Node-Depth Encoding for Evolutionary Algorithms Applied to Network Design

A.C.B. Delbem, Andre de Carvalho, Claudio A. Policastro, Adriano K.O. Pinto, Karen Honda, and Anderson C. Garcia

University of Sao Paulo - ICMC - USP, Sao Carlos - SP, Brazil
{acbd,andre,akogata,karen}@icmc.usp.br, claudio.policastro@terra.com.br, anderson@grad.icmc.usp.br

Abstract. Network design involves several areas of engineering and science. Computer networks, electrical circuits, transportation problems, and phylogenetic trees are some examples. In general, these problems are *NP-Hard*. In order to deal with the complexity of these problems, some alternative strategies have been proposed. Approaches using evolutionary algorithms have achieved relevant results. However, the graph encoding is critical for the performance of such approaches in network design problems. Aiming to overcome this drawback, alternative representations of spanning trees have been developed. This article proposes an encoding for generation of spanning forests by evolutionary algorithms. The proposal is evaluated for degree-constrained minimum spanning tree problem.

1 Introduction

Network design problems (NDPs) are present in several areas. The minimum spanning tree problem (MSTP) is an example. In the real world, many NDPs can be seen as extensions of the MSTP like, for example, telecommunication network design, computer network restoration, transportation problems, and determination of phylogenetic trees [1], [2], [3], [4], [5]. However, these extensions are in general *NP-Hard* [6], [7]. In order to deal with such complexity, some alternative strategies have been developed. Many of them utilize evolutionary approaches, with relevant results [2], [4], [5], [7], [8], [9].

Nevertheless, evolutionary algorithms (EAs) using conventional encodings produce many unconnected components or acyclic graphs when applied to large systems. The production of such graphs may consume a large amount of computational time, which reduces the efficiency of the EA approach. Depending on the adopted encoding, the generated spanning trees may be very different from their parents. Consequently, the convergence of EAs may be very slow.

To overcome this problem, an efficient representation should produce only connected components and acyclic graphs. Moreover, child spanning trees should resemble their parents. In this paper, we propose a representation with such required encoding features[1].

[1] References [10], [11] discuss more about representation characteristics for Evolutionary Algorithms.

The proposed approach generates spanning forests [12], while the available ones focus on the production of spanning trees [10], [6], [7], [8]. Its purpose is a more general representation, since several NDPs can be modelled as forests. The proposal performance was evaluated for degree-constrained minimum spanning tree problem.

The next section introduces the proposed representation. Sections 3 and 4 describe the correspondent operators. Section 5 presents experimental results for the degree-constrained minimum spanning tree. Final considerations are presented in Section 6.

2 The Encoding

The proposed encoding is based on the concepts of *chains* and *node depth* in a graph tree. The representation consists basically of linear lists containing the tree **nodes** and their **depths**. The order the pairs *(node,depth)* are disposed in the list is important and depends on the node position in an intermediate representation, which utilizes a special kind of graph chain[2].

Next Section presents the intermediate representation. Section 2.2 introduces the node-depth encoding proposed.

2.1 Intermediate Representation

The intermediate representation utilizes a special set of graph chains. This set consists of the chains connecting a leaf to a root. This type of chain is called *main chain*. Trees may be represented by main chains. Moreover, a forest can be represented by the union of the main-chain encodings of its trees.

A tree has a number of main chains equal to its number of leaves l. The set of all l main chains is a tree representation. For example, the representation of the tree from Figure 1(a) is in Figure 1(b). Figure 1(c) shows the same set of chains disposed in a different order. In this arrangement, nodes repeated in different chains are side by side. Such chains are said to be *properly grouped*.

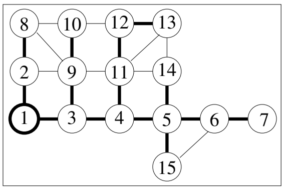
The node-depth encoding requires a tree to be represented by its main chains properly grouped, which will be called *intermediate representation*.

2.2 The Node-Depth Representation

From the intermediate representation, we obtain the proposed encoding as follows.

1. Eliminate the repeated nodes in different chains. This simplification is illustrated in Figures 2(a) and 2(b);
2. Store the remaining nodes with their depths in a linear list (which may be an array T). In this way, each element of the list should be a pair containing a node and its depth (see Figure 2(c)).

[2] An edge-sequence in which all the edges and nodes are distinct is called a **chain**.



(a) A graph with a spanning tree indicated by 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

Fig. 1. A graph with a spanning tree and its main chains

The pairs must be disposed in the list in the same order they are in the intermediate representation, considering the chains from top to bottom and, in each chain, the nodes from left to right (see Figure 2(c)).

```
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
```

(a) Properly grouped main chains of a spanning tree

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

(b) Representation by main chains without repeated nodes

$$\begin{bmatrix} \text{depth} \\ \text{node} \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 2 & 1 & 2 & 3 & 2 & 3 & 4 & 5 & 3 & 4 & 4 & 5 & 4 \\ 1 & 2 & 8 & 3 & 9 & 10 & 4 & 11 & 12 & 13 & 5 & 14 & 6 & 7 & 15 \end{bmatrix}$$

(c) Node-depth Representation

Fig. 2. Node-depth encoding from the intermediate representation

2.3 Forest Encoding

The proposed forest encoding is composed by the union of the encodings of all trees of a forest. In this way, the forest data structure can be easily implemented using an array of pointers, where each pointer indicates the node-depth encoding of a tree of the forest.

3 Operators

This Section presents two operators (called **operator 1** and **operator 2**) to generate new spanning forests using the node-depth encoding. Both operators generate a spanning forest F' of a graph G when they are applied to another spanning forest F of G.

The results produced by the application of both operators are similar. The application of the operator 1 (or 2) to a forest is equivalent to transfer a subtree from a tree T_{from} to another tree T_{to} of the forest. Applying operator 1, the root of the pruned subtree will be also the root of this subtree in its new tree (T_{to}). On the other hand, the transferred subtree will have a new root (any node of the subtree different from the original root) when applying operator 2.

In this way, the operator 1 can produce simple and small changes in the forest. The operator 2 can generate larger and more complex alterations.

The operator 1 requires a set with two nodes previously determined: the prune node p, which indicates the root of the subtree to be transferred; and the adjacent node a, which is a node of a tree different from T_{from} and that is also adjacent to p in G.

The operator 2 requires a set with three nodes: the prune node p, the adjacent node a, and the new root node r of the subtree.

In the following, we explain both operators considering that the required set of nodes were previously determined. We show how to efficiently obtain these sets of nodes in Section 4.

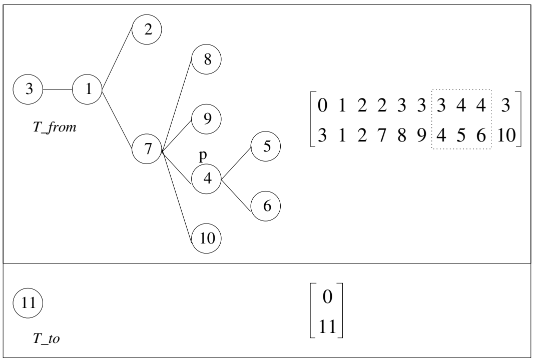
3.1 Operator 1

In the description of the operator 1, we consider that the nodes p and a were previously chosen and that the node-depth representation were implemented using arrays. Besides, we assume the indices of p (i_p) and a (i_a) in the arrays T_{from} and T_{to}, respectively, are also known.

The operator 1 can be described by the following steps (see Figures 3(a), 3(b) and 3(c)):

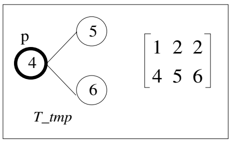
1. Determine the range (i_p-i_l) of indices in T_{from} corresponding to the subtree rooted at the node p. Since we know i_p, we only need to find i_l. The range (i_p-i_l) corresponds to the consecutive nodes x in the array T_{from} such that $i_x \geq i_p$ and $d_x \geq d_p$ (the dashed lines in Figure 3(a)), where d_x is the depth of the node x;
2. Copy the data in the range i_p-i_l from T_{from} into a temporary array T_{tmp} (containing the data of the subtree being transferred), see Figure 3(b). The depth of each node x from the range i_p-i_l is updated as follows: $d_x = d_x - d_p + d_a + 1$;

3. Create an array T'_{to} containing the nodes of T_{to} and T_{tmp} (i.e., generate a new tree connecting the pruned subtree to T_{to}), see Figure 3(c);
4. Construct an array T'_{from} comprising the nodes of T_{from} without the nodes of T_{tmp};
5. Copy the forest data structure F to F' exchanging the pointers to the arrays T_{from} and T_{to} for pointers to the arrays T'_{from} and T'_{to}, respectively.



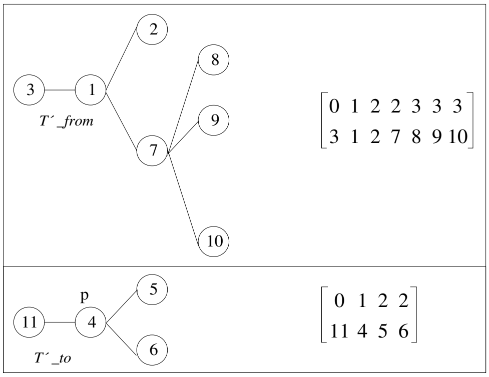
$$\begin{bmatrix} 0 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 3 \\ 3 & 1 & 2 & 7 & 8 & 9 & 4 & 5 & 6 & 10 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 11 \end{bmatrix}$$

(a) T_{to}, T_{from} and their node-depth representations



$$\begin{bmatrix} 1 & 2 & 2 \\ 4 & 5 & 6 \end{bmatrix}$$

(b) T_{tmp} and its node-depth representation



$$\begin{bmatrix} 0 & 1 & 2 & 2 & 3 & 3 & 3 \\ 3 & 1 & 2 & 7 & 8 & 9 & 10 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 & 2 \\ 11 & 4 & 5 & 6 \end{bmatrix}$$

(c) T'_{to}, T'_{from} and their node-depth representations

Fig. 3. Example of application of the operator 1

3.2 Operator 2

The operator 2 requires a set of three nodes: the prune node p, the new root node r and the adjacent node a. The nodes p, r are in the tree T_{from} and a is in T_{to}.

The differences between operator 1 and operator 2 are in the steps 2 and 3 (see the operator-1 procedure, Section 3.1), i.e. only the formation of pruned subtrees and their storing in temporary arrays are different.

In the sequel, we described the steps 2 and 3 for the operator 2. Figures 4(a), 4(b) and 4(c) provide an illustrative example of these steps.

The procedure of copy of the pruned subtree for the operator 2 can be divided in two steps: The first step is similar to the step 2 for the operator 1 and differs from it in the exchanging of i_p by i_r. The array returned by this procedure is called T_{tmp1}.

The second step considers the nodes in the chain from r to p (i.e. r_0, r_1, $r_2, \ldots, r_n$, where $r_0 = r$ and $r_n = p$) as roots of subtrees (see the highlighted nodes in Figure 4(a)). The subtree rooted at r_1 contains the subtree rooted at r_0. The subtree rooted at r_2 contains the subtree rooted at r_1, and so on (see Figure 4(a)). The algorithm for the second step should copy the subtrees rooted at r_i $(i = 1, \ldots, n)$ without the subtree rooted at r_{i-1} (see Figure 4(b)) and store the resultant subtrees in a temporary array T_{tmp2} (see Figure 4(c)).

The step 3 of the operator 1 creates an array T'_{to} from T_{to} and T_{tmp}. On the other hand, the operator 2 utilizes both temporary arrays T_{tmp1} and T_{tmp2} to construct T'_{to}.

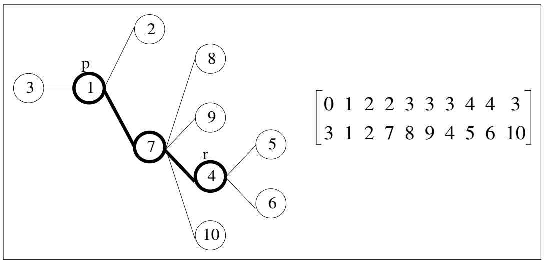
3.3 Operators for One-Tree Forests

The proposed operators require a forest with at least two trees. However, it is possible to utilize the same operators for forests with one tree. First, we add to the original forest with one tree (denoted T_{uniq}) an auxiliar tree T_{aux} containing only one node.

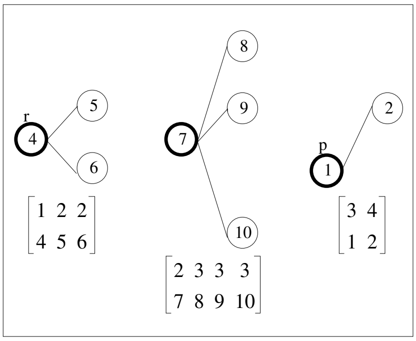
Second, the application of the operator 1 (2), given p and a (p, r, and a), is divided into two steps. Initially, the operator 1 is utilized to transfer the pruned subtree to the auxiliar tree ($T_{from} = T_{aux}$) using the node a equal to the unique node in T_{aux}. Afterward, we apply the operator 1 (or 2) to transfer the subtree from T_{aux} (T_{from}) to the tree T_{uniq} (T_{to}) using the original value of a.

4 Determination of the Nodes p, r, and a

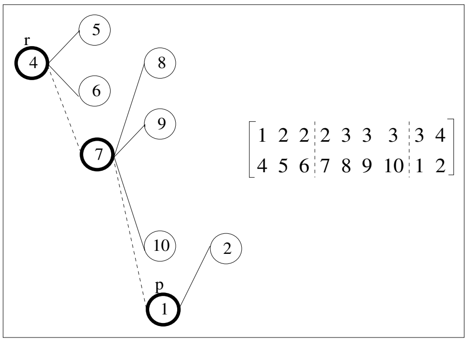
As described in Section 3, the operators 1 and 2 require a set of predefined nodes and their positions in F. Next, we present a strategy to locate a given node in a forest F. Subsection 4.2 describes an efficient procedure to find adequate nodes p, r, and a.



(a) T_{to} and its node-depth representation



(b) Subtrees rooted at the nodes in the chain from r to p



(c) Node-depth Representation of the pruned subtree

Fig. 4. Example of determination of T_{tmp2}. The thick lines highlight the nodes in the chain from r to p. The depth values shown in this Figure consider the depth of the node a equal to zero

4.1 The Node Position in F

The determination of the position of a node in F can be efficiently achieved using auxiliar matrices, here named Π_x's, and a vector, here named π. Each node x of G possesses its correspondent matrix Π_x. For the original spanning forest F_0 of G, Π_x is a column matrix: $\Pi_x = \begin{bmatrix} 0 \\ i_0 \\ j_0 \\ k_0 \end{bmatrix}$, where i_0 is the index of the tree that contains x (T_{i_0}), j_0 is the index corresponding to x in the array T_{i_0} and k_0 is the depth of x in its tree.

Suppose a forest F_h is being generated from F_g ($g < h$) and x is in the subtree that will be transferred to a new tree generating F_h. Then, x will have a new position in F_h different from its position in F_g. So, we insert a new column in Π_x with the indices correspondent to 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 transferred subtree (see Section 3).

The vector π stores the forest g, from which the forest F_h was generated, at the rank h of π, 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 x occured in one of predecessors of h. In this way, we can look for the predecessors of h in the columns of Π_x. We start searching for $\pi(h)$. If this column is not found in Π_x, we try the column $\pi(\pi(h))$, and so on. The process of looking for such columns in Π_x can be achieved efficiently by running a binary search [13] on the list given by $\Pi_x(0, \cdot)$ (the first row of Π_x).

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

4.2 Choice of the Nodes *p*, *r*, and *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, an index of a tree in F and, for this tree, pick up, randomly, a node index that is not the tree root. Call p the correspondent node.
2. Pick up, randomly, a node adjacent to p (using the node adjacent list of G). Call this node a. If $a \notin T$, determine its position in F using the vector π and matrix Π_a; otherwise pick up, randomly, another a or return to step 1.

The strategy for the determination of p and a for the operator 2 works as follows:

1. Pick up randomly an index of a tree in F and, for this tree, pick up randomly a node index that is not a root. Call p the correspondent node.
2. Determine the range of nodes in the subtree rooted at p as in the step 1 of operator 1.Choose randomly an index of the selected range. Call r the correspondent node;
3. Pick up randomly a node adjacent to r (using the node adjacent list of G). Call this node a. If $a \notin T$, determine its position in F using the vector π and matrix Π_a; else pick up randomly another a or return to step 1.

5 Tests

This Section presents an evaluation of the proposed procedure for the degree-constrained minimum spanning tree problem [7]. The tests consider 11 complete graphs with number of vertices from 15 to 1000. For each graph, the constraint degree varies from 3 to 5. The edge weights were randomly obtained from the interval ranging from 1 to the number of vertices. The proposed approached was also compared with a Genetic Algorithm using the Prufer Encoding (GAPE) [6], [4].

Table 5 shows the results, where best cost is the mean of the best individuals in 20 trails and time is the mean running time corresponding to these individuals. Tournament was used for selection in both methodologies to reduce the running time.

The results suggest that the proposed procedure can deal with the degree constrained minimum spanning tree. Besides this methodology seems adequate to work with large graphs.

Table 1. Results from the proposed algorithm and GAPE applied to the degree constrained minimum spanning tree problem

			Proposed Algorithm		GAPE	
Graph	Vertices	MST degree	BestCost	Time(s)	Best Cost	Time(s)
1	15	3	23.0	0.14	37.5	0.63
		4	23.0	0.14	34.4	0.63
		5	23.0	0.13	35.0	0.64
2	20	3	36.0	0.15	70.4	0.71
		4	36.0	0.14	64.5	0.71
		5	35.5	0.12	67.4	0.70
3	25	3	41.5	0.16	103.4	0.79
		4	41.6	0.16	102.4	0.82
		5	41.3	0.16	95.7	0.80
4	30	3	51.7	0.18	136.5	0.89
		4	53.0	0.20	131.0	0.90
		5	53.7	0.18	127.5	0.92
5	50	3	107.6	0.21	419.1	1.42
		4	112.2	0.21	398.3	1.43
		5	112.3	0.21	404.7	1.43
6	100	3	477.1	0.28	1937.1	3.74
		4	495.5	0.28	1822.9	4.06
		5	509.0	0.28	1816.8	4.08
7	200	3	3006.3	0.51	9868.9	15.72
		4	2838.0	0.49	9628.3	17.24
		5	2776.4	0.49	9702.2	17.55
8	300	3	9216.0	0.94	26712.2	36.96
		4	9394.0	0.93	25838.5	41.07
		5	9407.1	0.93	25650.5	41.79
9	400	3	21074.0	1.39	53089.5	69.10
		4	20802.4	1.38	52114.0	73.80
		5	20820.0	1.38	51834.2	77.72
10	500	3	37445.4	1.88	89886.2	113.16
		4	37518.9	1.88	88421.2	123.67
		5	37445.4	1.88	87649.3	127.12
11	1000	3	247474.0	6.49	424783.2	525.71
		4	245404.0	6.49	420479.1	571.83
		5	247605.0	6.49	417868.4	583.01

6 Conclusions

EAs for network layout problems require special chromossome encoding. This paper presented a forest representation, named node-depth encoding. Based on this representation, we have developed two new operators capable to manipulate a forest generating a new one.

The proposed approach was evaluated for the degree-constrained minimum spanning tree problem. The results suggest that the proposed technique can deal with this problem and work with large graphs using relatively small running time. In this way, this paper may encourage the development of new EA approaches using the node-depth encoding for other NDPs.

References

1. Kershenbaum, A.: Telecommunications Network Design Algorithms. McGraw-Hill, New York (1993)
2. Chou, H.H., Premkumar, G., Chu, C.H.: Genetic algorithms for communications network design - an empirical study of the factors that influence performance. IEEE Transactions on Evolutionary Computation **5** (2001) 236–249(3)
3. Harary, F., Gupta, G.: Dynamic graph models. Mathl. Comput. Modelling **25** (1997) 79–87
4. Gen, M., Li, Y.Z., Ida, K.: Solving multiobjective transportatin problem by spanning tree-based genetic algorithm. IEICE Transactions on Fundamental of Electronics Communications and Computer Sciences **E82A** (1999) 2802–2810
5. Reijmers, T.H., Wehrens, R., Daeyaert, F.D., Lewi, P.J., Buydens, L.M.C.: Using genetic algorithm for the construction of phylogenetic trees: Application to g-protein coupled receptor sequences. Biosystems **49** (1999) 31–43
6. Gen, M., Cheng, R.: Genetic Algorithms and Engineering Design. Ashikaga Institute of Technology, Ashikaga, Japan (1997)
7. Knowles, J., Corne, D.: A new evolutionary approach to the degree-constrained minimum spanning tree problem. IEEE Transaction on Evolutionary Computation **4** (2000) 125–134(2)
8. Carvalho, P.M.S., Ferreira, L.A.F.M., Barruncho, L.M.F.: On spanning tree recombination in evolutionary large-scale network problems - application to electrical distribution planning. IEEE Transactions on Evolutionary Computation **5** (2001) 623–630(6)
9. Delbem, A.C.B., de Carvalho, A., Bretas, N.G.: Optimal energy restoration in radial distribution systems using a genetic approach and graph chain representation. Electric Power Systems Researchs **67/3** (2003) 197–205
10. Palmer, C., Kershenbaum, A.: An approach to a problem in network design using genetic algorithms. Networks **26** (1995) 101–107
11. Droste, S., Wiesmann, D.: On representation and genetic operators in evolutionary algorithms. Technical Report CI–41/98, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund (1998)
12. Delbem, A.C.B., de Carvalho, A.: A forest encoding for evolutionary algorithms applied to design problems. Genetic Algorithm and Evolutionary Computation Conference 20003, Lecture Notes in Computer Science **2723** (2003) 634–635
13. Goodaire, E.G., Parmenter, M.M.: Discrete Mathematics with Graph Theory. Prentice Hall, Upper Saddle River, USA (1998)