

Chapter 5

VARIABLE NEIGHBOURHOOD SEARCH

VNS for Training Neural Networks

José Andrés Moreno Pérez, Pierre Hansen, Nenad Mladenovic, Belén Melián
Batista, Ignacio J. García del Amo
Universidad de La Laguna

Abstract: The basic idea of VNS is the change of neighbourhoods in the search for a better solution. VNS proceeds by a descent method to a local minimum exploring then, systematically or at random, increasingly distant neighbourhoods of this solution. Each time, one or several points within the current neighbourhood are used as initial solutions for a local descent. The method jumps from the current solution to a new one if and only if a better solution has been found. Therefore, VNS is not a trajectory following method (as Simulated Annealing or Tabu Search) and does not specify forbidden moves.

Key words: Estas son las palabras claves

1. INTRODUCTION

Artificial neural networks allow to approximate non-linear mappings from several input variables to several output variables. In order to do it, the structure of the network has to be fixed and a set of parameters known as weights have to be tuned. If the outputs are continuous variables then it is prediction or approximation problem while in classification the output is a single categorical variable. Most of the key issues in the net functionality are common to both.

The main goal in the fitting process is to obtain a model which makes good predictions for new inputs (i.e. to provide good generalization). Once the structure of the network is given, the problem is to find the values of the weights w that optimize the performance of the network in the classification of prediction task. In the supervised learning approach, given with a training

data set, the network is trained for the classification or prediction task by tuning the values of the weights in order to minimize the error across the training set. The training set T consists of a series of input patterns and the corresponding outputs. If the function f to be approximated or predicted has an input vector of variables $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and the output is represented by $f(\mathbf{x})$, the error of the prediction is the difference between the output $p(\mathbf{w}, \mathbf{x})$ provided by the network to the inputs \mathbf{x} using the weights \mathbf{w} and real value $f(\mathbf{x})$. The usual way to measure the total error is typically by the root mean squared difference between the predicted output $p(\mathbf{w}, \mathbf{x})$ and the actual output value $f(\mathbf{x})$ for all the elements x in T (RMSE; Root Mean Squared Error)

$$RMSE(T, \mathbf{w}) = \sqrt{\frac{1}{|T|} \sum_{x \in T} (f(x) - p(\mathbf{w}, x))^2}$$

Therefore, the task of training the net consisting in tuning the weights in interpreted as the non-linear optimization problem of minimizing the RMSE on the training set through an optimal set of values \mathbf{w}^* for the weights. That is to solve the problem

$$RMSE(T, \mathbf{w}^*) = \min_{\mathbf{w}} RMSE(T, \mathbf{w}), \mathbf{w} \geq 0.$$

To this problem, one can apply specific or general optimization techniques.

However, the main goal in the design of an artificial neural network is to obtain a design which makes best predictions for future inputs (i.e. which provide achieves the best possible generalization). Therefore the design must allow the representation of the systematic aspects of the data rather than their specific details. To evaluate the generalization provided by the network, the standard way consists of introducing another set V of pairs input/outputs in order to perform the validation. Once the training has been performed and the weights have been chosen, the performance of the design is given by the root mean squared error across the validation set V , i.e., the *validation error*, computed by:

$$RMSE(V; T, \mathbf{w}) = \sqrt{\frac{1}{|V|} \sum_{y \in V} (f(y) - p(\mathbf{w}^*, y))^2}$$

The net must exhibit a good fit between the target values and the output (prediction) in the training set and also in the testing set. If the RMSE in T is significantly higher than that one in E , we will say that the net has

memorized the data, instead of learning them (i.e., the net has over-fitted the training data). In order to avoid over-fitting by stopping the training before the network starts to memorize the data instead of learning the general characteristic of the instance, is useful to use a third disjoint set of instance V_T . Then, from time to time (i.e., when the number of iterations reach to some values) the percentage of the accuracy on this set obtained; when this value increased instead of decreases the training stops.

If the design of the model has few parameters it is difficult to fit the network to the data and if it has too many parameters and the structure is enough general, it would over-fit the training data set excessively minimizing the training errors against the improvement of the validation errors. However, there is not a well established criterion to determine the appropriate number of parameters. Moreover there is not consensus on what are the good architectures of the networks.

The usual neural networks used for the approximation or prediction of a function consists of a series of input nodes or neurones (one for each variable of the function), one output neuron (or several if the function is multi-dimensional) that are interconnected by variable set of hidden network. A kind of structure that has been widely used is the multilayer architecture where the neurons are organized in a set of layers with interconnections only between neurons of different layers. The multilayer neural networks for prediction have, in addition the input layers and the output layers a series of layers with a finite set of neurons from the input to the output where there is a link from every neuron each neuron of the next layer.

The simplest model consists of a network with only a hidden network with h neurons, therefore the set of neurons N of the network consists of n input neurons (n is the number of variables of the function to be approximated) in the input layer N_i , h neurons in the hidden layer N_H , and a single neuron in the output layer N_o . Moreover all the connections go from the neurons of the input layer to the neurons of the hidden layer and from the neurons of the hidden layer to the neuron of the output layer.

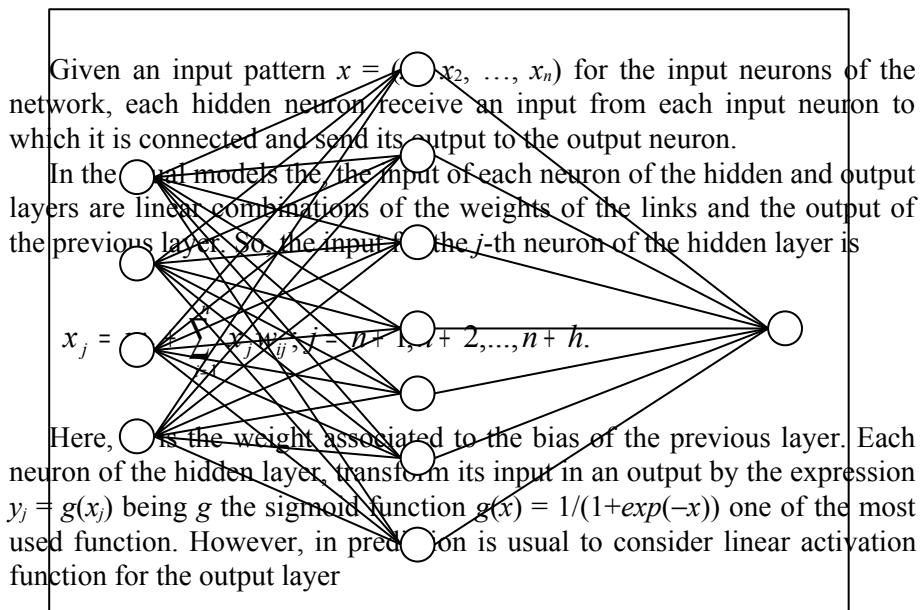
Blum and Li (1991) proved that a neural network having two layers and sigmoid hidden units can approximate any continuous mapping arbitrarily well. As consequence, regarding the classification problem, two layer networks with sigmoid units can approximate any decision boundary to arbitrary accuracy. However, Gallant and White (1992) showed that, from a practical point of view, the number of hidden units must grow as the size of the data set to be approximated (or classified) grows.

Within a multilayer neural network, the neurons can be enumerated consecutively through the layers from the first to the last layers. So we consider a network with a hidden layer to predict a real valued function with

n variables consisting of a set of input neurons $N_I = \{ 1, 2, \dots, n \}$, a set of hidden neurons $N_H = \{ n+1, n+2, \dots, n+h \}$ and the output neurons $n+h+1$. The links are:

$$L = \{ (i,n+j): i = 1, \dots, n, j = 1, \dots, h \} \cup \{ (n+j,n+h+1): j = 1, \dots, h \}.$$

This network is shown in the figure



2. THE VNS METHODOLOGY

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

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

In this notation, X denotes the *solution space*, x represents a *feasible solution* and f the *objective function* of the problem. It is a combinatorial optimization problem if the solution space is discrete or partially discrete. An *optimal solution* x^* (or a global minimum) of the problem is a feasible solution for which the minimum of (1) is reached. Therefore, $x^* \in X$ satisfies that $f(x^*) \leq f(x)$, $\forall x \in X$. A *Neighbourhood Structure* in X is defined by a function $N: X \rightarrow 2^X$ where, $\forall x \in X$, $N(x) \subseteq X$ is the set of *neighbours* of x . Then, a *local minimum* x' of the problem (1), with respect to (w.r.t. for short) the neighbourhood structure N , is a feasible solution $x' \in X$ that satisfies the following property: $f(x') \leq f(x)$, $\forall x \in N(x')$. Therefore any *local* or *neighbourhood search* method (i.e., method that only moves to a better neighbour 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 (1993) and Glover and Kochenberger (1993)). Variable Neighbourhood Search (VNS) (Mladenović (1995), Mladenović and Hansen (1997), Hansen and Mladenović (1999), Hansen and Mladenović (2000), Hansen and Mladenović (2001), Hansen and Mladenović (2003)) is a recent metaheuristic that exploits systematically the idea of neighbourhood 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 exploring then a series of different predefined neighbourhoods of this solution. Each time one or several points of the current neighbourhood are used as starting points to run a local descent method this 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 neighbourhood) as Simulated Annealing or Tabu Search.

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. Despite its simplicity it proves to be effective. VNS exploits systematically the following observations:

1. A local minimum with respect to one neighbourhood structure is not necessary so for another;
2. A global minimum is a local minimum with respect to all possible neighbourhood structures.
3. For many problems local minima with respect to one or several neighbourhoods are relatively close to each other.

The last observation, which is empirical, implies that a local optimum often provides some information about the global one. There 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 neighbourhoods of this local optimum is therefore performed in order, until a better one is found.

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

VND method

1. Find an initial solution x .
2. Repeat the following sequence until no improvement is obtained:
 - (i) Set $l \leftarrow 1$;
 - (ii) Repeat the following steps until $l = l_{\max}$:
 - (a) Find the best neighbor x' of x ($x' \in N_l(x)$);
 - (b) If the solution x' thus obtained is better than x , set $x \leftarrow x'$ and $l \leftarrow 1$; otherwise, set $l \leftarrow l+1$;

Figure -1. Variable Neighbourhood Descent Method \label{figVND}

Another simple application of the VNS principles appears in the reduced VNS. It is a pure stochastic search method: solutions from the pre-selected neighbourhoods are chosen at random. Its efficiency is mostly based on Fact 3 described above. A set of neighbourhoods $N_1(x)$, $N_2(x)$, ..., $N_{k_{\max}}(x)$ around the current point x (which may be or not a local optimum) is considered. Usually, these neighbourhoods are nested, i.e., each one contains the previous. Then a point x' is chosen at random in the first neighbourhood. 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 neighbourhood. After all neighbourhoods have been considered, the search begins again with the first one, until a stopping condition is satisfied (usually it is 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 \ref{figRVNS}.

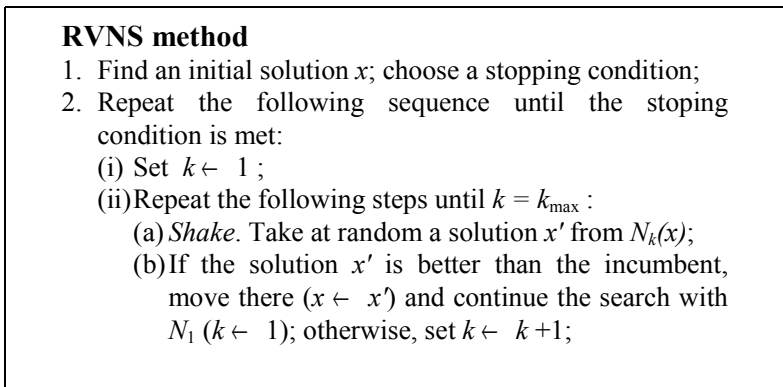


Figure -2. RVNS

In the two previous methods, we examined how to use variable neighbourhoods 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 Neighbourhood Search scheme. We first discuss how to combine a local search with systematic changes of neighbourhoods around the local optimum found. We then obtain the Basic VNS scheme of figure \ref{figBVNS}.

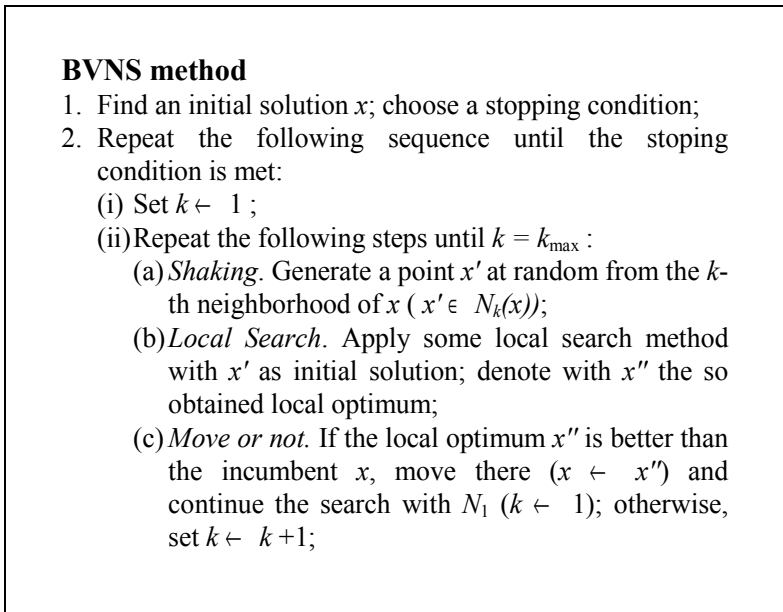


Figure -3. BVNS

The simplest basic VNS, where the neighbourhood for shaking is fixed, is called Fixed Neighbourhood Search (FSN) (see Brimberg et al. (2000)) and sometimes called Iterated Local Search, (see Lourenco et al. (2003)). The method selects by a perturbation a neighbour of the current solution, runs a local search from it to reach a local optimum, and moves to it if there has been an improvement. Therefore, the definition of different neighbourhood structures is not necessary or, one can fix only one among them (i.e., by fixing k) and jump (or 'kick the function') in the shaking (or perturbation) step to a point from that fixed neighbourhood. For example in Johnson and Mc-Geosh (1997) a new solution is always obtained from 4-opt (double-bridge) neighbourhood in solving TSP. Thus, k is fixed to 4. The steps of the simplest VNS are obtained taking only one neighbourhood (see figure \ref{figSimpleVNS})

FNS method1. Initialization:

Find an initial solution x ; Set $x^* \leftarrow x$;

2. Iterations:

Repeat the following sequence until a stopping condition is met:

(a) *Shake.*

Take at random a neighbor x' of x ($x' \in N_k(x)$);

(b) *Local Search.*

Apply the local search method with x' as initial solution; denote x'' the so obtained local optimum;

(c) *Move or not.*

If x'' is better than x^* , do $x^* \leftarrow x''$

Figure -4. FNS

If one uses Variable Neighborhood Descent instead of simple local search and if one improves the initial solution found by Reduced VNS, one obtains the General Variable Neighborhood Search scheme (GVNS) shown in figure \ref{figGVNS}.

GVNS method1. Initialization:

Select the set of neighborhood structures N_k , for $k = 1, \dots, k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_l for $l = 1, \dots, l_{max}$ that will be used in the local search; find an initial solution x and improve it by using RVNS; choose a stopping condition;

2. Iterations:

Repeat the following sequence until the stopping condition is met:

(i) Set $k \leftarrow 1$;

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

(a) *Shaking.*

Generate at random a point x' in the k -th neighborhood of x ($x' \in N_k(x)$);

(b) *Local Search by VND.*

Set $l \leftarrow 1$; and repeat the following steps until $l = l_{max}$;

1. Find the best neighbor x'' of x in $N_l(x')$

2. If $f(x'') < f(x')$ set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise set $l \leftarrow l + 1$;

(c) *Move or not.*

If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;

Figure -5. GVNS

Then a C code for the simple version of the Variable Neighborhood Search is shown in figure \ref{figVNScode}.

SVNS Code

```

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,best_sol)
7:   if improved(cur_sol,best_sol)
8:     best_sol = cur_sol ;
9:     k = 0 ;
10:  } /* if */
11: } /* while */

```

Figure -6. FNS code

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.

2.1 Parallelization of the VNS

The application of parallelism to a metaheuristic may 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 carry out this task, we need to know the parts of the code of an appropriate size that can be partitioned to be solved simultaneously. Several strategies for parallelizing a VNS algorithm have been proposed and analyzed in the literature.

Four different parallelization strategies have been reported in the literature (see Hansen et al. (2004) and Hansen et al. (2005)); two are simple parallelizations and the other two are more complex strategies. The two simple parallelizations of the VNS consist of parallelizing the local search and replicating the whole VNS in the processors respectively. The other two additional parallel strategies are proposed in García et al. (2002) and in Crainic et al. (2004).

The first of the two simple parallelization strategies (analyzed in García 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 two additional parallelization strategies use

cooperation mechanisms to improve the performance. The *Replicated-Shaking VNS parallelization* (RSVNS) of the VNS proposed in García 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 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 *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 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 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 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 continued starting from the best overall solution

in the current neighborhood. The CNVNS procedure is summarized as follows:

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 García et al. (2002) that applies a synchronous cooperation mechanism through a classical master-slave approach and additionally improved by the *Cooperative Neighborhood VNS* (CNVNS) proposed in Crainic et al. (2004) that applies a cooperative multi-search method based on a central-memory mechanism.

3. APPLICATION OF THE VNS TO THE TRAINING PROBLEM

When considering the problem of training an artificial neural network from the metaheuristics point of view, it is useful to treat it as a global optimization problem. The error E is a function of the adaptative parameters of the net, which all can be arranged (i.e., weights and biases) into a single W -dimensional weights vector \mathbf{w} with components $w_1 \dots w_w$. Then the problem consists in finding the weights vector with the lowest error. In this chapter a solution is referred to as a vector of weights since there are no restrictions in the values a weight can have. Therefore, every point of the W -dimensional space is a feasible solution. In this section, we try to explain how a VNS variant can be applied to solve this problem.

The key point of the VNS is that it searches over neighborhood structures in the solution space. A neighborhood of a solution s_1 is simply a group of solutions that are related to s_1 through a neighborhood function. Usually, this function is the classical euclidean distance, so as a solution s_2 belongs to the k -th neighborhood of a solution s_1 if the euclidean distance from s_2 to s_1 is less than or equal to k . From this example we can also deduce that this function generates nested neighborhoods, in the sense that if a solution s_2 belongs to the k -th neighborhood of s_1 , it also belongs to the $(k+1)$ -th neighborhood. This is a desirable property, because if we find that a certain solution s_3 is a local minimum of the k -th neighborhood, it is also most likely a local minimum of all the previous neighborhoods $1 \dots k-1$.

If the solution space has no inner structure, or this structure is unknown to us, the euclidean distance is a neighborhood function as suitable as any other can be, with the advantage that it is an intuitive function and generates nested neighborhoods. However, if the solution space shows some kind of inner structure and we have information about it, we can use it to generate a neighborhood function that is better fitted to the problem. In the artificial neural network training problem, it seems obvious that the solutions have a certain structure. In fact, we know the structure, since the elements of a solution (the weights) have to be arranged spatially according with the architecture of the net. Therefore the goal is to use this information to generate a good neighborhood function.

Our proposal is to define a neighborhood of a solution as all the solutions that share all their weights with the original solution but a finite number of them. More specifically, a solution s_2 belongs to the k -th neighborhood of a solution s_1 if they only differ in the values of the weights that arrive to the k -th neuron. This definition implies that a search through the k -th neighborhood of a solution will only allow changes in the weights that feed the k -th neuron, improving each neuron locally to best fit the global performance of the net. This approach has one main drawback: it is strongly local (it improves each neuron one at a time), and thus, it does not generate nested neighborhoods.

An alternative proposal is to consider for the k -th neighborhood not only the weights that arrive to the k -th neuron, but also the weights that arrive to the $(k+1)$ -th neuron. This approach does not still generate nested neighborhoods, but at least it generates overlapping neighbourhoods. These neighbourhoods make the method less local than the previous ones.

After deciding the way in which we are going to define a neighborhood, the next step is to consider the stopping criterion of the search process. The usual criterion for stopping the training process for an artificial neural network can be used here: stop when the net starts over-fitting by using a test data set to decide this. The best solution found until the moment by the VNS would be used with this test set, and the error would be logged. If the test error decreases, it means that the solution is good enough both for the train and test sets, and the search process can continue. But if the test error starts to increase, it may mean that the net is over-fitting, and the search should stop. This stopping criterion may require more iterations than neurons in the net, which can lead to inconsistencies in the way neighborhoods are selected (k can reach a value larger than the neurons in the net, producing an undefined neighborhood). To avoid this situation, we can slightly modify the neighborhood function to make it modular, thus selecting the k -th neighborhood modulus W .

The VNS relies on the local search procedure to find a local minimum in a certain neighborhood. The great advantage of the VNS is that it greatly reduces the search space for the local search procedure (with this approach, from W to the number of weights arriving to a neuron, which is typically much smaller than W). Local search procedures may use no gradient information at all (as, for example, the Simplex method), first order derivative information (gradient descent, line search, conjugate gradients), or even second order derivatives (Quasi-Newton methods, Levenberg-Marquardt method, etc). The key concept here is that gradient information is fast to calculate with the backpropagation technique, and that it is independent of the method used to search the weights-space.

4. EXPERIMENTAL TESTS

5. CONCLUSIONS

6. REFERENCES

- J.E. Beasley. A note on solving large p -median problems, *European Journal of Operational Research*, 21 (1985) 270-273.
- M.L. Brandeau and S.S. Chiu. An overview of representative problems in location research. *Management Science* 35 (1989) 645-674.
- 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 Sci.* 23 (1977) 789-810.
- 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.
- T.G. Crainic, M. Gendreau, Cooperative parallel tabu search for the capacitated network design. *Journal of Heuristics* 8 (2002) 601-627.
- J.A. Díaz, E. Fernández, Scatter search and Path relinking for the capacitated p -median problem, *European Journal of Operational Research* (2004), forthcoming.
- Drezner, Z. (ed.) *Facility location: A survey of applications and methods*, Springer, 1995.
- F. García López, B. Melián 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.
- F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics*, Kluwer, 2003.

- 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.
- P. Hansen and N. Mladenovic. Variable neighborhood search for the p -median. *Location Science*, 5 (1997) 207--226.
- 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.
- P. Hansen and N. Mladenović. Developments of variable neighborhood search, C. Ribeiro, P. Hansen (eds.), *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, Boston/Dordrecht/London, (2001) 415--440.
- P. Hansen and N. Mladenović, Variable neighborhood search: principles and applications, *European Journal of Operational Research* 130 (2001) 449-467.
- P. Hansen, N. Mladenovic. Variable Neighborhood Search. In F. Glover and G. Kochenberger (eds.), *Handbook of Metaheuristics* Kluwer (2003) 145--184.
- P. Hansen, N. Mladenovic and D. Pérez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics* 7 (2001) 335-350.
- 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.
- A.A. Kuehn, M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9 (1963) 643-666.
- L.A.N. Lorena, E.L.F. Senne, A column generation approach to capacitated p -median problems *Computers and Operations Research* 31 (2004) 863-876.
- 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.
- F.E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly*, 12 (1964) 138-139.
- P. Mirchandani and R. Francis, (eds.) *Discrete location theory*, Wiley-Interscience, (1990).
- N. Mladenovic. A variable neighborhood algorithm-A new metaheuristic for combinatorial optimization. Presented at Optimization Days, Montreal (1995) pp. 112.
- N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers Oper. Res.* 24 (1997) 1097--1100.
- N. Mladenovic, J.A. Moreno-Pérez, and J. Marcos Moreno-Vega. A chain-interchange heuristic method, *Yugoslav J. Oper. Res.* 6 (1996) 41-54.
- C.R. Reeves, *Modern heuristic techniques for combinatorial problems*. Blackwell Scientific Press, (1993).
- M.G.C. Resende and R.F. Werneck, A hybrid heuristic for the p -median problem, *Journal of Heuristics* 10 (2004) 59--88.
- 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.
- 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.
- K.E. Rosing and C.S. ReVelle, Heuristic concentration: Two stage solution construction, *European Journal of Operational Research* 97 (1997) 75-86.
- 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.

- 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*, To appear (2002)
- M.B. Teitz, P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16 (1968) 955-961.
- S. Voss. A reverse elimination approach for the p -median problem. *Studies in Locational Analysis*, 8 (1996) 49-58.
- R. Whitaker, A fast algorithm for the greedy interchange of large-scale clustering and median location problems, *INFOR* 21 (1983) 95-108.
- Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, New York.
- Chambers, J. and T. Hastie (1991), *Statistical models in S*, Wadsworth/Brooks Cole, Pacific grove, CA.
- Cleveland, W.S. (1979) "Robust Locally Weighted Regression and Smoothing Scatterplots," *Journal of the American Statistical Association*, Vol. 74, pp. 829-836.
- Fahlman, S.E. (1988) "An empirical study of learning speed in back-propagation networks", In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *Connectionist Models Summer School*, San Mateo, CA, Morgan Kaufmann, pp. 38-51.
- Friedman, J. H. and Silverman, B. W. (1989), Flexible parsimonious smoothing and additive modelling (with discussion), *Technometrics*, 31, 3-39.
- Glover, F. (1989) "Tabu Search-Part 1", *ORSA Journal on Computing*, vol 1, pp. 190-206.
- Hastie, T., R. Tibshirami and J. Friedman (2001), *The elements of statistical learning*, Springer, New-York.
- Hornik, K., M. Stinchcombe and H. White (1989) "Multilayer feedforward networks are universal approximators", *Neural networks*, vol. 2, pp. 359-366.
- Jacobs, R.A., (1988) "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, 1, pp. 295-307.
- Laguna, M. and R. Martí (2002) "Neural Network Prediction in a System for Optimizing Simulations," *IIE Transactions*, vol. 34(3), pp. 273-282.
- Laguna, M. and R. Martí (2003) *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers.
- Martí, R. and A. El-Fallahi (2004) "Multilayer neural networks: an experimental evaluation of on-line training methods" *Computers and Operations Research* 31, pp. 1491-1513.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1992) *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press (www.nr.com).
- R Development Core Team. (2003), "R: A Language and Environment for Statistical Computing", "http://www.R_project.org".
- Sexton, R. S., B. Alidaee, R. E. Dorsey and J. D. Johnson (1998) "Global Optimization for Artificial Neural Networks: A Tabu search Application," *European Journal of Operational Research*, vol. 106, pp. 570-584.
- Smith, S. and L. Lasdon (1992), "Solving Large Nonlinear Programs Using GRG," *ORSA Journal on Computing*, Vol. 4, No. 1, pp. 2-15.