Les Cahiers du GERAD ISSN: 0711–2440

Variable neighborhood programming for symbolic regression

S. Elleuch, B. Jarboui,
J. Pei, N. Mladenović

G–2018–53

July 2018

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Variable neighborhood programming for symbolic regression

Souhir Elleuch [a]
Bassem Jarboui [b]
Jun Pei [c]
Nenad Mladenović [d]

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

[b] *Emirates College of Technology, Abu Dhabi, United Arab Emirates*

[c] *School of Management, Hefei University of Technology, China*

[d] *GERAD & Mathematical Institute, SANU, Belgrade, Serbia*

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

July 2018
Les Cahiers du GERAD
G–2018–53

Abstract: In the field of Automatic Programming (AP), the solution of a problem is a program, which is usually presented by a tree with a specific structure. This tree contains different types of nodes, and is called an *AP tree*. For solving AP problems, we propose a new local search procedure that adapts the known 'elementary tree transformation' (ETT) into this specific tree. Our results indicate that the neighborhood size of an AP tree is for the order of magnitude smaller than the neighborhood size of a tree with one type of node. As our new ETT local search can be part of many meta-heuristics, it can be used to solve various AP problems. In this paper, we incorporate it into the Basic variable neighborhood programming (BVNP) scheme to solve the Symbolic regression problem. Computational experiments were conducted to test the performance of our proposed method; it was compared with the three well known automatic programming methods, i.e., the VNP method without ETT, Genetic programming, and Artificial bee colony programming. The obtained results show clearly the greater ability of our method, with respect to convergence speed and computational stability.

Keywords: Artificial intelligence, automatic programming, variable neighborhood programming, elementary tree transformation, symbolic regression

Acknowledgments: This work is partially supported by the National Natural Science Foundation of China (Nos. 71601065, 71231004, 71501058, 71690235, 71690230, 71531008), and Innovative Research Groups of the National Natural Science Foundation of China (71521001), the Humanities and Social Sciences Foundation of the Chinese Ministry of Education (No. 15YJC630097), Anhui Province Natural Science Foundation (No. 1608085QG167).

1 Introduction

There is a fast-growing interest in developing techniques to solve problems automatically in the research fields of Artificial intelligence and Machine learning. The term Automatic Programming (AP) indicates that programs are automatically generated, and no human intervention is needed. The solution to an AP problem is a program that is usually presented by the tree with a specific structure, called an *AP tree*. Such a tree has different types of nodes or vertices, i.e., nodes that present operations, variables, and constants.

Genetic programming (GP), one of the well-known techniques in AP, is based on Genetic Algorithm (GA) operators (Koza in 1992 [1]). GP is used in many fields, such as Symbolic regression [2, 3, 4], Forecasting [5, 6], Data-mining [7], and Classification [8]. There are many GP based methods that include local searches [6, 9]. Good results are obtained by using such methods for solving the Symbolic regression problem [9] and Energy consumption forecasting problem [6]. In [10], Automatic programming via Iterated local search (APRILS) is proposed, where the local search is presented by a selection of random solutions obtained by mutation.

Recently, a new AP method, called Variable Neighborhood Programming (VNP), based on the Variable neighborhood search (VNS) meta-heuristic [11] is introduced by Elleuch et *al* [12]. A set of neighborhood structures, defined for the current incumbent tree T, is systematically changed in search for a better tree T'. The VNP method is applied to solve Forecasting and Classification problems, using three neighborhoods, but only in the *Shaking* step. Since the local search step is omitted, VNP proposed in [12] is a Reduced VNP. A preliminary version of the Basic VNP, applied to Forecasting problem, was presented in [13].

In this paper, we have made the following contributions:

i) *New local search.* We propose a new neighborhood structure, and therefore, a new local search in AP, which can be used within several AP techniques: Genetic programming, Artificial bee colony programming, and Swarm intelligence programming. We use this neighborhood structure to adapt the well known 'elementary tree transformation' (ETT) of a regular tree graph to this specific AP tree. We proved that the cardinality $|N(T)|$ of an AP tree is smaller than the number of neighbors of an undirected tree with one type of node in the order of magnitude.

ii) *Basic VNP for solving Symbolic regression.* The new local search is included into the Reduced VNP [12] to get Basic VNP (BVNP). In order to check the quality of BVNP, we choose the most common problem from AP literature, i.e., *the Symbolic regression* problem.

iii) *Comparative analysis.* Extensive comparative analysis was performed on ten test functions used in the literature. The computational experiments are conducted to test the performance of our proposed method; it was compared with three well known automatic programming methods, i.e., the VNP method without ETT, Genetic programming, and Artificial bee colony programming. The computational results show clearly the greater ability of our method with respect to convergence speed as well as computational stability.

The paper is organized as follows. The steps of ETT for solving the AP problem are described in Section 2. In Section 3, the implementation of this local search within the basic VNP is explained. Experimental results and the comparison among the Reduced VNP, the new Basic VNP, GP, and ABCP algorithms are given in Section 4. Finally, Section 5 presents conclusions and outlines possible directions for the future work.

2 Elementary tree transformation in Automatic programming

2.1 ETT in the tree of an undirected graph

ETT is a well known transformation technique of a tree in an undirected graph $G(V, A)$, consisting of two sets, a node set V and an edge set A. Let $T(V, E)$ denote any spanning tree of G, ETT transforms a tree T into a tree T' in two steps (T'=ETT(T)):

(1) add an edge a to T such that it belongs to A, but not to E ($a \in A \setminus E$);

(2) detect a unit circle obtained, and remove any edge from that circle to get a subgraph T'.

The following obvious statement holds.

Proposition 1 *The resulting subgraph T', obtained from T by steps (1) and (2), is a spanning tree.*

All possible edge additions, followed by all possible removals from the corresponding circles, constitute the neighborhood of T, i.e., the set of trees $N(T) = \{T' \mid T' = \mathrm{ETT}(T)\}$. If the solution of a combinatorial optimization problem is a spanning tree, the definition of a neighborhood $N(T)$ allows us to establish a local search procedure with respect to ETT neighborhood structure:

(1) find a tree $T^* \in N(T)$ with the best value of the objective function;
(2) if a better solution is obtained, then $T \leftarrow T^*$, and return to (1); otherwise, stop with T^* as the local optimum.

Proposition 2 *The cardinality of $N(T)$ is less than $\frac{n(n-1)(n-3)}{2}$.*

Proof. If we assume that the initial graph is complete, then the number of possible edges to be added to T is $\frac{n(n-1)}{2} - n = \frac{n(n-3)}{2}$. The largest circle to remove an edge from has $n-1$ edges. After multiplying this two numbers, we get the result from above. □

2.2 ETT in AP tree

In this section, we try to answer the question whether ETT is possible in the specific AP tree. First, we give an example of a small tree.

Figure 1 illustrates the expression of:

$$\max\left\{\frac{3}{x_3 \cdot 2.4}; x_1 - x_2\right\}, \tag{1}$$

which can be presented as an AP tree with the functional set $F = \{\phi_1, ..., \phi_4\} = \{\max, +, -, \cdot\}$, and the terminal set $Y = X \cup C$, where $X = \{x_1, x_2, x_3\}$ is the set of the problem variables, and $C = \{c_1, c_2\} = \{3, 2.4\}$ denotes the set of constants; $k = 3 + 2 = 5$ is the cardinality of Y.

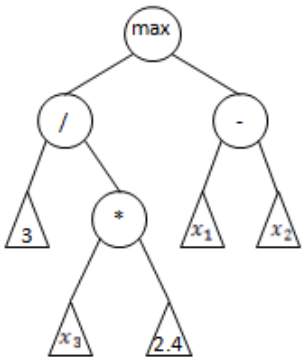

Figure 1: Solution presentation as AP tree

Let us first denote an AP tree rooted in r as $T(r, F, V, A, d))$, where F is the set of functional nodes, and V is the set of terminal nodes. The set A includes all edges of the tree T, and array $d = (d_1, ..., d_n)$ denotes the node degrees of T. Note that in a binary tree, the degree of a functional node is 3 if the functional node represents a binary operation, otherwise it is 2. The degree of a terminal node is equal to 1 (there is no child).

The steps of the ETT move on the AP tree are as follows.

- **Add an edge** $a = (i, j)$ to the current tree T, where nether i nor j is a root node $(i, j \neq r)$; in addition, at least i, or j must be a functional node; update degrees d of the tree;
- **Drop an edge** b $(b \in A, b \neq a)$ from the cycle of T,
- **If** both i and j are functional nodes, then choose b such that it is either connected to i or j; update degrees d;
- **If** one of the nodes is a terminal node (let it be j for example), then choose b such that its one end point is j; update degrees d;
- **Drop an edge** $c \neq a$, connected to the node having higher degree than its original degree, update d;
- **Add an edge** to the tree T to keep degree of each node feasible and update d.

In Algorithm 1, a pseudo code of our AP-ETT move is given.

Algorithm 1 ETT(T)

let $i \in F$ be a current node
Find (a node $j \in F \cup V$ and the one non adjacent to i) knowing that (i and j are not previously linked; j does not belong to the subtree rooted in i, and i does not belong to the subtree rooted in j)
$T' \leftarrow T \cup \{(i, j)\}$; $d_i \leftarrow d_i + 1$; $d_j \leftarrow d_j + 1$
if $j \in F$ **then**
  // case 1
  Find (the edge (l, k) of a cycle in T') where $(l \in F \cup V)$, and $(k \in i, j)$, and $((l, k) \neq (i, j))$ // two possibilities
  $T' \leftarrow T' \setminus \{(l, k)\}$; $d_k \leftarrow d_k - 1$; $d_l \leftarrow d_l - 1$
  Find (an edge (m, k')) where $(m \in F \cup V)$, and $(k' \in i, j)$, and $((m, k') \neq (i, j))$, and (m is not the parent of k') // two possibilities
  $T' \leftarrow T' \setminus \{(m, k')\}$; $d_{k'} \leftarrow d_{k'} - 1$; $d_m \leftarrow d_m - 1$
else
  // case 2
  Let l be the parent node of j
  $T' \leftarrow T' \setminus \{(l, j)\}$; $d_j \leftarrow d_j - 1$; $d_l \leftarrow d_l - 1$
  Find (an edge (m, i)) where $(m \in F \cup V)$, and $((m, i) \neq (i, j))$, and (m is not the parent of k') // two possibilities
  $T' \leftarrow T' \setminus \{(m, i)\}$; $d_i \leftarrow d_i - 1$; $d_m \leftarrow d_m - 1$
end if
$T' \leftarrow T' \cup \{(l, m)\}$; $d_l \leftarrow d_l + 1$; $d_m \leftarrow d_m + 1$
Return T'

Figure 2 illustrates the steps presented in Algorithm 1 at the example from Figure 1, when the first added edge connects two functional nodes.

From the example above, it can be observed that we made two edge additions and two edge deletions. The second add-drop pair is made to keep the degree of functional nodes equal to three, which is the necessary condition for any AP tree. Indeed, all operators in set F are binary and need 2 child nodes. From Algorithm 1, we can conclude that the following proposition holds.

Proposition 3 *Following the steps of Algorithm 1, the resulting AP subgraph T', obtained from the AP tree T, is an AP tree as well.*

Proof. By adding an edge to the AP tree T, we get a unique circle. By dropping an edge from that circle, we again have a tree T''. However, T'' is not a feasible AP tree since the desired degrees of functional nodes are ruined. Neither added nor dropped edges cannot contain terminal nodes at their both ends. Indeed, if for example, both ends of an added edge are terminal nodes, then that edge should be dropped immediately from the circle to assure that terminal nodes are leaves. Since at least one end point of the added or dropped vertex is functional, an additional transformation of T'' is necessary. It is easy to show that one more add-drop move is sufficient to recover the AP feasibility. □

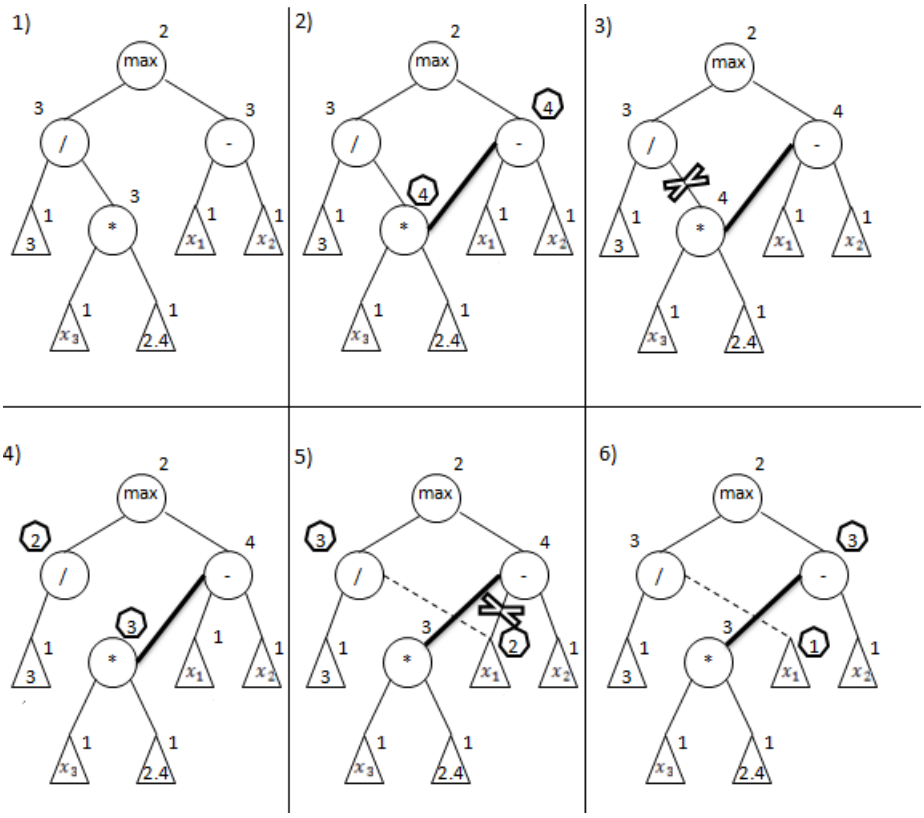

Figure 2: An example of ETT move in AP tree: from the expression $T = \max\left\{\frac{3}{x_3 \cdot 2.4}; x_1 - x_2\right\}$, the neighboring $T' = \max\left\{\frac{3}{x_1};\ x_3 \cdot 2.4 - x_2\right\}$ is obtained.

2.3 Enumeration of all neighborhood trees at the example

To explain better the progress of the Algorithm, we suppose that the current tree is the one illustrated in Figure 1. Firstly, we have to choose node $\phi \in F$, which is a functional node different from the root. Therefore, ϕ should belong to the set $F = \{/, -, \cdot\}$. Let us enumerate the complete ETT neighborhood of T.

We assume that $i =' /'$ is a current node. We have to find a node j belonging to $F \cup V$, not adjacent to i, i and j are not previously linked, j does not belong to the subtree rooted in i and i is not a node from the subtree rooted in j. As a result, j can be $'-'$, x_1, or x_2. If $j =' -'$, which is a functional node, we can get four expressions (2· 2), as given in Algorithm 1, case 5: $\max\{\frac{3}{x_1 - x_2}; x_3 \cdot 2.4\}$, $\max\{3; \frac{x_3 \cdot 2.4}{x_1 - x_2}\}$, $\max\{x_1; (\frac{3}{x_3 \cdot 2.4} - x_2\}$, $\max\{x_2; \frac{3}{(x_3 \cdot 2.4)} - x_1\}$. If $j = x_1$, i.e., if $j \in V$, we can deduce two different expressions (case 11 from Algorithm 1): $\max\{\frac{3}{x_1}; x_3 \cdot 2.4 - x_2\}$, $\max\{\frac{x_3 \cdot 2.4}{x_1}; 3 - x_2\}$. If $j = x_2$, we have two resulting expressions: $\max\{\frac{x_3 \cdot 2.4}{x_2}; 3 - x_1\}$, $\max\{\frac{3}{x_2}; x_3 \cdot 2.4 - x_1\}$.

We gave all the possibilities of the generated tree when $i =' /'$. We suppose now that there is $i =' -'$. Thus, j can get one of these node values: $\{/, \cdot, 3, x_3, 2.4\}$. When $j =' /'$, we notice that the nodes $'/'$ and $'-'$ are linked previously. Thus, we move to the case when $j = \cdot$, where j is a functional node. Therefore, we will apply case 5 from Algorithm 1. The possible expressions are : $\max\{\frac{3}{x_1}; x_3 \cdot 2.4 - x_2\}$, $\max\{\frac{3}{x_2}; x_3 \cdot 2.4 - x_1\}$, $\max\{\frac{3}{x_3 \cdot (x_1 - x_2)}; 2.4\}$, and $\max\{\frac{3}{2.4 \cdot (x_1 - x_2)}; x_3\}$. The first function is also found when $i = /$ and $j = x_1$, and the second function is already generated when $i = /$ and $j = x_2$. If $j = 3$, the moves can result in two expressions that are respectively generated when $i =' /'$, $j = x_1$, and $i = /$, $j = x_2$: $\max\{\frac{x_3 \cdot 2.4}{x_1}; 3 - x_2\}$, $\max\{\frac{x_3 \cdot 2.4}{x_2}; 3 - x_1\}$. If $j = x_3$: $j \in V$, then we follow the steps given in the case 11 of Algorithm 1. The expressions of the new trees are: $\max\{\frac{3}{2.4 \cdot x_1}; x_3 - x_2\}$, $\max\{3/(2.4 \cdot x_2); x_3 - x_1\}$. If $j = 2.4$, then $j \in V$, and the possible generated expressions are: $\max\{\frac{3}{x_3 \cdot x_1}; 2.4 - x_2\}$, $\max\{\frac{3}{x_3 \cdot x_2}; 2.4 - x_1\}$.

We assume now that $i =' \cdot'$. So, j can be $'-'$, x_1 or x_2. We have already linked the node $'\cdot'$ with the node $'-'$. Now, we will give the expressions when $j = x_1$ or $j = x_2$. These are terminal nodes, therefore, for each value, we obtain two expressions: $\max\{\frac{3}{2.4 \cdot x_1}; x_3 - x_2\}$, $\max\{\frac{3}{x_3 \cdot x_1}; 2.4 - x_2\}$, $\max\{\frac{3}{x_3 \cdot x_2}; 2.4 - x_1\}$, $\max\{\frac{3}{2.4 \cdot x_2}; x_3 - x_1\}$.

2.4 Bound on cardinality of ETT(T)

To calculate the number of possible neighbors generated from a tree $T(r, F, V, A, d)$ when applying $ETT(T)$, we need to know the structure of T. With the same sets, F and V, we can get more than one AP tree. Thus, we consider that the AP tree has the largest possible number of neighbors to obtain upper bound on cardinality. In other words, all terminals need to have the same depth and the degree of each functional node should be equal to 3.

Proposition 4 *The cardinality of $N(T)$, where T is an AP perfect tree, is less than $(n-1)(n-2)$.*

Proof. Let T be an AP tree, and let n_1 and n_2 denote the cardinality of F and V, respectively (F includes all functional nodes except the root). In a perfect tree, the following holds:

$$n_1 = \frac{n}{2} - 1, \quad n_2 = \frac{n}{2} + 1, \tag{2}$$

where $n = |F| + |V|$. Note that n should be an even number due to the fact that root is excluded from F and T is a perfect tree.

Let us first calculate the number of all possible added edges when their both end points belong to F. It is obviously equal to $n_1(n_1+1)/2 - n_1 = n_1(n_1-1)/2$. As explained earlier, for each added edge from F, there is only one way to drop an edge from the circle, but there are four possibilities to perform another add-drop move to recover feasibility. Thus, the cardinality of this type of move is $4[n_1(n_1-1)/2] = 2n_1(n_1-1)$.

The number of added edges in the case where one end point is in F and the other in V, is obviously $n_1 \cdot n_2$. As illustrated in the given example, there are just two options to recover feasibility in this case. Finally, we have

$$|N(T)| \;\leq\; 2n_1(n_1-1) + 2n_1n_2 = 2n_1(n_1 - 1 + n_2) = (n-1)(n-2),$$

after substituting n_1 and n_2 from (2). □

Comparing cardinalities of $N(T)$ from Proposition 2 and Proposition 4, the following statement is obvious.

Theorem 1 *The upper bound on cardinality of ETT neighborhood in an AP tree is smaller than that of a tree in an undirected graph in the order of magnitude.*

It should be noted that the result of Theorem 1 holds in worst case in both problems. If an undirected graph is sparse, then the number of possible added edges within the ETT neighborhood could be much smaller than $O(n^2)$.

2.5 Local search

The ETT based local search is presented in Algorithm 2. We employ the first improvement strategy: when an improving solution T' is found, the local search is stopped and T' is set to be the new incumbent solution. Note that the ETT move, and consequently ETT local search (ETT-LS), can be applied to both the GP individual and the VNP solution. Next, we present the new ETT-LS within a VNP heuristic.

Algorithm 2 ETT-LS(T)

terminate $\leftarrow$ *false*
repeat
  $T' \leftarrow$ ETT(T);
  if $f(T') \prec f(T)$ **then**
    $T \leftarrow T'$;
    terminate $\leftarrow$ *true*
  end if
until *terminate* = *true*

3 Basic VNP

3.1 VNP ingredients

By following the steps of the basic VNS algorithm, which includes the local search, the BVNP can be easily derived. The following four components of the BVNP need to be defined:

(i) *Shaking* or *perturbation* that aims to resolve possible local minima traps. A solution from the k^{th} neighborhood of the currently best solution T (incumbent) is selected randomly. Thus, we need to define a set of k_{max} (a parameter) neighborhoods that will be used in this step;

(ii) *Local search* that we use is our ETT-LS, which implements the *first improvement* search strategy (see Algorithm 1 and Algorithm 2);

(iii) *Neighborhood change* is performed in a usual way: if the new obtained solution based on local search is better, then it becomes a new incumbent and the next neighborhood to be explored is the first one from the list ($k \leftarrow 1$); otherwise, the next neighborhood from the list is explored in step to come ($k \leftarrow k+1$);

(iv) *Stopping condition* can be the maximum number of iterations or the maximum allowed CPU time t_{max}.

3.2 Neighborhood structures

Besides the new ETT move, we refer three neighborhood structures in Elleuch et *al* [12]. In details, the following three neighborhoods are used in our BVNP method:

- $N(T)$ denotes ETT neighborhood structure used in the local search;
- $\mathcal{N}_1(T)$ or *Changing a node value*. It conserves the shape of the tree and changes only one value of a functional or a terminal node. If we apply $\mathcal{N}_1(T)$ to the example of the program presented in Figure 1, then we find an expression with the same shape of the original one, but differing in just one value. The original expression is $\max\{\frac{3}{x_3 \cdot 2.4}; x_1 - x_2\}$. Thus, such a move can give the following expressions: $\max\{3 + x_3 \cdot 2.4; x_1 - x_2\}$, $\max\{3 \cdot x_3 \cdot 2.4; x_1 - x_2\}$, etc. This neighborhood alters the terminal node values as well. The terminal set of the example is $\{3, 2.4, 0.66, x_1, x_2, x_3, x_4\}$. Hence, possible expressions generated after one move are: $\max(\frac{2.4}{x_3 \cdot 2.4}; x_1 - x_2)$, $\max(\frac{0.66}{x_3 \cdot 2.4}; x_1 - x_2)$, $\max\{\frac{x_1}{x_3 \cdot 2.4}; x_1 - x_2\}$, etc.
- $\mathcal{N}_2(T)$ *Swap operator*. More details are given in [12]. The cardinality of $\mathcal{N}_2(T)$ is larger than the cardinality of the other two neighborhood structures. Swap operator affects the shape and the content of the current tree. In fact, starting from the tree illustrated in Figure 1, an infinite number of trees can be generated. We give here some examples, based on the same functional and terminal sets chosen as in the previous neighborhood structure: $\max(\frac{3}{x_1}; x_1 - x_2)$, $\max(\frac{3}{x_3 \cdot 2.4}; x_4 \cdot 0.66)$, $\max(\frac{3}{x_3 \cdot 2.4}; (x_4 + 3) \cdot 2.4)$, etc. We notice that in one move, one branch of the original tree is kept.

3.3 Shaking

The shaking step is given in Algorithm 3. In this procedure, we obtain the k^{th} neighbor of a tree T after k repetitions of one move. In other words, we first choose some neighborhood randomly (either *Node value change* or *Swap*), and then repeat the move k times using the selected neighborhood. After a brief experimentation, we set $k_{max} = 4$.

Algorithm 3 $Shake(T, k)$

$s \leftarrow$ *random value* $\in \{1, 2\}$
for $i = 1, k$ **do**
  Choose randomly $T' \in \mathcal{N}_s(T)$
  $T \leftarrow T'$
end for
return(T)

3.4 Basic VNP

Algorithm 4 gives the basic VNP procedure.

Algorithm 4 BasicVNP(T, k_{max})

repeat
  $k \leftarrow 1$
  while $k \leq k_{max}$ **do**
    $T' \leftarrow Shake(T, k)$
    T" $\leftarrow$ ETT-LS(T')
    Neighborhood_change$(T, T", k)$
  end while
until the stopping condition is met

To find the best representative model of a given problem, the optimization procedure should take into account a tree structure and the node values. Here, we propose three neighborhood structures, although it is clear that many others can be included. We follow the principles of recently proposed *Less is more approach* (LIMA) [14, 15, 16, 17]. In LIMA, we try to find the minimum number of ingredients that would make the new method, at least, as good as those from the state-of-the-art. As a result, we removed some neighborhoods proposed in [12]. This LIMA approach is applied, in the next section, to the Symbolic regression problem, where its effectiveness is demonstrated again.

4 Basic VNP for Symbolic regression problems

AP methods are widely applied to the symbolic regression. A large number of benchmark data sets can be found in the literature. This fact allows us to test the performance of our new approach. Symbolic regression is in fact a mathematical modeling method for analyzing numerical data in order to find an approximation of a mathematical function in the symbolic form [2]. Symbolic regression is a search for the mathematical expression that minimizes error metrics. There are several classical approaches in this field such as the regression by the Bernstein polynomial technique [18, 19], and the regression based on splines [20, 21]. Symbolic regression deals simultaneously with the search of the parameters such as linear or non-linear regression and the shape and structure of the function.

4.1 Test instances and parameter values

In this work we evaluate the performance of our algorithm on ten real-valued symbolic regression benchmark data sets. We have studied the following problems: polynomial, trigonometric, square-root, logarithmic and bi-variate functions. Table 1 shows the ten benchmark functions, which were used by Uy in 2011 [22]. This paper includes the largest set of issues in the symbolic regression, which are mostly taken from [23, 24, 25].

For solving an AP problem, Artificial bee colony programming (ABCP) heuristic is proposed in [26] and compared with GP. This method is an adaptation of Artificial bee colony meta-heuristic proposed in [27]. The specific parameters of ABCP and GP are given in Table 2, whose adjustments are taken from [26].

To make a reliable and fair comparison, we have chosen the same parameter values of BVNP as in [26] and [22], and adapted the same fitness function (Table 2, Table 3).

GP and ABCP are population based algorithms, while VNP is a trajectory based algorithm. Thus, they cannot be compared with VNP using population size parameter. However, we can use a measure based on the

number of function node evaluations to limit the number of iterations of the old VNP (RVNP) and the Basic VNP (Table 2, Table 3). This measure is not only employed by the different GP techniques [22] and ABCP method [26] but also by recent GP investigations [28, 29]. Table 3 gives the functional set and the terminal. The functional set is defined after the experimental study. In our case, we use the same functional set as in [22, 26]. Based on the target functions of this problem, we find that the solution presented in Figure 1 is adequate for this symbolic regression problem.

Table 1: Symbolic regression benchmark functions

Benchmark functions	Instances
$F_1 = x^3 + x^2 + x$	20 Random points $\subseteq [-1, 1]$
$F_2 = x^4 + x^3 + x^2 + x$	20 Random points $\subseteq [-1, 1]$
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	20 Random points $\subseteq [-1, 1]$
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 Random points $\subseteq [-1, 1]$
$F_5 = sin(x^2)cos(x) - 1$	20 Random points $\subseteq [-1, 1]$
$F_6 = sin(x) + sin(x + x^2)$	20 Random points $\subseteq [-1, 1]$
$F_7 = log(x + 1) + log(x^2 + 1)$	20 Random points $\subseteq [0, 2]$
$F_8 = \sqrt{x}$	20 Random points $\subseteq [0, 4]$
$F_9 = sin(x) + sin(y^2)$	100 Random points $\subseteq [-1, 1] \times [-1, 1]$
$F_{10} = 2sin(x)cos(y)$	100 Random points $\subseteq [-1, 1] \times [-1, 1]$

Table 2: Symbolic regression parameters adjustment of ABCP and GP

GP parameters	Value	ABCP parameters	Value
Population size	500	colony size	500
#of node evaluations	$15 \cdot 10^6$	#of node evaluations	$15 \cdot 10^6$
Crossover	0.9	Limit	500
Mutation	0.5		
Tournament size	3		

Table 3: Reduced VNP and basic VNP parameters adjustment

Parameter	**Value**
The functional set	$F = \{+, -, \cdot, /, sin, cos, exp, log\}$
The terminal set	$\{x_i \cup c\}, i \in [1, n], n$ is the number of variables, $c \in [-5, 5]$
fitness function	Sum of absolute error on all fitness cases
#of node evaluations	$15 \cdot 10^6$

Uy [22] presents various crossover operators of GP, where standard crossover (SC), semantics aware crossover (SAC), no same mate (NSM) selection, soft brood selection (SBS), context aware crossover (CAC), and semantic similarity-based crossover (SSC) are applied respectively. The explanation and the details of these techniques are available in [22].

4.2 Comparison of BVNP with other methods

Both Basic VNP and Reduced VNP are coded in *Java* programming language, and executed on Intel Core i3 cpu (330M / 2.13 GHz). To compare the performance of Basic VNP with GP and ABCP based methods, we used the same evaluation criteria from [22] and [26]: (i) *the percentage of successful runs*, (ii) *the mean best fitness*, and (iii) *best fitness*. In addition, we add (iv) *running time* criterion.

(i) **The percentage of the successful runs** for each approach are summarized in Table 4. This measure represents the total number of successful runs. A run is considered to be successful when at least one individual fitness value is lower than the threshold value (hits criterion) for all instances. To ensure a fair comparison, we use the same hits criterion 0.01 as in [22, 26]. We can see that Basic VNP algorithm has a large number of successful runs for the first three problems (F_1, F_2, F_3), and also for F_8. For F_4, SSC12 and ABCP score better percentage of successful run. For F_5, F_6, and F_7, the Basic VNP ranks the second. The last column of Table 4 clearly indicates the order of methods based on this criterion: BVNP (49.2% average success rate), ABSP (46.6%), RVNP (39.5%), SSC20 (38.3%), etc.

Table 4: Number of successful runs out of 100, within $15 \cdot 10^6$ node evaluations of each

	Functions										Average %
Methods	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	of succes
ABCP	89	50	22	**12**	**57**	**87**	58	37	33	21	46.6
SC	48	22	7	4	20	35	35	16	7	18	21.2
NSM	48	16	4	4	19	36	40	28	4	17	21.6
SAC2	53	25	7	4	17	32	25	13	4	4	18.4
SAC3	56	19	6	2	21	23	25	12	3	8	17.5
SAC4	53	17	11	1	20	23	29	14	3	8	17.9
SAC5	53	17	11	1	19	27	30	12	3	8	18.1
CAC1	34	19	7	7	12	22	25	9	1	15	15.1
CAC2	34	20	7	7	13	23	25	9	2	16	15.6
CAC4	35	22	7	8	12	22	26	10	3	16	16.1
SBS31	43	15	9	6	31	28	31	17	13	33	22.6
SBS32	42	26	7	8	36	27	44	30	17	27	26.4
SBS34	51	21	10	9	34	33	46	25	26	33	28.8
SBS41	41	22	9	5	31	34	38	25	19	33	25.7
SBS42	50	22	17	10	41	32	51	24	24	33	30.4
SBS44	40	25	16	9	35	43	42	28	33	34	30.5
SSC8	66	28	22	10	48	56	59	21	25	47	38.2
SSC12	67	33	14	**12**	47	47	66	38	**37**	51	41.2
SSC16	55	39	20	11	46	44	**67**	29	30	**59**	40
SSC20	58	27	10	9	52	48	63	26	39	51	38,3
RVNP	87	42	38	11	33	50	43	46	26	19	39.5
BVNP	**91**	**63**	**40**	11	49	57	60	**66**	30	25	**49.2**

(ii) **The mean best fitness** metric is more meaningful when comparing the effectiveness of AP techniques. For each problem, the *mean best fitness* of all indicated methods are given in Table 5, where the best-performance found for each function is boldfaced.

Table 5: Mean best fitness values in 100 trials, within $15 \cdot 10^6$ node evaluations for each

	Functions										Averg
Methods	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	mean
ABCP	0.01	**0.05**	0.07	0.10	0.05	0.02	0.06	0.10	0.47	1.06	0.236
SC	0.18	0.26	0.39	0.41	0.21	0.22	0.13	0.26	5.54	2.26	1.156
NSM	0.16	0.29	0.34	0.40	0.19	0.17	0.11	0.19	5.44	2.16	1.105
SAC2	0.16	0.27	0.42	0.50	0.22	0.23	0.15	0.27	5.99	3.19	1.345
SAC3	0.13	0.27	0.42	0.48	0.18	0.23	0.15	0.27	5.77	3.13	1.303
SAC4	0.15	0.29	0.41	0.46	0.17	0.22	0.15	0.26	5.77	3.03	1.284
SAC5	0.15	0.29	0.40	0.46	0.17	0.21	0.15	0.26	5.77	2.98	1.276
CAC1	0.33	0.41	0.51	0.53	0.31	0.42	0.17	0.355	7.83	4.40	1.783
CAC2	0.32	0.41	0.52	0.53	0.31	0.42	0.17	0.35	7.38	4.30	1.716
CAC4	0.33	0.41	0.53	0.53	0.30	0.42	0.17	0.35	7.82	4.32	1.773
SBS31	0.18	0.29	0.30	0.36	0.17	0.30	0.15	0.19	4.78	2.75	1.105
SBS32	0.18	0.23	0.28	0.36	0.13	0.28	0.10	0.18	4.47	2.77	1.052
SBS34	0.16	0.23	0.31	0.33	0.13	0.21	0.11	0.19	4.17	2.90	1.024
SBS41	0.18	0.26	0.27	0.38	0.12	0.20	0.13	0.20	4.40	2.75	1.037
SBS42	0.12	0.24	0.29	0.30	0.12	0.18	0.10	0.16	3.95	2.76	0.964
SBS44	0.18	0.24	0.33	0.35	0.15	0.16	0.11	0.19	2.85	1.75	0.724
SSC8	0.09	0.15	0.19	0.29	0.10	0.09	0.07	0.15	3.91	1.53	0.776
SSC12	0.07	0.17	0.18	0.28	0.10	0.12	0.07	0.13	3.54	1.45	0.720
SSC16	0.10	0.15	0.23	0.26	0.10	0.10	0.06	0.14	3.11	1.22	0.640
SSC20	0.08	0.18	0.23	0.30	0.09	0.10	0.06	0.14	2.64	1.23	0.588
RVNP	0.037	0.069	0.052	0.17	0.028	0.016	0.03	0.088	0.43	0.68	0.183
BVNP	**0.0038**	**0.050**	**0.052**	**0.096**	**0.015**	**0.0071**	**0.0024**	**0.015**	**0.41**	**0.512**	**0.136**

Based on the results presented in Table 5, it is clear that the Basic VNP provides the mean best fitness value in all cases. Therefore, it gives the most adequate function outputs in comparison with RVNS, GP, and ABCP. ABCP obtains the same value as the Basic VNP only for F_2. The last column of Table 5 clearly demonstrates the superiority of VNP based approach. The order of the methods is: BVNP (0.136); RVNP (0.183); ABCP (0.236); SSC20 (0.588).

(iii) **The best fitness value.** There is another way of comparing the methods based on the output functions of the individual, i.e., the solution with the best fitness value. This information is not available for GP. Thus, Table 6 illustrates the generated functions obtained by the ABCP and the BVNP.

Table 6: Functions generated by Basic VNP and by ABCP

Function	Generated function by Basic VNP	Generated function by ABCP
F_1	$x^3 + x^2 + x$	$x^3 + x^2 + x$
F_2	$x^4 + x^3 + x^2 + x$	$x^4 + x^3 + x^2 + x$
F_3	$\frac{1.6x^3+1.54x^2}{-056x^2+1.34} + x$	$x^5 + x^4 + x^3 + x^2 + x$
F_4	$1.6 \cdot x^4 + 1.26 \cdot x^3 + 0.16 \cdot x^2 + \frac{1.19 \cdot x}{1.41 - x}$	$x \cdot e^{((\frac{(1-cos(sin(x)))}{sin(1)}+1)\cdot(x+1)-1)}$
F_5	$cos(-3.15 + sin(-1.5x - 3.115))$	$sin(x^2)cos(x) - 1$
F_6	$sin(x) + sin(x + x^2)$	$sin(x) + sin(x + x^2)$
F_7	$log(x + 1) + log(x^2 + 1)$	$\frac{x}{e^h} \cdot$
F_8	$e^{log(x)\cdot 0.499}$	$e^{\frac{rlog(x)}{2}}$
F_9	$sin(x) + sin(y^2)$	$sin(x) + sin(y^2)$
F_{10}	$2sin(x)cos(y)$	$2sin(x)cos(y)$

Note: $\cdot h = (\frac{1}{(1+sin(x)+\frac{((1-cos(sin(cos(x))+1))\cdot cos(1-sin(x))\cdot cos(x))}{(\frac{1}{sin(e^{cos(1)})}+sin(1))})}$

These two techniques generate seven functions equal to the original ones. For F_8, the output of VNP is $F_8 = e^{log(x)\cdot 0.499}$. However, it can be written as $F_8 \approx e^{\frac{log(x)}{2}}$, which is equal to the target function. VNP did not find the exact function F_3, however, as can be observed from Figure 3, the generated function has the same shape as F_3.

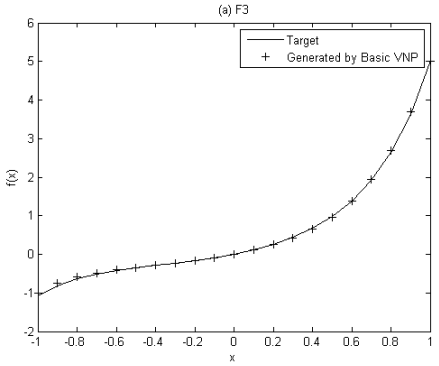

Figure 3: Target and Generated F3 functions obtained by Basic VNP

ABCP and VNP generate a function different from function F_4. The two generated functions and the real function are given in Figure 4. The curve shape of the function generated by VNP is closer to the curve of the target function than that by ABCP.

(iv) **Running times.** It is interesting to note that most of AP techniques do not include running time in their computational results. Note that in our comparison, all the methods have the same stopping condition, the maximum number of node evaluation (equal to $15 \cdot 10^6$). Using the same stopping rule, in Table 7, we report CPU times that RVNS and BVNS spent until their best values were obtained. Note that running times of the other methods were not available.

Table 7: Average running times in seconds for Reduced VNP and Basic VNP, within the same maximum number of node evaluations, on Intel Core i3 330M / 2.13 GHz

Functions	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
RVNP	0.14	0.13	0.29	0.057	7.10	4.88	6.06	3.62	7.23	6.41
BVNP	5.64	10.68	4.34	18.31	16.91	30.19	73.34	46.22	82.69	66.20

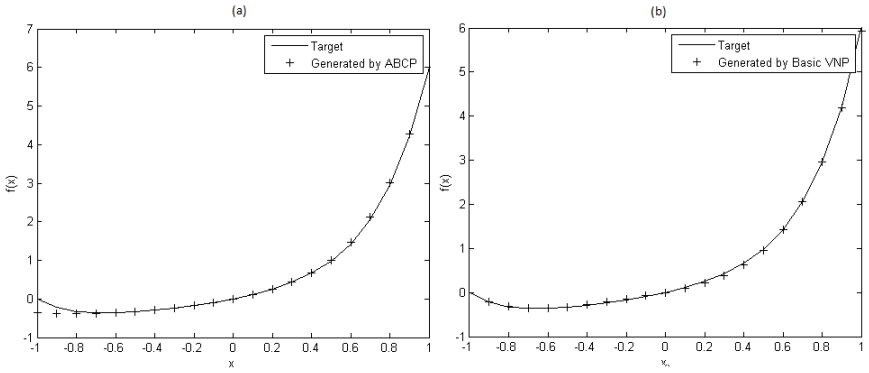

Figure 4: Target and Generated F4 functions obtained (a) by ABCP (b) by Basic VNP

The results indicate that RVNS obtains the solutions fast. However, they cannot be improved by using additional CPU times.

In conclusion, the basic VNP provides the best results for most studied problems from the literature under the same conditions. Our method is an effective way of improving the old VNP algorithm and creating a new research environment in the automatic programming field. The only limitation is the manual choice of the functional and terminal sets for both VNP and other automatic programming techniques. Furthermore, the parameter values influence the convergence speed.

5 Conclusions

We propose a new local search in the Automatic programming (AP) area. In AP, the solution is a program that is usually presented as a tree with a specific structure. There are two types of nodes, i.e., functional and terminal nodes. This tree is called as an AP tree. The new local search is based on Elementary tree transformation (ETT), a well-known transformation of a spanning tree of an undirected graph. We show that such an adaptation of ETT on an AP tree is possible, and the steps are given in details. Moreover, we prove that the cardinality of ETT implemented on an AP tree is n times smaller than that on a regular tree of a graph in the worst case. To the best of our knowledge, this is the first full local search proposed in AP area. We include ETT local search in the recent Variable neighborhood programming (VNP). Previous version of VNP did not include local search, and only Reduced VNP algorithm existed. Thus, we propose Basic VNP that incorporates our new ETT local search. Our Basic VNP is applied in ten separate sets of Symbolic regression benchmark instances. It is successfully compared with three recent meta-heuristic based approaches, i.e., Reduced VNP, Genetic Programming, and Artificial bee colony programming. Moreover, remarkable results show clearly the greater ability of our method with respect to convergence speed as well as computational stability.

In the future research, we can further extend to apply the current work into new challenging areas. Moreover, it is clear that the new ETT local search could be incorporated with other meta-heuristics, such as Genetic programming.

References

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

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

[3] W. Cai, A. Pacheco-Vega, M. Sen, K. Yang, Heat transfer correlations by symbolic regression, International Journal of Heat and Mass Transfer 49 (23–24) (2006) 4352–4359.

[4] 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 Recognition 44 (8) (2011) 1761–1776.

[5] 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.

[6] M. Castelli, L. Trujillo, L. Vanneschi, Energy Consumption Forecasting Using Semantic-Based Genetic Programming with Local Search Optimizer., Computational Intelligence and Neuroscience 2015 (2015) 971908.

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

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

[9] F. Lane, R. Azad, C. Ryan, On Effective and Inexpensive Local Search Techniques in Genetic Programming Regression, in: Parallel Problem Solving from Nature – PPSN XIII, Vol. 8672 of Lecture Notes in Computer Science, Springer International Publishing, 2014.

[10] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Automatic programming via iterated local search for dynamic job shop scheduling, IEEE transactions on cybernetics 45 (1) (2015) 1–14.

[11] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (11) (1997) 1097–1100.

[12] S. Elleuch, B. Jarboui, N. Mladenovic, Variable neighborhood programming - A new automatic programming method in artificial intelligence, Tech. rep., G–2016–92, GERAD, Montreal (2016).

[13] S. Elleuch, P. Hansen, B. Jarboui, N. Mladenović, New VNP for Automatic Programming, Electronic Notes in Discrete Mathematics 58.

[14] J. Brimberg, N. Mladenović, R. Todosijević, D. Urošević, Less is more: Solving the Max-Mean diversity problem with variable neighborhood search, Information Sciences 382-383 (2017) 179–200.

[15] L. R. Costa, D. Aloise, N. Mladenović, Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering, Information Sciences 415-416 (2017) 247–253.

[16] N. Mladenović, R. Todosijević, D. Urošević, Less is more: Basic variable neighborhood search for minimum differential dispersion problem, Information Sciences 326 (2016) 160–171.

[17] K. Concalves-de Silva, D. Aloise, S. Xavier-de Souza, N. Mladenovic, Less is more: Shorten Nelder-Mead method for large unconstrained optimization, Yugoslav journal of operations research (accepted).

[18] U. Stadtmüller, Asymptotic properties of nonparametric curve estimates, Periodica Mathematica Hungarica 17 (2) (1986) 83–108.

[19] B. M. Brown, S. X. Chen, Beta-Bernstein Smoothing for Regression Curves with Compact Support, Scandinavian Journal of Statistics 26 (1) (1999) 47–59.

[20] J. H. Friedman, Multivariate Adaptive Regression Splines, The Annals of Statistics 19 (1) (1991) 1–67.

[21] C. De Boor, A practical guide to splines : with 32 figures, Springer, 2001.

[22] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, E. Galván-López, Semantically-based crossover in genetic programming: application to real-valued symbolic regression, Genetic Programming and Evolvable Machines 12 (2) (2011) 91–119.

[23] N. Hoai, R. McKay, D. Essam, R. Chau, Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results, in: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), Vol. 2, IEEE, pp. 1326–1331.

[24] C. G. Johnson, Genetic Programming Crossover: Does It Cross over?, Springer Berlin Heidelberg, 2009, pp.97–108.

[25] M. Keijzer, Improving Symbolic Regression with Interval Arithmetic and Linear Scaling, Springer Berlin Heidelberg, 2003, pp. 70–82.

[26] D. Karaboga, C. Ozturk, N. Karaboga, B. Gorkemli, Artificial bee colony programming for symbolic regression, Information Sciences 209 (2012) 1–15.

[27] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. rep., Erciyes University, Engineering Faculty, Computer Engineering Department (2005).

[28] T.-H. Hoang, D. Essam, B. McKay, N.-X. Hoai, Building on Success in Genetic Programming: Adaptive Variation and Developmental Evaluation, in: Advances in Computation and Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 137–146.

[29] P. Wong, M. Zhang, SCHEME: Caching subtrees in genetic programming, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 2678–2685.