Bad Codings and the Utility of Well-Designed Genetic Algorithms

Franz Rothlauf
Chair of Information Systems
University of Bayreuth
Universitaetsstr. 30
D-95440 Bayreuth/Germany
rothlauf@ieee.org
phone: ++49 921 552819

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

Abstract

This paper compares the performance of the Bayesian optimisation algorithm (BOA) to traditional Genetic Algorithms (GAs) such as the simple GA, or an $(\mu + \lambda)$ Evolution Strategy for a real-world telecommunication problem. Users often notice that GAs perform well for real-world problems, but when the problem is slightly scaled up or modified, they sometimes fail unexpectedly. Competent GAs, such as BOA, however promise to overcome this problem more efficiently, and to behave more robustly on demanding problems. In this practical case study we use the pruefernumber encoding as an example of a bad encoding, that causes GAs difficulty in finding a good solution. The results of the experiments show that traditional GAs sometimes succeed and sometimes fail for different parameter settings or modifications of the encoding. The behaviour could not be predicted. The BOA however is able to perform as well or better than the best traditional GA, and more importantly does not fail once in this case study. It seems that the BOA is a step along the long road towards more robust and competent GAs, that are easier to use by real practitioners on problems with unknown complexity.

1 Introduction

The development of more competent and robust GAs has received more attention over the last few years. These types of GAs promise the user to scale better when used on difficult problems or bad encodings.

Until now competent GAs were mostly applied to artificial lab environments. The purpose of this paper is to use a competent GA (ComGA) for a real-world telecommunication tree network design problem and to compare its efficiency to traditional GAs. For the encoding of the tree networks, the pruefernumber encoding is used. This encoding is highly suitable for representing trees, but has the large disadvantage of low locality (Palmer & Kershenbaum, 1994; Rothlauf & Goldberg, 1999), making it very difficult for traditional GAs to find good solutions.

The paper is structured as follows. In the following section we take a closer look at the motivation for competent GAs. This is followed by a short overview of the tree network design problem and the pruefernumber encoding used. In section 4 we present the different GAs (4.1) and the four different optimisation cases (4.2). The results of the comparison between the simple GAs and a competent GA (BOA) in section 5 are followed by concluding remarks.

2 The promise of competent GAs

This section provides a detailed description about some problems with encodings and traditional GAs, and why ComGAs could help a user to apply GAs to problems of unknown complexity and properties.

Users are often confronted with the situation that a GA does a good job for a small or easy problem, but when scaled up to bigger or more complicated problems, the traditional GA degrades or even breaks down. The user is frustrated and starts knob twiddling until he gets good solutions (or not!). Research is under way to help the user to overcome this problem and to adjust the parameters of a GA autonomously (Harik & Lobo, 1999). Nevertheless, finding good encodings, operators, and parameters that all fit together is a difficult task and needs work, time, and sometimes luck. The process of matching the encoding and the

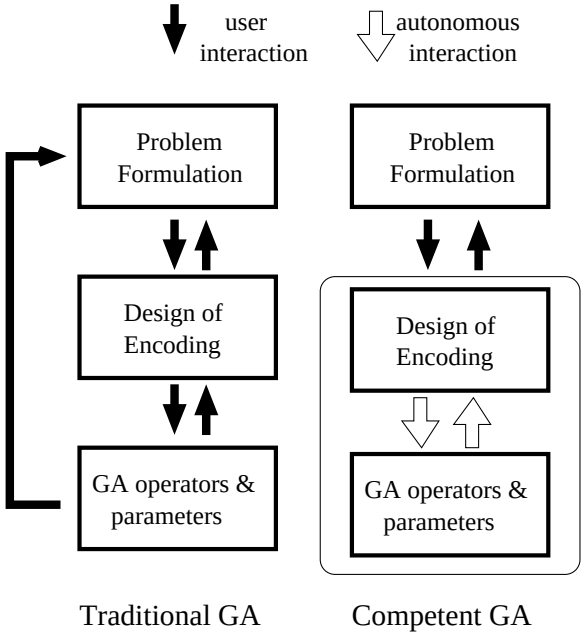


Figure 1: Necessary user interaction for traditional GAs and ComGAs

GA often draws the line between failure or success of optimisation. Figure 1 illustrates that when using a traditional GA, the fitting between the GA, the encoding and the problem must be done iteratively by hand. With ComGAs, at least the fitting of the encoding and the GA should be done autonomously by the ComGA.

We must ask why a good fitting between the encoding and the GA is necessary for the success of a GA? Some of the requirements on good encodings can be postulated following the theory of building blocks (Goldberg, 1989). For most of the current GAs the encoding must be chosen such that the existing building blocks (BB) in the gene are close together and not scattered over the whole string. Otherwise the traditional recombination operators would frequently break up the BB and the GA would not be able to detect the correct BBs (Harik, 1997). Also a high locality of the encoding helps the GA to find its way to good solutions. A high locality means that small changes in the encoding lead to small changes in the encoded tree. Traditional GA operators such as crossover and mutation degrade on low locality encodings because in this situation they behave like random search and do not produce offspring which have much in common with their parents.

To overcome this problem and to help the user, new and more intelligent GA methods are necessary. These kind of GAs should do the matching process between the encoding and the GA autonomously (fig. 1), and should be more robust when used on bad encodings. Two different approaches to solve these problems are applicable: Either the encoding can be modified in such a manner that it is easier for a traditional GA to find the correct BBs or the GA can be adjusted to the properties of the encoding.

Changing the encoding means reordering the representation that the BB are short and closed together. When this is done successfully, a traditional GA should be able to do a good job. Examples for this line of research are fmGa (Goldberg et al., 1993) or GemGA (Bandyopadhyay et al., 1998). When adjusting the GA to the encoding, the GA must be able to detect the linkage between the different alleles. During a GA run new generations are produced according to the gathered linkage informations. These kinds of competent GAs (ecGA (Harik, 1999), BOA (Pelikan et al., 1999)) are able to "learn" the structure of the encoding. They produce new individuals according to the linkage they detect between the different alleles. In the current case study we want to focus on the Bayesian optimisation algorithm (BOA) for a telecommunication tree network design problem.

3 A case study in bad encodings: Telecommunication tree network design

This section gives a short description of the tree network design problem and the chosen encoding.

3.1 The design problem

Finding good solutions for the tree network design problem is important for many fields such as telecommunications, computers, transportation and distributing networks.

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 minimise the overall cost of the telecommunication tree network. The location of the nodes, the level of traffic between the different locations, and the cost structure of the lines that can be used for constructing the network are given. For fulfilling the demands between the nodes different line types with different capacities and costs are available. The cost of a line consists of a fixed share for installing the line and a length dependent share. The only design variable is

the structure of the tree. The structure is represented by the pruefernumber (fig. 2) and describes between which nodes in the network a link is established or not. After determining the links between the nodes, the overall flow over each link is calculated and a line with the next higher available capacity is assigned to the link. The overall cost of the network is calculated by summing up the costs of all lines over all existing links.

3.2 The pruefernumber encoding

The pruefernumber has the remarkable benefit that it represents trees only and that all possible trees can be represented by a pruefernumber.

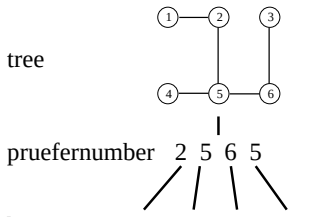


Figure 2: A tree, its pruefernumber and the corresponding bitstring

In addition all trees are represented equally. No repair operators are necessary when using GAs. Every possible number represents a tree and is valid (fig. 2).

Unfortunately, the pruefernumber has the major drawback that the locality of the encoding is low and irregular over the entire solution space (Rothlauf & Goldberg, 1999). Only for star networks do small changes of the pruefernumber lead to small changes in the encoded tree. For all other networks a slightly different pruefernumber often represents a completely different network. As a result the pruefernumber causes problems for a GA and seduces the user to start knob twiddling until he is satisfied with the results.

The pruefernumber itself is a number with $n-2$ digits of base n for a tree with n nodes. For our investigation the pruefernumber is represented as a bitstring.

A good description of the construction of the pruefernumber can be found in Palmer and Kershenbaum (1994).

4 Experiments

4.1 Three different evolutionary optimisation algorithms

For our investigation we compare the efficiency of the Bayesian optimisation algorithm (BOA) (Pelikan, 1999) with a modified, crossover based $(\mu + \lambda)$ Evolution Strategy (Bäck & Schwefel, 1995) and a simple Genetic Algorithm.

4.1.1 A Simple Genetic Algorithm

The simple GA (SGA) is based on Goldberg's basic implementation (Goldberg, 1989).We must tell the GA the probability of crossover and mutation and what kind of selection and recombination scheme should be used. In all our runs we use tournament selection. For recombination either one-point or uniform crossover is used. The crossover rate was normally set to 1 and no mutation was used.

4.1.2 An Evolution Strategy

The $(\mu + \lambda)$ Evolution Strategy (ES) produces new generations mainly with crossover and not mutation. Because the locality of the pruefernumber encoding is low, a mutation based approach would result more often in a random rather than a precise search. In each generation λ individuals are created by recombination (uniform or one-point crossover) and mutation from the μ parent individuals. Then the new λ individuals are examined and the μ best are chosen from all $\lambda + \mu$ individuals. One consequence of this strategy is that a once found good solution can only be replaced by better solutions.

4.1.3 The Bayesian Optimisation Algorithm

The BOA must be told the population size n , the maximum length k of the Building Blocks, the percentage of parents and offspring and the threshold ϵ for fixing the alleles in the gene. It terminates, when all alleles are fixed and the population is converged. In all runs the truncation selection with a percentage of parents $\tau = 50\%$ was used. The worst 50% of the population are replaced by the offspring. For the length of the BBs equal to zero (no interactions between the bits) the BOA should behave like a simple GA with uniform crossover and a proportional selection scheme. A more complete description of the algorithm can be found in Pelikan et al., 1999.

4.2 Four network design problems

Our network problems are derived from a real-world 26-node problem from a company with locations all over Germany. To make our case study more representative we have chosen four different design problems: First we have a 16-node problem with all traffic ending in node 1. In the second problem, one of the nodes is removed and some additional traffic is added. The third problem 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. Also for installing a link some money must be paid. Possible line capacities are 64 kBit/s, 512 kBit/s and 2,048 MBit/s. The optimal solution for this problem (fig. 3) is 60883 DM/month. The complexity of the problem is low.

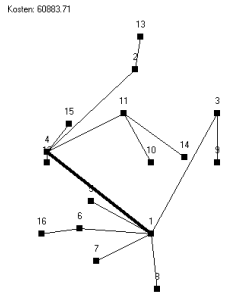


Figure 3: Best solution for problem 1

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 a little bit more involved than in problem 1.

We illustrate in figure 2 that each pruefernumber is encoded as a binary bitstring. Therefore we can get invalid solutions.[1] The probability of getting an invalid digit is about $\frac{1}{16} = 6.25\%$. With 13 digits in each pruefernumber the probability of getting a valid bitstring is about $0.9375^{13} = 43.2\%$. Invalid pruefernumbers are repaired by shifting the highest order bit from 1 to 0 to get valid numbers. This causes the encoding to be a little bit more complicated and disruptive and makes it harder for the GA.

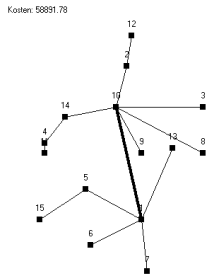
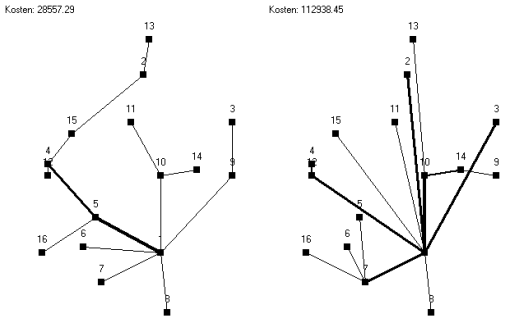


Figure 4: Best solution for problem 2

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

In this scenario (fig. 5(a)) the costs for installing a line is only 10% of the costs in problem 1. Therefore the cost of a link is mainly determined by the length of the links. 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 is the minimum spanning tree. Otherwise the problem is exactly like problem 1.

[1] The binary string 1111 encodes the node number 16, but there are only 15 nodes.



(a) problem 3 (b) problem 4

Figure 5: Optimal solutions for problem 3 and 4

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

In problem 4 (fig. 5(b)) 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[2]. 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 resp. the 2048 kBit/s line.

5 Experimental results

This section presents computer experiments investigating the performance of the different optimisation methods. The goal is not to find exactly the "optimal" parameter settings, but more to have a closer look at the robustness. To get more generality, all four different optimisation scenarios (4.2) are studied. For optimisation we use the simple Genetic Algorithm (SGA), a crossover-based Evolution Strategy (ES) and the Bayesian optimisation algorithm (BOA) from subsection 4.1.

[2] 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.

The section starts with a description of the optimisation parameters. Then the performance of the simple GA is compared with the efficiency of the ES. That is followed by an investigation of the performance of the BOA. Also some parameter twiddling is done to get an idea of the robustness of the optimisation methods.

5.1 The design of the computer experiments

Because of the high complexity of the problem and to gain statistical evidence from the results, 100 runs were performed for each parameter setting. The runs terminated after convergence, or if no convergence is possible, after 200 generations.[3] The population size is varied from 1000 to 7500 individuals[4].

In Abuali et al. (1995) it was shown that a GA works well on the pruefernumber encoding for one-point or uniform crossover. As we get the best results using tournament selection we use this selection scheme with a tournament size of 3. In the $(\mu + \lambda)$ Evolution Strategy the size of the parents μ is equal to the size of the offspring λ. For the BOA the fittest 50% of the population are treated as parents. The offspring replace the worst 50% of the population in each generation. The threshold ϵ is set to 0.01. The maximum length k of the Building Blocks is set to $0, 1, 2, 3$ or 5. The mutation rate for the SGA and the ES is set to zero. For recombination either one-point or uniform crossover is used. For the encoding of the pruefernumber we use the bitstring from fig. 2. As an extension and to make the encoding more demanding for a GA the positions of the bits are scrambled randomly. This eliminates the effort of the position of an allele in the string.

5.2 "Traditional" GAs

We want to compare the efficiency of a simple GA and a crossover-based $(\mu + \lambda)$ Evolution strategy for different crossover schemes and pruefernumber encodings.

In figure 6 the averaged lowest cost for problem 1 is shown. The cost of the cheapest solution at the end of the run is plotted over the number of fitness calls till convergence. It can be seen that the SGA with uniform crossover performs poorly and is not able to detect good solutions. Whereas an ES with uniform crossover performs as well as an ES or SGA with one-point crossover. The SGA and as well an ES degrade for problem 2 (figure 7) when using uniform crossover. Only GAs with one-point crossover can find good solutions. The situation is similar for problem 3 (fig. 8) and problem 4 (fig. 9). The ES with one-point crossover performs best and the SGA with one-point crossover is slightly worse. An ES and a SGA with uniform crossover fail. If the positions of the pruefernumber bitstring are scrambled randomly the ES and the SGA fail for each of the four scenarios (fig. 6-9). For onepoint crossover, where we have seen good results for the unscrambled encoding, the GAs degrade. Looking at the effort of different crossover operators reveals that one-point crossover is mostly better than uniform crossover. Scrambling the positions of the bitstrings causes the GAs to fail. The GAs are not able to detect the correct BBs.

Reasons for this unexpected result could be due to the low locality of the pruefernumber and the importance of the order of the digits in the bitstring[5]. Using two long substrings (one-point crossover) is better for the fitness of the offspring than using a lot of small "substrings" of length 1 (uniform crossover).

Trying to adjust the GA more to the problem can improve the results a little bit. In figure 10 and 11 different plots for problem 2, a crossover rate of 0.8 and a mutation rate of 0.01 and 0.02 are shown for one-point and uniform crossover. It can be seen that finding the best parameters is tricky. Adding mutation leads for one-point crossover to better results for the ES and to worse results for the SGA. For uniform crossover only a mutation rate of 0.01 increases the performance of the SGA. When using an ES the performance of uniform crossover is better with mutation.

The problems with "traditional" GAs are illustrated with this case study very clearly. The robustness of the GAs could not be foreseen and they could degrade on problems by slightly changing the parameters or modifying the encoding.

5.3 Competent GAs

As we have seen in subsection 5.2 that uniform crossover is bad for the performance of a GA, the BOA is expected to have a hard time. We have seen that it is advantageous to transfer longer bit chains from the parents to the offspring, but the BOA must construct this chains from the linkage between the different alleles in the string.

Figure 12 shows the performance of the BOA for different BB lengths k and compares it to the performance of the best traditional GAs for problem 1. Increasing

[3] Only very few runs were stopped after 200 generations. As we use no mutation the populations converged typically after 50 to 120 generations.

[4] The populations converged typically after 50 000 (1000 ind.) respective 400 000 (7500 ind.) fitness calls

[5] Compare the performance of the GA for the scrambled and unscrambled encoding.

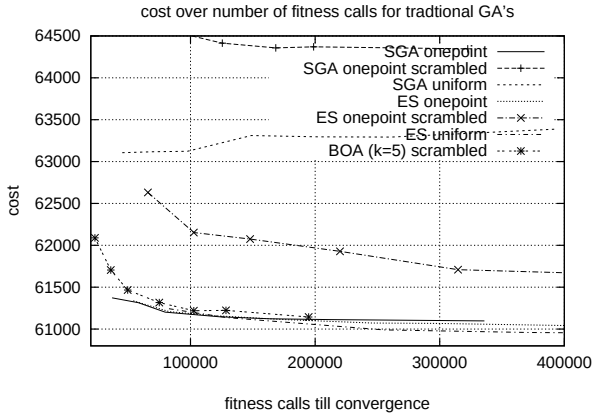


Figure 6: The performance of the SGA and the ES on problem 1. The SGA fails for uniform crossover. Both GAs fail for a scrambled encoding.

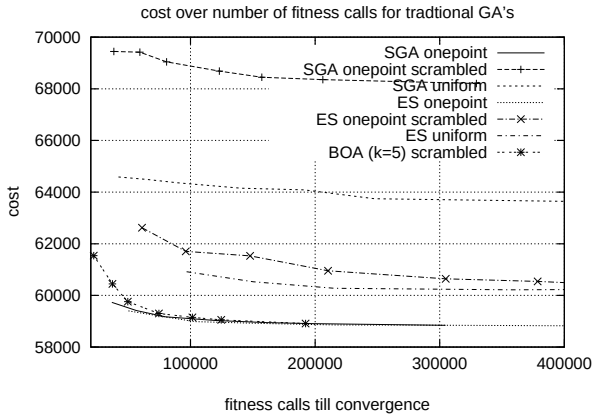


Figure 7: The performance of the SGA and the ES on problem 2. For one-point crossover and unscrambled encoding they perform well. Otherwise they fail.

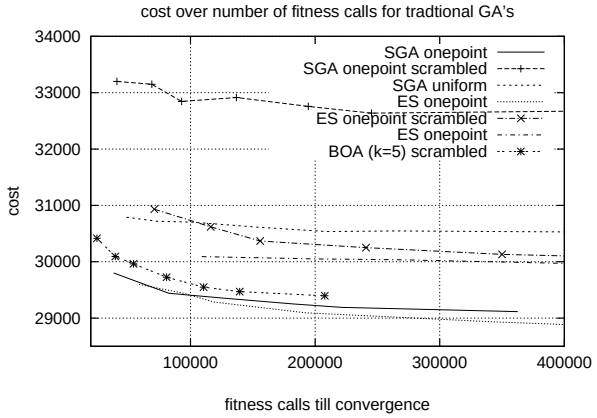


Figure 8: The performance of the SGA and the ES on problem 3. They perform poorly for uniform crossover or scrambled encoding.

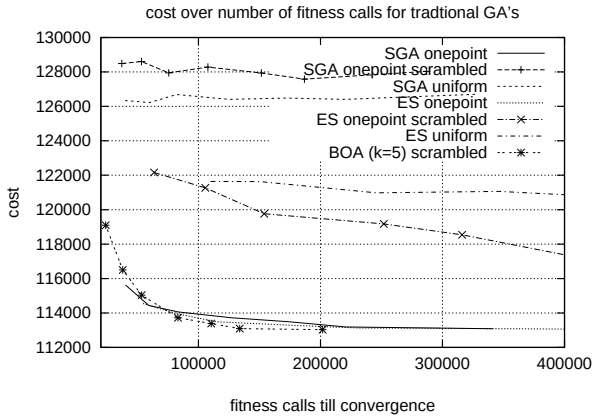


Figure 9: The performance of the SGA and the ES on problem 4. They perform well for onepoint crossover. Both fail for uniform crossover or scrambled encoding.

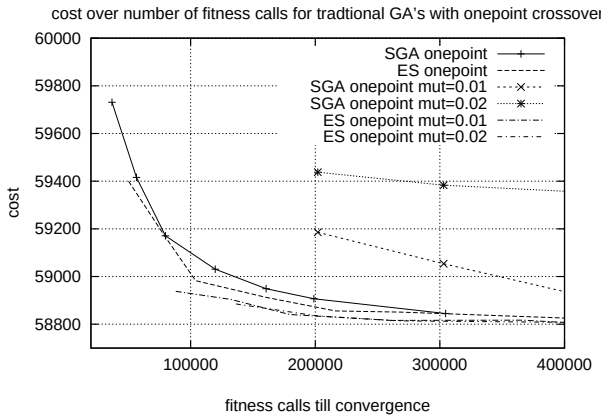


Figure 10: "Traditional GAs" with one-point crossover and additional mutation on problem 2

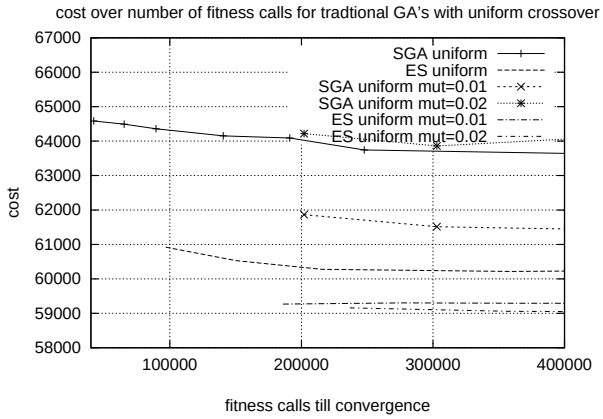


Figure 11: "Traditional GAs" with uniform crossover and additional mutation on problem 2

k leads to better results and for $k = 5$ the performance of the BOA is about the same as the best traditional GA (ES with uniform crossover). For $k = 0$ the performance of the BOA is approximately the same as a SGA with uniform crossover, so this case is missing in all figures. In figure 13, 14 and 15 the performance of the BOA is shown for problem 2, 3 and 4. In all plots the BOA behaves for high k's about equally and performs as well as the best "traditional" GA. For the scrambled pruefernumber bitstring the performance of the BOA for $k = 5$ is a little bit worse than for the unscrambled bitstring. Nevertheless the performance is in comparison to traditional GAs (fig. 6-9) quite good. The BOA performs well, the traditional GAs fail.

To have a look at the robustness of the BOA the parameters are varied for problem 2 and the effect on the solution quality is examined. Modifying the ratio between parents and offspring of the BOA for $k = 1$ in figure 16 shows no large changes. The BOA performs a little bit better, but we get no dramatic improvements or deterioration.

Comparing all results shows that the BOA is able to find good solutions. The effort for the more complex statistical model is larger, but the algorithm is in comparison to traditional GAs able to find good solutions for all reasonable parameter settings and it does not degrade on bad encodings. The behaviour of the BOA is robust. Even for a scrambled pruefernumber bitstring the BOA can detect good solutions, whereas the traditional GAs fail.

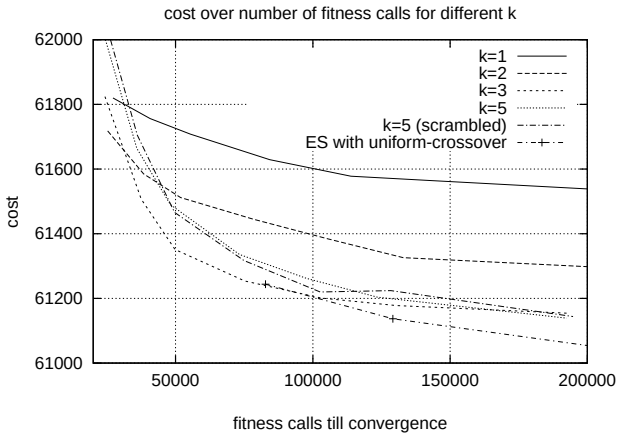


Figure 12: The performance of the BOA on problem 1

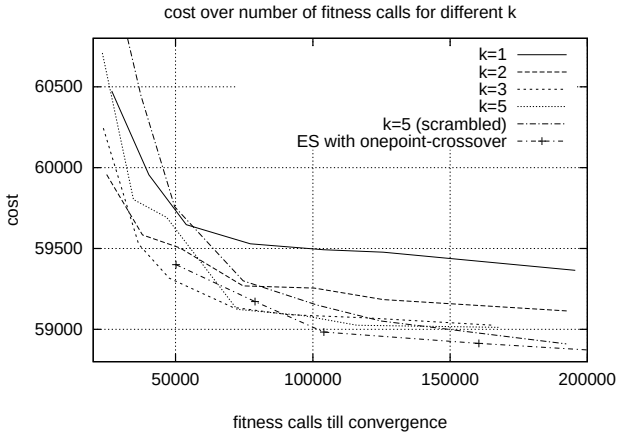


Figure 13: The performance of the BOA on problem 2

6 Conclusion

The experiments have shown that the BOA, as an example for competent GAs, is able to keep the promise of more robust GAs for a special real-world telecommunication problem. The BOA always performs well independently of the parameter settings or the modifications of the encoding, whereas the traditional GAs like simple GA and Evolution Strategy sometimes fail and sometimes succeed. Even when scrambling the positions of the pruefernumber bitstring randomly the BOA was able to find good solutions. The traditional GAs however failed. The BOA seems to be able to detect the linkage also for bad encodings. Using the BOA means more computational effort, but the user is rewarded with a more robust algorithm. This should increase the acceptance of GAs for users who solely want to solve their problems and are not interested in finding exactly the optimal parameter settings or encodings. It was shown that the BOA is a step on the long road to user-friendly, easy to handle, robust and reliable Genetic Algorithms.

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. See Eschelman (1995), pp. 470–477.

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.

Bandyopadhyay, S., Kargupta, H., & Wang, G. (1998). Revisiting the GEMGA: Scalable evolutionary optimizaiton through linkage learning. See Institute of Electrical and Electronics Engi-

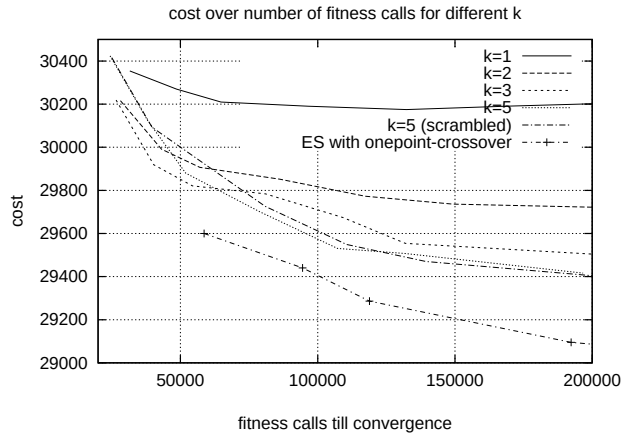


Figure 14: The performance of the BOA on problem 3

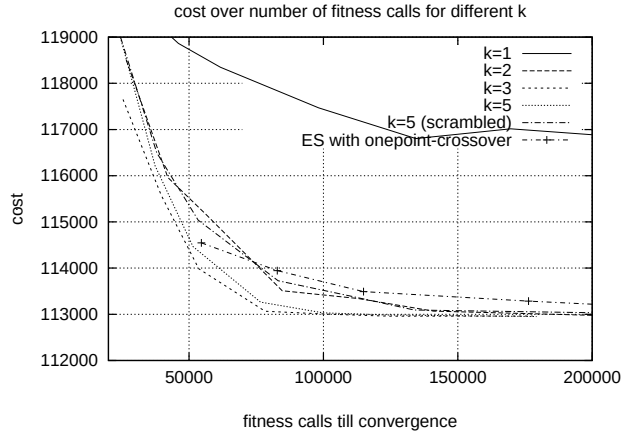


Figure 15: The performance of the BOA on problem 4

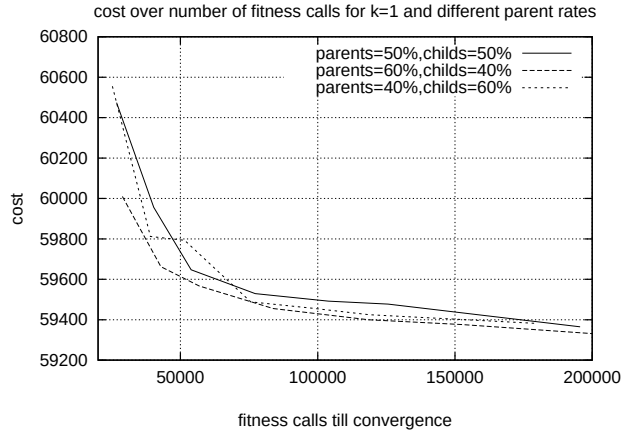


Figure 16: The performance of the BOA on problem 2 for different parents and offspring rates.

neers (1998), pp. 603–608.

Eschelman, L. (Ed.) (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann.

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

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.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL Report No. 99010). Urbana, IL: University of Illinois at Urbana-Champaign.

Harik, G., & Lobo, F. (1999). *A parameter-less genetic algorithm* (IlliGAL Report No. 99009). Urbana, IL: University of Illinois at Urbana-Champaign.

Harik, G. R. (1997). *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. Doctoral dissertation, University of Michigan, Ann Arbor. Also IlliGAL Report No. 97005.

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

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.

Pelikan, M. (1999). *A simple implementation of the bayesian optimization algorithm BOA* (IlliGAL Report No. 99011). Urbana, IL: University of Illinois at Urbana-Champaign.

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). *BOA: The Bayesian optimization algorithm* (IlliGAL Report No. 99003). Urbana, IL: University of Illinois at Urbana-Champaign.

Rothlauf, F., & Goldberg, D. E. (1999). Tree network designn with genetic algorithms - an investigation in the locality of the pruefernumber encoding. In Wu, A. S. (Ed.), *Late Breaking Papers at the Genetic and Evolutionary Computation Conference 1999* (pp. 238–244). Madison, WI: Omni Press.