# New Data Structure and Spanning Forest Operators for Evolutionary Algorithms

**A. C. B. Delbem** **A. de Carvalho**
University of Sao Paulo, ICMC - USP
Sao Carlos, Brazil, 13560-970
{acbd, andre}@icmc.usp.br

**Abstract**

Network design is, in general, a combinatorial optimization problem. The minimum spanning tree (MST) is possibly one of the simplest problems in network design. Polynomial-time algorithms have been proposed for solving MST problem. However, other similar problems are generally NP-hard. Due to their complexity, recent researches have proposed the use of evolutionary algorithms (EAs) to solve them. Very encouraging results have been achieved when compared with conventional algorithms.

Nevertheless, EAs approaches still require enormous computational time when considering large-scale network problems. The tree encoding (the data structure) for EAs is a critical factor in large systems. In order to overcome this drawback, the authors propose a new tree representation and its related genetic operations. The authors apply the proposed EA approach using the new encoding to solve the electrical distribution restoration problem. The algorithm performance suggests that the contribution of the approach is relevant.

**Keywords:** Main Chain Representation, Evolutionary Algorithms, Minimum Spanning Tree, Dynamic Graphs, Distribution Systems Restoration.

## 1 Introduction

Many network design problems are combinatorial. The transportation problem, the telecommunication network design, the distribution system reconfiguration, the computer network restoration are some examples [15, 11, 5, 3]. Relevant advances in the solution of network design occured after the first formulation of the minimum spanning tree problem by Boruvka in 1926 [8]. At this time, he was working in the solution of the most economical layout of a power-line network [10]. Since then, some polynomial-time algorithms for solving MST problem were developed by Kruskal, Prim and Dijkstra [9].

However, several network design problems in the real world have some differences from the MST problem. Since this problems are related to the MST, they are also called extensions of the MST. In general, the extensions are NP-hard [8]. The degree-constrained MST, the quadratic MST, the multicriteria MST and their correspondents for spanning forest are some examples. Due to their complexity, some researches have recently proposed evolutionary algorithms (EAs) to solve them. Some relevant results have been achieved compared with conventional algorithms.

Nevertheless, the EAs approaches still require enormous computational time when considering large-scale network problems. The tree encoding (the data structure) for EAs is a critical factor in large systems. This fact is easy to see. Consider an array of $m$ bits where each index corresponds to an edge of a graph $G$ with $n$ nodes (this representation is called *edge encoding*). In this encoding, a bit value indicates presence (1) or absence (0) of an edge in a subgraph of $G$. Then, select randomly $n+1$ edges of $G$ (i.e., $n+1$ bits receive the value one and the remaining ones receive zero) producing the subgraph $S$ of $G$. It is quite likely that $S$ would be not a tree, since $S$ may not be connected or may have loops. Moreover, the probability of producing a tree is drastically reduced as $n$ increases. Thus, edge encoding is not an efficient data structure to represent trees in EAs approaches for large-scale network design.

Other tree representations are available in the literature [8]. They can be classified in three categories: edge encoding, node encoding and edge-node encoding. Node encoding is clearly the most suitable one. However, it has some drawbacks. The main disadvantage is that it does not possess *locality*, since small changes in a representation of a tree make drastic changes in the tree.

In order to improve the EAs performance, we propose a new data structure for tree encoding. This representation is based on graph chains. We also present the genetic operations using the new encoding to generate new individuals (solutions).

We apply the proposed EA approach using the new tree encoding to solve the electrical distribution restoration. In fact, the proposed encoding also allows the representation of forests. The algorithm performance solving the restoration problem suggests the contribution of the approach is relevant.

The next Section briefly review some basic EAs concepts. Section 3 presents the main tree encodings available. In Section 4, we show the proposed data structure for forest representation. Section 5 explains the proposed genetic operations using the new encoding. In the Section 5, we present the resultant EA approach. Section 7 considers tests with this methodology in the restoration of electrical distribution systems. Finally, we sum up the main characteristics of the proposed approach in Section 8.

## 2 Evolutionary Algorithms

This Section presents an overview of EAs. These algorithms are procedures based on the theory of species evolution [6]. The most famous EA may be the Genetic Algorithm (GA) based on the genetic principles of crossover and mutation. The EA should have basically the following characteristics: each solution of a problem is treated as an individual of a population (set of possible solutions). Based on criteria, some individuals of the population are selected. From these individuals, new ones with some changes are created.

Each new individual is evaluated according to a chosen criterion. These individuals may be included in the population depending on the results of the evaluation analysis. The old individuals may be extracted from the population. These changes in the population produce a new population. This process of creating individual and changing the population is repeated an arbitrary number of times. The last population is supposed to have improved individuals (better solutions for the problem).

Each individual is computationally represented in a data structure called *chromossome*. The most common chromossome structure is a string of characters (for example, a string of bits). In general, an encoding algorithm translates a solution to its chromossome form. A decoding algorithm may also be used for interpretation of the chromossome meaning.

Although the optimality of the final solutions can not be guaranteed, this kind of methodology has shown relevant results for optimization problems with non-linear (even discontinuous) objective functions as well as for combinatorial problems.

The EAs can be separated in three basic categories: Evolutionary Programming (EP), Genetic Algorithms (GA) and Evolutionary Strategies (ES). They are basically distinguished by the combination of some algorithm strategies: types of population selection, genetic operators used, selection process of survivors, and so on [6, 14, 12].

## 3 Tree Encoding Approaches

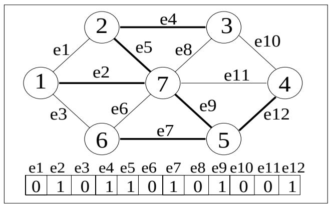
The tree encoding in EAs is critical for network design problems since any new generated chromossome should correspond to a tree (Appendix A presents an overview of the main graph concepts used in this paper). The encodings developed for representing trees can follow three different approaches:

1. Edge encoding;
2. Node encoding;
3. Edge-node encoding.

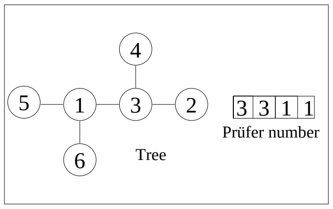
An effective representation for a tree should possess the following encoding features [16]:

1. It should be capable of representing all possible trees and only trees;
2. It should be easy to encoding and to decoding for the fitness function evaluation;
3. It should possess locality in the sense that small changes in the representation make small changes in the tree;
4. It should be capable of encouraging short schemata so that the population can evolve toward more fit chromossomes;
5. It should be unbiased in the sense that all trees are equally represented, i.e. all trees should be represented by the same number of encodings.

The next Subsections present the main characteristics of each approach for of tree representation.



(a) A graph and its edge encoding for a spanning tree.



(b) A tree and its Prüfer number.

Figura 1: Examples of the tree encodings: (a) edge encoding; (b) node encoding.

### 3.1 Edge Encoding

Consider an array $e$ of $m$ bits where each index corresponds to an edge of a graph $G$ with $m$ edges and $n$ nodes. Each bit value indicates presence (1) or absence (0) of an edge in a subgraph of $G$. See the example shown in Figure 1(a).

As we argued in Section 1, this encoding has low probability to obtaining a tree. Thus, the encoding feature (1) (required for an effective representation, see the introduction of Section 3) is not preserved by the edge encoding.

### 3.2 Node Encoding

It is possible to obtain a node encoding for trees using the Prüfer number [8]. Prüfer established an one-to-one correspondence between each possible spanning tree of a complete graph with $n$ nodes and each string that can be produced with $n-2$ digits. That is, we can use only $n-2$ digits permutation to uniquely represent a tree where each digit is an integer between 1 and $n$ inclusive. Each permutation produces a string, which is also called Prüfer number.

This paper will use an example (extracted from [8]) to explain how to obtain a Prüfer number for a tree. Using the same example, we also show the reverse operation, i.e. how a tree can be generated from a Prüfer number. The Prüfer number (3 3 1 1) corresponds to a spanning tree on a six-node complete graph represented in Figure 1(b). The encoding procedure begins locating the leaf node with the smallest label. In this case, it is node 2 (see Figure 1(b)). Since node 2 [1] is adjacent to node 3 in the tree, assign 3 to the first digit in the Prüfer number, and then remove node 2 and edge $(2,3)$ producing a subtree. Now, node 4 is the leaf with the smallest label. Since node 4 is also adjacent to node 3 in the subtree, assign 3 to the second digit in the Prüfer number, and then remove node 4 and edge $(4,3)$. Repeat the process on the resultant subtree until the subtree has only one edge (in this case, $(1,6)$) and the Prüfer number of this tree with four digits is finally produced.

Conversely, it is possible to generate the tree of Figure 1(b) from the Prüfer number $P = (3\ 3\ 1\ 1)$. Let $\overline{P}$ be the set of all nodes not included in $P$, which designates the eligible nodes for the construction of a tree. For $P = (3\ 3\ 1\ 1)$, the eligible nodes are 2, 4, 5 and 6, then $\overline{P} = \{2,4,5,6\}$. Node 2 is the eligible node with the smallest label. Node 3 is the leftmost digit in $P$. Add edge $(2,3)$ to the tree, remove node 2 from $\overline{P}$ for further consideration, and remove the leftmost digit 3 of $P$ leaving $P = (3\ 1\ 1)$. Node 4 is now the eligible node with the smallest label and node 3 is now the leftmost digit in remaining $P$. Then add edge $(4,3)$ to the tree, remove node 4 from $\overline{P}$ for further consideration, and remove digit 3 from $P$ leaving $P = (1\ 1)$. Since node 3 is now no longer in the remaining $P$, it become eligible and is put in $\overline{P} = \{3,5,6\}$. Repeat this process until $P$ is empty and all edges have been added to the tree.

It is easy to see that one of the main drawbacks of the Prüfer number is its relatively little locality, since changing even one digit of a Prüfer number can change the resultant tree dramatically [16][2]. Another disadvantage of the Prüfer number is that it may generate trees that are not spanned from a graph $G$ if $G$ is not complete. The worst case occurs if $G$ is sparse, the generation of valid trees using the Prüfer number becomes unlikely. It is also important to note several network design problems do not correspond to complete graphs.

Examples of other node encodings can be seen in [16, 1].

[1]Node 2 has only one node adjacent to it since node 2 is a leaf.

[2]In order to verify the little locality of the Prüfer number encoding, just increment by one the last digit of Prüfer number of our example $P = (3\ 3\ 1\ 1)$ producing $P = (3\ 3\ 1\ 2)$. Then, generate the corresponding tree using the procedure illustrated above.

### 3.3 Edge-node Encoding

The edge-node representation requires that the chromossome hold values for each node and each edge. These values are in the range from 0 to 255. The spanning tree corresponding to the encoding is found by running the Prim's minimum spanning tree algorithm on a modified cost matrix [16, 1]. This encoding has three main disadvantages:

1. It requires a very long encoding (memory cost);
2. It needs a conventional minimum spanning tree algorithm to generate a tree from an encoding (computational cost);
3. It contains no useful information about a tree, such as degree, connections, and so on.

## 4 Representing Forests by Graph Chains

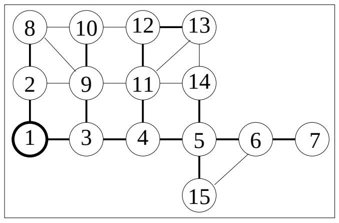
Graph chains (see Appendix A) can be used to represent a forest. In Section 4.1, we introduce a representation for trees, and then we show that a forest encoding (Section 4.2) can be obtained by the union of separated encodings of the trees of a forest.

In order to represent a tree, we use only the chains that connect a leaf with a root. This chains were called **main chains** (see Appendix A).

### 4.1 Tree Encoding by Main Chains

A tree has a number of main chains equal to its number of leaves $l$. The tree representation is composed by the set of all $l$ main chains. For example, the representation of the tree from Figure 2(a) is in Figure 2(b). Figure 2(c) shows the same set of chains disposed in a different order. In this arrangement, the same node in different chains are side by side. This property is important and it is highlighted in the following definition.

**Property 1**: Let $S$ be the set of main chains representing a tree $T$. If all the equal nodes in different chains are side by side, the chains are said to be *properly grouped*.



(a) A graph $G$ with a spanning tree $T$. $T$ is indicated by the thick edges.

```
1 3 4 5 14
1 2 8
1 3 4 11 12 13
1 3 4 5 6 7
1 3 9 10
1 3 4 5 15
```

(b) The main chains of the spanning tree.

```
1 2 8
1 3 9 10
1 3 4 11 12 13
1 3 4 5 14
1 3 4 5 6 7
1 3 4 5 15
```

(c) The main chains properly grouped.

Figura 2: A graph with a spanning tree and its main chains.

In order to encoding a tree adequately, it is important that the tree have a representation by main chains with the chains *properly grouped*. In this way, the following proposition must be true.

**Proposition 1**: Any tree can be *uniquely* represented by main chains with the chains *properly grouped*.

The justification for the **Proposition 1** is presented in Appendix B.

We can implement the proposed tree representation using $l$ vectors to store the $l$ main chains and an array of pointers addressing each one of the vectors. In this way, the array of pointers determine the order that the main chains are disposed.

### 4.2 Forest Encoding by Main Chains

The forest encoding is composed by the union of the separated encodings of the trees of a forest. In this way, the forest data structure can be easily implemented using an array of pointers where each pointer indicates a tree represented by main chains.

We presented in Section 3 five encoding main features that a tree representation should possess. It is easy to see that the proposed representation using main chains hold such encoding characteristics.

## 5 Operations on Forests Encoded by Main Chains

The proposed operators will be described using a didactical example and several illustrations. The references [4, 5] provide some additional explanation about the operators.

This Section presents two operators to generate a new forest using the forest encoding based on main chains. They will be called **operator 1** and **operator 2**. Both operators generate a spanning forest $F'$ of a graph $G$ when they are applied to a other spanning forest $F$ of $G$.

The result produced by the application of both operations are similar. That is, one application of the operator 1 (or 2) to a forest is equivalent to transfer a subtree from an tree $T_{from}$ to another tree $T_{to}$ of the forest. The difference between them is that, applying operator 1, the root of the pruned subtree will be also the root of this subtree in its new tree ($T_{to}$). Nevertheless, the transfered subtree will have a new root (any node of the subtree different from the original root) when applying operator 2.

In this way, we provided two operators: one capable to produce simple and small changes in the forest (operator 1); and a second one that can generate larger and more complex changes (operator 2).

The operation 1 requires a set of two nodes: the prune node ($n_p$), which indicates the root of the subtree to be transfered; and the adjacent node ($n_a$), which is a node of a tree different from $T_{from}$ and that is also adjacent to $n_p$ in $G$.

The operation 2 requires a set of three nodes: the prune node ($n_p$), which indicates the root of the subtree to be transfered; the new root node ($n_r$) of the subtree; and the adjacent node ($n_a$), which is a node of a tree different from $T_{from}$ and that is also adjacent to the new root $n_r$ in $G$.

In the following, we explain both operations considering that the required set of nodes are previously determined. We show how to efficiently obtain these sets of nodes in Appendix D.

### 5.1 Operator 1

The operator 1 generates a spanning forest $F'$ of a graph $G$ when it is applied to another spanning forest $F$ of $G$. The result of the application of the operator 1 is equivalent to transfer a subtree from an tree $T_{from}$ to another tree $T_{to}$ of the forest. The root of the pruned subtree will be also the the root of this subtree in its new tree $T_{to}$.

We assume that two nodes are previously known: the prune node $n_p$ and the adjacent node $n_a$. Since we know the nodes $n_p$ and $n_a$, we also know their respective trees $T_{from}$ and $T_{to}$. Besides we know the first chain of $n_p$ in $T_{from}$ and its position in this chain. Analogously, we also know the first chain of $n_a$ in $T_{to}$ and its position in this chain (see Appendix C).

The operator steps will be illustrated using the graph of Figure 3(a). We will consider $n_p = 4$ (in $T_{from}$) and $n_a = 20$ (in $T_{to}$) in the examples for operator 1.

The operator 1 can be described by the following steps:

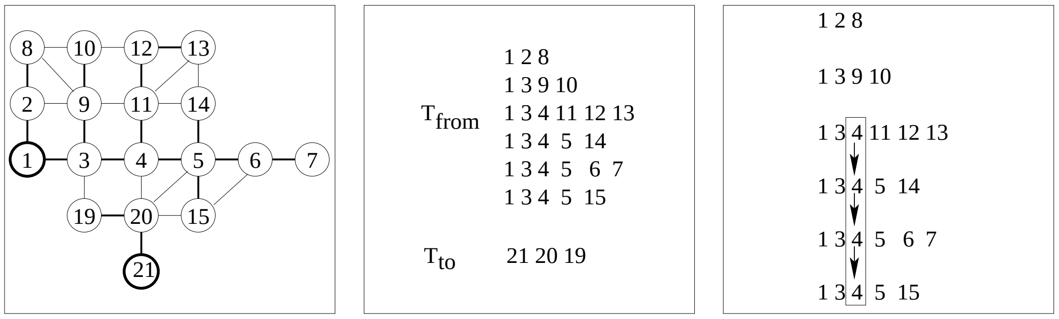
1. Identify the chains of $T_{from}$ with $n_p$. Since we know the position of $n_p$ in the first chain ($c_f$) of $T_{from}$, we just have to go down through the chains below $n_p$ looking for other node(s) equal to $n_p$. Figure 3(c) illustrates this step;

2. Concatenate the chain segment on the left of $n_b$ ($T_{to}$) inclusive with the chain segments on the right of $n_a$ in the chains identified in the step (1), generating the list $T_{tmp}$ with the concatenated chains. The chains should be concatenated following the order they appear in $T_{from}$ to preserve the **Property 1**. The concatenation is exemplified in Figure 4(a);

3. Create an array of pointers called $T'_{to}$. Create a pointer in this array to each main chain in $T_{to}$. Create also a pointer to each main chain in $T_{tmp}$;

4. Create an array of pointers called $T'_{from}$. Create a pointer in this array to each main chain in $T_{from}$ that does not have $n_p$;

5. Create an array of pointers called $F'$. Create a pointer to each tree data structure of $F$ except to $T_{from}$ and $T_{to}$. Create pointers to $T'_{from}$ and $T'_{to}$;

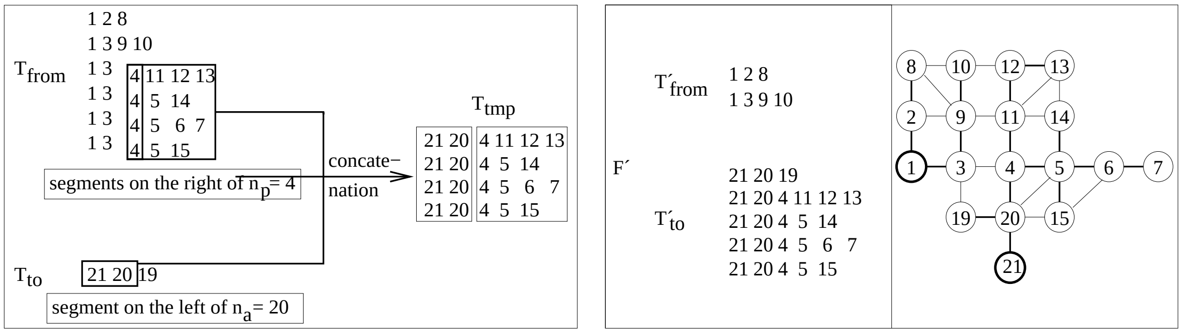


(a) A graph $G$ with an initial spanning forest $F$. $T$ is indicated by the thick edges.

(b) The main chains of the spanning forest.

(c) Determination of the chains with $n_p = 4$ in $T_{from}$.

Figura 3: An spanning forest and the determination of the chains with $n_p$.



(a) Concatenation carried out by the operator 1.

(b) Representation of the spanning forest $F'$ and its graph.

Figura 4: Example of operator-1 concatenation and the representation of the new forest $F'$.

### 5.2 Operator 2

The operator 2 also generates a spanning forest $F'$ of a graph $G$ when the operator is applied to another spanning forest $F$ of $G$. Similarly with operator 1, the result of the application of the operator 2 is equivalent to transfer a subtree from an tree $T_{from}$ to another tree $T_{to}$ of the forest. However, the transfered subtree will have a new root (any node of the subtree different from the original root).

The authors assumed that a set of three nodes are previously known: the prune node $n_p$, the new root node $n_r$ and the adjacent node $n_a$. The nodes $n_p$, $n_r$ are in the tree $T_{from}$ and $n_a$ is in $T_{to}$. Besides we also know the first chain of $n_p$ and $n_r$ in $T_{from}$ and their positions in these chains. Analogously, we also know the first chain of $n_a$ in $T_{to}$ and its position in this chain (see Appendix C).

The operator 2 is similar to the operator 1. The main difference is that the pruned subtree will have a new root after its transference to another tree. The differences between operator 1 to operator 2 occur just in the steps 1 and 2 (see the operator 1 procedure), i.e. only the process to obtain the concatenated chain list $T_{tmp}$ is different.

The operator steps will be also illustrated using the graph of Figure 3(a). The experiments in this paper considered $n_p = 3$ ($T_{from}$), $n_r = 5$ ($T_{from}$) and $n_a = 20$ ($T_{to}$) in the examples for operator 2.

It is possible to obtain the list $T_{tmp}$ reformulating the steps 1 and 2 of the operator 1 as follows:

1. Identify the segments of chains on the right of $n_p$ (inclusive) in $T_{from}$. Since we know the position of $n_p$ in the first chain ($c_f$) of $T_{from}$, we just have to go down through the chains below $c_f$ looking for other node(s) equal to $n_p$ as it was described for the operator 1. These identified segments of chains should be separated in three lists:

   (a) The list $L_1$ of segments of chains without $n_r$;

   (b) The list $L_2$ with the chain segment on the left of $n_r$;

(c) The list $L_3$ of the segments of chains on the right of $n_r$.
Figure 5 illustrates these lists for our example;

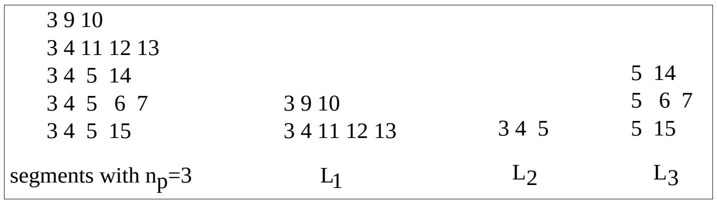


Figura 5: Identification of the Lists $L_1$, $L_2$ and $L_3$.

2. Concatenate properly the chain segment on the left of $n_a$ inclusive (in $T_{to}$) with the chain segments of $L_1$, $L_2$ and $L_3$, generating the list $T_{tmp}$. The concatenation is executed in three parts:

   (a) Concatenate the segment of chains on the left of $n_a$ in $T_{to}$ (denoted by $s_l$) with the segments in $L_3$, generating the list $T_{tmp3}$ of concatenated chains;

   (b) Reverse the segment in $L_2$ and concatenate $s_l$ with the segment inverted, generating the list $T_{tmp2}$. The concatenation of steps 2.(a) and 2.(b) are illustrated in Figure 6;

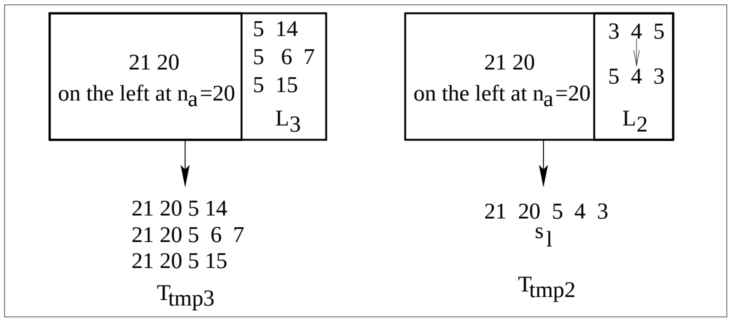


Figura 6: Concatenations for $L_2$ and $L_3$.

   (c) Concatenate the segment $s_l$ with the segments in $L_1$, generating the list $T_{tmp1}$. In this step, we need to determine the adequate parts of each chain segment that will concatenate and also the order (to preserve the **Property 1**) which the concatenated segments will be stored in the list $T_{tmp1}$.

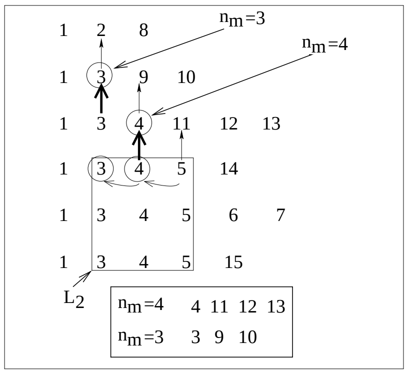
   Figures 7(a) and 7(b) illustrates the determination of the part of each segment in $L_1$ for the concatenation. The procedure corresponding to Figure 7(a) tries to find nodes equal to the nodes of $L_2$ (which contains the nodes from $n_p$ to $n_r$) in the chains above the first chain in $T_{from}$ with $L_2$. We begin looking for the last element of $L_2$ ($n_r$). If it is not found, we try the last but one element and so on. That is, when a trail fails (thin vertical arrows in Figure 7(a)), we try again using the predecessor element of $L_2$. When a trial succeeds (thick verical arrows in Figure 7(a)), the segment of chain on the right of the match node ($n_m$) is concatenated with the segment of $s_l$ on the left of $n_m$. Then, the concatenated chain is put in $T_{tmp1}$.

   In fact, it is also necessary to verify for the elements of $L_2$ in the chains below the last chain in $T_{from}$ with $L_2$. In this way, for each element of $L_2$ (from $n_r$ to $n_p$), it is required to verify first the chains above and also below the chains with $L_2$ before trying another element of $L_2$.

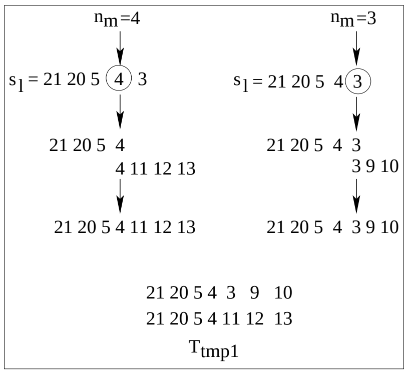
   (d) Create an array of pointers $T_{tmp}$ (see Figure 8) with pointers to the chains in $T_{tmp3}$, $T_{tmp2}$ and $T_{tmp1}$ following this order of lists and the order that the chains are disposed in each list. If the last node (at position $k$) of the chain in $T_{tmp2}$ (i.e., $s_l$) is equal to the node at position $k$ of the first chain in $T_{tmp1}$, then the chain $s_l$ is redundant and, thus no pointer should be addressed to it.

## 6 EA Approach Using the Proposed Tree Encoding

In this Section, we discuss the main characteristics of the proposed EA and present its pseudo-code. The approach using the new encoding has more similiarities to Evolutionary Programming than Genetic Algo-



(a) The thick vertical arrows indicates a match and the thin vertical ones represents a fail.



(b) Concatenation of the proper segments with $s_l$.

Figura 7: Determination of the part of segments to concatenate and the order of concatenation.

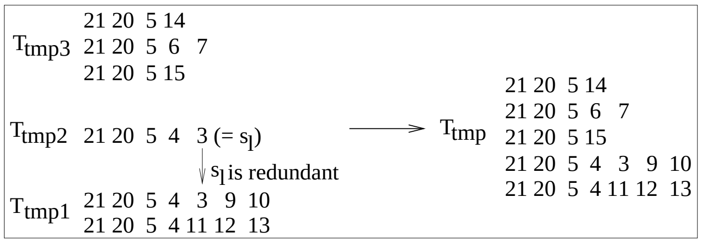


Figura 8: Attainment of $T_{tmp}$ from $T_{tmp1}$, $T_{tmp2}$ and $T_{tmp3}$.

rithms (GA) or Evolutionary Strategies (ES) [14]. However, we can not classify this evolutionary approach as a typical Evolutionary Programming (EP).

A typical EP operator does not try to emulate the genetic operators as observed in nature. The proposed operators 1 and 2 also do not intend to imitate the genetic operators of the nature. Moreover, since the EP is an abstraction of *evolution* at the level of reproductive populations (i.e., species), no recombination mechanisms are typically used because recombination does not occur between species [12]. It must be remembered that the operators 1 and 2 (see Section 5) do not carry out recombination. Nevertheless, an EP commonly uses stochastic tournament to choose the survivors, while this process is deterministic in the proposed EA.

The Algorithm 6.1 presents the pseudo-code of the EA using the encoding based on main chains.

## 7 Application to Restoration of Electrical Distribution Systems

The proposed EA approach using forest encoding based on main chains is applied to automatic elaborate plans for service restoration in electrical distribution systems. This problem considers situations that leave network regions out-of-service. The service interruption may be caused by faults in the distribution circuit or by isolation of circuit zones for maintenance task.

The restoration of the energy supply to the consumers is a multiobjective problem, with a certain degree of conflict. The restoration problem considers functions whose characteristics, in general, make difficult the use of mathematical programming techniques to obtain restoration plans. Moreover, the approaches using such functions are intensively affected by the combinatorial explosion problem.

Moreover, a proper restoration plan must be obtained rapidly since the restoration is an on-line problem.

The Evolutionary Algorithms have shown relevant results for this problem [7, 2, 13]. Nevertheless, these techniques still have difficulties to fast obtain restoration plans for real size (large-scale) networks.

Severals tests were executed, using networks with different sizes, to evaluate the potential of the proposed technique. In the following, we present the general problem formulation and then we sum up in a comparative table the main results.

**Algorithm 6.1** Pseudo-code for the proposed EA.

```
ALGORITHM proposedEA($F_0$)
// start with an initial time
$t := 0$;
// generate an initial population $P(t_0)$ based on the original spanning tree $F_0$
$P(t)$ := INITIAL_POP($P(t_0, F_0)$);
// evaluate the individuals of the initial population
EVALUATE($P(t)$);
// test for termination criterion (a time $t_{max}$)
WHILE not done DO
    // stochastically select an individual ($F_s$) in the population
    $F_s$ := STOCHASTICALLY_SELECT($P(t)$);
    // stochastically select an operator 1 or 2
    $OP$ := SELECT_OPERATOR(op1,op2);
    // apply $OP$ to produce a new individual $F_t$ from $F_s$
    $F_t$ := OP($F_s$);
    // evaluate the new individual $F_t$
    EVALUATE($F_t$);
    // deterministically select the survivors among $P(t)$ and $F_t$
    $P(t+1)$ := ALTER_POP($P(t)$,$F_t$);
    // increase the time counter
    $t := t + 1$;
END
```

The objective function considered for the restoration problem is expressed in the next equation:

$$f = \psi(F, F^0) + \omega_s||s|| + \omega_v||\Delta v|| + \omega_x||x||; \qquad (1)$$

where $\psi(F, F^0)$ is the number of different edges (switch operations) between the forest $F^0$ (the forest corresponding to the original configuration of the system) and the forest $F$ being evaluated; $s$ is a vector with the substations loads; $v$ is a vector of the voltages of the substation buses; $x$ is a vector with the current flow in the distribution lines; $\omega_s$ is the weight for the constraining violation of the maximal load of the substations; $\omega_v$ is the weight for constraining violation of the maximal voltage drop; and $\omega_x$ is the weight for the constraining violation of the maximal current flow. The weights are obviously positive values and $||\cdot||$ is the usual infinite norm. If a constraint is not violated, the corresponding penalty is not applied, i.e. $||\cdot|| = 0$.

The tests were carried out in a *Pentium III 450Mhz* with *512MRAM* using the Operational System *Debian GNU/Linux 2.2* and the *C* compiler *gcc*. For the tests, we use $\omega_s = 2$, $\omega_v = 10$ and $\omega_x = 1$. We set as 1/4 the probability of the SELECT_OPERATOR(op1,op2) (see Algorithm 6.1) to choose the operator 1, i.e. the use of operator 2 is 4 times more frequent than operator 1.

The authors applied the proposed approach to produce restoration plans for all possible situations of service interruption on each test system.

Table 1 shows the test results together with the computational performance of other approaches available in the literature. In order to construct this Table, we consider that all the algorithms achieve satisfactory restoration plans if all solutions found by them restore entirely the out-of-service regions and do not have violation of the constraints. Note that, in the comparison of these algorithms, we did not take account the perfomance of minimizing the number of switch operations. The solutions used to construct Table 1 are all satisfactory ones. Each computation time presented in this Table is the largest one found for each system tested.

| Approach | Processor | Network Scale (number of nodes) | Maximal Running Time (s) |
|---|---|---|---|
| Proposed EA | Pentium III 450Mhz | 204 | 10 |
| GA-*Fuzzy* [13] | Pentium-CELERON 300 | 102 | 150 |
| EA with *fuzzy* sets [2] | Pentium 133 | 32 | 21 |
| Parallel GA [7] | Transputer with 16 processors MIMD | 30 | 33 |

Tabela 1: The computational performance of the proposed approach and other algorithms with running time available.

The results suggest a reduction of the running time for the energy restoration problem when we apply the EA approach using the proposed forest encoding based on main chains. It is important to note that few papers about energy restoration provide the running time required by their approaches. Besides, due to the larger amount of data involved in a distribution system, the test networks are in general not available in the papers. These drawbacks complicate a more precise performance comparison of different approaches.

Fortunately, the approaches proposed in [7] provides some information about the approach performance as network size increases. Figure 9 shows graphs based on the running time required by proposed EA and the approaches presented in [7]. These graphs suggests that the proposed EA requires relatively small running time when the network size increases.

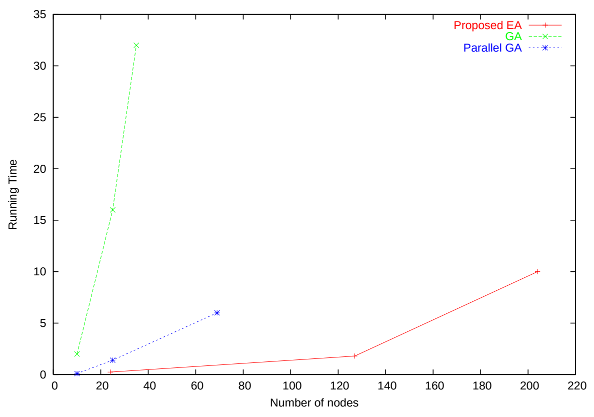


Figura 9: Relationship between the number of network nodes and the running time for three approaches: the Proposed EA, a GA and a Parallel GA (the last two are presented in [7]).

## 8 Conclusion

EAs for network layout problems requires special chromossome encoding. The authors propose a new forest encoding aiming to overcome some drawbacks of the available representations. The proposed encoding uses main chains to represent a forest. Based on this representation, we developed two new operators capable to manipulate a forest generating a new one. Besides, the proposed encoding possesses the main encoding features required for a effective tree representation (see Section 3) according to *Palmer & Kershenbaum* [16].

Moreover, the proposed operator do not require that $G$ is complete (as required by the Prüfer number encoding) to produce only feasible spanning trees of $G$. Since many practical problems do not involve complete graphs (in fact, several networks correspond to sparse graphs), the proposed encoding might improve the performance of EA approaches for a variety of problems.

The authors also elaborated an EA approach using the new forest representation. This approach was applied to solve the restoration of electrical distribution system. The computational performance of the proposed EA suggests that it is able to produce satisfactory solutions with relatively small running time. It is important to note that the restoration in an on-line problem.

At this moment, the authors are working with the development of a new operator, which is also based on main chains. Such operator intend to *recombine* two spanning forests of a graph $G$ generating a new spanning forest of $G$. The new operator might improve even more the performance of the EAs for some extensions of the MST problem.

## Acknowledgements

The authors would like to acknowledge FAPESP and CNPq for the financial support given to this research.

## Referências

[1] F Abuali, R Wainwright, and D Schoenefeld. *A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem, in Eshelman, L J, editor.* Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, 1995.

[2] A. Augugliaro, L. Dusonchet, and E. Riva Sanseverino. Multiobjective service restoration in distribution networks using an evolutionary approach and fuzzy sets. *Electrical Power and Energy Systems*, 22:103–110, 2000.

[3] P M S Carvalho, L A F M Ferreira, and L M F Barruncho. On spanning tree recombination in evolutionary large-scale network problems - application to electrical distribution planning. *IEEE Transactions on Evolutionary Computation*, 5:623–630(6), 2001.

[4] A. C. B. Delbem. *Energy Restoration in Distribution Systems using Evolutionary Algorithm with Graph Chain Representation.* University of Sao Paulo (EESC-USP)/PhD Thesis written in portuguese, feb, 2002.

[5] A. C. B. Delbem, A. C. P. L. F. Carvalho, and N. G. Bretas. A fast algorithm for generation of forests: Application to distribution system reconfiguration. *IEEE Porto Powertech 2001*, CD-ROM, 2001.

[6] A. E. Eiben and G. Rudolph. Theory of evolutionary algorithm: a bird's eye view. *Theorical Computer Science*, 229:3–9, 1999.

[7] Y Fukuyama, H-D Chiang, and K Nan Miu. Parallel genetic algorithm for service restoration in electric power distribution systems. *Electrical Power and Energy Systems*, 18:111–119, 1996.

[8] M Gen and R Cheng. *Genetic Algorithms and Engineering Design.* Ashikaga Institute of Technology, Ashikaga, Japan, 1997.

[9] E G Goodaire and M M Parmenter. *Discrete Mathematics with Graph Theory.* Prentice Hall, Upper Saddle River, USA, 1998.

[10] R Graham and P Hell. On the history of minimum spanning tree problem. *Annals of History of Computing*, 7:43–57, 1985.

[11] F. Harary and G. Gupta. Dynamic graph models. *Mathl. Comput. Modelling*, 25:79–87, 1997.

[12] Joerg Heitkoetter and David Beasley. The hitch-hiker's guide to evolutionary computation: A list of frequently asked questions (faq). *USENET: comp.ai.genetic. Available via anonymous FTP from rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic*, pages 1–100, 2002.

[13] Ying-Tung Hsiao and Ching-Yang Chien. Enhancement of restoration service in distribution systems using a combination fuzzy-ga method. *IEEE Transactions on Power Systems*, 15:1394–1400, 2000.

[14] Christian Jacob. *Illustrating Evolutionary Computation with Mathematica.* Morgan Kaufmann Publishers, San Francisco, USA, 2001.

[15] A Kershenbaum. *Telecommunications Network Design Algorithms.* McGraw-Hill, New York, 1993.

[16] C Palmer and A Kershenbaum. An approach to a problem in network design using genetic algorithms. *Networks*, 26:101–107, 1995.

## A The Graph Nomenclature Used

This section defines some graph concepts [9]. This may prevent from misunderstandings since the definitions of graph concepts are not completely standardized in the literature. It also introduces the definition of main chains, which is used in the data structure proposed for forest encoding, see Section 4.

A **graph** G consists of a pair (N(G),E(G)), where N(G) is a non-empty finite set of elements, called **nodes** and E(G) is a finite set of unordered pairs of distinct elements from N(G), called **edges**. N(G) is sometimes called the node-set of G. A pair {x, y} is said to join nodes x and y. For illustration purpose, Figure 10 represents a simple graph G whose node-set N(G) is represented by the set {u, v, w, z}, and whose edge-set E(G) consists of the pairs {u, v}, {v, w}, {u, w} and {w, z}. The **degree** of a node is the number of edges incident with it.

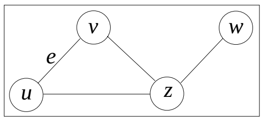


Figura 10: Example of a Graph.

Given any graph G, an edge-sequence in G is a finite sequence of edges in the form {$v_0$, $v_1$}, {$v_1$, $v_2$},..., {$v_{m-1}$, $v_m$} (also denoted by $v_0 \to v_1 \to v_2 \to \ldots \to v_m$). An edge-sequence in which all the edges are distinct is called a **path**. If, in addition, the nodes $v_0$, $v_1$,...,$v_m$ are distinct (except, possibly, $v_0 = v_m$ ), then the path is called a **chain**.

A path or chain is closed if $v_0 = v_m$. A graph is said to be connected if, given any pair of nodes x, y of G, there is a chain from x to y. An arbitrary graph can be split up into disjoint connected subgraphs called **connected components**. A connected graph has only one component; a graph with more than one component is called a **disconnected graph**.

A component without closed chains is called a **tree**. A graph with one or more trees is called a **forest**.

One of the tree nodes is usually called **root**. Such node is in general a reference where the tree initiates from. The degree of the root can be one or greater than one. We call the nodes with degree one as **leaves**, except if this node is the root.

Given a root $r$ of a tree $T$ and a leaf $t$ of $T$, there is only one chain connecting nodes $r$ and $t$. Each chain with such characteristic will be called **main chain**. Given a tree $T$, the set of all main chains is an way to represent $T$, as shown in Section 4.

A graph where for every pair of nodes $v$ and $w$ $(v,w) \in E(G)$ is called **complete**. When the number of edges of graph is much larger than the number of nodes, it is called **dense**. Conversely, when the number of edges is relatively small, the graph is called **sparse**.

## B Justification of Proposition 1

**Proposition 1**: Any tree can be represented by main chains with the chains *properly grouped.*

**Justification 1**: It is easy to see that the order in which the chains from root to the leaves are disposed in a graphic representation of a tree corresponds to the chains *properly grouped.* In this way, in order to obtain a tree representation with the chains *properly grouped*, we can proceed as follows: take the chain of the leftmost (rightmost) leaf in the graphic representation of the tree and put it in a stack. Then, take the second leftmost (rightmost) leaf in the graphic representation of the tree and put it in the stack. Repeat this procedure until all leaves have been visited. We can execute the previous procedure for any tree, thus any tree has a representation by main chains with the chains *properly grouped.*

## C Node Position in $F$

The determination of the position of a node in $F$ can be efficiently achieved using the matrices $\Pi_x$ and a vector $\pi$. Each node $n_x$ of $G$ possesses its corresponding matrix $\Pi_x$. For the original spanning forest $F_0$ of $G$, $\Pi_x$ is a column matrix: $\Pi_x = \begin{bmatrix} 0 \\ i_0 \\ j_0 \\ k_0 \end{bmatrix}$, where $i_0$ is the tree of $n_x$ $(T_{i_0})$, $j_0$ is the first chain of $n_x$ in $T_{i_0}$ and $k$ is the position of $n_x$ in this chain.

Suppose a forest $F_h$ is been generated from the $F_g$ $(g < h)$ and $n_x$ is in the subtree that will be transfered for a new tree generating $F_h$. Then, $n_x$ will have a new position in $F_h$ different from its position in $F_g$. So, we insert a new column in $\Pi_x$ with the indices of this new position. The altered matrix results in $\Pi_x = \begin{bmatrix} 0 & h \\ i_0 & i_h \\ j_0 & j_h \\ k_0 & k_h \end{bmatrix}$. The position update is carried out for all nodes of the transfered subtree in the operations 1 and 2.

The vector $\pi$ stores the parent $g$ of the forest $F_h$ in the rank $h$ of $\pi$, i.e. $\pi(h) = g$. The parent of $g$ is $\pi(g)$, the parent of $\pi(g)$ is $\pi(\pi(g))$, and so on. This constitutes a linked list with all precedessors of $F_h$. Obviously, the last position change of $n_x$ occured in one of predecessors of $h$. In this way, we can look for the predecessors of $h$ in the columns of $\Pi_x$. We start searching for $\pi(h)$. If this column is not found, we try the column $\pi(\pi(h))$, and so on. The process of looking for such columns in $\Pi_x$ can be achieved efficiently by running a binary search [9] on the list given by $\Pi_x(0,\cdot)$ (the first row of $\Pi_x$).

Once identified a column with a predecessor of $h$, we only need to read the position indices of $n_x$ stored in the same column.

## D Determination of the nodes $n_p$, $n_r$ and $n_a$

The proposed operators require a special set of nodes in order to generate a spanning forest $F'$ of $G$ based on another spanning forest $F$ of $G$.

For the operator 1, this set can be efficiently obtained by the following strategy:

1. Pick up randomly a node of a forst $F$ that is not a root. Call this node $n_p$ and determine its first position in $F$ using the matrix $\Pi_{n_p}$;

2. Pick up randomly a node adjacent to $n_p$ (using the node adjacent list of $G$). Call this node $n_a$. If $n_a \notin T$ [3], determine its position in $F$ using the matrix $\Pi_{n_a}$; else pick up randomly another $n_a$ or return to step 1.

The strategy for the determination of $n_p$ and $n_a$ for the operator 2 is:

1. Pick up randomly a node of $G$ that is not a root. Call this node $n_p$ and determine its first position in $F$ using the matrix $\Pi_{n_p}$;

2. Pick up randomly one of the chains with $n_p$[4]. Choose randomly a node of the selected chain on the right of $n_p$. Call this node $n_r$;

[3]To know whether $n_a \notin T$, we verify whether $n_a \in T$. This verification can be easily accomplished by moving through the adjacent nodes of $n_a$ in the main chain representation of $T$.

[4]An efficient procedure for the determination of the chains with $n_p$ can be seen the description of the operator 1 in Section 5.

3. Pick up randomly a node adjacent to $n_r$ (using the node adjacent list of $G$). Call this node $n_a$. If $n_a \notin T$, determine its position in $F$ using the matrix $\Pi_{n_a}$; else pick up randomly another $n_a$ or return to step 1.