Network Random Keys - A Tree Representation Scheme for Genetic and Evolutionary Algorithms

Franz Rothlauf* rothlauf@uni-bayreuth.de
Department of Information Systems, University of Bayreuth, Universitätsstr. 30, D-95440 Bayreuth, Germany

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

Armin Heinzl heinzl@uni-bayreuth.de
Department of Information Systems, University of Bayreuth, Universitätsstr. 30, D-95440 Bayreuth, Germany

Abstract

When using genetic and evolutionary algorithms for network design, a good choice of the representation scheme for the construction of the genotype is important for the performance of the algorithms. One of the most common representation schemes for networks is the characteristic vector representation. However, with encoding trees, and using crossover and mutation, invalid individuals occur that are either under- or over-specified. When constructing the offspring, or repairing the invalid individuals that do not represent a tree, it is not possible to distinguish between the importance of the links which should be used. These problems can be overcome by transferring the concept of random keys from scheduling and ordering problems, to the encoding of trees. This paper investigates the performance of a simple genetic algorithm (SGA) using network random keys (NetKeys) for the one-max tree and a real-world problem. The comparison between the network random keys, and the characteristic vector encoding, shows that despite the effects of stealth mutation, which favors the characteristic vector representation, selectorecombinative SGAs with NetKeys have some advantages for small and easy optimization problems. As soon as it comes to more complex problems, SGAs with network random keys significantly outperform SGAs using characteristic vectors.

This paper shows that random keys can be used for the encoding of trees, and that genetic algorithms using network random keys are able to solve complex tree problems much faster than when using the characteristic vector. Users should therefore be encouraged to use network random keys for the representation of trees.

Keywords

network random keys, characteristic vector encoding, tree network, representation, stealth mutation, random keys.

1 Introduction

Random keys (RKs) are an efficient method for encoding ordering and scheduling problems. They were introduced by Bean (1994) and are advantageous when used for problems where the relative ordering of tasks is important.

*Visiting student at the Illinois Genetic Algorithms Laboratory.

In this paper the use of random keys is extended to a different class of problems. Random keys should be used for the encoding of the topology of trees. A tree is is an undirected, fully connected graph with no cycles. Finding good solutions for tree network problems is important in many fields like telecommunication, electric, gas, computer, and transportation networks. The performance of genetic and evolutionary algorithms (GEAs) used with a traditional encoding scheme (characteristic vector) is compared to the network random key encoding (NetKey). Both representations have the same length and similar construction complexity. However, when using characteristic vectors (CVs) the genetic operators mutation and crossover produce invalid solutions which are under- or over-specified. A repair mechanism is necessary that restores valid solutions, but because genetic algorithms (GAs) with CVs cannot distinguish between the importance of links, the repair mechanism must rely on random link insertion or deletion. Using NetKeys means a transition from binary and hard decisions of establishing a link in the CV, to describing the importance of a link with a continuous value $[0, 1]$ in the NetKeys. The information about the individual is not stored with discrete values $\{0, 1\}$, but by the relative ordering of the positions in a sequence of keys. In this paper we want to investigate whether RKs are a good solution to use for encoding trees. We also want to examine both theoretically and empirically the performance of the NetKey encoding on some test and real world problems, and show the differences in comparison to the characteristic vector encoding.

The paper is structured as follows. In section 2 we take a closer look at the characteristics of random keys. This is followed by an illustration on how to use the characteristic vector encoding, and how the problems of invalid solutions, and the missing ability of GAs to distinguish between the importance of links, can be overcome when using NetKeys. In section 4 we present the one-max tree problem and some real-world scenarios. A theoretical investigation into population sizing for the one-max tree problem, and SGAs using NetKeys in section 5 is followed by a comparison of the performance of the NetKeys and CVs when using GAs for solving the test problems. In section 6 we present the direction of future work. The paper ends with concluding remarks.

2 A Short Introduction on the Properties of Random Keys

This section summarizes the history and properties of the random key encoding.

The random key representation for representing permutations was first presented by Bean (1992). Later, the encoding was also proposed for single and multiple machine scheduling, vehicle routing, resource allocation, quadratic assignments, and traveling salesperson problems (Bean, 1994). Norman and Bean (1994) refined this approach (Norman & Bean, 2000) and applied it to multiple machine scheduling problems (Norman & Bean, 1997). An overview of using random keys for scheduling problems can be found in Norman (1995). In Norman and Smith (1997) and Norman et al. (1998), random keys were used for facility layout problems. In Knjazew (2000) and Knjazew and Goldberg (2000a) a representative of the class of competent GAs (fast messy GA (Goldberg et al., 1993)) was used for solving ordering problems with random keys.

The random key representation uses random numbers for the encoding of a solution. A key sequence of length l is a sequence of l distinct real numbers (keys). The values are initially chosen at random, are floating numbers between zero and one, and are only subsequently modified by mutation and crossover. An example for a key sequence is $r = (0.07, 0.75, 0.56, 0.67)$. Of importance for the interpretation of the key sequence is the position and value of the keys in the sequence. If we assume that $Z_l = \{0, \ldots, l-1\}$ then a permutation σ can be defined as a surjective function $\sigma : Z_l \rightarrow Z_l$. For any

key sequence $r = r_0, \ldots, r_{l-1}$, the permutation σr of r is defined as the sequence with elements $(\sigma r)_i = r_{\sigma(i)}$. The permutation r^s corresponding to a key sequence r of length l is the permutation σ such that σr is decreasing (i.e., $i < j \Rightarrow (\sigma r)_i > (\sigma r)_j$). The ordering corresponding to a key sequence r of length l is the sequence $\sigma(0), \ldots, \sigma(l-1)$, where σ is the permutation corresponding to r. This mathematical definitions describes that the positions of the keys in the key sequence r are ordered according to the values of the keys in descending order. In our example we have to identify the position of the highest value in the key sequence (0.75 at position 2). The next highest value is 0.67 at position 4. We continue ordering the complete sequence and get the permutation $r^s = 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$. In the context of scheduling problems this permutation can be interpreted as a list of jobs that are executed on one machine (We start with job 2, then continue with job 4, job 3, and job 1.) From a key sequence of length l, we can always construct a permutation of l numbers. Every number between 1 and l (resp. 0 and $l-1$) appears in the permutation only once as the position of each key is unique. Here are some properties of the encoding.

- A valid permutation r^s of l numbers can be created from all possible key sequences as long as there are no two keys r_i that have the same value.[1] Therefore, every random key sequence represents a permutation of l numbers.
- There are many possibilities for the construction of a key sequence r from a permutation of numbers r^s. Every element r_i of the sequence can be scaled up by the same factor and r still represents exactly the same permutation. As long as the relative ordering of the keys in the key sequence is the same, different key sequences always represent the same permutation. Furthermore, it is necessary that r^s is a permutation of l numbers otherwise no key sequence r can be constructed from r^s.
- RKs encode both the relative position of a number in the permutation r^s (encoded by the value of the key at position i in comparison to all other keys), and the absolute position of i in r^s. The relative position of a number i in the permutation r^s is determined by the numbers that precede and follow i. It is determined directly by the weights of the keys r_i. All numbers j in the sequence r^s that follow i correspond to lower-valued keys ($r_j < r_i$), whereas all numbers j that precede i correspond to higher-valued keys ($r_j > r_i$). In the context of scheduling problems all jobs where the corresponding key has a higher value than the ith key are executed before job i, and all jobs with a corresponding key with lower value are executed after i. In contrast, the absolute position of a number i in the permutation r^s cannot be encoded directly, but is only indirectly determined by the value of the ith key. The absolute position describes at which position in the permutation r^s a number i appears. A large value at the ith position leads us to a position at the beginning of the permutation, and a low value leads to a position at the end.
- The distinction between relative and absolute position of a number in the permutation r^s is important for the locality of RKs. The locality of an encoding describes how well the genotypic neighbors correspond to the phenotypic neighbors. A coding has high locality if mutating a genotype changes the corresponding phenotype only slightly. A look at RKs shows that the locality of RKs is high for ordering problems. A small change in the genotype (the key sequence r) leads to a small change in the phenotype (the permutation r^s). The change of one key changes the relative position of exactly one number. However, one must be careful with the definition of the

[1] $r_i \neq r_j$ *for* $i \neq j$ *and* $i, j \in [1, l]$

neighborhood. If the absolute position of the numbers in r^s would be important, a change of one key is disastrous. If the value of the key r_i with the highest value is modified, only the number i changes its relative position in the permutation r^s, but up to l numbers change their absolute position in the permutation. However, as we use RKs to represent a sequence of numbers, only the relative, and not the absolute positions of the numbers in the permutation must be considered. And for problems where the relative positions of numbers are important the locality of RKs is high.

- When using genetic and evolutionary algorithms with RKs, standard crossover and mutation operators can be used and are expected to work well. No repair mechanism, or problem-specific operators, are necessary when using this encoding for ordering problems. The standard one- or multi-point crossover schemes work well (Bean, 1994) because the relative ordering of the positions in the parents is preserved and transferred to the offspring (Fox & McMahon, 1991). Due to the high locality of the encoding we expect standard mutation operators to work well and to construct offspring that are similar to their parents.

RKs show interesting properties for the encoding of scheduling problems. In the following section the tree network design problem is presented and we want to illustrate how the RKs can be used.

3 Tree Network Design with NetKeys and the Characteristic Vector Encoding

This section starts with a short overview of the tree network design problem. It is followed by a description of the CV encoding, which problems arise with its use, and how these can be overcome by using NetKeys.

3.1 The Design Problem

Finding good solutions for tree network design problems is important in many fields such as telecommunication, computer, backbone access, transportation and distributing networks.

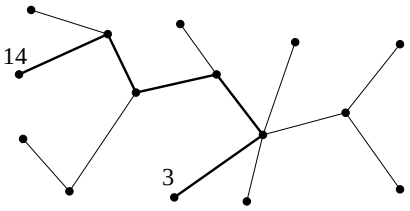


Figure 1: A spanning tree on 15 nodes, with the path connecting nodes 3 and 14 emphasized.

For a graph with n nodes there are $n(n-1)/2$ possible links. This results in $2^{n(n-1)/2}$ different graphs. A spanning tree is defined as a connected graph with exactly $n-1$ links. There are no loops or rings in a tree. Between any two nodes there exists only one possible path for the flow. Figure 1 shows a spanning tree on 15 nodes and emphasizes the path connecting nodes 3 and 14. The aim of the design process is to minimize the overall cost for constructing and maintaining the tree network and is calculated by summing-up the costs of all links. The location of the nodes, the demanded flow between the different locations, and the cost structure of the links that can be used for constructing the tree are given. The only design variable is the list of links which are used for the tree.

3.2 Encoding Trees with the Characteristic Vector Encoding

The characteristic vector encoding is one of the most common approaches for encoding graphs (Davis, Orvosh, Cox, & Qiu, 1993). Examples for the use of the CV encoding

can be found in Tang et al. (1997) and Sinclair (1995). In Berry, Murtagh, and Sugden (1994) examples are given for the use of the CV encoding for the encoding of trees.

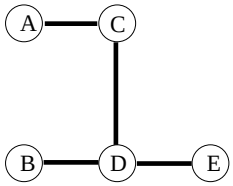


Figure 2: A five node tree.

A characteristic vector is a binary vector that indicates if a possible link is used or not in the graph. For an n-node graph exist $n(n-1)/2$ possible links, and a characteristic vector of length $l = n(n-1)/2$ is necessary for encoding an n-node graph. All possible links must be numbered, and each link must be assigned to a position in the vector. In Table 1 we give an example of a characteristic vector for a 5 node tree. The nodes are labeled from A to E. The link from node A to B is assigned to the first position in the string, the link from A to C is assigned to the second position and so on. To indicate if the ith link is established, the value at position i is set to one. If no link is established the value at position i is set to zero. The tree that is represented by Table 1 is shown in Figure 2.

0	1	0	0	0	1	0	1	0	1
A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

Table 1: The characteristic vector for the tree in Figure 2.

The locality of the encoding is high. Changing a zero to a one in the characteristic vector adds an additional edge to the graph. However, if the graph was a tree, a cycle would have been created. A repair mechanism is necessary that removes one of the links in the cycle. For a tree, either none, (the previously added link is removed) or two edges are different (another link in the cycle is removed). When changing a one to a zero the situation is similar. The locality is the same for small changes in the phenotype (the graph). A change of two edges (minimal mutation step) changes two bits in the corresponding CV. Furthermore, in contrast to other encodings like Prüfer numbers (Rothlauf & Goldberg, 2000), the locality of CVs is independent of the specific structure of the represented tree (for example star, list, or arbitrary tree).

Every characteristic vector that represents a tree must have exactly $n-1$ ones, the represented graph must be connected, and there are no cycles allowed. This makes the construction of trees from randomly chosen characteristic vectors demanding as most of the randomly generated characteristic vectors are invalid, and not trees. For an n-node network, there are $2^{n(n-1)/2}$ possible characteristic vectors, but only n^{n-2} valid trees (Prüfer, 1918). The probability of randomly getting a tree is $\frac{n^{n-2}}{2^{n(n-1)/2}} < \frac{2\ln(n)}{\ln(2)n} < 3\ln(n)/n$. The chance of randomly creating a CV that represents a tree is low (Palmer, 1994).

Randomly chosen characteristic vectors which represent a tree can be invalid in two different ways:

- There are cycles in the tree.
- The tree is not connected.

We get a cycle for the example in Table 1 if we set the first allele (A-B) to one. This characteristic vector does not represent a tree any more because a cycle of A-C-D-B-A exists. If we alter any of the ones in the characteristic vector to zero we get two disconnected trees. If A-C is switched to zero we get two disconnected trees. One consists of the node A and the other consists of a star with center D.

Moreover, when GEAs are used, and the genetic operators mutation and crossover come into operation, the use of the characteristic vector is faced with some serious problems:

- Under-specification,
- Over-specification and redundancy,
- Distinction between important and unimportant links.

Creating a new individual randomly, or by genetic operators, can produce a graph that is not connected. The tree is not completely specified. Some links which are necessary to connect the tree are missing. A similar problem arises with over-specification. An individual is over-specified if there are more than $n-1$ ones in the CV. The encoding specifies more links as necessary for a valid tree. Therefore, to get a valid tree it is necessary to remove some of the spare links. With the over-specification we get redundant information, but no help on which links (redundant information) should be removed. Some GEA approaches are not affected by over- and under-specification, and accept invalid solutions to some extent (Orvosh & Davis, 1993; Davis, Orvosh, Cox, & Qiu, 1993). However, most of the traditional GEAs repair all infeasible solutions and use some kind of repair mechanism. The repairing of invalid solutions is mostly done in two steps (Berry, Murtagh, & Sugden, 1994):

- Remove links that cause cycles.
- Add links to obtain a connected graph.

When repairing a characteristic vector that should represent a tree, the cycles in the corresponding graph must be identified. If we randomly choose the characteristic vector $\vec{c} = 1100010100$ of length $l = 10$ for a 5 node graph, we are faced with the cycle A-C-D-B-A. We could choose one of the links A-C, C-D, D-B or B-A and remove it randomly. When we choose the link A-C we get $\vec{c} = 1000010100$. As there are no more loops we can stop removing links and continue checking if the graph is fully connected. As there are only three ones in $\vec{c}$, and we have a tree with 5 nodes, the graph could not be connected and we have to randomly add one link. As the node E is separated from the rest of the tree, a link from E to a randomly chosen node A, B, C or D, has to be added. The link C-E is chosen and we finally get $\vec{c} = 1000010110$. A closer look at the repair mechanisms shows that the order of the repair steps does not matter.

When using the CV representation, it is difficult to decide during a GA run as to whether a link in an infeasible offspring should be established or not, because the algorithm can only add or delete links of an infeasible solution in a random fashion. No information exists in the representation about which link should be removed if over-specification happens, or which links should be added if the tree is under-specified. Therefore, when using CVs it is not possible to distinguish between those links that are important and those that are not. Due to the random deletion and insertion of links the resulting offspring do not necessarily inherit the high-quality links from their parents, but often have only few links in common with them. However, if the offspring do not inherit the links from their parents, and the offspring mainly consist of randomly created links, GAs can not work efficiently and perform in a similar way to random search. To overcome this problem Palmer and Kershenbaum (1994) presented an approach for encoding trees that uses biases for each node and modifies the cost matrix of the links according to these biases. The represented tree is found by constructing a minimum spanning tree using the modified cost matrix. With this approach GAs are able to distinguish between expensive and cheap nodes and no repairing is necessary. This link

and node biased encoding shows interesting results and outperforms other approaches where no biasing was used (Palmer, 1994; Abuali, Wainwright, & Schoenefeld, 1995). To extend this idea and to consider not only the importance of the nodes, but also to enable GAs to distinguish between the importance of links, and to overcome the problems of CVs with redundancy and under-specification, we want to use the more general concept of random keys for the encoding of trees.

3.3 Encoding Trees with NetKeys

We want to demonstrate how the problems that arise with the use of the characteristic vector can be solved by using the random key encoding we described in section 2.

With NetKeys we are able to give priority to links that should be used for the construction of a tree. As NetKeys use continuous variables that can be interpreted as the importance of the link, it is possible to distinguish between more and less important links. The higher the value of a key r_i, the higher the probability that the link i is used for the construction of the tree. Because NetKeys encode the importance of the links, no problems of redundancy, over or under-specification occur. Every key sequence r represents a valid tree. It is not necessary to use problem-specific operators. Standard mutation and crossover operators can be used. With no under- or over-specification no repair mechanism is necessary when using NetKeys.

The positions of the keys in the key sequence are interpreted in the same way as for the characteristic vector. The positions are labeled and each position represents one possible link in the tree. From a key sequence r of length $l = n(n-1)/2$, a permutation r^s of l numbers can be constructed as described in section 2. The tree is constructed from the permutation as follows:

1. Let $i = 0$, G be an empty graph with n nodes, and r^s the permutation of length $l = n(n-1)/2$ that can be constructed from the key sequence r. All possible links of G are numbered from 1 to l.

2. Let j be the number at the ith position of the permutation r^s.

3. If the insertion of the link with number j in G would not create a cycle, then insert the link with number j in G.

4. Stop, if there are $n-1$ links in G.

5. Increment i and continue with step 2.

With this calculation rule, we can construct a unique, valid tree from every possible key sequence.

We want to illustrate this construction rule with an example. We use the key sequence r from Table 2 which represents a 5 node tree. The permutation $r^s = 10 \rightarrow 8 \rightarrow 6 \rightarrow 9 \rightarrow 2 \rightarrow 7 \rightarrow 1 \rightarrow 5 \rightarrow 4 \rightarrow 3$ can be constructed from r. We start constructing the graph G by adding the link D-E (position 10) to the tree. This is followed by adding C-D (position 8) and B-D (position 6). If we add the link C-E (position 9) to the graph, the cycle C-E-D-C would be created, so we skip C-E and continue by adding A-C (position 2). Now we have a tree with four edges and terminate the construction algorithm. We have constructed the tree shown in Figure 2.

The computational effort for constructing the phenotype from the genotype is similar for the NetKey and the characteristic vector representation. The calculation of the permutation r^s of length l from the key sequence r can be done in $O(l \log(l))$ (sorting

position	1	2	3	4	5	6	7	8	9	10
value	0.55	0.73	0.09	0.23	0.40	0.82	0.65	0.85	0.75	0.90
link	A-B	A-C	A-D	A-E	B-C	B-D	B-E	C-D	C-E	D-E

Table 2: A possible key sequence for the tree in Figure 2.

an array of l numbers). The process of constructing the graph from the permutation r^s is comparable to repairing an invalid graph that is constructed from a characteristic vector.

The locality of the NetKey encoding is high. A mutation (changing the value of one key) means either no change if the relative ordering is not changed, or the change of two edges if the position of a number that corresponds to a used link changes in the permutation r^s. As the construction of the tree is based on the relative ordering of the keys, the locality is high.

4 Test Problems

The following section presents the one-max tree problem and describes the properties of some real-world test instances.

4.1 The One-Max Tree problem

To test the performance of optimization algorithms for the topological design of trees, standard test problems should be used. But standard test problems are not very popular.

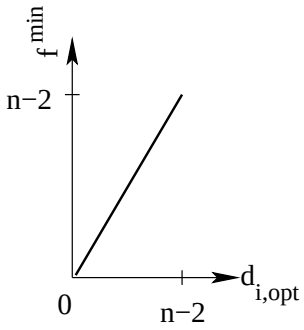


Figure 3: The cost function for the one-max tree minimization problem.

Most researchers define their own problems and rarely use other test problems. To overcome this, and to motivate researchers to build up an open library of network design problems of different size and complexity, we introduce the standard one-max tree problem.

It is based on the one-max problem for binary representations (Ackley, 1987). An optimal solution (tree) is chosen either randomly or by hand. The structure of this tree can be determined: It can be a star, a list, or an arbitrary tree with n nodes.

For the calculation of the fitness of the individuals, the distance d_{ab} between two trees G_a and G_b is used. It is defined as

$$d_{ab} = \frac{1}{2} \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} |l_{ij}^a - l_{ij}^b|,$$

where l_{ij}^a is 1 if the link from node i to node j exists in tree G_a and 0 if it does not exist in G_a. This definition of distance between two trees is based on the Hamming distance (Hamming, 1980) and $d_{ab} \in \{0, 1, \ldots, n-2\}$.

When using this distance metric for a minimization problem (Figure 3) the fitness of an individual G_i is defined as the distance $d_{i,opt}$ to the optimal solution G_{opt}. Therefore, $f_i^{min} = d_{i,opt}$, and $f_i^{min} \in \{0, 1, \ldots, n-2\}$. An individual has fitness (cost) of $n-2$ if it has only one link in common with the best solution. If the two individuals do not differ ($G_i = G_{opt}$), the fitness (cost) of G_i is $f_i^{min} = 0$. When defining a maximization problem, the fitness f_i^{max} of an individual G_i is defined as the number of edges it has

in common with the best solution G_{opt}. Therefore, $f_i^{max} = n - 1 - d_{i,opt}$. Because this test problem is similar to the standard one-max-problem it is easy to solve for mutation-based GEAs, but somewhat harder for recombination-based GAs (Goldberg, Deb, & Thierens, 1993). The knowledge about the standard one-max problem can be used for this one-max tree problem.

If our example tree from Figure 2 is chosen as the optimal solution and we have a minimization problem, the star with center D would have fitness (cost) of 1, because the two trees differ at two edges[2] ($d_{i,opt} = 1$).

In our runs we use for G_{opt} either arbitrary trees, stars, or lists as best solutions.

4.2 Four Real-World Communication Tree Network Design Problems

Our communication network problems are derived from a real-world 26-node problem from a company with locations all over Germany.

For fulfilling the demands between the nodes, different line types with discrete capacities and cost are available. The cost for installing a line consists of a fixed and length dependent share. Both depend on the capacity of the link. The cost are based on the tariffs of the German Telecom from 1996 and represent the amount of money (in German Marks) a company has to pay for renting a telecommunication line of a specific length and capacity per month. For a detailed description of the cost structure, demands and location of the nodes the reader is referred to Rothlauf (2001).

In problem 1 we have a 16-node problem with traffic ending only at node 1. In the second problem, one of the nodes is removed and some additional traffic is added. The third problem is similar to problem 1, but uses a modified cost-function for the lines. Finally we look at a 16-node problem with traffic between all nodes.

4.2.1 Problem 1: One headquarter and 15 branch offices

This problem is the original design problem. All 15 branch offices (node 2 to 16) communicate only with the headquarter (node 1). Possible line capacities are 64 kBit/s, 512 kBit/s and 2048 kBit/s. The optimal solution for this problem is shown in Figure 4(a). The complexity of the problem is low.

4.2.2 Problem 2: One headquarter and only 14 branches

If one node is left out and some additional traffic is added, finding the best solution is slightly more involved than in problem 1. The optimal solution is shown in Figure 4(b).

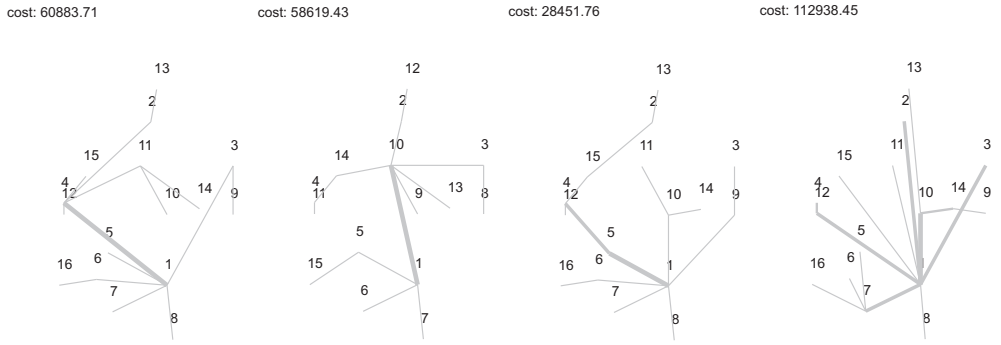
4.2.3 Problem 3: One headquarter, 15 branches and cheap lines for everybody

In the scenario shown in Figure 4(c) the cost for installing high capacity lines is only 10% of the cost in problem 1. Therefore, the cost of a link is mainly determined by its length. Hence, the optimal structure is more like a minimum spanning tree. If the cost of the link would be solely determined by the length of the link, and there was only one possible capacity, the optimal solution would be the minimum spanning tree. Otherwise the problem is exactly like problem 1.

4.2.4 Problem 4: 4 headquarters, 12 branches and all are working together

In problem 4 depicted in Figure 4(d) the demand matrix is completely filled. Some traffic exists between every node i and j. Between the four headquarters (node 1, 2, 3 and 4) the traffic is uniformly distributed between 256 kBit/s and 512 kBit/s. Every other node communicates with the four headquarters and has a uniform demand between 0

[2] A-C respectively A-D



(a) Problem 1 (b) Problem 2 (c) Problem 3 (d) Problem 4

Figure 4: Optimal solutions for the four real world problems.

and 512 kBit/s. This demand is split into the headquarters at a ratio of 0.4, 0.3, 0.2 and 0.1 for the node 1, 2, 3 and 4[3]. Between all 12 branch offices the demand of the traffic is uniformly distributed between 0 and 64 kBit/s. To make the problem more realistic two additional line types are available. It is possible to use lines of 128 kBit/s and 4096 kBit/s with twice the cost of 64kBit/s respectively 2048 kBit/s lines.

5 Experimental Results

This section presents a population sizing model for the one-max tree problem, and discusses the effects of stealth mutation. Furthermore, it compares the performance of the NetKey and the CV representation for the one-max tree problem defined in subsection 4.1, and the real-world problem from subsection 4.2.

5.1 Population Sizing for the One-Max Tree Problem and the Effects of Stealth Mutation

In the following we develop a population sizing model for the one-max tree problem. For our model we only consider crossover ($p_{crossover} = 1$) and no mutation ($p_{mut} = 0$). Because GAs without mutation are not able to reproduce links (BBs) once they are lost, the supply of good building blocks in the first generation must be ensured (Goldberg, 1989b; Goldberg, Deb, & Clark, 1992).

When extending the population sizing equation in Harik, Cantú-Paz, Goldberg, and Miller (1997) from a binary alphabet to a χ-ary alphabet we get:

$$N_{min} = -\frac{\chi^k}{2} \ln(\alpha) \frac{\sigma_f}{d} \sqrt{\pi},$$

where χ is the cardinality of the alphabet, α is the probability of failure, σ_f is the overall variance of the function, and d is the signal difference between the best and second best BB. For calculating σ_f we have to investigate how to decide among the competing BBs. For the one-max tree problem we have to find these $n-1$ links the optimal solution is constructed from. The key sequence r that represents a tree with n nodes consists of

[3]Node 1 is the most important node and 40% of the traffic of the branches ends there, in node 2 30% of the traffic ends, and so on.

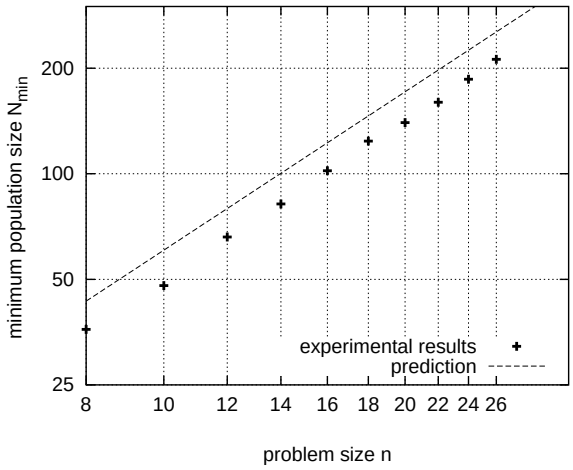


Figure 5: Minimum pop size N_{min} for NetKeys over the problem size n for the one-max tree problem. The probability of finding the optimal solution is $P_n = 0.95$. The population size goes with $O(n^{1.5})$.

$l = n(n-1)/2$ different keys. For the construction of the tree these $n-1$ keys r_i are used that have the highest value. Therefore, we can split the $n(n-1)/2$ different keys r_i in $n/2$ different groups of size $(n-1)$. Finding the optimal solution means that all keys r_i with the links i contained in the optimal solution can be found in one group which is considered for the construction of the tree and thus contains the keys with the $n-1$ highest values of r_i. A good decision among competing BBs means deciding between the $n/2$ different groups of size $n-1$ and identifying the correct one. A key k_i can belong either to the one group that is considered for the construction of the tree (the key has a high value), or to one of the $n-2$ groups that are not considered. This is similar to the needle in a haystack model and the standard deviation for such a case is (Goldberg, Deb, & Clark, 1992)

$$\sigma_f = \frac{\sqrt{2l(n-2)}}{n} = \sqrt{\frac{(n-1)(n-2)}{n}} \approx \sqrt{n}.$$

As we have $n/2$ different groups, the cardinality χ of the alphabet can be assumed to be $n/2$ (compare Goldberg (1990). Using these results and with $k = 1$ (the one-max tree problem is fully easy, and there are no interdependencies between the alleles), and $d = 1$, we get an approximation for the population size N_{min}:

$$N_{min} = -\frac{\sqrt{\pi}}{4}\ln(\alpha)\sqrt{n(n-1)(n-2)} \approx -\frac{\sqrt{\pi}}{4}\ln(\alpha)n^{1.5}.$$

The necessary population size N_{min} goes with $O(n^{1.5})$.

In Figure 5, both the theoretical prediction from above, and the experimental results for the minimum necessary population size N_{min} that is necessary for solving the one-max tree problem with probability $P_n = 1-\alpha = 0.95$, are shown over the problem size n. We use a simple GA with tournament selection without replacement of size 3, uniform crossover and the NetKey encoding. We perform 500 runs for each population size and for $N > N_{min}$ the GA is able to find the optimum with probability $p = 0.95$ ($\alpha = 0.05$). Although, we have to make some assumptions in our derivation, and the exact influence

of the construction algorithm of the phenotype from the encoding is difficult to describe theoretically, the population sizing model gives us a good approximation of the expected population size N which goes with $O(n^{1.5})$.

When comparing the results for NetKeys with those of the CV encoding we have to consider the influence of the repair process. In subsection 3.2, we have seen that the repair process fixes under-specified individuals by inserting links randomly. As a result, we still get mutation when we use characteristic vectors though we only want to consider the performance of selectorecombinative GAs and therefore used no mutation. This effect caused by the repair process should be denoted as *stealth mutation*. New links are created during the GA-run, even if they are not present in the start population, or not properly mixed and lost. Therefore, stealth mutation results in some supply of BBs during the run when using CVs.

We have seen that the one-max tree problem is similar to the standard one-max problem. Because mutation-based search performs better for this easy problem in comparison to crossover-based search alone (Goldberg, Deb, & Thierens, 1993), stealth mutation should also increase the performance of GAs using CVs for the easy one-max tree problem. To investigate this effect some experiments concerning the influence of mutation and stealth mutation were performed. The results have shown that even with small population sizes GAs using CVs always find the optimal solution due to stealth mutation. However the run duration strongly increases and GAs need many more generations to find the optimal solution. The same behavior was obtained for GAs using NetKeys when we added some additional mutation. As a result we expect stealth mutation to improve the performance of GAs using CVs especially when solving small and easy one-max tree problems.

5.2 Optimization Parameters

For our empirical investigation we use a simple genetic algorithm with different selection schemes and crossover operators. The simple GA is based on Goldberg's basic implementation (Goldberg, 1989a) with the roulette wheel selection procedure replaced with one of two possibilities described in the next paragraph. For recombination either one-point or uniform crossover is used. We do not use mutation as we want to investigate the effect of recombination alone, and to eliminate finding the optimal crossover-mutation ratio for every specific problem. Furthermore, we assume that mutation only works as a local search operator in the background, and that no significant differences in the results when adding some additional mutation are expected.

For selection we use either tournament selection of size 3, or a deterministic $(\mu + \lambda)$ selection (Bäck & Schwefel, 1995). In each generation λ individuals are created from the μ parent individuals. Then the new λ individuals are examined and the μ best are chosen from all $\mu + \lambda$ individuals. One consequence of this strategy is that a once found good solution can only be replaced by a better one.

To gain statistical evidence from the results, 100 runs were performed and the significance of the results was tested with a t-test for each parameter setting. For minimizing the one-max tree problem the population size N for the runs was set to $2 * N_{min}$. N_{min} was determined by experiment for one-point crossover and for the probability of failure $\alpha = 0.01$. As a result, when using the population size $N = 2 * N_{min}$ the optimal solution for the one-max problem was found in all GA runs. For the real world telecommunication problem the population size N for the SGA was set to 2000 in all runs.

5.3 Performance of the NetKey Encoding for the One-Max Tree Problem

We want to compare the efficiency of the NetKey encoding, and the characteristic vector representation for the one-max tree problem from subsection 4.1.

We investigated the performance of the two representations for a 12, 16, 20 and 26 node one-max tree minimization problem with G_{opt} as an arbitrary tree (12, 20, and 26 nodes), or a list, star, or arbitrary tree (16 nodes). As the population size is set to $N = 2 * N_{min}$ the optimal solution is found in all runs. In Table 3 the results for different selection and crossover schemes are presented. The mean μ and the standard deviation σ of the number of generations that are necessary for finding G_{opt} is shown. A t-test is performed on the means μ_{NetKey} and μ_{cv} and the probability p_T that the two samples are taken from the same universe is shown.

For 20 and 26 node trees, SGAs using NetKeys are always highly significantly better than when using CVs ($p_T < 0.001$). A SGA with tournament selection, uniform crossover and NetKeys needs the smallest number of generations for finding the optimal solution for all different tree sizes (highly significantly with $p_T < 0.001$).

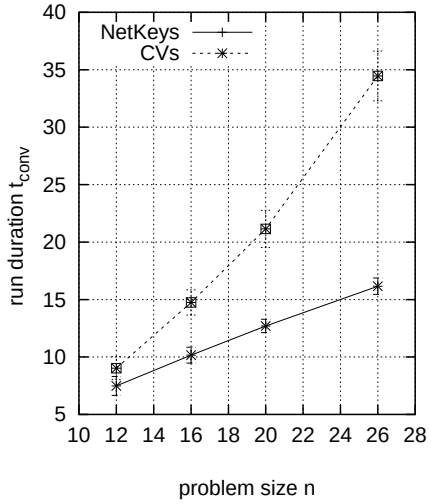


Figure 6: Run duration t_{conv} over problem size n for the one-max tree problem using tournament selection and uniform crossover.

In Mühlenbein and Schlierkamp-Voosen (1993), and Thierens and Goldberg (1994), the time until convergence is defined as $t_{conv} = \pi\sqrt{l}/2I$ with the selection intensity I and the string length l. I depends only on the used selection scheme. With $l = n(n-1)/2$ we get $t_{conv}/n \approx const.$ In Table 3 it could be seen that when using NetKeys, t_{conv}/n is about constant for different n. NetKeys behave according to the predicted behavior. When using CVs, t_{conv}/n increases with larger n which indicates that GAs with CVs struggle more because of more repair and stealth mutation. The search for good solutions depends mainly on the random effects of mutation than on recombination. With larger problem size n of the one-max tree problem, the performance of GAs using NetKeys increases in comparison to when using CVs. This can be explained by assuming that for simple problems, the stealth mutation of the CV helps the GA to regain lost links randomly. With higher problem size, the probability of randomly finding the correct link decreases, the performance of the characteristic vector encoding is reduced, and the NetKey representation becomes better in comparison to the CV. To illustrate this more clearly in Figure 6 we show, using the results from Table 3, the run duration t_{conv} over the problem size n for tournament selection and uniform crossover. For NetKeys t_{conv} grows as predicted linearly with increasing n, whereas with CVs the increase is much larger and not linear any more.

To investigate if there are any structural dependent influences on the performance of GAs, we compare in Table 3 the case that G_{opt} is a tree with G_{opt} is a list, or a star, for a 16 node tree. SGAs show the same behavior as for trees and their performance is independent of the structure of the optimal solution. Tournament selection with uniform crossover and NetKeys performs significantly better than all other parameter settings.

In Figure 7 the averaged lowest cost for the 100 runs is plotted over the number of

generations for 16, 20 and 26 node trees and G_{opt} is a tree. As an extension to Table 3 we wanted to know not only how many generations are necessary to find G_{opt}, but if there are any differences in the convergence behavior of the GAs for the different encodings. The convergence and search behavior of the GAs for the different parameter settings show the same properties as in Table 3. GAs using uniform crossover and NetKeys perform the best, NetKeys are better than characteristic vectors, uniform is better than one-point crossover, and tournament selection outperforms $(\mu + \lambda)$ selection. It could be seen that with increasing problem size n, GAs with NetKeys are significantly better than with CVs.

n	pop size N	type		t_{conv} (in generations)							
				tournament selection				$\mu + \lambda$ selection			
				1-point		uniform		1-point		uniform	
				NetKey	CV	NetKey	CV	NetKey	CV	NetKey	CV
12	600	tree	μ	9.43	9.71	7.48	9.03	15.25	14.4	11.6	12.73
			σ	1.27	1.30	0.82	1.00	1.76	1.68	1.13	1.31
			p_T	0.13		< 0.001		< 0.001		< 0.001	
			t_{conv}/n	0.79	0.81	0.62	0.75	1.27	1.20	0.97	1.06
16	1600	tree	μ	13.62	14.55	10.16	14.75	20.73	21.37	15.89	18.96
			σ	1.29	1.34	0.69	1.10	1.86	1.84	1.11	1.24
			p_T	< 0.001		< 0.001		0.015		< 0.001	
			t_{conv}/n	0.85	0.91	0.64	0.92	1.30	1.34	0.99	1.19
		star	μ	14.84	16.09	9.78	18.08	22.45	23.56	14.99	21.08
			σ	3.30	5.21	0.66	1.61	4.52	5.95	1.02	1.24
			p_T	0.044		< 0.001		0.14		< 0.001	
			t_{conv}/n	0.93	1.01	0.61	1.13	1.40	1.47	0.94	1.32
		list	μ	14.03	13.86	10.54	13.53	21.56	20.23	16.42	18.28
			σ	1.18	1.12	0.83	1.05	1.87	1.63	0.82	1.25
			p_T	0.30		< 0.001		0.015		< 0.001	
			t_{conv}/n	0.88	0.87	0.66	0.85	1.35	1.26	1.03	1.14
20	3200	tree	μ	17.2	19.74	12.7	21.15	26.25	28.37	19.34	25.81
			σ	1.35	1.90	0.58	1.62	2.03	2.25	1.00	1.33
			p_T	< 0.001		< 0.001		< 0.001		< 0.001	
			t_{conv}/n	0.86	0.99	0.64	1.06	1.31	1.42	0.97	1.29
26	5700	tree	μ	22.61	28.24	16.16	34.46	35.15	39.14	24.6	36.32
			σ	1.53	2.52	0.71	2.17	2.89	2.95	0.93	1.38
			p_T	< 0.001		< 0.001		< 0.001		< 0.001	
			t_{conv}/n	0.87	1.09	0.62	1.33	1.35	1.51	0.95	1.40

Table 3: A comparison of the performance of a simple GA using NetKeys versus CVs for the one-max tree problem. We show for different problem sizes n and corresponding population sizes $N = 2N_{min}$ the mean μ and standard deviation σ of the number of generations that are necessary for finding G_{opt}. p_T is the probability that μ_{NetKey} and μ_{cv} are taken from the same universe. The results show that GAs using NetKeys find the optimal solution faster independently of the used crossover and selection scheme than when using CVs. As predicted t_{conv}/n stays constant for NetKeys, whereas it increases for CVs with increasing problem size.

SGAs using the NetKey representation show some advantage in comparison to the CV representation. However, the results are muddied by stealth mutation. With increasing problem size the effect of the stealth mutation increases, SGAs using CVs perform more random search-like, and SGAs using NetKeys always perform significantly better than with CVs.

5.4 Performance of the NetKey Encoding for a Real-World Telecommunication Tree Network Design Problem

We want to test the performance of the NetKey encoding on the real-world problem from subsection 4.2. We only show results for uniform crossover, because in our runs uniform crossover generally outperformed one-point crossover.

Figure 8 compares the performance of GAs using NetKeys with using the CV encoding for the telecommunication tree network design problem. The problem complexity increases from problem 1 (figure 8(a)) to problem 4 (figure 8(d)). We show the performance of GAs with different selection schemes (tournament and $\mu + \lambda$ selection) for NetKey versus CV encoding.

The comparison between NetKeys and CVs reveals performance differences for different problem complexity. In all four problems the convergence speed was significantly higher when using NetKeys in comparison to CVs. Because of the stealth mutation, SGAs with CVs continue finding better solutions even when SGAs with NetKeys are already converged. But as the stealth mutation only works very slowly it could not compensate the lack of performance of the CVs in comparison to the NetKeys. For the simple problem 1 (figure 8(a)) the NetKeys perform slightly better than CVs. With increasing problem complexity the differences become larger. For problems 3 (figure 8(c)) and 4 (figure 8(d)) CVs could not find the optimum independently of the selection scheme in a reasonable amount of time.

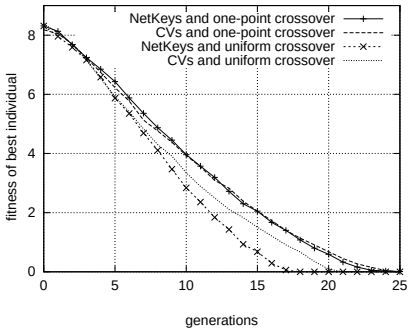
In Table 4 we summarize the results and present the mean μ and the standard deviation σ of the best solution after 50 generations averaged over 100 runs. The percentage by which the best solutions' cost at the end of the run exceeded the cost of the known best solution is indicated by $p_w = cost/cost_{opt} - 1$ in percent. $p_w = 0$ indicates that the optimal solution is found in all runs. A SGA with $(\mu + \lambda)$ selection and NetKeys is able to find solutions that are on average no more than 0.1% worse than the optimal solution in all four problems. For problem 2 the optimal solution even was found in all runs. However, SGAs with CVs and $(\mu + \lambda)$ selection find solutions that are on average 2.8% worse than the optimum. The results are even worse for CVs with tournament selection. For all four problems GAs using NetKeys find solutions that are on average not worse than 0.77% in comparison to the optimum. When using CVs the distance to the optimum is up to more than 13%. Both the convergence speed and the solution quality at the end of the run show a strong advantage for the NetKey representation.

For the real-world problem we observe the same behavior of GAs as for the one-max tree problem. Due to the stealth mutation, NetKeys are only slightly better than CVs for long run durations and simple problems, whereas they generally converge much faster to the optimal solution. However, if the problems become more complex, SGAs with NetKeys remain very robust and are able to solve the problem quickly and easily, whereas CVs rarely find the optimal solution.

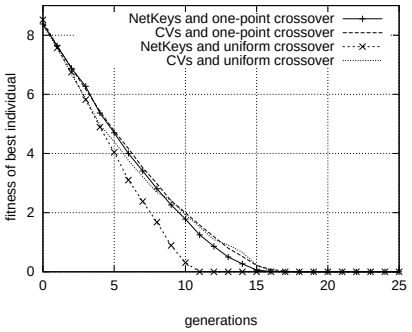
6 Future Research

Based on this study, the following topics require further investigation:

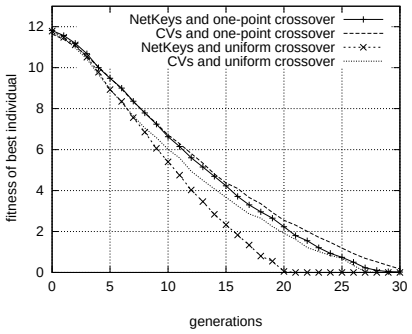
- Use of evolution strategies and mutation with the NetKey representation.



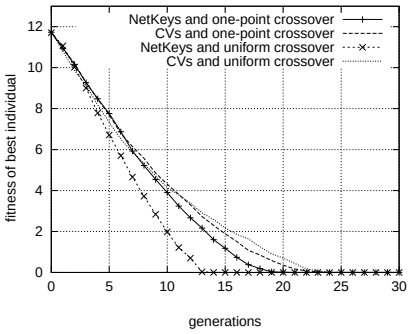
(a) $n = 16$ and $(\mu + \lambda)$ selection



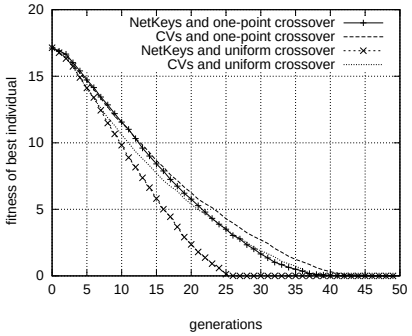
(b) $n = 16$ and tournament selection



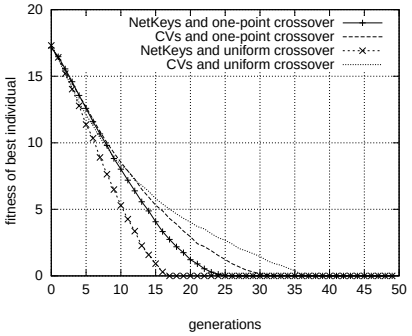
(c) $n = 20$ and $(\mu + \lambda)$ selection



(d) $n = 20$ and tournament selection

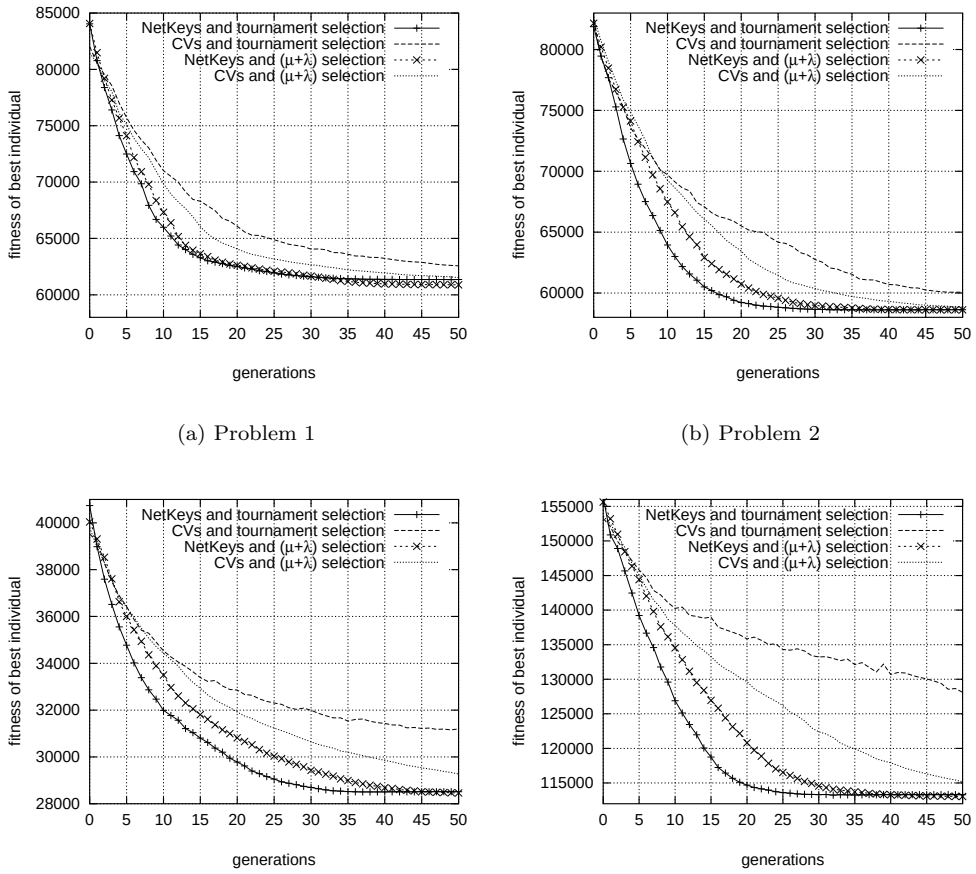


(e) $n = 26$ and $(\mu + \lambda)$ selection



(f) $n = 26$ and tournament selection

Figure 7: Fitness of the best individual over the number of generations for the 16 (top), 20 (middle) and 26 (bottom) node one-max tree problem. The best solution is an arbitrary tree. SGAs using NetKeys generally perform better than when using characteristic vectors. The higher the problem size, the better NetKeys are in comparison to CVs.



(a) Problem 1

(b) Problem 2

(c) Problem 3

(d) Problem 4

Figure 8: We compare the performance of SGAs for the four real-world problems using NetKeys and CVs. The plots show the fitness (cost) of the best individual over the number of generations. SGAs generally perform better with NetKeys than with CVs. The higher the problem complexity, the better SGAs perform with NetKeys. For complex problems (problem 4), GAs using CVs could not find the optimal solution in a reasonable amount of time.

problem number	Cost of best solution		cost of best solution after 50 runs			
			tournament selection		$(\mu + \lambda)$ selection	
			NetKey	CV	NetKey	CV
1	60 883.71	μ	61 351.33	62 566.87	60 886.62	61 525.47
		σ	145	431	40	225
		p_w	0.77%	2.76%	0.00%	1.05%
2	58 619.43	μ	58 623.06	60 062.01	58 619.43	58 829.77
		σ	20	290	0	108
		p_w	0.01%	2.46%	0.00%	0.36%
3	28 451.76	μ	28 513.59	31 195.79	28 462.53	29 268.9
		σ	145	431	40	225
		p_w	0.22%	9.64%	0.04%	2.87%
4	112 938.45	μ	113 310.25	128 106.06	113 056.74	115 156.54
		σ	880	3806	93	816
		p_w	0.33%	13.43%	0.10%	1.96%

Table 4: Fitness of the best individual after 50 generations for the four real-world problems. We show the mean μ and the standard deviation σ of the cost of the best solution after 50 generations. p_w is the distance to the optimal solution in percent.

- Use of NetKeys for general network topologies.
- Creation of test library for network problems.
- Properties of the NetKeys in comparison to other tree encodings.
- More theoretical investigation in representations.
- Use of NetKeys in combination with competent GAs.
- Different tree representations in the field of genetic programming.

In the remainder of this section, each of these items is briefly considered.

In this work we have used NetKeys only with selectorecombinative GAs. Based on the observations made, and the good performance of mutation-based approaches for continuous problems, we strongly believe that using evolution strategies or introducing mutation, could improve the performance of GEAs with NetKeys, and would give us an even more powerful optimization method for many real-world applications.

In this work the use of random keys is restricted to trees. In principle NetKeys could also be used for meshed networks. This could easily be done by introducing a measurement for the meshedness of the network. The higher the meshedness of the network, the more links that are used for constructing the graph, and the more links which are present in the network.

Trying to compare the encodings under controlled conditions has convinced us that more boundedly difficult standard test problems are needed. The one-max tree problem was one attempt to develop such a library of problems with different sizes and complexities for optimizing network structures. The development of deceptive problems (compare Kargupta, Deb, and Goldberg (1992)) could make life for GEAs more difficult, and comparisons between different operators and encodings easier. Deceptive tree problems could be created in a similar manner as that presented in the one-max tree problem.

In this work NetKeys were compared only to the characteristic vector representation. Elsewhere (Rothlauf & Goldberg, 2000) it has been shown that the Prüfer number

encoding, that can also be used for representing trees, performs worse than the characteristic vectors because of the low locality of this encoding. Palmer and Kershenbaum (1994) proposed the link and node biased encoding that uses a similar, but less general concept, for distinguishing between important and unimportant links. It was compared in Abuali, Wainwright, and Schoenefeld (1995) to some other representations and showed good performance. A more exhaustive investigation into the properties and performance of the existing tree encodings is necessary.

The theoretical results about population sizing and convergence time we get for the NetKeys using available theory give us good predictions of the empirical performance of the representation. This kind of approach should be used more widely because it could help researchers in evaluating and designing encodings in a more theory-guided manner. If existent theoretical models were more frequently used by practitioners, the choice between different encodings could be performed more systematically. When using a network representation for a problem of unknown complexity, a good theoretical understanding of encodings is often more helpful than only empirical results.

Competent GAs were used successfully for scheduling problems using random keys (Knjazew & Goldberg, 2000b). It would be interesting to use these more efficient GAs for network design problems and to overcome some of the problems caused by the use of traditional GAs. Especially if the problems are boundedly deceptive, those algorithms would result in a more powerful optimization method.

An interesting question in this context is how different encoding possibilities of trees affect the field of genetic programming. One strength of GEAs is that the search process works on the genotype and not on the phenotype of a problem. Therefore, it is not necessary to define specific operators for every problem, and standard operators with known properties could be used. In the field of genetic programming, currently the encoding of the tree structure is not important because the operators modify the phenotypical structure of the tree and not the underlying representation. However, to develop and use genetic programming operators not on the phenotypical structure of the tree but on the encoding, could result in more general operators, and in less effort for developing specific operators for each different problem.

7 Conclusions

This paper performed a number of functions. We started by reviewing the use of random keys in ordering and permutation problems. Some of the properties of random keys were then described and a method of using random keys for tree network design called network random keys (NetKeys) was developed. We described the characteristic vector representation that could also be used for the encoding of trees and illustrated an effect called stealth mutation that can arise in intersection with genetic algorithms using selection and crossover. We analyzed the properties of the characteristic vector and compared it to the NetKeys. An easy test function (one-max tree problem) was developed in analogy to the one-max problem (Ackley, 1987). In addition to this, a real-world problem with four different scenarios was presented to investigate the performance of the two different encodings. We made some predictions about population sizing and run duration for the one-max tree problem by using existing GA theory for the NetKey encoding. A comparison of the NetKeys to the characteristic vector representation was done using a variety of selection and crossover operators both for the real-world and the one-max tree problem. Finally we presented directions of future research.

These results suggest that the use of the characteristic vector encoding is affected by some serious theoretical problems with over- and under-specification, as well as dis-

tinction between the importance of the links that should be used for constructing a tree. NetKeys overcome those problems in theory as they do not use repair mechanisms, and they are able to distinguish between important and unimportant links. This is possible because NetKeys encode the structure of a tree as an ordered sequence of links, whereas the characteristic vector representation only encodes whether a link is established or not. Although the empirical results are muddied by stealth mutation that favors the CV representation, it could be seen that for small and very simple tree problems a selectorecombinative simple genetic algorithm using NetKeys shows some advantage relative to CVs especially with tournament selection and uniform crossover. For larger and more complex problems, using NetKeys generally speeds up the SGA significantly in comparison to characteristic vectors both for the one-max tree problem and the real-world problem. The empirical results support the theoretical assertions and show that genetic and evolutionary algorithms work faster and more reliably with NetKeys than with characteristic vectors.

Based on the presented results we encourage further study of NetKeys for encoding both trees and other networks. Our simple application of time-to-convergence and population-size theory gave good predictions in the empirical investigations. We believe that deeper application of this theory should be advantageous. For a controlled comparison and test of representations, operators, and algorithms a test-suite of scalable and boundedly difficult problems is necessary. We recommend that such a library be built because only with bounding test problems can the performance of genetic and evolutionary algorithms be investigated systematically and be assured to work in other less taxing circumstances. Finally, even though more work is needed, we believe that the results presented are sufficiently compelling to immediately recommend increased application of the NetKey encoding in real-world applications.

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.

Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing*. Boston: Kluwer Academic.

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.

Bean, J. C. (1992, June). *Genetics and random keys for sequencing and optimization* (Technical Report 92-43). Ann Arbor, MI: Department of Industrial and Operations Engineering, University of Michigan.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*(2), 154–160.

Berry, L. T. M., Murtagh, B. A., & Sugden, S. J. (1994). A genetic-based approach to tree network synthesis with cost constraints. In Zimmermann, H. J. (Ed.), *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, Volume 2 (pp. 626–629). Promenade 9, D-52076 Aachen: Verlag der Augustinus Buchhandlung.

Davis, L., Orvosh, D., Cox, A., & Qiu, Y. (1993). A genetic algorithm for survivable network design. See Forrest (1993), pp. 408–415.

Forrest, S. (Ed.) (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Fox, B. R., & McMahon, M. B. (1991). Genetic operators for sequencing problems. In Rawlins, G. J. E. (Ed.), *Foundations of Genetic Algorithms* (pp. 284–300). San Mateo, CA: Morgan Kaufmann.

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

Goldberg, D. E. (1989b). Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 70–79). San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (1990, September). *Real-coded genetic algorithms, virtual alphabets, and blocking* (IlliGAL Report No. 90001). Urbana, IL: University of Illinois at Urbana-Champaign.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. See Forrest (1993), pp. 56–64.

Goldberg, D. E., Deb, K., & Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, *32*(1), 10–16.

Hamming, R. (1980). *Coding and information theory*. Prentice-Hall.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Bäck, T. (Ed.), *Proceedings of the Forth International Conference on Evolutionary Computation* (pp. 7–12). New York: IEEE Press.

Kargupta, H., Deb, K., & Goldberg, D. E. (1992). Ordering genetic algorithms and deception. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature- PPSN II* (pp. 47–56). Amsterdam: Elsevier Science.

Knjazew, D. (2000). *Application of the fast messy genetic algorithm to permutation and scheduling problems* (IlliGAL Report No. 2000022). Urbana, IL: University of Illinois at Urbana-Champaign.

Knjazew, D., & Goldberg, D. E. (2000a). *Large-scale permutation optimization with the ordering messy genetic algorithm* (IlliGAL Report No. 2000013). Urbana, IL: University of Illinois at Urbana-Champaign.

Knjazew, D., & Goldberg, D. E. (2000b). *OMEGA- ordering messy ga: Solving permutation problems with the fast messy genetic algorithm and random keys* (IlliGAL Report No. 2000004). Urbana, IL: University of Illinois at Urbana-Champaign.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Norman, B. A. (1995). *Scheduling using the random keys genetic algorithm*. unpublished PhD thesis, University of Michigan, Ann Arbor, Michigan.

Norman, B. A., & Bean, J. C. (1994). *Random keys genetic algorithm for job shop scheduling* (Tech. Rep. No. 94-5). Ann Arbor, MI: The University of Michigan.

Norman, B. A., & Bean, J. C. (1997). Operation sequencing and tool assignment for multiple spindle CNC machines. In *Proceedings of the Forth International Conference on Evolutionary Computation* (pp. 425–430). Piscataway, NJ: IEEE.

Norman, B. A., & Bean, J. C. (2000). Scheduling operations on parallel machines. *IEE Transactions*, *32*(5), 229–460.

Norman, B. A., & Smith, A. E. (1997). Random keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems. In *Proceedings of the Forth International Conference on Evolutionary Computation* (pp. 407–411). Piscataway, NJ: IEEE.

Norman, B. A., Smith, A. E., & Arapoglu, R. A. (1998). Integrated facility design using an evolutionary approach with a subordinate network algorithm. In Eiben, A. E., Bäck, T., Schoenauer, M., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature, PPSN V* (pp. 937–946). Berlin: Springer-Verlag.

Orvosh, D., & Davis, L. (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. See Forrest (1993), pp. 650.

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.

Prüfer, H. (1918). Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, *27*, 742–744.

Rothlauf, F. (2001). *Towards a theory of representations for genetic and evolutionary algorithms: Development of basic concepts and their application to binary and tree representations*. Doctoral dissertation, University of Bayreuth/Germany.

Rothlauf, F., & Goldberg, D. E. (2000). Prüfernumbers and genetic algorithms: A lesson on how the low locality of an encoding can harm the performance of GAs. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., & Schwefel, H.-P. (Eds.), *Parallel Problem Solving from Nature, PPSN VI* (pp. 395–404). Berlin: Springer-Verlag.

Sinclair, M. C. (1995). Minimum cost topology optimisation of the COST 239 European optical network. In Pearson, D. W., Steele, N. C., & Albrecht, R. F. (Eds.), *Proceedings of the 1995 International Conference on Artificial Neural Nets and Genetic Algorithms* (pp. 26–29). New York: Springer-Verlag.

Tang, K. S., Man, K. F., & Ko, K. T. (1997). Wireless LAN desing using hierarchical genetic algorithm. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 629–635). San Francisco: Morgan Kaufmann.

Thierens, D., & Goldberg, D. E. (1994). Convergence models of genetic algorithm selection schemes. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature- PPSN III* (pp. 119–129). Berlin: Springer-Verlag.