995). `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`.

Voss, S. (1996). "A Reverse Elimination Approach for the *p*-Median Problem." *Studies in Locational Analysis* 8, 49–58.