Les Cahiers du GERAD ISSN: 0711–2440

Variable neighborhood programming – a new automatic programming method in artificial intelligence

S. Elleuch, B. Jarboui,
N. Mladenović

G–2016–21

April 2016

Variable neighborhood programming – a new automatic programming method in artificial intelligence

Souhir Elleuch [a]
Bassem Jarboui [a]
Nenad Mladenovic [b]

[a] *MODILS, University of Sfax, Tunisia*

[b] *GERAD & LAMIH, Université de Valenciennes, France & Mathematical Institute SANU, Belgrade, Serbia*

elleuchsouhir1@gmail.com
bassem_jarboui@yahoo.fr
nenadmladenovic12@gmail.com

April 2016

Les Cahiers du GERAD
G–2016–21

Abstract: Automatic programming is an efficient technique that has contributed to an important development in the artificial intelligence field. In this paper, we introduce a new technique called Variable neighborhood Programming (VNP) that was inspired by the principle of Variable Neighborhood Search (VNS) algorithm. VNP starts from a single solution presented by a program, and the search for the good quality global solution continues by exploring different neighborhoods. The goal of our algorithm is to generate a good representative program adequate to a selected problem. VNP takes the advantages of the systematic change of neighborhood structures within a local search of the VNS algorithm to explore more research space. To explain more the algorithm process we apply VNP in a simple sample in symbolic regression problem. Then the effectiveness and the good convergence of this algorithm is proved by testing it on benchmark problems drawn from time series prediction and classification areas, and we compared it with the related techniques.

Key Words: Automatic programming, Variable Neighborhood Search, Variable Neighborhood Programming, Forecasting and Classification.

Acknowledgments: The work of the third author is partly supported by Brazilian grant CNPq (PVE) – 304041/2014-9.

1 Introduction

Building an intelligent machine is an old dream that, thanks to computers, began to take shape. Several scientific are as contribute to this search direction. One of the well-known techniques is automatic programming that consists of generating new programs in automatic fashion. If a machine is able to generate programs with minimal help from the humans, then, the human level intelligent machine becomes reality.

Genetic programming (GP) was introduced by Koza in 1992 [19]. It is inspired by genetic algorithm (GA) operators and it manipulates a generation of programs. GP is a widely used evolutionary algorithm which successfully solves many problems. The main difference between GP and GA is the presentation of a solution. An individual in GA can be a string, while GP's individuals are programs. The usual way to present program within GP is by using a tree. An example is shown in Figure 1a.

From its appearance, the GP has increasingly been used in artificial Intelligence area [26]. Symbolic regression is the first such kind of problem [20] which continues to be commonly studied by other authors [4, 13, 18]. Moreover, there are many other areas where GP approach is successfully applied, such as data mining [1], forecasting [5, 3, 17],and classification [7, 1, 26]. The papers cited above may be divided into two groups: pure GP algorithms, and hybrids that combine GP with other general ideas.

A few number of algorithms is developed related to the idea of GP. We cite for example Grammatical Evolution algorithm [25] and immune programming [24].

All evolutionary algorithms based on population explore the search space horizontally. Therefore, these algorithms risk passing the optimal solution without succeeding to reach it, which leads to a slow convergence to a near-optimum solution [30]. This fact has encouraged the appearance of local search (LS) based algorithms that apply a sequence of local changes of an initial solution, which improves each time its quality until there are no more changes that produce a better solution, i.e., until attaining a local optimum. Variable Neighborhood Search (VNS) metaheuristics are based on local search but does not follow a single trajectory. The neighborhood structures are defined to explore the vicinity of the current incumbent (best known so far) solution in the search space. In fact, VNS explores search space in depth, and conduct the process to reach a good solution in reasonable time [12]. This approach was suggested by Mladenovi and Hansen in 1997 [22]. It has many degrees of freedom, and as being very general, it can be applied to solve different problems [23]. Besides, many related works in solving global optimization problems have proved that VNS based heuristic, compared with population based algorithms [2], gives better results [14].

In the present paper we create an original technique that uses the principles of VNS algorithm to automatically generate programs. A general and enhanced algorithm is proposed and is called Variable Neighborhood Programming (VNP). Its performance is evaluated by testing it on three popular research problems, which are symbolic regression, time series forecasting and classification. We use benchmark related datasets, and compare our new approach with different algorithms from the literature.

We have organized the rest of this paper in the following way. First, we present VNP process in general where a new way to present solution as a tree is introduced. Then we give detailed description of possible neighborhood structures or moves. We conclude this subsection by highlighting the steps of our new VNP algorithm. The third section explains first the application of VNP in a learning process. Then, we discuss computer results obtained in solving time series forecasting problem and in solving classification problem. The final section is reserved to conclusions.

2 Variable neighborhood programming (VNP)

Systematic change of neighborhood structures is the basic idea of VNS metaheuristic. Its effectiveness is proved by solving many discrete and continuous global optimization problems. Following the simple idea of neighborhood change, several VNS extensions are developed for resolving many kind of problems, as indicated in the first section. The VNP algorithm is inspired by this power algorithm. In this section, a basic scheme of VNP process is presented. It includes solution representation, suggested neighborhood structures, and the algorithm steps.

2.1 Solution representation

The majority of the previous studies usually present program, as tree rather than lines of a code. This tree is modeled to represent the rule, the function, the model, etc. Let $F = \{fn_1, fn_2, \ldots, fn_n\}$ denotes the functional set and $E = \{x_1,\ x_2, \ldots, x_n\}$ denotes the terminal set. In fact, functional set can include logical, arithmetic operators... Although variables and constants belong to the terminal set. The example shown in Figure 1a illustrates the expression $\frac{x_1}{x_2+x_3} + x_4 - x_5$.

We found that the tree representation is not the adequate (not enough flexible) one for presenting a problem in general, especially in cases when variables are multiplied by different coefficient values. In fact in the model above all variables have the same importance with a coefficient equal to 1, which is not necessary the case for many situations. So, we suggest an extended solution presentation by adding coefficients to variables. Each terminal node is attached to its own parameter value. These parameters give a weight for each terminal node. They take values from the interval [0, 1]. This form of presentation allows VNP to examine parameter values and tree structures in the same iteration, as will be shown later. Let $G = \{a_1,\ a_2, \ldots, a_n\}$ denotes a parameter set. In Figure 1b an example of our VNP solution representation is presented.

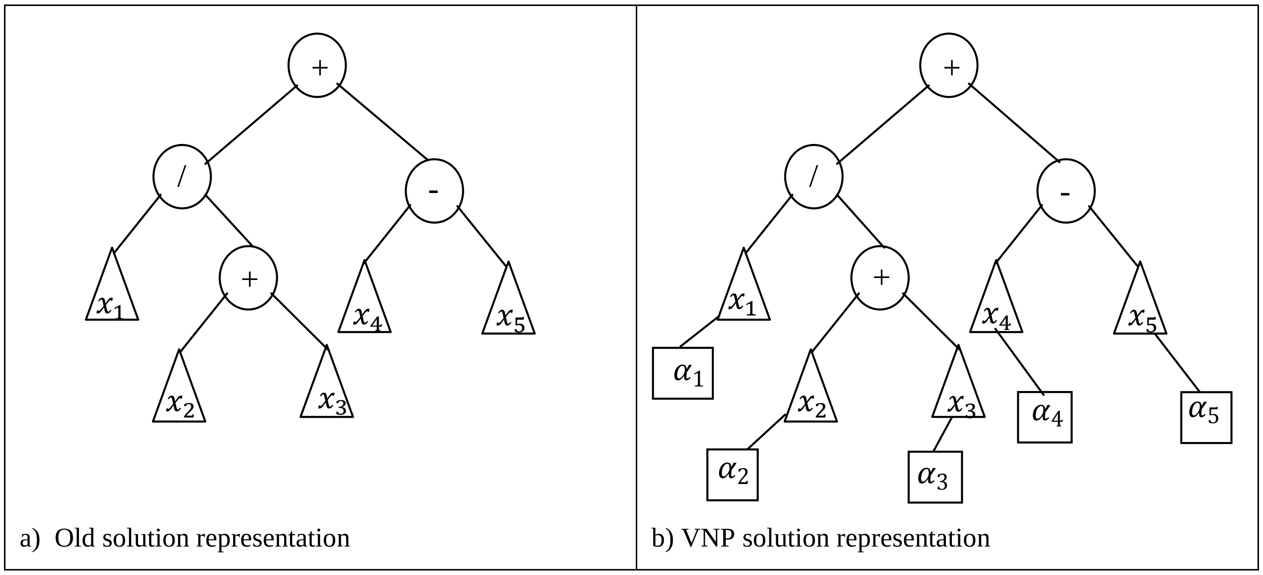

Figure 1: Solution representation in GP (left) and VNP (right)

2.2 Neighborhood structures

Here we propose nine possible neighborhood structures of a given solution (program). For a given problem, we can either use all suggested neighborhood structure or choose between them, this choice depends to the application context. A neighborhood structure movement allows visiting a new solution slightly different to the original one. Therefore, to select a set of neighborhood structures, we have to consider the following criteria: (i) its complementarity; (ii) its diversity and (iii) its cardinality (i.e., the number of visited solutions has to be maximized).

2.2.1 Changing a node value operator ($\mathcal{N}_1$)

This neighborhood structure conserves the skeleton of the tree and changes only the values of a functional or a terminal node. Each node can obtain many values from its corresponding set. Let x_i be the current solution, its neighbor x_{i+1} differs from x_i by just a single node. Figure 2 shows a move within this neighborhood structure.

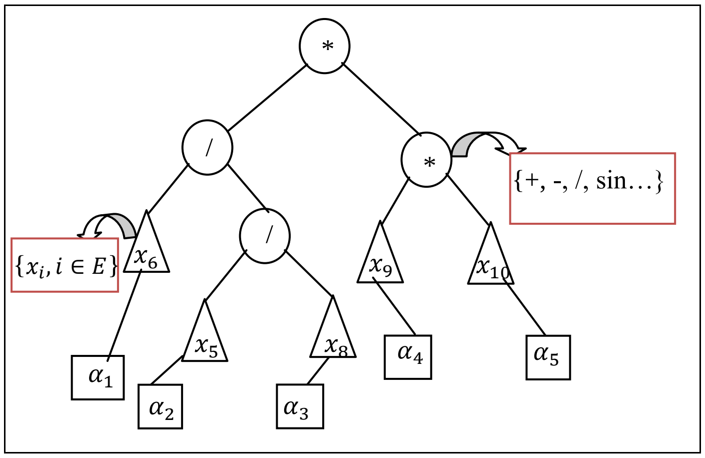

Figure 2: Neighborhood structure $\mathcal{N}_1$: Changing a node value

2.2.2 Swap operator ($\mathcal{N}_2$)

In this operator we choose first a node from the current tree at random and generate a new sub-tree as presented in Figure 3a1 and Figure 3a2. Then, we attach it in the place of the sub-tree, corresponding to the selected node. In this move, we must respect the constraint related to the maximum tree size. More details are given in Figure 3.

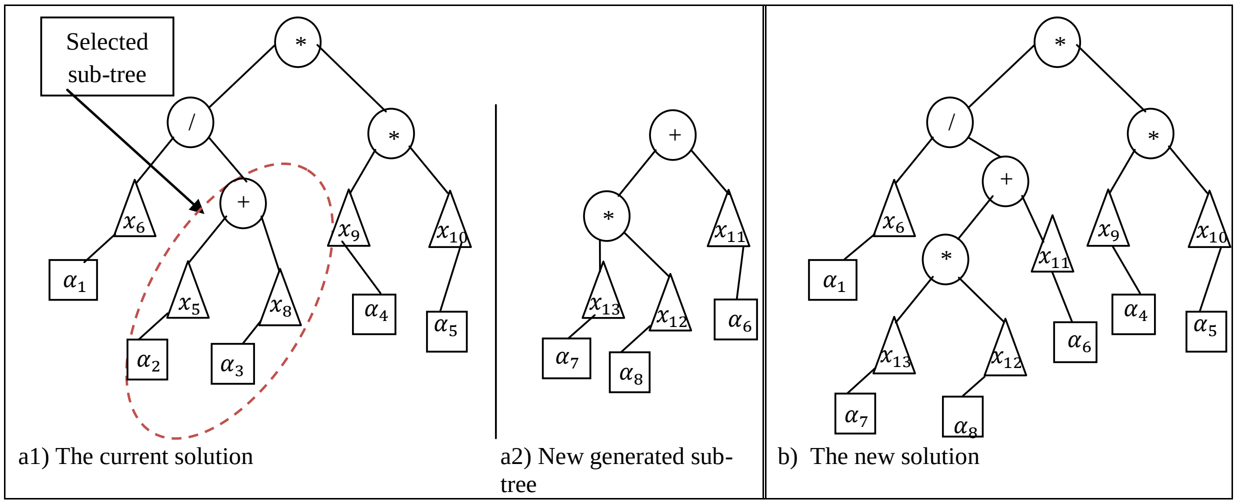

Figure 3: Neighborhood structure $\mathcal{N}_2$: Swap

2.2.3 Changing a parameter value operator ($\mathcal{N}_3$)

In the previous neighborhood structures we just considered the tree form and its node values. Here we focus on the optimization parameters. So, we keep the position and value of nodes to search the neighbors in the parametric space. Figure 4 shows illustrate details. The change from one value to another is random.

2.2.4 Inversion operator ($\mathcal{N}_4$)

Two nodes are selected randomly from a given solution, and then their values are exchanged. The type of the chosen nodes must belong to the same set (functional or terminal, see Figure 5.

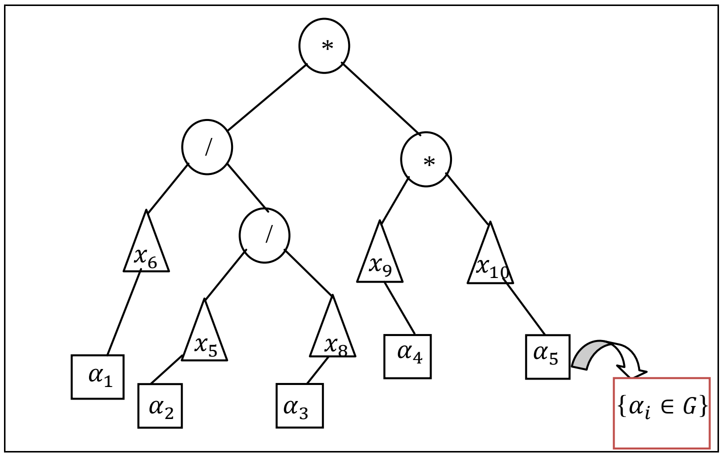

Figure 4: Neighborhood structure $\mathcal{N}_3$: Changing a parameter value

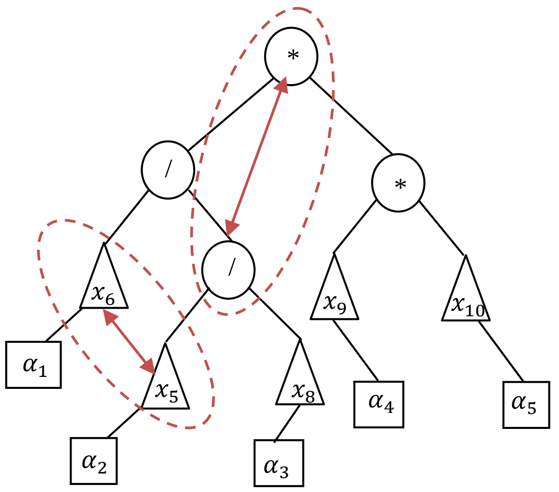

Figure 5: Neighborhood structure $\mathcal{N}_4$: Inversion

2.2.5 Adjacent swap operator ($\mathcal{N}_5$)

In this neighborhood structure, one node is selected randomly and changed with its adjacent node, which is of the same type. This neighborhood structure is not always applicable. For example in Figure 6, we cannot apply adjacent swap move to terminal node x_6 because it has no any adjacent node, though we can apply it to the terminal nodes x_9 and x_{10}.

2.2.6 Three exchange operator($\mathcal{N}_6$)

This neighborhood structure movement is like the inversion operator, but here three nodes are selected randomly and then their values are exchanged.

2.3 Remove operator ($\mathcal{N}_7$)

The remove operator has a critical influence on the quality of a given solution x_i.It can be applied only if the size of the current solution is larger than the fixed size. This size is defined according to the addressed problem. This process begins by choosing randomly a node from the original tree different to the root node. Then, whatever the type of the selected node is, we drop the whole corresponding sub-tree. In Figure 7, these movements are explained in more detail.

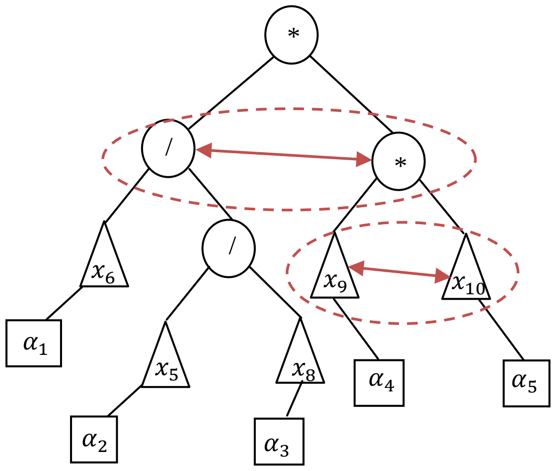

Figure 6: Neighborhood structure ($\mathcal{N}_5$): Adjacent swap

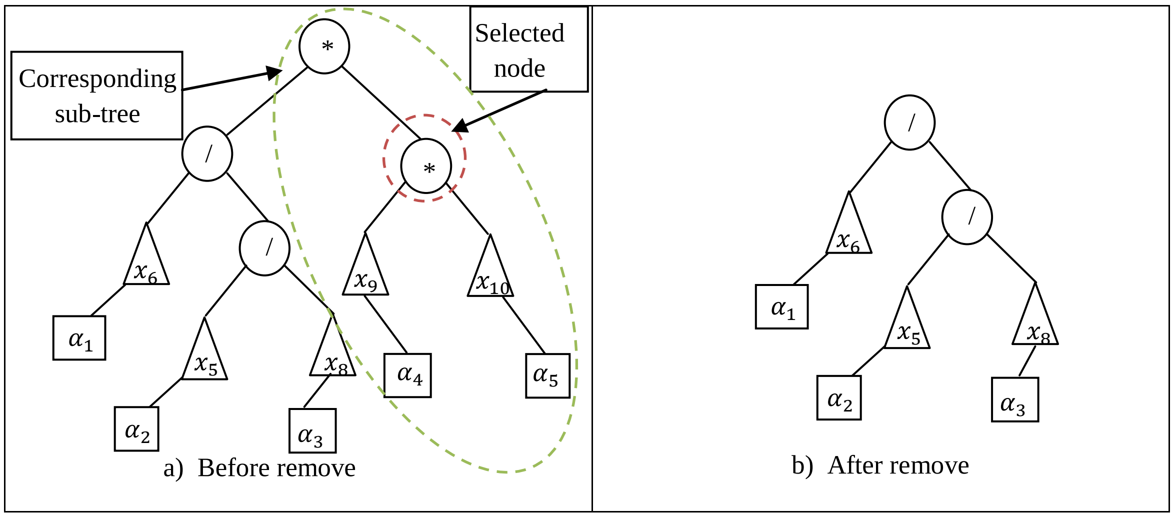

Figure 7: Neighborhood structure $\mathcal{N}_7$: Remove

2.3.1 Move/Inversion operator ($\mathcal{N}_8$)

For the beginning, two nodes different from the root are selected and their associated sub-trees are defined. Then, the positions of these sub-trees are exchanged. One node, at least, should be a functional node to avoid the likelihood of falling into the inversion neighborhood structure explained above. Figure 8 shows an example of this operator.

2.3.2 Shuffle operator ($\mathcal{N}_9$)

A sub-tree is selected and rearranged randomly. This operation affects values and the position of nods. However, the organization of nodes in a given solution x_i should be respected. Therefore, a functional or a terminal node is replaced by another one of the same type (Figure 9).

2.4 Description of the VNP method

To find the best representative model of a given problem, the optimization procedure must take into account a tree structure and a parameter vector. VNP algorithm is developed to optimize simultaneously the tree

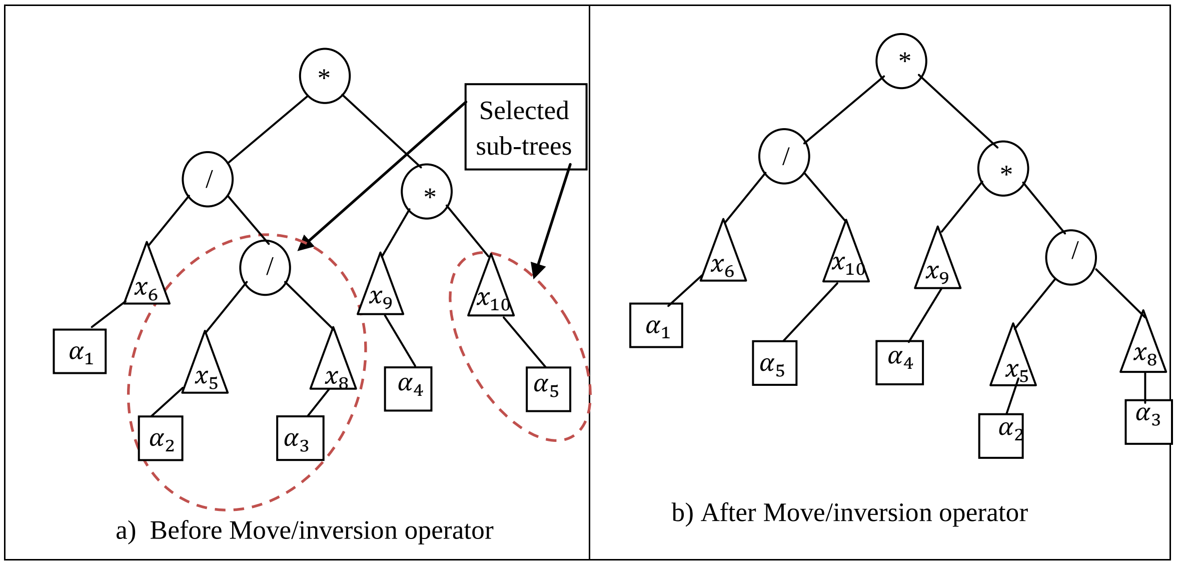
Selected sub-trees
a) Before Move/inversion operator
b) After Move/inversion operator

Figure 8: Neighborhood structure $\mathcal{N}_8$: Move/inversion operator

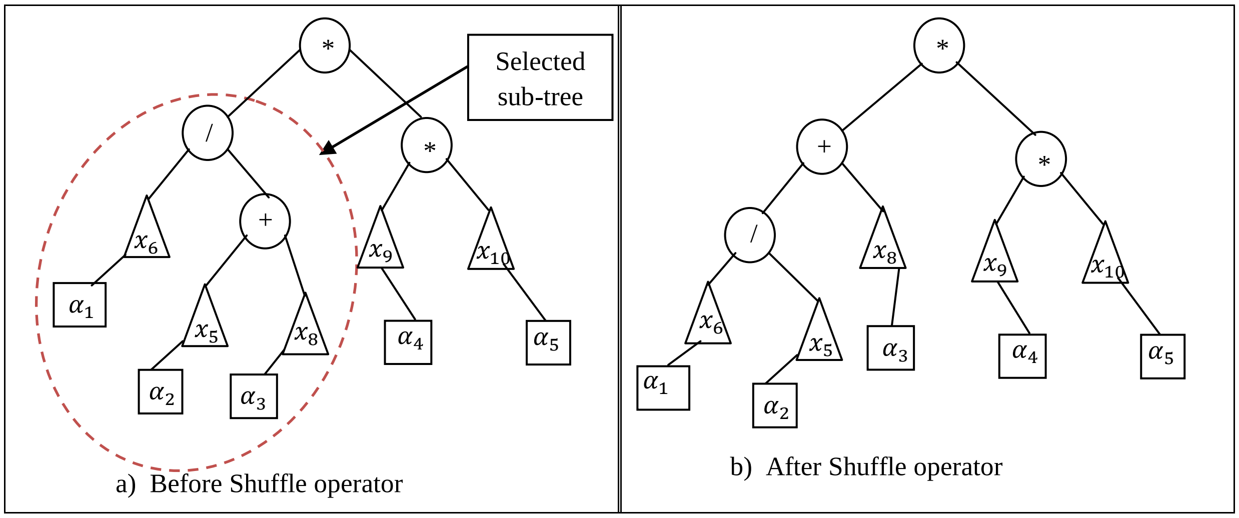

Figure 9: Neighborhood structure $\mathcal{N}_9$: Shuffle

and the corresponding parameter values. So, besides the neighborhoods we choose from the set of operators listed earlier, we may include a neighborhood structure specified for parameter optimization. However, if the structure of the implemented model is not the appropriate one, it is not interesting to pay much attention to the parameter optimization.

2.4.1 VNP Descent

Variable neighborhood descent (VND) is deterministic variant of VNS. There different neighborhood structures (or different moves) are placed in the list and used one after another until no improving move is possible. Following this analogy, we call the deterministic variant of VNP as VNP Descent (VNPD for short). Since we propose nine neighborhood structures in getting new solution, we can use all of them or just some of them deterministically in some order. VNPD pseudo code is presented at Algorithm 1.

Algorithm 1: VNPD (T, l_{max})

```
Input: the set of neighborhood structures: $\mathcal{N}_l, l = 1, \ldots, l_{max}$ and an initial solution $T$
$l \leftarrow 1$;
while $l < l_{max}$
        Find the best neighbor $T'$ of $\mathcal{N}_l(T)$
        Move or not:
            if fitness $(T')$ is better than fitness $(T)$ then
                move $T \leftarrow T'; l \leftarrow 1$;
            else
                $l \leftarrow l + 1$;
End while
Return $T$
```

In Algorithm 1 the solutions are denoted by T and T'. The single parameter l_{max} determines the number of neighborhoods to be used. Solution T and neighborhood structures are at the input. At the output we have a new solution T that is locally best with respect to all structures.

2.4.2 General VNP

General VNS is a variant of VNS where VND based heuristic is used as a local search. Again, following the analogy, we call General VNP variant of VNP that uses VNPD as a local search. The Algorithm 2 summarizes the different steps of the general VNP.

Algorithm 2: General VNP (k_{max}, l_{max})

```
Initialization:
  (1) Fix the set of neighborhood structures for the tree structure optimization and the parameter vector
      optimization, applied to the local search phase: $N_k, k = 1, \ldots, k_{max}$ and the set of neighborhood
      structures for the shaking phase: $N_k, k = 1, \ldots, k_{max}$.
  (2) Select the set of functions and terminals adequate for the studied problem.
  (3) Generate randomly an initial tree $T$ as presented in Figure 1b.
  (4) Choose the stopping condition.
Repeat
    $k \leftarrow 1$;
    while $k < k_{max}$
        (a) $T' \leftarrow$ Shake$(T)$// Find the first neighbor $T'$ in $N_k(T)$
        (b) $T'' \leftarrow$ VNPD$(T', l_{max})$ // Local Search
        (c) Move or not:
                if fitness $(T'')$ is better than fitness $(T)$ then
                    move $T \leftarrow T''; k \leftarrow 1$;
                else
                    $k \leftarrow k + 1$;
End while
until termination condition is met
Return $T$;
```

Like VNS algorithm, the VNP alternates between a stochastic element, which is the random generation of a neighbor in the shaking phase, and a deterministic element, which is the local search, by using VNP descent algorithm (see Algorithm 1). These two steps are followed by the neighborhood change, based on the acceptance criterion.

2.4.3 VNP shaking

The shaking step allows the diversification in the search space. Our proposed VNP algorithm does not use exactly the same neighborhood structures as for the local search. That is why they are denoted differently than those in VNPD algorithm, namely, we denoted by $N_k(T)$, $k - 1, \ldots, k_{max}$, neighborhoods to be used in the shaking step. $N_k(T)$ may be reached by repeating k times one or more moves $\mathcal{N}_l(T)$ explained earlier.

Nevertheless, in the shaking phase, we use a neighborhood structure that mainly affects the skeleton of a presented solution with its different forms for the perturbation. For more explanation, we take the swap operator as an example presented in the Section 2.2.2. Let m be the maximum size of a solution. We can find k neighborhood structures using the swap operator: $k \in [1, k_{max}]$ represents the size of the new generated sub-tree. If n denotes the size of the original tree after deleting the old sub-tree, then $n + k_{max} \leq m$. The objective of this phase is to provide a good starting point for the local search.

2.4.4 Solution evaluation

The evaluation consists of defining a fitness or objective function assessing the proposed problem. This function is defined according to the considered problem. After running each solution (program) on training data set, fitness is measured by counting how many training cases the current solution result is correct or near to the exact solution.

3 VNP applications

This section illustrates computational results that we get by applying the proposed interwoven algorithm VNP. The raised research questions are: (1) How to apply the proposed algorithm to solving problems of different complexity, i.e., how to include the problem specific knowledge of each particular problem, within general framework of VNP? (2) How does the proposed algorithm compare with other artificial intelligence algorithms, when the same problem is considered? To discuss these questions, we will begin first with a simple application in symbolic regression area than two popular problems are examined: (i) time series forecasting, and (ii) classification. Experiments are based on programs (trees) in the form of arithmetic expressions.

Time series forecasting problem consists on predicting the future behavior of a sample data based on previously observed values. In other words, it's finding a mathematical function, containing problem variables, that approximates the sample data.

Classification is the processing of searching a set of models which distinguish and characterize data classes or concepts. These features in our paper are collected in a mathematical function presented as a program.

In these two cases we apply our approach on the following benchmark datasets: Mackey-Glass and Box-Jenkins instances in the forecasting problem, and Iris, Wine, Statlog, Glass identification, and Yeast datasets in the classification problem.

These two problems need a training phase to extract statistical features, and a testing phase to evaluate resulting model. The employed fitness function in the two phases is given in the following sub-section.

3.1 Learning process

The proposed problems are resolved by a learning process which is described as follows: First, the original dataset is divided into two parts. Then, one of the given datasets serves to train VNP algorithm and the second is employed to test the best model. The data should be treated and conversed to numerical data before using it. A sample in a data includes input values and desired output value. After training the algorithm and testing the obtained model we attain finally the fitness error validating the selected tree. More details are shown in Figure 10.

3.2 Symbolic regression problem

Symbolic regression is a Mathematical modeling method and numerical data analysis, which aims to find an approximation of a mathematical function in symbolic form. Its general process tries from a set of experimental values, finding the curve which represents well the variations of the quantity to be studied and which passes as close as possible from points that are available, to determine the general behavior of the system. In other terms, the symbolic regression is a method of research in the space of mathematical expressions which minimizes the error metric. Therefore the symbolic regression is trying to find functions

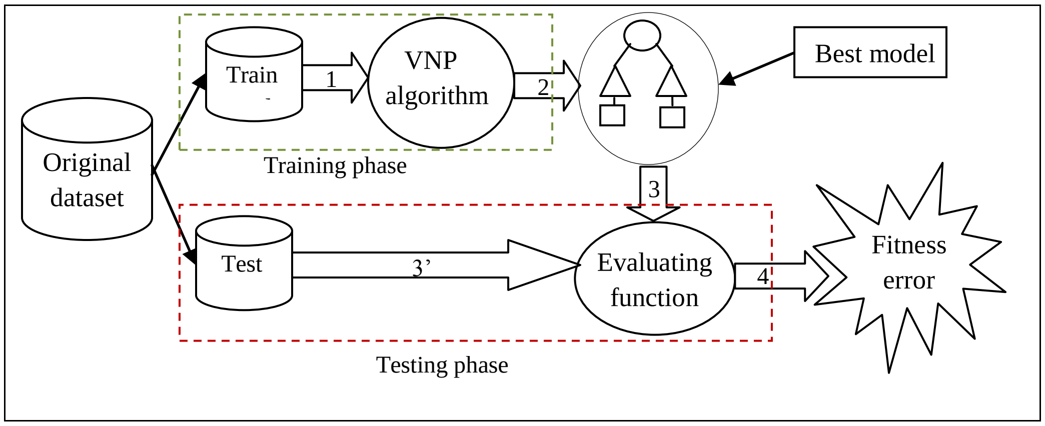

Figure 10: Learning process schema

whose output has some desired properties. We try in most cases, to find a function which coincides with the given points.

We present a simple application in this area to explain more the process of VNP algorithm. As an example we ran our algorithm on the dataset described in Table 1.

Table 1: Symbolic regression problem dataset

Instance	Input	Desired Output
1	1	2
2	2	6
3	4	20
4	7	56
5	9	90

We search a function satisfying the inputs and outputs given in the previous table. Each row represents a fitness case. The run of our algorithm will result a program (tree) which can calculate for each input the corresponding output.

To resolve this problem we have to follow four steps:

1. Terminal and function sets identification
2. Neighborhood structure choice with due respect to selection criteria
3. Parameter adjustments
4. Definition of an adaptability function.

This problem is not complicated so we limit the functional set to addition and multiplication operations. Terminal set includes the single input attribute x_1 and random real numbers. To run our algorithm we employ the simple solution representation (Figure 1a) and we choose $\mathcal{N}_1$ and $\mathcal{N}_2$ as neighborhood structures. The selected neighborhood structures guarantee the complementarity criterion. The Root Mean Square Error (RMSE) is a good fitness function to our problem. The formula of this function is :

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_t^j - y_{out}^j)^2}$$

where n represents the total number of samples, y_{out}^j and y_t^j are the VNP's given model, and the desired output of the sample number j.

The next table summarizes these different choices.

Table 2: VNP parameters adjustment for symbolic regression problem

The functional set		$F = \{+, *\}$
The terminal sets		$\{x_1, c\}$, $c \in \mathbb{R}$
Neighborhood structures		$\mathcal{N}_1$,$\mathcal{N}_2$
Parameter tuning	Minimum tree length	3 *nodes*
	Maximum tree length	200 *nodes*
	Maximum iteration number	50 *iterations*

The VNP algorithm output is something like the following table.

Table 3: VNP outputs

VNP iteration	Current program	RMS Error
First iteration	(((-3.48 * (-1.89 + (1.98 * x_1))) * (-1.54)- 2.23)	29.6181
Third iteration	(((((-0.47 / ((1.06 - 1.72) * 4.6)) * (((x_1/ (-0.5/ 4.61)) * 1.18) - -0.71)) * (-4.27 – (-4.71))) / -4.5) * ((((x_1 - (-4.12 - 3.56)) - ((-2.31 + -0.6) / 4.23)) – (-4.12)) + (((-2.13 + x_1) * (2.73 * 1.83)) - (-3.6 + -1.54))))	0.28414
Last iteration	((x_1 + (-0,13/ -0,13)) * x_1) $\;{x_1}^2 + x_1$	0.0

After five iterations, the VNP algorithm finds the best solution with an error of 0.0. The last output can be simplified to: $x_1 + {x_1}^2$. This program provides a function which relates inputs with the desired outputs. The results of three iterations of VNP algorithm are available in the Table 4.

Table 4: VNP algorithm results

Instance	Input	Desired Output	First Iteration	Third Iteration	Last Iteration
1	1	2	1,78152174	1,99598151	2
2	2	6	8,91829325	6,04860724	6
3	4	20	30,3179232	20,0989852	20
4	7	56	62,4173682	56,0373686	56
5	9	90	83,8169982	89,904835	90

3.3 Time series forecasting problem

For studying this problem, we use two widely used benchmark datasets: Mackey-Glass series, and Box-Jenkins set.

3.3.1 Fitness function

We use the Root mean square error as a fitness function:

$$fitness = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(y_t^j - y_{out}^j\right)^2}$$

where n represents the total number of samples, $y_{out}^j$and y_t^j are the VNP's given model, and the desired output of the sample number j.

3.3.2 Initialization step

To begin with, we have to fix the neighborhood structure. Using complementary criteria, after some trials, we decide to apply the swap operator ($\mathcal{N}_2$) and the changing node value operator ($\mathcal{N}_\mathbf{1}$) for tree neighborhood structure, and the changing parameter value operator ($\mathcal{N}_3$) for parameter neighborhood structure.

The experiment adjustments are given in the next table. The values are fixed after several tests.

Table 5: VNP parameters adjustment for forecasting problem

The functional set	$F = \{+, *, /, pow\}$
The terminal sets	$\{x_i, c\}\ ,\ i \in [1 \ldots m]\,, m = input\ numbers,\ c \in \mathbb{R}$
Neighborhood structures	$\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$
Minimum tree length	20 *nodes*
Maximum tree length	200 *nodes*
Maximum iteration number	50000 *iterations*

3.3.3 The Mackey-Glass series results

The Mackey-Glass series, based on the Mackey-Glass differential equation [21] are chaotic time series generated from the following time-delay ordinary differential equation:

$$\frac{d(x\,(t))}{dt} = -bx\,(t) + a\frac{x(t-t)}{1+x^c(t-t)}$$

Following the majority of studies, the samples have been generated using the following values for the parameters: $a = 0.2$, $b = 0.1$ and $c = 10$. The task of VNP is to predict the value of the Macke-Glass at point $x(t+6)$ using $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. The obtained dataset contains 1000 samples. The first 500 are used for the training phase and the remainder for the testing phase.

It should be noted that the VNP algorithm execution leads to an output model with an accurate precision. Different results for forecasting Mackey-Glass data are shown in Table 6. The proposed system is compared with a variety of methods by using a simple approach or hybridization between different algorithms. As far as we know, the basic Genetic Programming is not applied to the Mackey-Glass dataset under the same mentioned conditions. We compare our work with PSO BBFN [9], HMDDE–BBFNN [8], Classical RBF [6], CPSO [28], HCMSPSO [16], and FBBFNT [3]. The related results are shown in Table 6.

Table 6: Comparison of training and testing error of Mackey-Glass dataset

Method	Training error RMSE	Testing error RMSE
PSO BBFN	----	0.027
HMDDE–BBFNN	0.0094	0.0170
Classical RBF	0.0096	0.0114
CPSO	0.0199	0.0322
HCMSPSO	0.0095	0.0208
FBBFNT	0.0061	0.0068
VNSP	**0.0021**	**0.0042**

From the computational results, it can be seen that the proposed VNP algorithm works well for generating prediction models of Mackey-Glass time series.

3.3.4 Box and Jenkins' Gas Furnace Problem

The gas furnace data of Box and Jenkins were collected from a combustion process of a methane–air mixture [10]. This time series has found a widespread application as a benchmark example in many practical sciences for testing prediction algorithm. The data set contains 296 pairs of input-output values. The input $u(t)$ corresponds to the gas flow, and the output $y(t)$ presents the CO2 concentration in outlet gas. The inputs are $u\,(t-4)$, and $y(t-1)$, and the output is $y(t)$. In this work, 200 samples are used in the training phase and the remaining samples are used for the testing phase. The performance of the evolved model is evaluated by comparing it with the above mentioned approaches. The RMSE achieved by VNP output model is 0.0038. Table 7 shows the RMSE achieved by ODE [27], HMDDE [8] and FBBFNT [3]. Our approach proves its effectiveness and generalization ability.

Table 7: Comparison of testing error of Box and Jenkins dataset

Methods	Prediction error RMSE
ODE	0.5132
HHMDDE	0.3745
FBBFNT	0.0047
VNP	**0.0038**

3.4 Classification problem

Classification consists on predicting the appropriate class of an input vector based on a set of attributes. Available samples are divided to form the training dataset and the testing dataset. Training dataset instances enables algorithms to extract features and to find the most representative model.

For our experiments, we choose five datasets of radically different nature: Iris, Wine, Statlog, Glass identification, and Yeast datasets. The five datasets are accessible at the UCI machine learning repository. They are used for demonstrating the performance of classification algorithms.

3.4.1 Performance measure

To measure the efficiency of our method in classification problems, we employ Accuracy performance measures. Accuracy presents the percentage of correctly classified instances. It is one of the most popular metrics in this area:

$$Acc = \frac{TN + TP}{TP + FP + FN + TN}$$

where TP, TN, FP, and FN are the True Positive, True Negative, False Positive, and False Negative, respectively.

3.4.2 Initialization step

We have to fix a set of parameters in the initialization step. For the neighborhood structures we employ the same selection as in forecasting problem. The parameter adjustments set is shown in the next table.

Table 8: VNP parameters adjustment for classification problem

The functional set	$F = \{+, *, /, pow, sin\}$
The terminal sets	$\{x_i, c\}$, $i \in [1..m]$, $m = input\ numbers,\ c \in \mathbb{R}$
Neighborhood structures	$\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$
Minimum tree length	20 *nodes*
Maximum tree length	100 *nodes*
Maximum iteration number	50 *iterations*

Each dataset is randomly partitioned to ten-fold and VNP is applied ten times to each fold, a single fold is retained for testing the model, and the remaining 9 subsamples are employed to training algorithms. This process is called the cross-validation. Each algorithm is executed 10 times. So, each fold is used exactly once as a validation data.

3.4.3 Results

We have compared our approach with a method based on GP algorithm, which is two stage genetic programming method (S2GP) [15].We have also compared the proposed classification algorithm with the popular classification algorithms: Support Vector Machines (SVM) [29], and k-Nearest Neighbors (KNN) with parameters [11]. Accuracy values shown in the Table 10 are the average results obtained after ten runs.

Table 10 shows that the performance of the proposed scheme is compatible with other classification algorithms. These results demonstrate again that the proposed approach can generalize well, and the resulting trees with medium sizes can also be easily used in classification problems.

Table 9: Datasets characteristics

Datasets	**Classes**	**Attributes**	**Type**	**Size**
Iris	3	4	Real	150
Statlog (Vehicle Silhouettes)	4	18	Integer	946
Yeast	10	8	Real	1484
Wine	3	13	Integer, Real	178
Glass identification	6	10	Real	214

Table 10: Results of dataset classification

Dataset	KNN (%)	SVM(%)	S2GP (%)	VNP (%)
IRIS	95	94	96	96.7
VEHICLE	54	51	56	55.3
YEAST	50	58	61	58.2
WINE	84	83	85	89.1
GLASS	60	63	64	66

4 Conclusions and perspectives

The purpose of the present study is two-fold. First, we departed from genetic programming and introduced a new algorithm based on local search. Second, we suggest a new way to represent the solution, which ameliorates the property of generalization. In fact, we combined simultaneously parameters with structure in the same tree. This method is applied to two types of time series problems and five datasets of classification. The application of VNP provides satisfying results compared to BBFN [9], HMDDE–BBFNN [8], Classical RBF [6], CPSO [28], HCMSPSO [16], and FBBFNT [3] in the Mackey-Glass time series problem and to ODE [27], HMDDE [8] and FBBFNT [3] in Box and Jenkins time series problem. In the classification section, we select Support Vector Machines (SVM) [29] and k-Nearest Neighbors (KNN) with parameters [11] algorithms and compared them with our work. VNP algorithm gives again good results in classification field.

References

[1] M. de Arruda Pereira, C.A. Davis Júnior, E. Gontijo Carrano, J.A. de Vasconcelos, A niching genetic programming-based multi-objective algorithm for hybrid data classification, Neurocomputing. 133 (2014) 342–357.

[2] C. Blum, A. Roli, Metaheuristics in combinatorial optimization, ACM Comput. Surv. 35 (2003) 268–308.

[3] S. Bouaziz, H. Dhahri, A.M. Alimi, A. Abraham, A hybrid learning algorithm for evolving Flexible Beta Basis Function Neural Tree Model, Neurocomputing. 117 (2013) 107–117.

[4] W. Cai, A. Pacheco-Vega, M. Sen, K.T. Yang, Heat transfer correlations by symbolic regression, Int. J. Heat Mass Transf. 49 (2006) 4352–4359.

[5] Y. Chen, B. Yang, J. Dong, A. Abraham, Time-series forecasting using flexible neural tree model, Inf. Sci. (Ny). 174 (2005) 219–235.

[6] K.B. Cho, B.H. Wang, Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction, Fuzzy Sets Syst. 83 (1996) 325–339.

[7] W.-J. Choi, T.-S. Choi, Genetic programming-based feature transform and classification for the automatic detection of pulmonary nodules on computed tomography images, Inf. Sci. (Ny). 212 (2012) 57–78.

[8] H. Dhahri, A.M. Alimi, A. Abraham, Hierarchical multi-dimensional differential evolution for the design of beta basis function neural network, Neurocomputing. 97 (2012) 131–140.

[9] H. Dhahri, A.M. Alimi, F. Karray, Designing beta basis function neural network for optimization using particle swarm optimization, in: 2008 IEEE Int. Jt. Conf. Neural Networks (IEEE World Congr. Comput. Intell., IEEE, 2008: pp. 2564–2571.

[10] G.M.J. G.E. P Box, Time series analysis, forecasting and control, Holden-Day, San Francisco, 1976.

[11] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes, Pattern Recognit. 44 (2011) 1761–1776.

[12] M. Garca-Torres, F. Gómez-Vela, B. Melián-Batista, J.M. Moreno-Vega, High-dimensional feature selection via feature grouping: A Variable Neighborhood Search approach, Inf. Sci. (Ny). 326 (2016) 102–118.

[13] S. Gustafson, E.K. Burke, N. Krasnogor, On Improving Genetic Programming for Symbolic Regression, in: 2005 IEEE Congr. Evol. Comput., IEEE, n.d.: pp. 912–919.

[14] S. Hanafi, J. Lazic, N. Mladenovic, C. Wilbaut, I. Crevits, New VNS based 0-1 MIP Heuristics, Yugosl. J. Oper. Res. ISSN 0354-0243 EISSN 2334-6043. 25 (n.d.).

[15] H. Jabeen, A.R. Baig, Two-stage learning for multi-class classification using genetic programming, Neurocomputing. 116 (2013) 311–316.

[16] C.-F. Juang, C.-M. Hsiao, C.-H. Hsu, Hierarchical Cluster-Based Multispecies Particle-Swarm Optimization for Fuzzy-System Optimization, IEEE Trans. Fuzzy Syst. 18 (2010) 14–26.

[17] A. Kattan, S. Fatima, M. Arif, Time-series event-based prediction: An unsupervised learning framework based on genetic programming, Inf. Sci. (Ny). 301 (2015) 99–123.

[18] M. Keijzer, Scaled Symbolic Regression, Genet. Program. Evolvable Mach. 5 (2004) 259–269.

[19] J.R. Koza, Genetic programming: on the programming of computers by means of natural selection, MIT Press Cambridge, MA, USA. (1992).

[20] J.R. Koza, Genetic programming II: automatic discovery of reusable programs, MIT Press Cambridge, MA, USA. (1994).

[21] M. Lovbjerg, T. Krink, Extending particle swarm optimisers with self-organized criticality, in: Proc. 2002 Congr. Evol. Comput. CEC'02 (Cat. No.02TH8600), IEEE, 2002: pp. 1588–1593.

[22] N. Mladenovi, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.

[23] N. Mladenović, R. Todosijević, D. Urošević, Less is more: Basic variable neighborhood search for Minimum differential dispersion problem, Inf. Sci. (Ny). 326 (2015) 160–171.

[24] P. Musilek, A. Lau, M. Reformat, L. Wyard-scott, Immune programming, 176 (2006) 972–1002.

[25] M. O'Neill, C. Ryan, Grammatical evolution, Evol. Comput. IEEE Trans. 5 (2001) 349 – 358.

[26] C. De Stefano, G. Folino, F. Fontanella, A. Scotto di Freca, Using Bayesian networks for selecting classifiers in GP ensembles, Inf. Sci. (Ny). 258 (2014) 200–216.

[27] B. Subudhi, D. Jena, A differential evolution based neural network approach to nonlinear system identification, Appl. Soft Comput. 11 (2011) 861–871.

[28] F. VandenBergh, A.P. Engelbrecht, A Cooperative Approach to Particle Swarm Optimization, IEEE Trans. Evol. Comput. 8 (2004) 225–239.

[29] V.N. Vapnik, Statistical Learning Theory, Wiley, 1998.

[30] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (1999) 82–102.