Pruefernumbers and Genetic Algorithms: A lesson how the low locality of an encoding can harm the performance of GAs

Franz Rothlauf and **David E. Goldberg**

Working Paper 3/2000
February 2000

Working Papers in Information Systems
Editor: Prof. Dr. Armin Heinzl

University of Bayreuth
Chair of Information Systems
Universitaetsstrasse 30
D-95440 Bayreuth, Germany
Phone +49 921 552807, Fax +49 921 552216
E-Mail: wi@uni-bayreuth.de
Internet: *h*ttp://wi.oec.uni-bayreuth.de

Pruefernumbers and Genetic Algorithms: A lesson how the low locality of an encoding can harm the performance of GAs

Franz Rothlauf
Chair of Information Systems
University of Bayreuth
Universitaetsstr. 30
D-95440 Bayreuth/Germany
rothlauf@ieee.org

David E. Goldberg
Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Av. Urbana, IL 61801
deg@illigal.ge.uiuc.edu

February 26, 2000

Abstract

When handling tree networks, researchers have sometimes tried using the pruefernumber representation for encoding networks, but GAs often degraded or broke down when used on this encoding.

This paper investigates the locality of the pruefernumber and its effect on the performance of a Genetic Algorithm (GA). The locality describes how the neighborhood of the genotype is preserved, when constructing the phenotype (the tree) from the genotype (the pruefernumber). It is shown that the locality of the pruefernumber is highly irregular on the entire solution space and that the performance of a GA depends on the structure of the optimal solution. A GA is able to perform well only for networks that have a good locality (stars). For all other types of networks (lists, trees) the locality is low and a GA fails to find the best list or tree. Using a GA with the pruefernumber encoding can be useful, when the good solutions tend to be a star.

The locality of an encoding could have a strong influence on the performance of a GA. When choosing encodings for optimization problems, researchers should be aware of this and be careful with low locality encodings. If the locality of the encoding is low, a failure of the GA is often inescapable.

1 Introduction

When applying GAs to tree networks the pruefernumber is sometimes used for encoding the trees (Julstrom, 1993; Gen et al., 1998; Abuali et al., 1995). Often this type of encoding results in poor GA performance, which some have expected to be due to the bad locality of the encoding(Palmer & Kershenbaum, 1994).

In this paper we investigate the locality of the pruefernumber encoding and the performance of a GA for a simple One-Min problem. The locality describes how good the neighborhood of the genotype is preserved, when constructing the phenotype from the genotype. The problems of a GA with low locality is illustrated for different network types (stars, lists, trees). We get deeper insights why this encoding fails, illustrate the problem of GAs with low locality encodings and give hints to other researchers how they could use this representation or define new ones more effectively.

The remainder of the paper is organized as follows. We start with a short overview how the pruefernumber is constructed. Section 3 describes some properties of the pruefernumber encoding.

A description about the computer experiments is given in Section 4. The results for the locality and the performance of the GA are presented in section 5. The paper ends with a summary of results.

2 From the phenotype to the genotype of a tree and vice versa

This section gives a short review on how to get a pruefernumber (the genotype) from a tree (the phenotype) and how to get the tree back from the pruefernumber.

Tree networks are generally defined as an undirected and connected graph with no cycles. A tree with n nodes has exactly $n-1$ links. For a graph with n nodes there are exactly $n^{(n-2)}$ trees. The pruefernumber for a n-node tree is a number with $n-2$ digits and each digit is of base n (compare fig. 1).

2.1 The construction of the pruefernumber

Every tree has at least two nodes, which have only one connecting edge (the degree of the node is 1). All nodes are labelled with numbers from 1 to n. The pruefernumber can be constructed by the following algorithm:

1. Let i be the lowest numbered node of degree 1 in the tree.
2. Let j be the one node, which is connected to i (there is exactly one!). The number of the jth node is the rightmost digit of the pruefernumber.
3. Remove node i and the edge (i, j) from the tree and from further consideration.
4. Go to 1 until only two nodes (that means one link) are left.

After termination you have a pruefernumber with $n-2$ digits, that represents your tree. Illustrative examples for constructing the pruefernumber can be found in Palmer & Kershenbaum, 1994 or Gen et al., 1998.

2.2 The construction of the tree from the pruefernumber

The construction of the tree from the pruefernumber follows the construction of the pruefernumber.

1. Let P be a pruefernumber with $n-2$ digits. All node numbers which are not in P can be used for the construction of the network (are eligible).
2. Let i the lowest numbered eligible node. Let j be the leftmost digit of P.
3. Add the edge (i, j) to the tree.
4. Designate i as no longer eligible and remove the leftmost digit j from the pruefernumber.
5. If j does not occur anywhere else in the remaining pruefernumber, designate j as eligible.
6. Go to 2 until there remain no digits in the pruefernumber. If no digits left then there are exactly two numbers r and s eligible. Add at last the edge (r, s) to the tree.

3 Properties of the pruefernumber encoding

In this section we give a short overview on the properties of the pruefernumber particularly with regard to the use of genetic and evolutionary algorithms (Palmer, 1994).

The use of the pruefernumber has some remarkable benefits:

- every tree can be represented by a pruefernumber
- only trees are represented by pruefernumbers
- every pruefernumber represents exactly one tree
- all trees are equally represented

A look at the construction rule of the pruefernumber shows that the pruefernumber is able to represent all possible trees. Each tree has at least two nodes with degree 1 so the construction rule can be applied to all possible trees. It was shown by Pruefer (1918) that the pruefernumber represents only trees. This means random numbers can be created and they always represent a tree. In contrast to many other representations for trees, repairing of a randomly chosen representation is not necessary. In addition, all trees are represented equally. Each tree is represented by exactly one specific pruefernumber. The mapping from the pruefernumber to the tree is a one to one mapping. The number of different trees for a graph with n nodes is n^{n-2}, which is equal to the number of possible different pruefernumbers.

However the pruefernumber has also some disadvantages:

- complex calculation
- bad locality

The calculation of the pruefernumber is a little complex, but it can be done in $O(n \log n)$. This seems to be not too bad. The real disadvantage of the pruefernumber is the little locality of the representation. Small changes in the representation can lead to big changes in the represented network. This means the mapping from the genotype to the phenotype is not homogeneous. A small change in the pruefernumber could lead to a large change in the tree and vice versa. So most of the individuals that are created with basic operators like mutation or crossover are not similar to their parents. The operators work more like a random search than like a guided search. This behavior is illustrated in section 5.4.

4 Experiments

This section describes the computer simulations for evaluating the locality of the pruefernumber encoding and the performance of GAs for different network structures like stars, lists and trees.

We present results for four different types of networks:

- star: One node is of degree $n-1$ and the rest of the nodes have degree 1.
- ordered list: The nodes are connected according their number. Node n is connected to $n+1$, node $n+1$ is connected to $n+2$ and so on. Two nodes are of degree 1 (the first and the last of the list) and the rest of the nodes have degree 2.
- random list: Like the ordered list, but the order of the nodes in the list is random.

- tree: Every other tree network

We distinguish between ordered and random lists, because the locality of the pruefernumber depends strongly on the numbering of the nodes. We will see in section 5 that the neighborhood and also the performance of the GAs is different for ordered and random lists.

Fig. 1 shows the encoding of the network as a pruefernumber and the encoding of the pruefernumber as a bitstring. Finally, the GA works on the bitstring representation. Looking at the locality means either having a random walk through the search space or exploring the complete neighborhood of one individual. We investigate the locality in six different ways:

- random walk through the bitstring (one step is a one bit change in the bitstring)
- random walk through the pruefernumber (change of one digit in the pruefernumber)
- random walk through the phenotype (change of one edge in the tree)
- neighborhood of a bitstring (all neighbors that are different in one bit from the examined individual)
- neighborhood of a pruefernumber (all neighbors that differ in one digit)
- neighborhood of a tree network (all neighbors are different in one edge)

A random walk through the bitstring/pruefernumber representation of a tree means changing one bit/digit and examining how many edges in the network are different. A random walk through a tree means, that one edge of the tree is replaced by a randomly chosen edge and the difference of bits/digits in the bitstring/pruefernumber is examined. The start individual for the bitstring, pruefernumber or tree is chosen randomly. To get statistically significant information independent from the start individual, 400 steps were carried out in each of the 20 runs.

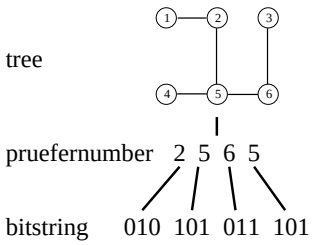

Figure 1: A tree, its pruefernumber and the corresponding bitstring

Exploring the neighborhood of the pruefernumber/bitstring means having a look at all individuals, which are different in one bit/digit in comparison to the examined individual and counting how many edges have changed in the tree. Investigating the neighborhood of a tree means looking at all individuals which are different in one edge from the examined network. For each problem 20 independent runs were performed.

For our investigation in the performance of a GA we defined a simple One-Min problem. The fitness function is defined as the distance to the optimal solution. The optimal solution has fitness 0. The fitness of all other networks is equal to the number of different edges in comparison to the optimal solution. This means for a $n = 16$ node problem that a tree, that has 12 links in common with the best solution, has fitness 3. The fitness values of the tree range between 0 and $n - 1$.

For finding the optimal solution we use a traditional simple GA (Goldberg, 1989) with onepoint-crossover, no mutation and a $(\mu + \lambda)$ selection scheme (Bäck & Schwefel, 1995). For each generation we produce from μ parents λ offspring and choose from all $(\mu+\lambda)$ individuals the μ best individuals[1]. For the 16 node problems $\mu = \lambda = 400$ and the GA stops after 50 generations. With 32 nodes μ and λ were set to 1500 and we stop after 70 generations. 20 runs were performed for each problem.

Results are presented for 16 and 32 node tree networks[2].

[1] We prefer the $\mu + \lambda$ selection scheme in comparison to traditional roulette wheel or tournament selection schemes with replacement, because some kind of elitism helps a lot in finding good solutions for this simple One-Min problem. Our experiments have shown that without elitism a GA can not find its way to the optimal solution.

[2] The 8 node problems show the same behavior and are missing due to limited space

nodes	tree		pruefernumber	bitstring
n	star	$(n-1)(n-2)$	$(n-1)\ (n-2)$	$(n-2) * \lceil log_2(n) \rceil$
	list	$\frac{1}{6}n(n-1)(n+1)-n+1$		
8	star	42	42	18
	list	77		
16	star	210	210	56
	list	665		
32	star	930	930	150
	list	5425		

Table 1: number of neighbors for tree, pruefernumber and bitstring

5 Results

This section presents results investigating the locality of the pruefernumber and the performance of a traditional GA. We start with looking how many neighbors a bitstring, a pruefernumber or a tree has. This is followed by an investigation in the locality by doing a random walk through the pheno- and genotypes of trees and looking at the neighborhood of different network types (stars, lists and trees). The section ends with results for the performance of the GA for different types of optimal solutions.

5.1 Number of neighbors

In table 5.1 we present the number of neighbors for different types of networks. Assume a tree network with n nodes. Each star network has exactly $(n-1)(n-2)$ neighbors. For a list network each tree has $\sum_{i=1}^{n-1} i(n-i)-1 = n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 - (n-1) = \frac{1}{2}n^2(n-1) - \frac{1}{6}n(n-1)(2n-1) - n + 1 = \frac{1}{6}n(n-1)(n+1) - n + 1$ neighbors. The number of neighbors for tree networks is between the number for star and list networks and must be calculated for each network separately. It depends on the different degree of the nodes in the tree. Each pruefernumber with $k = n-2$ digits has exactly $(n-1) * (n-2)$ neighbors. Each pruefernumber with $n-2$ digits is encoded as a bitstring of length $(n-2) * \lceil log_2(n) \rceil$. So each bitstring has $(n-2) * \lceil log_2(n) \rceil$ neighbors. A change of one digit in the pruefernumber can result in up to $\lceil log_2(n) \rceil$ different bits.

Comparing the number of neighbors of a pruefernumber to the numbers of neighbors of a tree shows a one to one correspondence for star networks. A star has as many neighbors as a pruefernumber. It can be seen in subsection 5.3 that for star networks the locality is perfect, too. When modifying the tree toward a list network the number of neighbors for the pruefernumber and bitstring stays constant, whereas the number of neighbors for the tree is increasing. This results in different locality for different areas of the solution space.

5.2 Random walks

A look at the results for a random walk through the bitstring (fig. 2) and the pruefernumber (fig. 3) shows that most of the one bit/digit changes lead to a change of more than one edge in the tree. For about 50% of all one bit/digit changes the phenotype changes in more than three different edges. The locality of the genotype (bitstring as well as pruefernumber) is quite low.

When walking through the solution space of the phenotype (tree networks) the plots in figure 4 show an accumulation between 1 and 5 bit changes for the bitstring, but also a remarkable high

possibility for high order changes, when changing one edge in the tree. For the pruefernumber (fig. 5) about 75% of the neighboring individuals are different in more than one digit. Also the locality of the phenotype is quite bad.

5.3 A look at the neighborhood

The neighborhood of a bitstring (fig. 2) and a pruefernumber (fig. 3) is different for different network types. The neighborhood of a bitstring/pruefernumber that encodes a star network shows that a change of one bit/digit results in a change of exactly one edge in the encoded star network[3]. The locality of the bitstring/pruefernumber is perfect for star networks. A look at the neighbors of bitstrings that encode ordered list networks shows that the change of one bit results in a maximum change of 4 edges (fig. 2) in the tree independent from the number of nodes. The pruefernumber (fig. 3) shows the same behavior. A change of one digit leads to a maximum change of 4 edges. The locality of the pruefernumber for ordered list networks is worse than for stars, but better as for tree networks. Bitstrings and pruefernumbers that encode random lists or random tree networks show very low locality. All neighbors that can be created by an one bit change of the bitstring (fig. 2) or a one digit change of the pruefernumber (fig. 3) have phenotypically not much in common with the considered network.

If the neighborhood of a star network with n nodes is investigated, the change of one edge results in the change of one digit in the pruefernumber[4], but up to $\lceil log_2(n) \rceil$ bits in the bitstring[5] that represents the pruefernumber (fig. 4). The locality of the phenotype is perfect for star networks. The locality for tree and list networks shows a different behavior. The change of one edge results most of the time in a complete different bitstring/pruefernumber. The change of one edge leads in about 80% to a bitstring that is different in more than 5 bits (fig. 4). For the pruefernumber (fig. 5) the results are similar. The locality of the phenotype is quite poor for tree and list networks.

5.4 The performance of a GA

In fig. 6 we show the performance of the GA from section 4 for different optimal solutions. All parameters of the encoding and the GA remains the same for the different problems. Only the type of the best solution is different. For star networks, the GA is able to find the optimal solution without problems. In all 20 runs, the GA finds the optimal solution in less than 24 (16 nodes) or 43 (32 nodes) generations. However if the optimal solution is a random list or a random tree, the GA can never find the optimal solution and is completely misled because of the low locality. For ordered lists, the GA performs a little bit better, but nevertheless the locality is so bad that the GA must degrade.

We see that when a GA explores the neighborhood of a star network, it is really a search around one individual. The GA is able to find its way to the optimum. When it comes to searching the solution space around a ordered list, or even worse a random list or a tree network however, the search is not more than a blind and random search.

[3]Because of better clarity this trivial case is missing in figure 2 and 3.

[4]case missing in fig. 4 because of better clarity

[5]The neighborhood of star networks for 32 node networks is missing to get more illustrative plots. It shows the same behavior as for 16 nodes.

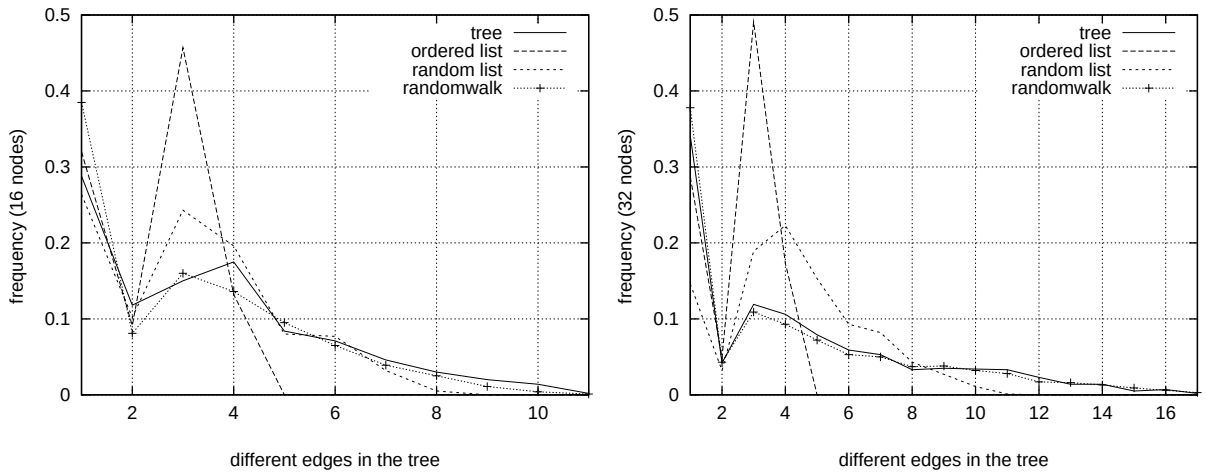

Figure 2: Locality of the bitstring for 16 (left) and 32 (right) node networks. The plots show the number of different edges in the network, if one bit in the bitstring is changed.

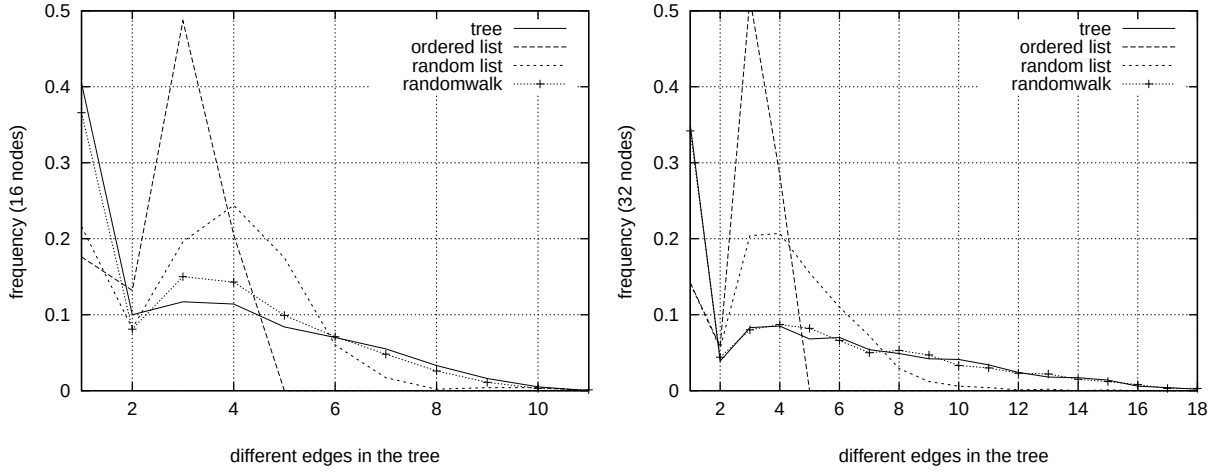

Figure 3: Locality of the pruefernumber for 16 (left) and 32 (right) node networks. The plots show the number of different edges, if one digit of the pruefernumber is changed.

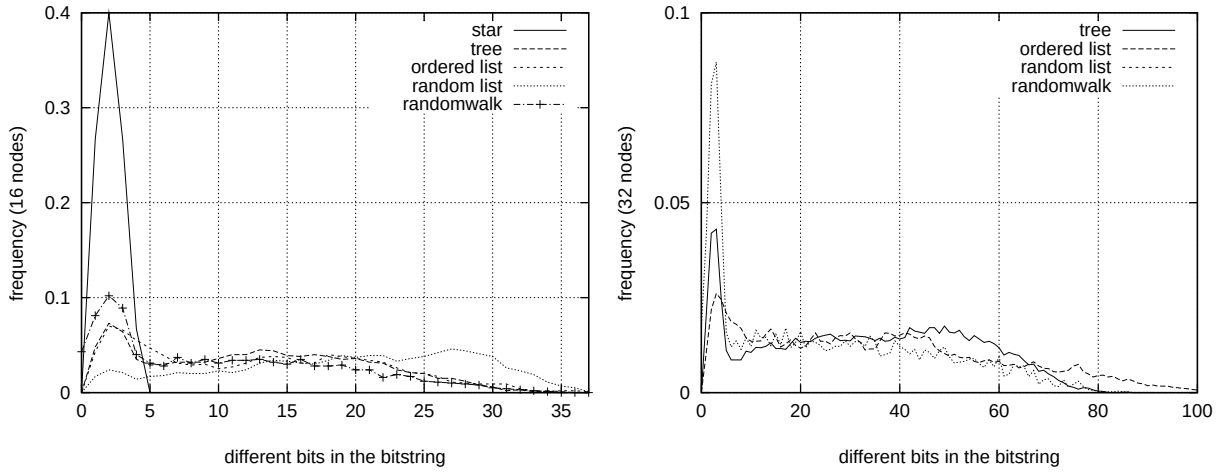

Figure 4: Locality of the tree for 16 (left) and 32 (right) node networks. The plots show the number of different bits in the bitstring, if one edge of the tree is changed.

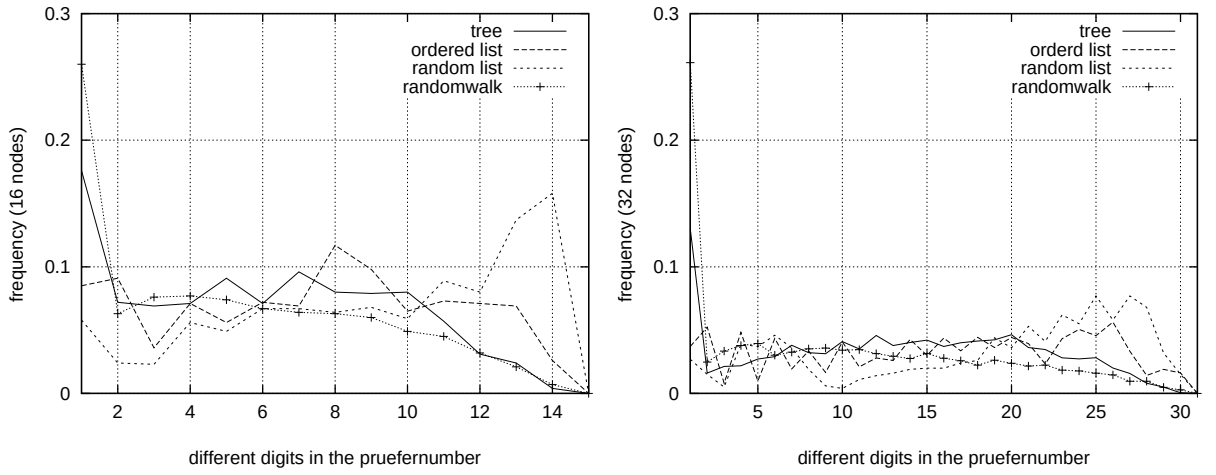

Figure 5: These plots show the locality of the phenotype (tree) for 16 (left) and 32 (right) node networks. The plots show the number of different digits in the pruefernumber, if one edge of the tree is changed.

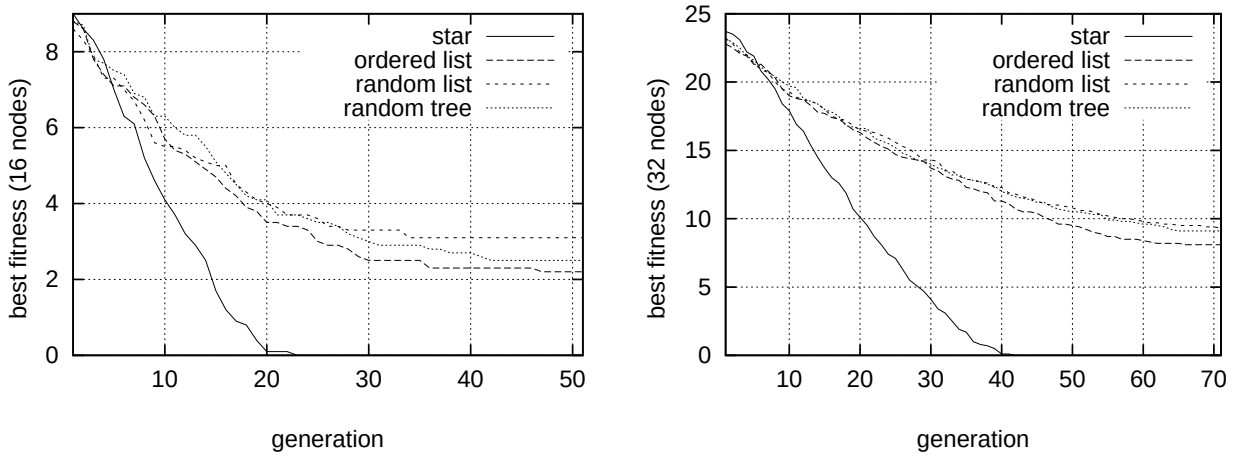

Figure 6: The performance of a Genetic Algorithm for 16 (left) and 32 (right) node networks. The structure of the best solutions has a big influence on the performance of the GA. If the best solution is a star, the GA performs well. If the GA has to find a best solution that is a list of a tree, it degrades and cannot solve the easy One-Min problem

6 Summary and Conclusions

This paper presents an investigation in the locality of the pruefernumber and the performance of GAs using this encoding.

We illustrate that the locality of the phenotype (the tree) and the genotype (either pruefernumber or bitstring) is not homogeneous. When looking at star networks the pruefernumber encoding shows a perfect locality. A change of one edge leads to the change of one digit of the pruefernumber and vice versa. A GA performs pretty well for the One-Min problem, if the optimal solution is star-like. Even though the locality is not too bad for ordered list networks, a GA has already big problems finding the optimal ordered list. For random lists and tree networks the locality is very low. Here the pruefernumber encoding shows its ugly face. Every small change of the pruefernumber leads to a big change in the encoded tree and vice versa. A GA cannot find the optimal solution or even come close to it. Unfortunately, every star network has only one edge in common with another star, so the areas of high locality are separated from each other by large areas of low locality.

We have seen that the locality of an encoding has a strong influence on the performance of a GA. When choosing encodings for problems, researchers should be aware of this and be careful with low locality encodings. If the locality of the encoding is low, a failure of the GA even for simple problems is often inescapable.

References

Abuali, F. N., Wainwright, R. L., & Schoenefeld, D. A. (1995). Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 470–477). San Francisco, CA: Morgan Kaufmann.

Bäck, T., & Schwefel, H.-P. (1995). Evolution strategies I: Variants and their computational implementation. In Winter, G., Périaux, J., Galán, M., & Cuesta, P. (Eds.), *Genetic Algorithms in Engineering*

and Computer Science (Chapter 6, pp. 111–126). Chichester: John Wiley and Sons.

Gen, M., Ida, K., & Kim, J. (1998). A spanning tree-based genetic algorithm for bicriteria topological network design. See Institute of Electrical and Electronics Engineers (1998), pp. 15–20.

Gen, M., Zhou, G., & Takayama, M. (1998). A comparative study of tree encodings on spanning tree problems. See Institute of Electrical and Electronics Engineers (1998), pp. 33–38.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Institute of Electrical and Electronics Engineers (Ed.) (1998). *Proceedings of 1998 IEEE International Conference on Evolutionary Computation.* Piscataway, NJ: IEEE Service Center.

Julstrom, B. A. (1993). A genetic algorithm for the rectilinear steiner problem. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 474–480). San Mateo, CA: Morgan Kaufmann.

Palmer, C. C. (1994). *An approach to a problem in network design using genetic algorithms.* unpublished PhD thesis, Polytechnic University, Troy, NY.

Palmer, C. C., & Kershenbaum, A. (1994). Representing trees in genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1 (pp. 379–384). Piscataway, NJ: IEEE Service Center.

Pruefer, H. (1918). Neuer beweis eines satzes ueber permutationen. *Arch. Math. Phys.*, *27*, 742–744.