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

Article *in* Evolutionary Computation · February 2002

DOI: 10.1162/106365602317301781 · Source: PubMed

3 authors, including:

Franz Rothlauf
Johannes Gutenberg-Universität Mainz
209 PUBLICATIONS **2,536** CITATIONS

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

Franz Rothlauf, **David E. Goldberg** and
Armin Heinzl

Working Paper 8/2000
July 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

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

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

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

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

July 31, 2000

Abstract

When using genetic and evolutionary algorithms for the design of network structures, a good choice of the representation scheme for the construction of the genotype is important for the performance of the algorithm. One of the most common representation schemes for networks is the characteristic vector representation. However, with encoding tree networks, 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 tree networks. This paper investigates the performance of a simple genetic algorithm (SGA) using network random keys (NetKeys) for a 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, a selectorecombinative GA with NetKeys has some advantages for small and easy optimization problems. As soon as it comes to more complex problems, a GA with network random keys significantly outperforms a GA using characteristic vectors.

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

*Visiting student at the Illinois Genetic Algorithms Laboratory.

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.

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 tree 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 encodings use vectors with the same length and have 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 a GA with CVs cannot distinguish between the importance of the 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 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.

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, 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 a One-Max-Tree problem and some real-world scenarios. A theoretical investigation in population sizing for the One-Max-Tree problem, and SGAs using NetKeys is followed in section 5 by a comparison of the performance of the NetKeys and CVs when using a GA 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 random key vector of length l consists of l random values (keys). The values are chosen initially at random, and are floating numbers between zero and one, and are only subsequently modified by mutation and crossover. An example for a random key vector is $\vec{r}_4 = (0.07, 0.75, 0.56, 0.67)$. Of importance for the interpretation of the random key vector is the position of the keys in the vector. The positions of the keys in the vector are ordered according to their values in ascending or descending

order. In our example we have to identify the position of the highest value in the vector (0.75 at position 2). The next highest value is 0.67 at position 4. We continue ordering the complete vector and get the sequence $\vec{r}_4^s = 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$. From a random key vector of length l, we can always construct a permutation of l numbers. Every number between 1 and l appears in the sequence only once as the position of each key is unique.

Here are some properties of the encoding.

- A valid sequence $\vec{r}_l^s$ of l numbers can be created from all possible random keys, as long as there are no two keys r_i in the vector which have the same value.[1] Every key can be interpreted as a permutation of l numbers.

- There are many possibilities for the construction of an RK vector $\vec{r}_l$ from a sequence of numbers $\vec{r}_l^s$. Every random key vector can be scaled up by some factor and represents exactly the same sequence of numbers. As long as the relative ordering of the positions in the vector is the same, different random keys always represent the same sequence. It is necessary that $\vec{r}_l^s$ is a permutation of l numbers, otherwise no random key can be constructed.

- RKs encode both the relative ordering of the numbers in $\vec{r}_l^s$ (encoded by the value of the key at position i in comparison to all other keys), and the absolute ordering of the position i (encoded by the absolute value of the key). The absolute position of a number i in $\vec{r}_l^s$ cannot be encoded directly, but the value at the ith position determines indirectly at which position in $\vec{r}_l^s$ the number i can be found. A large value at the ith position will lead us to a position at the beginning of the sequence, and a low value leads to a position at the end.

- A look at the neighborhood of an individual shows that the locality of the random keys is high for ordering problems. A small change in the genotype (the random key $\vec{r}_l$) leads to a small change in the phenotype (the sequence $\vec{r}_l^s$). The change of one key changes the relative position of exactly one number. However, one must be careful with the definition of the neighborhood. If the absolute position of the numbers in $\vec{r}_l^s$ is important, a change of one key is disastrous. If the value of the key with the highest value is modified, only the position of this key in $\vec{r}_l^s$ changes its relative position, but up to l numbers change their absolute position in the sequence. We want to emphasize that only the relative ordering, and not the absolute positions of the numbers in the sequence, is interesting (compare Knjazew & Goldberg, 2000b). Only concerning the relative positions the locality is high.

- When using genetic and evolutionary algorithms (GEA) with random keys, all kinds of standard crossover and mutation operators can be used. No repair mechanism, or special crossover-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.

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 present how the RKs can be used.

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

3 Tree network design with NetKeys and the characteristic vector representation

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 this could be overcome by using NetKeys.

3.1 The design problem

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

For a graph with n nodes there are $n(n-1)/2$ possible links. This results in $2^{n(n-1)/2}$ possible network structures. A tree network is defined as a connected graph with n nodes and $n-1$ links. There are no loops or rings in a tree. Between any two nodes there exists only one possible path for the flow. 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 network are given. The only design variable is the structure of the tree.

3.2 Encoding tree networks with the characteristic vector

The characteristic vector is one of the most common approaches for encoding the structure of a network (Davis, Orvosh, Cox, & Qiu, 1993). Examples for the use of the characteristic vector encoding can be found in Tang et al. (1997) and Sinclair (1995). In Berry, Murtagh, and Sugden (1994) examples can be found for the use of the characteristic vector for the encoding of tree networks.

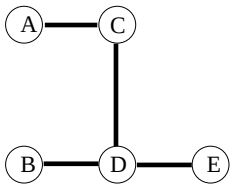


Figure 1: A five node tree network

A characteristic vector is a binary vector that indicates if a possible link is used or not in the network. For an n-node network exist $n(n-1)/2$ possible links, and a characteristic vector of length $l = n(n-1)/2$ is necessary for encoding the structure of an n-node network. 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 network. The nodes are labelled 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 of the allele is set to zero. The tree network that is represented by table 1 is shown in figure 1.

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 1

The locality of the encoding is high. Changing a zero to a one in the characteristic vector adds an additional edge to the graph. 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 just 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 (minimum step size) changes two bits in the characteristic vector. The locality of the encoding remains constant over the whole solution space. In comparison to other encodings, such as the pruefernumber, there exist no structure dependent differences for the locality of the encoding (Rothlauf & Goldberg, 2000).

Every characteristic vector that represents a tree network must have exactly $n-1$ ones, the represented graph must be connected, and there are no cycles allowed. This makes the construction of tree networks 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 (Pruefer, 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 network 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 there is a cycle of A-C-D-B-A. 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 genetic and evolutionary algorithms 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 network that is not connected. The tree is not specified completely. Some links which are necessary to connect the network are missing. A similar problem arises with the over-specification of a tree network. An open question is how to handle a network with too many links that is not a tree any more. To get a valid tree it is necessary to remove some of the links. With the over-specification we get some redundancy, but no help on which links 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). Most of the traditional GEAs do 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 network

When repairing a characteristic vector that should represent a tree, the cycles in the graph must be identified. If we randomly choose the characteristic vector $\vec{c}_{10} = 1100010100$ for a 5 node network, we are faced with the cycle A-C-D-B-A in the network. 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 $\vec{c}_{10} = 1000010100$. As there are no more loops we can stop removing links and continue checking if the network is fully connected. As there are only three ones in $\vec{c}_{10}$, and we have a tree with 5 nodes, the graph could not be connected and we have to 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 the $\vec{c}_{10} = 1000010110$. A closer look at the repair mechanisms shows that the order of the repair steps does not matter.

When using the characteristic vector representation, the decision of whether a link is established or not, is a difficult decision during the GA run, because the algorithm can store no information about the importance of a link. It has no idea which link should be removed if over-specification happens, or which links should be added if the tree is under-specified. For this reason it is not possible to distinguish between those links that are important and those that are not. In Palmer and Kershenbaum (1994) an approach for encoding trees is presented that uses biases for each node and modifies the cost matrix of the links according to the bias. The represented tree is found by constructing a minimum spanning tree using the modified cost matrix. With this approach the GA is able to distinguish between expensive and cheap nodes. 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 consider not only the importance of the nodes, but to enable a GA to distinguish between the importance of links, and to overcome the problems of redundancy and under-specification, we want to use the more general concept of random keys for the encoding of tree networks.

3.3 Encoding tree networks 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 the objects in the sequence. As NetKeys use continuous variables that could be interpreted as the importance of the link, it is possible to distinguish between more and less important links. The higher the value of the allele, the higher the probability that the link is used for the tree. Because NetKeys encode the importance of the links, no problems of redundancy, over or under-specification occur. Every NetKey vector represents a valid network structure. It is not necessary to use any special operators. Standard mutation and crossover operators could be used. With no under- or over-specification no repair mechanism is necessary for NetKeys. This means that no hard decisions about which link should be removed or added are necessary, and we loose no information due to the repair process during a GA-run.

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

1. Let $i = 0$, G be an empty graph with n nodes, and $\vec{r}_l^s$ the sequence with length $l = n(n-1)/2$ that could be constructed from the NetKey vector $\vec{r}_l$. All possible links of G are numbered from 1 to l.

2. Let j be the number at the ith position of $\vec{r}_l^s$.

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 network from every possible NetKey vector.

We want to illustrate this construction rule with an example. We use a NetKey for a 5 node network from table 2. The sequence $\vec{r}^{s}_{10} = 10 \rightarrow 8 \rightarrow 6 \rightarrow 9 \rightarrow 2 \rightarrow 7 \rightarrow 1 \rightarrow 5 \rightarrow 4 \rightarrow 3$ can be constructed from the random key vector. 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 remove 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 1.

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 NetKey vector for tree in figure 1

The computational effort for constructing the phenotype from the genotype is similar for the NetKey and the characteristic vector representation. The calculation of the sequence from the NetKey vector $\vec{r}_l$ can be done in $O(l \log(l))$ (sorting an array of l numbers). The process of constructing the graph from $\vec{r}^{s}_{l}$ 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 relative position is changed. However, a mutation of one key often dramatically changes the absolute positions of the numbers in the sequence. But as the construction of the tree is based on the relative ordering of the keys, the locality is high.

4 Test problems

4.1 The One-Max-Tree problem for topological network optimization

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

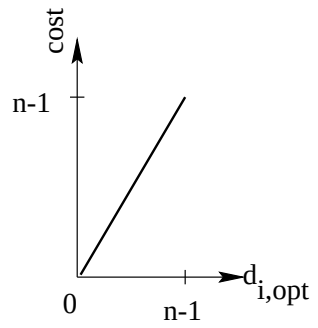


Figure 2: The cost function for the One-Max-Tree problem

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 is chosen either randomly or by hand. The structure of this tree network can be determined: It can be a star, a list, or a random tree network with n nodes.

For the calculation of the fitness of the individuals, the hamming distance for graphs is used. The distance d_{ij} between two graphs G_i and G_j is defined by half of the number of different edges in the two networks. With this metric the fitness of an individual is defined as the distance $d_{i,opt}$ to the optimal solution G_{opt}. The fitness of an individual varies between 0 and $n-1$ for a n-node network. For a minimization problem (figure 2) and a n-node tree network, an individual has a cost of $n-1$ if it has no links in common with

the best solution. If they do not differ ($G_i = G_{opt}$), the cost of G_i would be 0. For a maximization problem, the fitness f_i of an individual G_i is defined as the number of edges it has in common with the best solution.[2] Because this test problem is similar to the standard one-max-problem it is easy to solve for mutation-based GEA, but somewhat harder for recombination-based GAs (Goldberg, Deb, & Thierens, 1993). The knowledge about the one-max-problem can be used for this One-Max-Tree-problem.

If our example network from figure 1 is chosen as the optimal solution and we have a minimization problem, the star network with center D would have cost of 1, because the two networks differ at two edges[3] ($d_{i,opt} = 1$).

In our runs we use either randomly generated trees, star networks or list networks as best solutions.

4.2 Four real-world communication 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 link consists of a fixed and length dependent share. Both depend on the capacity of the link.

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). For the cost structure we use the cost model of the German Telecom from 1996. The cost of renting a line depends on the length and the capacity of the link. Possible line capacities are 64 kBit/s, 512 kBit/s and 2048 kBit/s. The optimal solution for this problem is shown in figure 3(a). The complexity of the problem is low.

4.2.2 Problem 2: One headquarter and only 14 branches

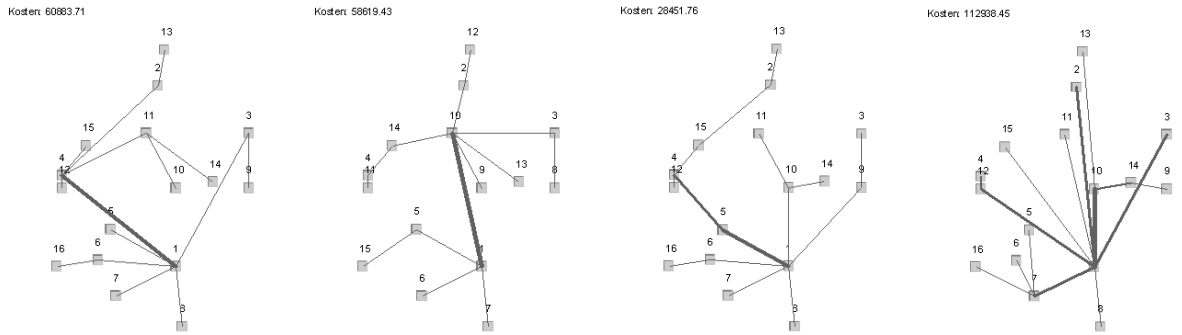
If one node is left out and some additional traffic is added, finding the best solution is slightly more involved than in problem 1.

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

In the scenario shown in figure 3(c) the cost for installing a line 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 determined only by the length of the link, and if there was only one possible capacity, the optimal solution would be the minimum spanning tree. Otherwise the problem is exactly like problem 1.

[2] $f_i = n - 1 - d_{i,opt}$

[3] A-C respectively A-D



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

Figure 3: Optimal solutions for the real world problem

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

In problem 4 depicted in figure 3(d) the demand matrix is completely filled. Between every node i and j exists some traffic. 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 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[4]. 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 a line of 128 kBit/s and 4096 kBit/s with twice the cost of a 64kBit/s respectively the 2048 kBit/s line.

5 Experimental results

This section presents experiments comparing the performance of the network random key representation and the characteristic vector representation for the standard test problem (subsection 4.1), and the real-world problem from subsection 4.2.

5.1 Optimization parameters

For our investigation we use a simple genetic algorithm with different selection schemes and crossover operators. The simple GA (SGA) is based on Goldberg's basic implementation (Goldberg, 1989a) with the roulette wheel selection procedure replaced with one of two possibilities depicted in the next paragraph. For recombination either one-point or uniform crossover is used.

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.

[4]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.

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 the One-Max-Tree problem the population size N for the runs was set to $2 * N_{min}$ (compare the following subsection). For the real world telecommunication problem the population size N for the SGA is set to 2000 in all runs.

5.2 Population sizing and stealth mutation

In all our runs we only used crossover ($p_{crossover} = 1$) and no mutation ($p_{mut} = 0$). Because a GA without mutation is 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 BBs. An upper bound for the cardinality χ of the alphabet is $n(n-1)/2$. For calculating σ_f we have got to investigate how to decide among the competing BBs. We can split the $n(n-1)/2$ possible values of one allele in $n/2$ different partitions of size $(n-1)$. A good decision means deciding between the $n/2$ different partitions and identifying the correct one. This is similar to the needle in a haystack model and the standard deviation for such a case is $\sigma_f = \sqrt{2l(n-2)}/n = \sqrt{(n-1)(n-2)/n} \approx \sqrt{n}$ (Goldberg, Deb, & Clark, 1992). Using this result and with $l = n(n-1)/2$, $k = 1$, $\alpha \approx 0.01$, $d = 1$ and $\chi = n(n-1)/2$ we get an upper bound for the population size:

$$N_{min} = -\frac{\sqrt{\pi}}{4} \ln(\alpha) \sqrt{n(n-2)}(n-1)^{1.5} \approx 2n^{2.5}$$

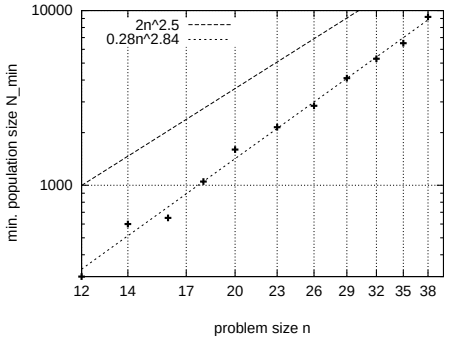


Figure 4: Minimum pop size N_{min} for NetKeys over different problem sizes n

In figure 4 the minimum necessary population size N_{min} that is necessary for solving the One-Max-Tree problem with a SGA with tournament selection, one-point crossover and NetKey encoding over the problem size n is shown. The regression line is $0.28n^{2.84}$. We performed 100 runs for each population size (step size 50) and for $N > N_{min}$ the GA is able to find the optimum in all 100 runs ($\alpha \approx 0.01$). However, we have to make some assumptions in our derivation, and the NetKey encoding is difficult to describe theoretically, the population sizing model gives us a good approximation of the expected complexity of N_{min} ($O(n^{2.5})$ versus $O(n^{2.84})$).

A comparison of the results for NetKeys with those of CV-encoding is difficult. As we have seen in subsection 3.2, the repair process fixes under-specified individuals by inserting links randomly. Although we only want to investigate recombination we still get mutation when we use characteristic vectors. This effect caused by the repair process should be called *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. Because there is some supply of BBs during the run with CVs, the comparison between NetKeys and CVs is unfair. Experiments have shown that because of stealth mutation of the CVs, a SGA can always find the optimal solution for the simple One-Max-Tree problem even with small population

sizes ($N > 300$), but needs many more generations. SGAs using NetKeys show the same behavior if we add some mutation.

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 network with G_{opt} as a tree. 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_{rk} 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 networks, a SGA using NetKeys is 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 network sizes (highly significantly with $p_T < 0.001$).

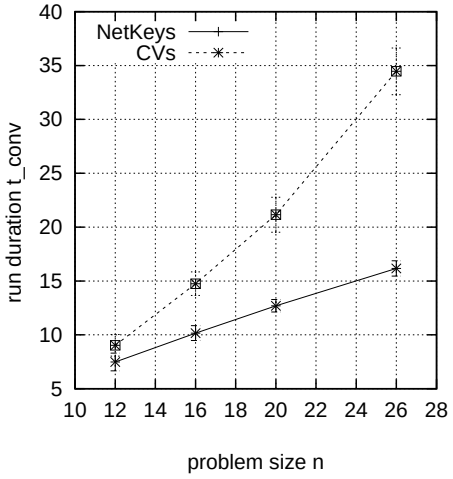


Figure 5: Run duration t_{conv} over problem size n for the One-Max-Tree problem using tournament selection and uniform crossover

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. With 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 more on the random effects of mutation than on recombination. As a result the more complex the problem is, the better NetKeys perform in comparison to 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 we show in figure 5 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 the GA, 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 network. In principle the SGA shows the same behavior as for trees. Tournament selection with uniform crossover and NetKeys performs significantly better than all other parameter settings.

In figure 6 the averaged lowest cost for the 100 runs is plotted over the number of generations for 16, 20 and 26 node networks 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 GA for the different encodings. The convergence and search behavior of the GA for the different parameter settings show the same properties as in table 3. GAs using

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

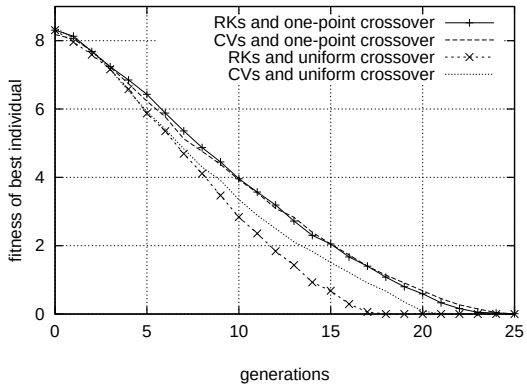
Table 3: A comparison of a SGA with NetKeys and CVs for the One-Max-Tree problem

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 CV.

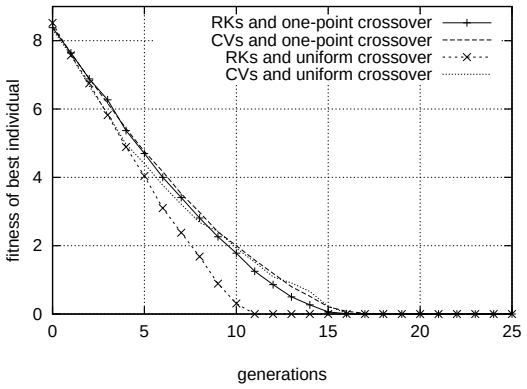
SGAs using the NetKey representation show some advantage in comparison to the characteristic vector representation. However, the results are muddied by stealth mutation. With increasing problem size the effect of the stealth mutation declines and a SGA always performs significantly better with NetKeys than with CVs.

5.4 Performance of the NetKey encoding for a real-world telecommunication network design problem

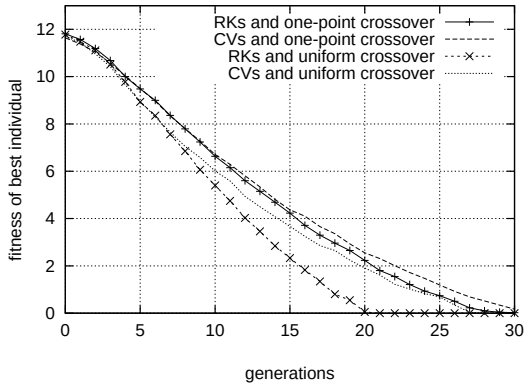
We want to test the performance of the NetKey encoding on the real-world problem from subsection 4.2. We only show results for uniform crossover, because in our runs uniform crossover generally outperforms one-point crossover.



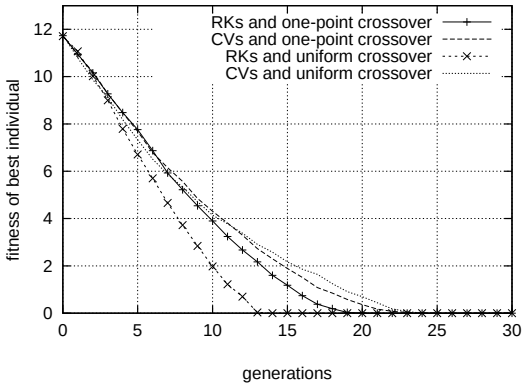
(a) $n = 16$ and $(\mu + \lambda)$ selection



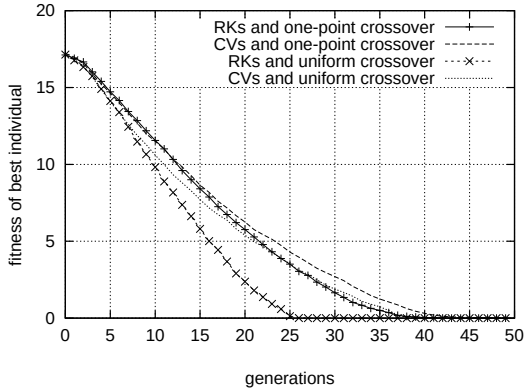
(b) $n = 16$ and tournament selection



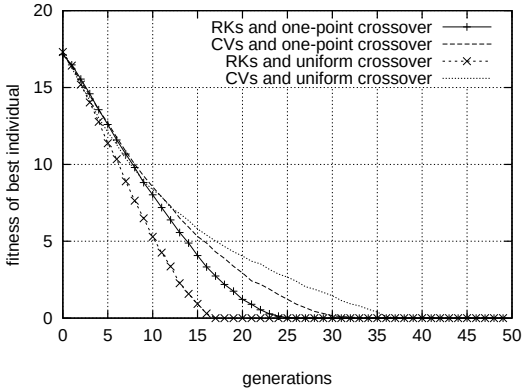
(c) $n = 20$ and $(\mu + \lambda)$ selection



(d) $n = 20$ and tournament selection

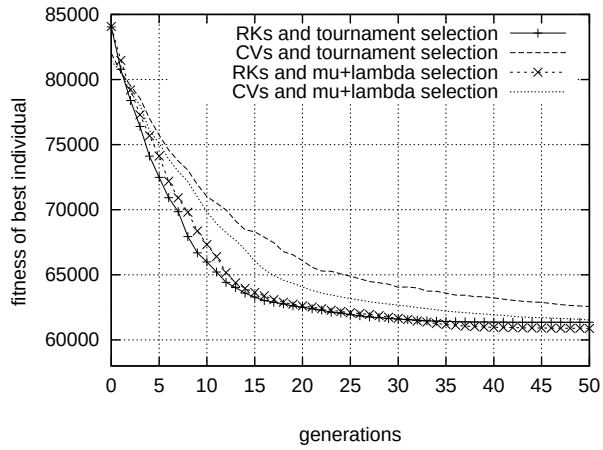


(e) $n = 26$ and $(\mu + \lambda)$ selection

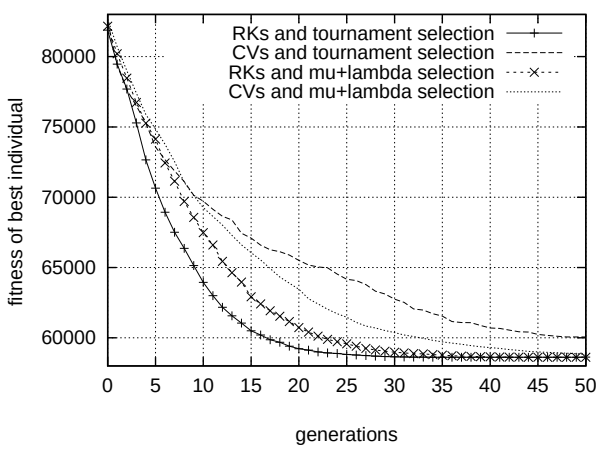


(f) $n = 26$ and tournament selection

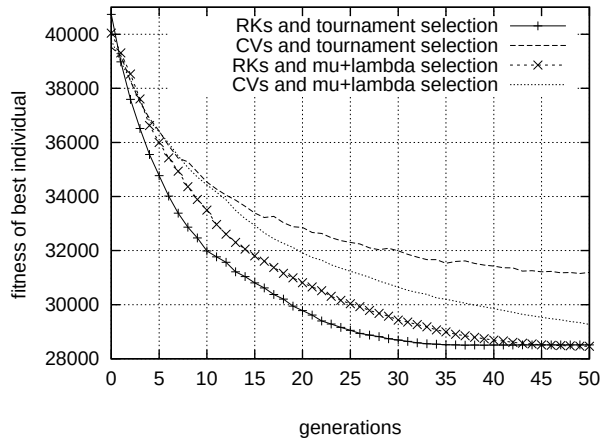
Figure 6: Fitness of the best individual over a number of generations for the 16 (top), 20 (middle) and 26 (bottom) node One-Max-Tree problem (Best solution is a tree). A SGA using NetKeys generally performs better than when using characteristic vectors. The higher the problem size, the better are NetKeys in comparison to CVs.



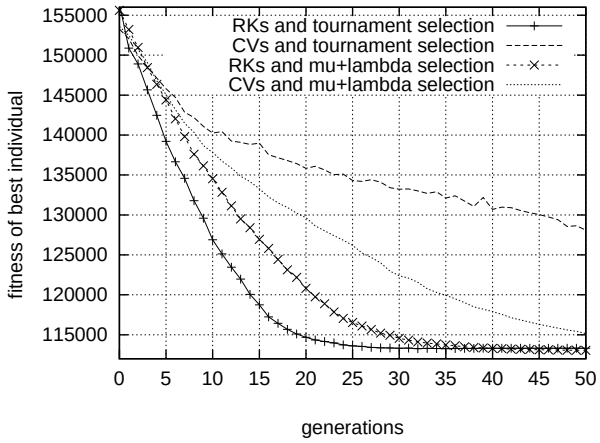
(a) problem 1



(b) problem 2



(c) problem 3



(d) problem 4

Figure 7: The performance of the SGA for the four real-world problems when using NetKeys in comparison to CVs. The SGA generally performs better with NetKeys than with the CVs. The higher the problem complexity, the better the SGA performs with NetKeys. For complex problems (problem 4), a GA with 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 real-world problem.

Figure 7 compares the performance of a GA using NetKeys with using the characteristic vector for the telecommunication network design problem. The problem complexity increases from problem 1 (figure 7(a)) to problem 4 (figure 7(d)). We show the performance of the GA with different selection schemes (tournament and $\mu + \lambda$ selection) for NetKey versus characteristic vector representation.

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 a SGA with CVs continues finding better solutions even when a SGA with NetKeys is 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 7(a)) the NetKeys perform slightly better than CVs. With increasing problem complexity the differences become larger. For problems 3 (figure 7(c)) and 4 (figure 7(d)) CVs could independently of the selection scheme not find the optimum in a reasonable amount of time.

In table 4 we present the mean μ and the standard deviation σ of the best solution after 50 generations averaged over 100 runs. The distance to the optimal solution is indicated by p_w in percent. A SGA with $(\mu + \lambda)$ selection and NetKeys is able to find solutions that are at the most 0.1% worse than the optimal solution in all four problems. For problem 2 the optimal solution was found in all runs. However, a SGA with CVs is up to 2.8% worse than the optimum. The results are even worse for CVs with tournament selection. For all four problems NetKeys are not worse than 0.77% in comparison to the optimum. With 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 the GA 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 converge generally 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 could 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
- Use of NetKeys for general network topologies
- Creation of test library for network problems
- Properties of the NetKeys in comparison to other network 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 tree networks. In principle NetKeys could also be used for meshed networks. This could easily be done by introducing a measure 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 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 network 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) was shown that the Pruefer number encoding that could also be used for tree network representation 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 network 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 evaluating encodings and verifying empirical results. If existent theoretical models would be used more often 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 GEA is that the search process works on the genotype and not on the phenotype of a problem. Therefore, it is not necessary to define for every problem specific operators 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 phenotyical 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 has done a number of things. We started 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 applying known theory about the one-max problem to simple GAs using 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 distinction between the importance of the links that should be used for constructing a network. NetKeys overcome this problems in theory as they use no repair mechanism, and they are able to distinguish between the important and unimportant links. This is possible because NetKeys encode the structure of the tree network as an ordered sequence of links, whereas the characteristic vector representation only encodes if 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 network 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 al-

gorithms 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. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 408–415). 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. (Also TCGA Report 88004).

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. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.

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.

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. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 650). 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.

Rothlauf, F., & Goldberg, D. E. (2000). *Pruefernumber and genetic algorithms: A lesson how the low locality of an encoding can harm the performance of gas* (IlliGAL Report No. 2000011). Urbana, IL: University of Illinois at Urbana-Champaign.

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.