

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kard. Mercierlaan 94, 3001 Leuven

ANALYSIS AND DESIGN OF GENETIC ALGORITHMS

Jury :
Prof. W. Dutré, vice-decaan, voorzitter
Prof. J. Vandewalle, promotor
Prof. B. De Moor, promotor
Prof. L. Boullart (UG)
Prof. L. De Raedt
Prof. D. E. Goldberg (UIUC, USA)
Prof. B. Manderick (VUB ; EUR, NL)
Prof. H. Van Brussel

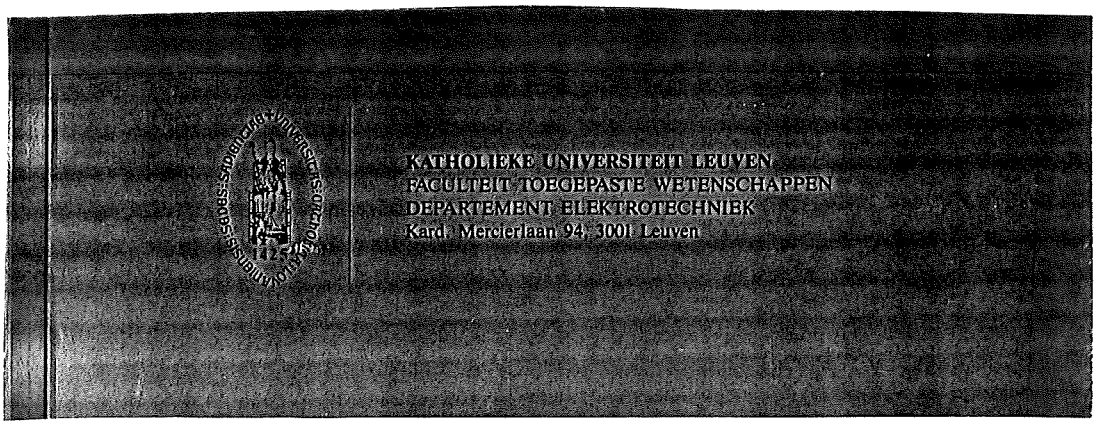
Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Dirk THIERENS

UDC 681.5.021

Mei 1995



ANALYSIS AND DESIGN OF GENETIC ALGORITHMS

Promotoren :
Prof. Dr. ir. J. VANDEWALLE
Prof. Dr. ir. B. DE MOOR

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Dirk THIERENS

Mei 1995

Voor Karine en Tajana

Voorwoord

Bij het beëindigen van mijn doktoraat wil ik graag iedereen bedanken die heeft bijgedragen tot het welslagen van dit werk.

In de eerste plaats wil ik mijn promotoren Prof. Joos Vandewalle en Prof. Bart De Moor bedanken voor hun steun en vertrouwen die ze mij de voorbije jaren geboden hebben. De onderzoeksgroep SISTA die zij leiden, is ongetwijfeld een unieke omgeving waar alles aanwezig is om het leven van een doktoraatsstudent eenvoudig te maken. Dat het pad van een doktoraatsstudent niet altijd over rozen loopt heb ik persoonlijk kunnen ervaren, en het is zeker niet de minste verdienste van Joos dat hij mij in die moeilijke periode heeft opgevangen. Zonder hem zou dit doktoraat er nooit gekomen zijn, en ik wil hem hierbij van harte bedanken.

It is a pleasure to acknowledge the profound influence Prof. David E. Goldberg has had on this work. I am very grateful that he has given me the opportunity to work so close with him. Collaborating with someone of his quality is a rare privilege, which I was fortunate to get. Our many hours of discussion learned me a great deal about what scientific research is all about. Even more importantly, he showed me how a passion for your work makes everything so easy. My five visits to Champaign-Urbana have meant a lot to me.
Thank you Dave.

Verder wil ik ook een bijzonder woord van dank richten tot Prof. Bernard Manderick. Ik heb Bernard reeds leren kennen toen ik nog student was aan de Vrije Universiteit van Brussel. Dank zij hem ben ik in kontakt gekomen met het boeiende onderwerp dat genetische algoritmen ongetwijfeld vormen. Later heb ik het genoegen gehad om samen met hem de wetenschappelijke en niet-wetenschappelijke aspekten van

verschillende internationale congressen te exploreren. Ik ben dan ook verheugd dat hij bereid was om deel te nemen aan het leescomité.

Ook Prof. Luc De Raedt wil ik bedanken voor zijn participatie in het leescomité. Het doet me een groot genoegen vast te stellen dat mensen uit het domein van symbolische machine learning, op diepgaande wijze geinteresseerd zijn in genetische algoritmen.

Prof. Luc Boullart wil ik in de eerste plaats bedanken voor het vertrouwen dat hij mij geschonken heeft toen ik verbonden was als wetenschappelijk medewerker aan het Gentse labo voor regeltechniek. Door externe omstandigheden zijn we echter verplicht geweest om deze samenwerking te stoppen. Ik ben hem dan ook erkentelijk dat hij als lid van de jury zijn steun betuigt voor mijn werk.

Verder ben ik Prof. Van Brussel dankbaar voor zijn brede wetenschappelijke interesse en zijn bereidheid om in de jury te zetelen.

Tenslotte wil ik de inzet van Prof. Dutré loven om als voorzitter van de jury, de verdediging van dit werk in goede banen te leiden.

Alle leden van SISTA wil ik bedanken voor de prettige sfeer die er heerst in de groep. De joviale en ongedwongen houding van Prof. Bart De Moor is hier zeker niet vreemd aan.

Johan Suykens en Jan Van Impe wil ik expliciet bedanken voor de vele interessante discussies.

I would also like to thank the members of the GA lab in Illinois: Kalyanmoy Deb, Hillol Kargupta, Jeff Horn, Andrew Horner, Andy Assad, Mike Rudnick, Nick Nafpliotis, Sam Mahfoud and Georges Harik. I will never forget the long, sometimes passionate discussions we had about almost everything.

Tijdens mijn Gentse periode heb ik eveneens het geluk gehad om samen met Leo Vercauteren een bureau te delen. De werkefficiency heeft er misschien een beetje onder geleden, maar onze veelvuldige discussies waren er zeker niet minder leerrijk om.

Graag wil ik langs deze weg ook mijn ouders bedanken voor alles wat ze de voorbije jaren voor mij gedaan hebben. Ze vragen zich soms af wat ik steeds uitspook op de universiteit. Ik hoop dat dit werk een tipje van de sluier oplicht.

Tot slot wens ik dit werk op te dragen aan twee bijzondere mensen: mijn echtgenote Karine die veel meer voor mij betekent dan ik onder woorden kan brengen, en ons dochtertje Tajana die nog te klein is om het allemaal te beseffen, maar die ik toch extra in de bloemetjes wil zetten: just for being there.

Acknowledgments

I sincerely would like to thank the following institutions for their indispensable help.

Financial support was provided by the Flemish Community through the Concerted Action Project *Applicable Neural Networks*, and by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture (IUAP-17 and IUAP-50).

My informative visits to the University of Illinois at Urbana-Champaign would have been impossible without the financial support by the US National Science Foundation under Grant ECS-9022007, the US Army under Contract DASG60-90-C0153, and the US Air Force Office of Scientific Research under Grant No. F49620-94-1-0103.

Abstract

Genetic algorithms are a class of optimization algorithms modeled in analogy with the mechanisms of natural evolution and genetics. Advantages of genetic algorithms are the global approach they take to explore a search space, their broad applicability, the almost trivial way to parallelize them and the ease with which they can be combined with local and heuristic search techniques.

In this thesis we study the dynamical or convergence properties of genetic algorithms. A lower and upper bound on the convergence time complexity is derived. Next we show the importance of the recombination speed of the crossover operators. The mixing time stands against the selection time, and their interaction determines whether the population will convergence to the desired optimum or not.

The analysis of the mixing and selection time gives us a clear understanding of the limits of basic genetic algorithms: without proper linkage information, the standard genetic algorithm has only a limited capacity to mix building blocks. A number of extensions - elitism, niching and restricted mating - are studied, but from a mixing point of view, they offer only marginal help. Only when the building block mixing is properly attended to, will the mixing becomes fast enough to allow effective building block exchange. One possible way, is to adaptively change the recombination probability distribution so that the building blocks are properly mixed.

The study of elitism also led to the design of an extremely simple, but very promising genetic algorithm implementation, called elitist recombination genetic algorithm. A last contribution is the presentation of a non-redundant genetic representation of feedforward neural networks that is ideally suited for recombining neural networks.

Contents

voorwoord i

acknowledgments iv

abstract v

0 Survey and Main Contributions 1
- 0.1 Overview . . . 3
- 0.2 Main contributions . . . 9
- 0.3 Software Environment . . . 13

1 Genetic Algorithm Review 15
- 1.1 Introduction . . . 16
- 1.2 Evolution as an Optimization Metaphor . . . 17
- 1.3 The Basic Genetic Algorithm . . . 19
- 1.4 Schema Theorem . . . 23
  - 1.4.1 Schemata . . . 23
  - 1.4.2 Schema Theorem . . . 25
  - 1.4.3 Optimal allocation of trials . . . 29
  - 1.4.4 Building block hypothesis . . . 32
- 1.5 Deception . . . 34
  - 1.5.1 Deceptive functions . . . 34
  - 1.5.2 Multimodal deceptive functions . . . 36
- 1.6 Schema Variance . . . 38
  - 1.6.1 Collateral noise . . . 38
  - 1.6.2 Population sizing . . . 39

1.7 Conclusion: Conditions for GA Optimization 41

2 Bounds On GAs Convergence Time Complexity 43
2.1 Introduction 44
2.2 A Lower Bound Through Normal Distribution Modeling 45
2.2.1 Introduction 45
2.2.2 Proportionate Selection 46
2.2.3 Tournament Selection 49
2.2.4 Truncation Selection 51
2.2.5 Conclusion 53
2.3 Effects Of Linkage Disequilibrium On Convergence Time 54
2.3.1 Introduction 54
2.3.2 Genic Covariance Introduction through Selection . 54
2.3.3 Genic Decorrelation through Crossover 57
2.3.4 Selection and Recombination Combined 61
2.3.5 Conclusion 66
2.4 An Upper Bound Through Domino Convergence Modeling 66
2.4.1 Introduction 66
2.4.2 Proportionate Selection 68
2.4.3 Tournament Selection 69
2.4.4 Conclusion 84
2.5 Conclusion 87

3 Dimensional Analysis of Mixing in Basic GAs 89
3.1 Introduction 91
3.2 Dimensional Analysis of Allele-Wise Mixing 93
3.2.1 Introduction 93
3.2.2 Recombination 93
3.2.3 Selection 96
3.2.4 Dimensional Model for Allele-Wise Mixing 97
3.2.5 Mixing and Linkage Biased Crossover 106
3.2.6 Conclusion 107
3.3 Dimensional Analysis of Building-Block Mixing 108
3.3.1 Introduction 108
3.3.2 Exchanging Building Blocks 110
3.3.3 Mixing Two Building Blocks 113

3.3.4 Mixing Several Building Blocks 118
3.3.5 Conclusion 122
3.4 Conclusion 123

4 Extensions to the Basic Genetic Algorithm 127
4.1 Introduction 128
4.2 Elitist Recombination 130
4.2.1 The Elitist Recombination Genetic Algorithm . . . 131
4.2.2 Example: Fully deceptive function with linkage . . 136
4.3 Niching 139
4.3.1 An overview of niching techniques 140
4.3.2 The generalized crowding algorithm 144
4.3.3 Example: multimodal deceptive function with linkage 146
4.3.4 Example: Royal Road function 149
4.4 Restricted Mating 155
4.5 Adaptive Crossover 158
4.5.1 Adaptive crossover algorithm 158
4.5.2 Adaptive crossover analysis 163
4.5.3 Example: fully deceptive function without linkage 166
4.5.4 Example: multimodal deceptive function without linkage 167
4.6 Conclusion 168

5 General Conclusions 171

A Genetic Weight Optimization of Feedforward Neural Networks 179
A.1 Introduction 180
A.2 Equivalence symmetries in neural networks 181
A.3 Genotype representation of neural networks 182
A.4 The hybrid GA+BPX algorithm 185
A.5 Example: two spirals classification problem 188
A.6 Conclusion 192

B Nederlandse Samenvatting **195**
B.1 Overzicht 197
B.2 Belangrijkste bijdragen 202
B.3 Software Omgeving 206

C Notation **207**

D References **209**

Chapter 0

Survey and Main Contributions

Since the 1960's people have tried to design computational systems in analogy with the principles of natural evolution. The profound interest in translating the evolutionary principles to the organization of artificial systems must be viewed in the light of the amazing properties of natural systems. Darwin's theory of natural selection made clear how a set of simple organizational principles can give rise to a completely self-organizing, continuously improving system, which stands in sharp contrast with the rigid, brittle and painfully designed artificial systems. Today, engineers are faced with an ever increasing design complexity and thus it becomes more and more tempting to borrow some of nature's problem solving capabilities to enlarge the engineer's problem solving toolbox. Moreover the widespread availability of high performance computing power facilitate the practical use of biologically or physically inspired algorithms.

Evolutionary algorithms are not the only members of the nature based computational family, and its two cousins - neural networks and simulated annealing - are also intensively studied and successfully applied to difficult real world problems. Although these algorithms draw their inspiration from nature, it is important to realize that this should only be considered as a metaphor to guide the design of complex problem solving methods. One sometimes hears the criticism that certain

extensions made to the basic algorithms are "unnatural" and therefore "uninteresting". However there is a huge difference between modeling natural systems and using them as a metaphor to design new algorithms. The former is certainly a very important scientific endeavor but it should not be confused with the no less important engineering task of creating powerful computational methods to assist in problem solving and design.

In this thesis we put on our engineering hat and study genetic algorithms from the viewpoint of problem solving. This leads us to study fundamental questions like "What are the limits of basic genetic algorithms ?", "What are the computational requirements ?", "How can we evaluate different genetic algorithm implementations ?" and "How can we extend the basic algorithm to increase its performance ?".

The first evolutionary algorithms made exclusive use of selection and mutation and had only limited success in terms of computational efficiency. A good review of this early work can be found in (Goldberg, 1989). The work of John Holland on adaptive systems theory (starting from the early sixties) led to the development and mathematical analysis of the genetic algorithm where great importance was given to the population based search mechanism obtained by combining selection and recombination. The mathematical foundation was formed by the schema analysis and its relation to the k-armed bandit problem which is very well described in Holland's influential book *Adaptation in Natural and Artificial Systems* that appeared in 1975. At that same year De Jong finished his important dissertation that coupled Holland's schema theory with an extensive set of computational experiments. Research on genetic algorithms gradually spread out from the University of Michigan, and in 1985 the first biannual International Conference on Genetic Algorithms was held.

It was not until 1989 however that the interest in genetic algorithms really took off: in that year David Goldberg published his book *Genetic Algorithms in Search, Optimization, and Machine Learning* which provided an ideal blend of theory, intuitive insight, implementation, historical setting, and applications to a wide audience.

Since then a lot has changed: the initial small family of GA researchers became a world-wide scientific field with many international

conferences such as the *International Conference on Genetic Algorithms*, the *Parallel Problem Solving from Nature* conference, the *IEEE Conference on Evolutionary Computation*, and the *Foundations of Genetic Algorithms* conference. Recently, MIT Press started a specific journal called *Evolutionary Computation*.

Despite the 20th anniversary of Holland's book, and the world-wide interest in evolutionary computation, genetic algorithms are still in need of theoretical studies that help us understand and give an intuitive insight into these nonlinear, population based, stochastic algorithms, applied on a large, diverse set of high dimensional and complex problems.

I sincerely hope that this thesis can make a small contribution to this endeavor.

0.1 Overview

The main subject of this thesis is to try to get an insight into the dynamical characteristics or convergence properties of genetic algorithms. The key topic is the interaction between selection and recombination in terms of their relative processing speed. More generally we look at the necessary balance between exploiting what we know by selecting previously good solutions, and exploring new parts of the search space by recombining the selected solutions.

Chapter 1: Genetic Algorithm Review

In this chapter we introduce the basic genetic algorithm and review the most important theoretical concepts of GA theory. First we consider the philosophy of seeing neo-darwinian theory as a metaphor for designing a computational system that can be used for optimization purposes. By abstracting the underlying principles into a computational framework we obtain the basic genetic algorithm.

The chapter continues with a discussion of John Holland's fundamental work on the schema theorem and the corresponding building block hypothesis. The schema theorem gives a lower bound on the expected number of instances that schemata will have at the next generation.

The schema theorem leads us to the building block hypothesis that states that short, low-order and highly fit schemata can be recombined to form higher-order schemata and complete strings with very high fitness.

Three potential problems for the building block hypothesis can be identified.

First, the building blocks might mislead the genetic search to what is called deceptive attractors. We explain the notion of deception and discuss fully deceptive functions and multimodal fully deceptive functions that are specifically designed as hard test problems for GAs, and that will also be used in this thesis.

A second problem that might occur is that the population size is too small to allow reliable selection of the building blocks. The minimal population size needed for reliable building block processing is computed by taking the schema variance into consideration.

Finally, a third potential problem for the building block hypothesis is the need for fast mixing or building block exchange. This has however largely been ignored by the research literature and will be the main topic of Chapter 3.

We conclude the review chapter by summarizing Goldberg's six conditions for reliable genetic optimization.

Chapter 2: Bounds On GAs Convergence Time Complexity

In this chapter we analyze the dynamical properties of several genetic algorithm implementations by building lower and upper bound convergence models for functions with independent gene fitness contributions.

First we build a lower bound convergence model by using normal distribution analysis for the bit counting function which has equal gene fitness contributions. The model and experimental results show that for ranked based genetic algorithms - such as tournament selection - the number of generations towards population convergence is of order $O(\sqrt{\ell})$, with ℓ the length of the binary string. Unscaled proportionate selection on the other hand has a convergence complexity of $O(\ell \ln \ell)$.

The next section shows how the covariance build-up between the alleles, slows down the convergence speed. This covariance is caused by the selection phase, and is only partly decorrelated by recombination.

In the last section we build an upper bound convergence model based on the domino convergence analysis for functions with exponentially scaled gene fitness contributions. It is shown that in domino convergence the genes are converging logistically with a time varying growth rate. A good approximation is obtained by taking an exponentially increasing growth rate which results in a wave equation model. The model and experimental results show that when using tournament selection, the number of generations needed for the population to converge is of order $O(\ell)$.

Chapter 3: Dimensional Analysis of Mixing in Standard GAs

In this chapter we study a largely ignored aspect of genetic optimization, namely the mixing or juxtapositioning of the alleles and building blocks, and its relation to selection. Although mixing has hardly received any attention in the literature, we will show that it is a key factor in order to determine when a genetic algorithm will converge to the global optimum. The limited size of the population is explicitly taken into account. We argue that from a dynamical or convergence point of view, selection and recombination are involved in a kind of race against time. The number of building blocks is quickly increased from generation to generation by the selection phase, but in order to create optimal solutions these building blocks have to be juxtaposed by the crossover operator and this also takes some time or generations to occur. If the selection of building blocks goes too fast - relative to the rate at which crossover can juxtapose or mix them - then the population will prematurely converge to a suboptimal solution. The GA will only reliably converge to the global optimum if the selection pressure is not too high, such that crossover has enough time to recombine the necessary building blocks.

We quantify this selection-recombination race by using the dimensional analysis methodology. Dimensional analysis gives us the functional relations between the key parameters, namely the selection pressure, the recombination rate, the population size, the string length and the building block length.

In Section 3.2 we present a dimensional model of the bit-wise or allele-wise mixing. First we discuss a random assortive mixing model

which is partly based on Holland's random assortive modeling of crossover. The random assortive mixing model gives the functional relation between the selective pressure and the recombination rate, but it is too crude to model the relation with the string length and the population size. Using our results of the normal distribution modeling in the previous chapter (Section 2.2), we present a complete allele-wise mixing model. The model agrees very well with experimental results, and indicates for what parameter combinations the genetic algorithm will fail to converge to the global optimum due to mixing problems. At the same time the region in the parameter space is identified where the GA will converge reliably to the global optimum.

In Section 3.3 we analyze the mixing at the level of building blocks. Traditional analysis of GA performance - as exemplified by the schema theorem - is mainly focused on selection and on low disruption characteristics of crossover operators. The mixing analysis shows however that the juxtapositioning power of recombination is also very important. The traditional view on genetic optimization considers crossover to be a source of building block disruption and therefore very disruptive operators such as uniform crossover are supposed to be of no use for problems where one needs to process building blocks as a whole. The most pronounced problem where this is the case are the fully deceptive functions, and general GA wisdom says that a basic GA can only solve these problems to global optimality if the building blocks are tightly linked and one uses one-point crossover, which has only a small disruptive effect. Assuming that the building blocks are tightly linked is however an extremely strong requirement which certainly limits the applicability of the basic genetic algorithm.

The schema theorem basically talks about the product of two terms: a factor representing the selection pressure and a factor representing the survival probability, which is the complement of the disruption loss. An interesting observation can now be made: by increasing the selection pressure we can use the highly disruptive uniform crossover operator that does not make any assumptions about linkage, and still satisfy the schema theorem. Put it another way, increasing the selection pressure ensures that we are able to grow the building blocks without any need of linkage information, so why would we worry about tight codings as in

the traditional GA approach, where whenever GA difficult or deceptive problems are faced, the emphasis is put on the use of low disruptive crossover operators and tight linkage so the disruptive factor is kept small. We have shown that fully deceptive functions indeed are solved to global optimality by using uniform crossover and a high selection pressure.

An alternative of increasing the selection pressure is to lower the disruptive factor and still use a crossover operator that is unbiased towards linkage, for instance uniform crossover with a very low allele swapping probability achieves this.

So do basic genetic algorithms need tightly linked building blocks after all ? As far as the schema theorem is concerned they do not.

Alas, increasing the selection pressure or decreasing the disruption factor does have a cost, and it is here that the importance of the need for mixing analysis shows itself most profoundly. This can easily be seen if we take either of these approaches to their logical conclusions and end up with an absurdity. If a selection pressure is chosen to counterbalance a very high loss, all population diversity will be lost almost immediately resulting in no exchange and an extreme form of premature convergence. If disruption is made very small, exchange will never take place and despite meeting the schema theorem criterion, the growing building blocks will never exchange. This points out an often ignored fact of GA practise: growing building blocks is one thing, mixing them is another.

Having a high selective pressure ensures that the building blocks increase in number, but in order to juxtapose or mix them we need a very large population size. Using dimensional analysis we present the mixing ladder climbing model that tells us that the minimal population size needed grows exponentially in the number of building blocks, which is of course computationally too expensive. Therefore, in practice the basic genetic algorithm cannot be used to solve fully deceptive problems unless the building blocks are tightly linked.

The basic GA is therefore severely limited in its mixing capacity and tightly linked building blocks are needed if we want fast convergence to the global optimum.

Chapter 4: Extensions to the Basic Genetic Algorithm

In this chapter we discuss extensions to the basic genetic algorithm and look at their impact in terms of convergence speed and mixing behavior. The extensions are tested on some hard benchmark problems which are known to pose challenging problems to most genetic algorithm implementations. In Section 4.2 we propose a locally implementable genetic algorithm - called elitist recombination GA - that intertwines the selection and recombination phase. Elitist recombination has a local elitist mechanism such that parent strings can only be replaced by children that have a better fitness value. We show that the selection pressure of the elitist recombination GA is equal to the selection pressure of tournament selection with tournament size two, which is widely used in practice. Due to the local elitism, elitist recombination has a number of additional benefits, and therefore promises to be a very useful GA implementation. Experimental results on a fully deceptive function with linked building blocks confirm the equivalence in convergence speed, and also show that elitist recombination is less sensitive to undersized populations than tournament selection.

Section 4.3 discusses the use of niching techniques in genetic optimization. After a short review of sharing and crowding techniques, a generalized crowding algorithm is outlined. The algorithm is tested on the difficult multimodal fully deceptive function with linked building blocks. Niching based genetic algorithms are usually only applied for multimodal function optimization. We argue that they have also an important role to play in tackling the selection-mixing race. To illustrate this we apply the generalized crowding algorithm on the hierarchical Royal Road function, proposed by Holland as a challenge to all GA implementations.

In Section 4.4 we analyze the use of restricted mating for increasing the mixing probability. When the mating parents are close together in Hamming space, then most building blocks will be the same in both parents and thus cannot be disrupted by any crossover operator. The mixing probability is therefore increased, but we show that this only replaces one problem with another. For restricted mating to work well, most strings need to have a Hamming neighbor in the current population, and this requirement again makes the problem intractable.

In Section 4.5 we propose to adapt the crossover operator in function of the search trajectory. By making the crossover exchange distribution dependent on the state of the current population, the building blocks are more efficiently preserved and exchanged. We test the extended genetic algorithm using the adaptive crossover operator on a fully deceptive function without tightly linked building blocks and on a multimodal fully deceptive function also without linkage information. In both cases the optimal solutions were found and maintained.

Appendix A: Genetic Weight Optimization of Feedforward Neural Networks

Appendix A looks at an application of genetic algorithms in optimizing neural networks. This problem is interesting because the neural network genotype representation gives some special problems for recombination. Structurally different feedforward neural networks can implement the same functional mapping due to representational symmetries. These redundancies however have caused major concern about the applicability of crossover to neural network optimization. A number of heuristic solutions have been proposed in the literature.

We propose a non-heuristic and very simple genotype representation that eliminates the redundancies. It is shown that crossover transmits more information from the parents to the children when using the non-redundant representation than with the straightforward but redundant coding.

0.2 Main contributions

The main contributions of this thesis can be summarized as follows. The references of results that have been published yet, are also given.

- *Normal distribution modeling of tournament selection:*

  Tournament selection is a widely used selection mechanism because it has a steady but moderate selection pressure and takes little computational overhead. We have build a convergence model

that specifies the proportion of optimal alleles in the population in function of time measured by the number of generations. The model uses normal distribution theory and is valid for fitness functions which have independent and equal gene fitness contributions. For genotype strings of length l, it is shown that the convergence time is of order $O(\sqrt{\ell})$, which is a lower bound on the convergence time complexity for functions with independent gene fitness values.

Comparisons are also made with proportionate and truncation selection. References are Thierens & Goldberg (1994b) and Thierens & Vandewalle (1994).

- *Effects of linkage disequilibrium on GA convergence:*

  Due to the selection of the better strings in a population, gene values or alleles become correlated. Crossover only partly decorrelates them again, so the net effect is a build-up of covariance between the alleles. Although this phenomenon has long been recognized in quantitative genetics - where it is called linkage disequilibrium - it has not been studied before in a genetic algorithm context. Unfortunately the results from quantitative genetics studies do not carry over to GA analysis.

  We have analyzed the effect of linkage disequilibrium on the convergence characteristics and incorporated it into the normal distribution model. We have argued that linkage disequilibrium is a key factor in understanding premature convergence, where the genetic algorithm fails to converge to the desired optimum.

  References are Thierens & Goldberg (1994d), and a short discussion can also be found in Thierens & Goldberg (1994b) and Thierens & Vandewalle (1994).

- *Domino convergence modeling of tournament selection:*

  The normal distribution model gave a lower bound on the convergence time complexity for functions with independent gene fitness values. An upper bound is obtained by modeling the convergence characteristics when optimizing functions with independent gene fitness values that are exponentially scaled. This exponentially scaling causes a domino-like convergence behavior (Rudnick, 1992). We have derived a domino convergence model where the proportion of all gene values are increasing logistically with a time varying growth rate. A good approximate model is given by considering the logistic growth rates to be exponentially increasing with time and exponentially decreasing with the position of the gene in the string. This results in a wave equation model with a constant wave speed and thus an order $O(l)$ convergence time complexity.

- *Dimensional analysis of allele-wise mixing:*

  We have identified and shown the importance of mixing analysis to determine when a genetic algorithm will converge to the desired optimum or when it will fail to do so. It is argued that selection and recombination are involved in a race, and when selection goes to fast relative to the recombination or mixing time, then the GA will fail to converge to the global optimum.

  The mixing is studied at two levels: first, at the level of independent bits or alleles, and second, at the building block level. Using dimensional analysis we derived an allele-wise mixing model that accurately predicts where the mixing failure boundary lies in the GA parameter space. These parameters are the population size, the string length, the recombination rate and the selection pressure.

- *Dimensional analysis of building block mixing:*

  The dimensional mixing analysis is also done at the building block level. Here it is shown that - contrary to what one might conclude from the schema theorem - a basic genetic is able to solve fully deceptive functions using uniform crossover, and thus would in principle have no need for tightly linked building blocks. The way to achieve this is by increasing the selection pressure. Unfortunately, the minimal population size needed to mix the building blocks, grows exponentially in the number of building blocks.

  An important conclusion is that the basic genetic algorithm is quite limited in its mixing capabilities. The building blocks thus need

to be tightly linked for mixing purposes and not for letting them grow in the population.

References are Thierens & Goldberg (1993).

- *The elitist recombination genetic algorithm:*

  We have proposed an extremely simple genetic algorithm implementation where the selection and recombination phase are intertwined. The elitist recombination GA holds only a competition between each mating pair and their pair of offspring. It is shown that the selection pressure is the same as with tournament selection. Parents can only be replaced by better offspring and due to this local elitist mechanism, elitist recombination has a number of additional benefits.

  References are Thierens & Goldberg (1994a).

- *Niching and the selection-recombination race:*

  The mixing analysis led us to the identification of the race between selection and recombination in the basic genetic algorithm. We have also studied how niching techniques can prevent this race. A generalized crowding algorithm is used to show that niching is not only useful for multimodal function optimization, but that it also leads to better results for optimizing uniglobal functions with hierarchically composed building blocks, for instance Holland's Royal Road function.

  We have studied to what degree the addition of elitism, niching and restricted mating helps improve the mixing capability of the basic genetic algorithm. Elitism makes sure that good solutions do not get lost unless they can be replaced by better ones. Niching eliminates the race against time that is going on in a basic GA between selection and recombination. Restricted mating ensures that the mixing probability is increased by restricting crossover to strings that are close together in Hamming space.

  Together these extensions allow the building blocks to stably occupy a certain proportion of the population, and this without tight linkage assumptions. The juxtapositioning of all building blocks into one string remains however computationally inefficient.

- *Adaptive crossover:*

  Adaptive crossover is designed to change the crossover exchange probability, such that it becomes tuned to the position of the building blocks in the population. The extensions to the basic genetic algorithm allow the building blocks to maintain a certain foothold in the population. However when they are not linked, the mixing problem remains. Adaptive crossover exploits the fact that a substantial amount of building blocks are present in the population, such that the probability of exchanging entire building blocks becomes significantly higher. As a result, the juxtapositioning of all building blocks into one string now becomes very fast. Analysis and experimental results for fully deceptive function without tightly linked building blocks, and for multimodal fully deceptive function without tight linkage, show that the algorithm is capable of solving these very hard problems.

- *Crossover friendly neural network representation:*

  A last contribution is a non-redundant neural network genotype representation and its relation to crossover efficiency. Due to symmetries, structurally different feedforward neural networks can implement the same functional mapping. These redundancies unnecessary enlarge the search space, and cause problems when recombining two neural networks. We propose a very simple genotype representation that eliminates the functional redundancies. It is shown that crossover transmits more information from the parents to the children when using the non-redundant representation than with the straightforward but redundant coding.

  References are Thierens & Vandewalle (1995).

0.3 Software Environment

All simulation results presented throughout this text are obtained using self-written C code, except for the neural network experiments in Ap-

pendix A where a combination of C and the Matlab Neural Network toolbox is used. We felt that discussing the specific implementations details would unnecessarily distract the reader from the main arguments, and thus we have omitted them. Nevertheless, all algorithms are concisely described such that the interested reader should be able to implement them in a straightforward manner.

The plots in this thesis are generated with *GNUPLOT*, while the text has been typeset with LaTeX.

Chapter 1

Genetic Algorithm Review

In this first chapter we introduce the basic genetic algorithm (GA) and review the most important theoretical concepts of GA theory. A key concept is Holland's schema theorem that gives a lower bound on the expected number of instances schemata will have at the next generation.

The schema theorem leads us to the building block hypothesis that states that short, low-order and highly fit schemata can be recombined to form higher-order schemata and complete strings with very high fitness. Next we discuss three potential problems for the building block hypothesis.

First, the building blocks might mislead the genetic search to what is called deceptive attractors. In Section 1.5 we explain the notion of deception and discuss fully deceptive functions and multimodal fully deceptive functions that are specifically designed as hard test problems for GAs, and that will also be used in this thesis.

A second problem that might occur is that the population size is too small to allow reliable selection of the building blocks. In Section 1.6 the minimal population size needed for reliable building block processing is computed by taking the schema variance into consideration.

Finally, a third potential problem for the building block hypothesis is the need for fast mixing or building block exchange. This has however largely been ignored by the research literature and will be the main topic of Chapter 3.

We conclude this review chapter by summarizing Goldberg's six conditions for a successful application of genetic algorithms.

1.1 Introduction

Nonlinear optimization is a key problem in many situations and consequently a great number of optimization algorithms have been designed. Although this is certainly not the place to give an overview of the vast number of techniques, it is instructive to identify a number of different classes of optimization algorithms. A major distinction between all optimization algorithms is whether they are local or global search algorithms. As their name indicate local optimization algorithms search only locally in the search space. The best known examples of this approach are the gradient based optimization algorithms that use either first order or second order derivative information (eg.: steepest descent, conjugate gradient, quasi-Newton, ...). Gradient based optimization algorithms converge to the nearest local optimum. To find other local optima one simply restarts the algorithm from a different initial point and continues doing this until a satisfactory optimum is found or the available amount of computation is exhausted. When we are dealing with low dimensional problems that do not have too many optima, restarted local optimization is an efficient approach to nonlinear optimization. Unfortunately this technique breaks down when we have a high dimensional complicated function. Due to the curse of dimensionality, local optimization algorithms are no longer feasible.

In order to find good solutions for complicated functions we can no longer let our search algorithm be guided by local information only. Taking a global view on the search space increases the likelihood of finding high valued solutions to the task at hand. In recent years two approaches to global optimization have had quite some success, namely simulated annealing and genetic algorithms. Both are stochastic algorithms derived in analogy with natural phenomena. Simulated annealing is designed according to the cooling process of metals, while genetic algorithms take their inspiration from evolution theory.

To understand how natural evolution can serve as a guideline in designing optimization algorithms, it is necessary to look at some basic

principles from neo-darwinian theory.

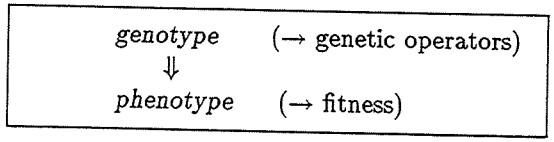
1.2 Evolution as an Optimization Metaphor

Genetic algorithms (GAs) are optimization algorithms based on principles of natural selection and natural genetics. In nature the genetic information of each organism is formed by a double string of four nucleotides A,U,C,G called the DNA. In complex organisms the entire genetic information consists of a different number of strings called the chromosomes and considering them all together we speak of the genotype.

The information stored in the genotype is used to develop the entire organism, also called the phenotype. This developmental or morphological process is extremely complicated but it is of no concern to us here. The only thing that matters is that the information or the blueprint of the organism is represented as a relatively simple syntactical structure - the genotype - and is somehow *decoded* into the organism itself - the phenotype.

The *evaluation* of each organism is performed at the phenotypic level: organisms have to struggle for their existence in a resource limited environment and the less adapted species will simply become extinct. Whereas the selection of well adapted species takes place at the phenotypic level, the search for better organisms takes place at the genotypic level: chromosomes are not perfectly copied to the offspring but are sometimes mutated. With sexual reproduction the genetic material is inherited from both the mother and father due to the crossover operator, so new combinations are tried and subsequently tested at the phenotypic level.

It is important to realize that the search process is merely performing blind syntactic operations at the structurally simple genotypic level, while the actual evaluation is performed at another level: the survival of the fittest phenotype.



To summarize we can identify five essential characteristics of a Darwinian process:

1. **Structures:** the structures are the basic entities that play a fundamental role in the system, yet are simple enough to allow syntactic manipulation (in natural evolution these are the genotypes).

2. **Structures are copied:** the structures need to be multiplied.

3. **Copies partially differ from the original:** the genotypes of the offspring are varied by genetic operators (mutation, crossover, inversion, ...).

4. **Structures are competing for a limited resource:** in nature the amount of food, water and shelter is always limited so organisms have to compete for it.

5. **Relative reproductive success depends on the environment:** whether or not organisms are successful in creating many offspring depends on how well they are adapted to the environment they live in. The same species can be very successful in one place but totally unfit in another.

These principles of Darwinian processes are very general and it is interesting to see that the same underlying mechanisms have been identified with other natural systems. The best known is the immune system where a foreign cell or molecule - called *antigen* - has to be recognized by a specific host defense molecule - called the *antibody* (Burnet, 1959). The problem the immune system faces is how it can generate an antibody for every possible antigen even if it has never encountered them before. For many years this was a highly debated subject, but it is now generally accepted that the immune system is a darwinian process: a large and diverse repertoire of antibodies is always present and whenever an unknown antigen enters the system, the antibodies that most closely match are copied such that their number increases rapidly. This copying process is not merely creating exact replicas but also varies the new antibodies and this continues until the antigen can be recognized well enough to trigger the host defense mechanisms and fight the intruder.

There are also some - albeit more hypothetical - theories that explain certain processes in the brain - especially during child development - as being darwinian (Calvin, 1983; Edelman, 1987).

The examples from nature show that darwinian systems possess some fundamental principles capable of sustaining complex adaptive systems. The primary goal of genetic algorithm research is to study the computational mechanisms underlying these systems and to design algorithms that can be used as problem solving tools.

Some of the advantages of such algorithms are:

- They are blind optimization algorithms, so they most likely are robust and generally applicable.

- They are inherently parallel which makes them very promising viewing the continuously increasing parallel computer power (even simply a local network of workstations) and the serious difficulties in developing parallel optimizations algorithms.

- They are extremely easy to combine with existing local optimization techniques, so we do not have to start from scratch and can benefit from years of experience in those domains.

1.3 The Basic Genetic Algorithm

Genetic algorithms are search algorithms based on the principles of natural evolution. Two fundamental mechanisms form the basis of GAs, namely selection (or reproduction) and variation (or recombination). In nature organisms are always in competition with one another because the environmental resources are limited. The individuals that are best adapted to the environmental constraints are most likely to survive and produce more offspring. This differential rate of creating offspring under environmental pressure is often called the "survival of the fittest" under natural selection. However, selection only does not lead to the development of continuing more adapted organisms. To obtain an evolutionary process we also need a variation mechanism that causes the offspring to be different from its parents. The new individuals that are well adapted to their environment will at their turn create other offspring, while the

lesser fit individuals will only have a low number of descendants. The interplay between selection and variation in a resource limited environment thus results in a profilation of organisms that are increasingly better adapted to that environment.

In their basic form genetic algorithms manipulate a set of fixed length strings. The number of strings in this set or population is held constant and called the population size. Every string is associated with a numerical value or its fitness. The fitness function is a function from the genotype space to the set of real numbers. In practice the genotype are usually binary strings, so the gene values or alleles are either 1 or 0, and the fitness function defines a mapping from l-bit binary strings to the set of real numbers ($f : \{0,1\}^l \rightarrow \mathbb{R}$). The basic execution of a GA is very simple: after an initialization step the algorithm is continuously alternating between the selection and variation step to generate an ongoing sequence of populations.

- initialization
  - create population of n l-bit strings at random
  - compute their respective function value
- successive generations
  - selection: only best strings are considered further
    $\Rightarrow$ exploitation phase
  - recombination: generate new strings with genetic operators
    * crossover
    * mutation
    $\Rightarrow$ exploration phase

Selection step: In the original genetic algorithm design individuals are chosen for reproduction in proportion to their fitness value. This

means that for each string S_i in the population at generation t the chance of being selected is computed as:

$$\frac{f(S_i)}{\sum_{j=1}^{n} f(S_j)}$$

Since the population size is kept constant, the expected number of offspring an individual can contribute to the next generation is:

$$n.\frac{f(S_i)}{\sum_{j=1}^{n} f(S_j)} = \frac{f(S_i)}{\overline{f}(t)}$$

where $\overline{f}(t)$ is the average fitness of the population at time t. Individuals that have a higher than average fitness are expected to have at least one copy in the mating pool, while strings having a lower than average fitness will have in expectation at most one single copy. As a result the next generation will consist primarily of strings that are created from the current above average individuals.

Variation step: To introduce new individuals in the population, genetic operators are applied to the selected parents. These genetic operators are simple syntactic operations on the genotypes. The basic GA uses two genetic operators, namely crossover and mutation.

1. **crossover**
   Crossover takes two strings as input and produces two new strings by exchanging substrings. The one-point crossover operator as defined by Holland (1975) works as follows:

   (a) two parent strings:

   $$p_1 : a_1 a_2 ... a_l$$
   $$p_2 : a'_1 a'_2 ... a'_l$$

   (b) select a crossover point at random:

   c = random(1,2, ..., l-1)

   (c) generate two children by exchanging the two substrings:

   $$c_1 : a_1 ... a_c a'_{c+1} ... a'_l$$
   $$c_2 : a'_1 ... a'_c a_{c+1} ... a_l$$

One-point crossover is a linkage-biased recombination operator, this means that for two gene values or alleles a_i and a_j, the probability that they stay together in the offspring is dependent on their relative position - their linkage - in the string. Two alleles that are close together on the string - thus that are tightly linked - will have a higher probability to stay together in one of the children, than two alleles that are not tightly linked. A common way to express this is that the disruption probability of tightly linked genes is lower than that of loosely coupled genes, when using a linkage-biased crossover operator.

Contrary to one-point crossover, the uniform crossover operator (Syswerda, 1989) is not linkage-biased, and determines for each gene individually whether or not it will be exchanged:

(a) two parent strings:

$$p_1 : a_1 a_2 a_3 a_4 a_5 ... a_l$$
$$p_2 : a'_1 a'_2 a'_3 a'_4 a'_5 ... a'_l$$

(b) generate two children by exchanging each gene with probability p_x :

$$p_1 : a_1 a'_2 a_3 a_4 a'_5 ... a_l$$
$$p_2 : a'_1 a_2 a'_3 a'_4 a_5 ... a'_l$$

Traditional GA theory considers only one-point crossover because it is a low-disruptive recombination operator. Uniform crossover is mostly used with an allele-swapping probability $p_x = 0.5$, and thus is a very disruptive recombination operator. However, it is clear that by lowering p_x we can also lower the disruption probability for the linkage-unbiased uniform crossover. In Chapter 3 we will return to this issue. In the rest of this chapter only one-point crossover is considered.

2. **mutation**
Crossover is capable of generating all possible combinations of alleles, except when a certain allele is completely absent from the population. Alleles that only appear in strings with low fitness are particularly vulnerable. This does not mean however that these

alleles themselves are of no use. It is quite possible that they could be part of a highly fit string if they would only be combined with the right alleles. However, suppose that a particular allele is eliminated from the population before it had the chance to prove its value in the right combination. Now there is no way that the crossover operator can ever discover the successful combination.

The mutation operator guarantees that no allele is permanently lost from a population. It takes as input one string and simply replaces an allele with another one:

$$p : a_1 a_2 ... a_l$$
select a gene at random: $0 < i < l$
$$c : a_1 ... a'_i ... a_l$$

It is obvious that the probability of mutating a particular gene is very small (eg. 0.0001), otherwise the GA would degenerate to a simple random walk. In fact mutation is only an insurance policy against the permanent loss of some alleles. The real power of GAs comes from the selection and crossover operator, as is shown by the schema theorem which is discussed in the next section.

Simple genetic algorithm:

1. CreateRandPop(Pop(0));
2. EvaluatePop(Pop(0));
3. for t=1:max_generation
   a) Select(Pop(t),TmpPop(t));
   b) Crossover(TmpPop(t),Pop(t+1));
   c) Mutate(Pop(t+1));
   d) EvaluatePop(Pop(t+1));

1.4 Schema Theorem

1.4.1 Schemata

To study the search performance of genetic algorithms Holland examined the processing of similarities among binary strings (Holland, 1975).

He therefore defined schemata, which are basically similarity subsets denoted by a ternary string of the same length l as the binary strings or $h = h_1 h_2 \ldots h_l$ with $h_i \in \{0, 1, *\}$. The $*$ symbol is a 'don't care' symbol so for instance the schema $h = 11*0*$ represents the four binary strings $\{11000, 11001, 11100, 11101\}$. Schemata thus have some fixed positions and some positions that can be filled in with any allele value or bit. The number of bits explicitly present in the schema is called the schema order $o(h)$. Another definition is the schema's defining length $\delta(h)$ which is the distance between the two outermost fixed bits or the number of points the schema might be cut by one-point crossover. The schema $h = 110**$ has a defining length $\delta(110**) = 2$ and is of order $o(110**) = 3$.

The purpose of studying GAs at the level of schemata as opposed to the level of individual bits is to focus the attention to interacting, co-adapted sets of bits or genes. Every binary string of length l is a member of 2^l different schemata (each bit can be replaced by a don't care symbol '$*$'), for instance the string 11000 is a member of the schemata 1100*, 110*0, 110**, 11*00, 11***, *****, and so on. In a population of n strings there are between 2^l and $n.2^l$ schemata present.

To see the effect of recombination on schemata let us look at the following one-point crossover operation between the two parents p_1 and p_2 to produce the two children c_1 and c_2. The crosspoint is taken just after the fourth gene:

$$p_1 : 11110000 \Rightarrow c_1 : 11111100$$
$$p_2 : 11001100 \Rightarrow c_2 : 11000000$$

For different schemata the crossover has different outcomes, for instance both parents and both children are instances of the schema 11******. In fact whenever both parents are member of a particular schema then the children will also since crossover does not change the particular bit values. The schema **11**00 has one parent and one child as member. Schema ***10*** is present in one parent but it has no instances under the children, therefore it is said that this particular schema is disrupted by the crossover operation. Finally, schema instances can also be created. For example the schema **1111** is not present in the two parents but the first child is an instance of it.

Schemata have also an associated fitness value, which is simply the average fitness of the member strings:

$$f(h) = \frac{1}{||h||} \sum_{i \in h} f(i)$$

where $||h||$ is the number of strings that are an instance of the schema h. A schema of order $o(h)$ has $l - o(h)$ don't care positions and thus has $2^{l-o(h)}$ member strings. Of course it would take prohibitively much computation time to explicitly evaluate the schemata fitness values. However in a random population of n strings the schemata have in expectation $n/2^{o(h)}$ instances present, thus the low order schemata have enough member strings evaluated to form a reliable estimate of the schema fitness. It is exactly at this level that Holland focuses when analyzing the behavior of genetic algorithms as expressed in the schema theorem.

1.4.2 Schema Theorem

Selection: To understand how genetic algorithms process schemata we first look at the effect of selection only. Consider an arbitrary schema h and call $m(h, t)$ the number of instances or member strings present in the population at generation t. The expected number of instances of schema h after using selection, $E[m(h, t^s)]$, is equal to the sum of the expected number of copies each member string will receive. Note that t^s indicates the intermediate generation at time t, this is after doing selection, but before doing recombination. The probability an individual i will be selected from a population $P(t)$, when using proportional selection is

$$p(i, t) = \frac{f(i)}{\sum_{j \in P(t)} f(j)}$$

so the expected number of copies c an individual i at generation t will get is (with n the population size):

$$\begin{aligned} E[c(i, t)] &= np(i, t) \\ &= \frac{f(i)}{\frac{1}{n} \sum_{j \in P(t)} f(j)} \\ &= \frac{f(i)}{\bar{f}(t)} \end{aligned}$$

where $\bar{f}(t)$ is the mean fitness of the population at generation t. The expected number of instances of schema h $E[m(h,t^s)]$ can now be computed as

$$\begin{aligned} E[m(h,t^s)] &= \sum_{i\in h} E[c(i,t)] \\ &= \sum_{i\in h} \frac{f(i)}{\bar{f}(t)} \\ &= \frac{m(h,t)}{\bar{f}(t)} \frac{\sum_{i\in h} f(i)}{m(h,t)} \\ &= m(h,t)\frac{\bar{f}(h,t)}{\bar{f}(t)} \end{aligned}$$

where $\bar{f}(h,t)$ is the mean fitness of the schema in the population at generation t.

Under the action of selection only, schemata with above-average fitness will receive a rapidly increasing number of instances. This growth of good schemata however would be of no use if the exploration of new strings by recombination would destroy them.

Recombination: The genetic operators *crossover* and *mutation* bring new strings in the population by syntactically changing copies of the better strings found so far. Crossover can effect schemata in a number of different ways:

- If the crossover point falls outside the defining length of a schema, then the number of instances of this schema remains the same. For instance if one parent is an instance of a particular schema then one of the children will also be an instance:

  p_1 : 1111|0000 $\Rightarrow$ c_1 : 1111|1100
  p_2 : 1100|1100 $\Rightarrow$ c_2 : 1100|0000
  h : 1**1|**** $\Rightarrow$ h : 1**1|****

- If the crossover point falls within the defining length of a schema and thus disrupts it, the produced offspring will in general no longer be a member of the schema:

  p_1 : 1111|0000 $\Rightarrow$ c_1 : 1111|1100
  p_2 : 1100|1100 $\Rightarrow$ c_2 : 1100|0000
  h : *11*|*00* $\Rightarrow$ h : *11*|*00*

There are however a number of exceptions:

1. If the parents are both instances of a schema h then the produced offspring will also be members of the schema no matter where the crossover point is located:

   p_1 : 1111|0000 $\Rightarrow$ c_1 : 1111|1100
   p_2 : 1100|1100 $\Rightarrow$ c_2 : 1100|0000
   h : 11**|**00 $\Rightarrow$ h : 11**|**00

2. If one parent is a schema instance and both parents have the same alleles at one side of the crosspoint then one of the children is also a member of the schema:

   p_1 : 1111|0000 $\Rightarrow$ c_1 : 1111|1100
   p_2 : 1100|1100 $\Rightarrow$ c_2 : 1100|0000
   h : 11**|00** $\Rightarrow$ h : 11**|00**

3. Even if none of the parents are a member of a schema h, one of the children can become an instance of it:

   p_1 : 1111|0000 $\Rightarrow$ c_1 : 1111|1100
   p_2 : 1100|1100 $\Rightarrow$ c_2 : 1100|0000
   h : **11|11** $\Rightarrow$ h : **11|11**

The Schema Theorem establishes a lower bound on the expected number of instances each schema h will have when going from one generation t to the next $t+1$. It thereby assumes that whenever a crosspoint falls within the defining length of the schema it will be destroyed. From the discussion above it is clear that this is a conservative estimate: it is not considered that schemata can survive a cut depending on the particular bit values and the possible member gains (case 4) are also not taken into account. The schema survival probability thus becomes

$$p_s = 1 - p_c \cdot \frac{\delta(h)}{l-1}$$

where p_c is the probability that the crossover operator is used.

Combining this with the estimate under selection only, we obtain for the expected number of member strings of each schema h:

$$E[m(h,t+1)] \leq m(h,t)\frac{\bar{f}(h,t)}{\bar{f}(t)}(1-p_c\cdot\frac{\delta(h)}{l-1}) \tag{1.1}$$

The growth or decay of the number of instances of a schema h under selection and crossover thus depends primarily on whether the schema fitness is above or below the population average and whether the schema has a relative short or long defining length.

The effect of mutation is simply to calculate: a schema member will survive mutation when all its alleles at the schema's fixed positions remain unchanged. Call p_m the mutation probability then a single allele survives with a probability $1-p_m$ and an instance of schema h of order $o(h)$ has a survival probability $(1-p_m)^{o(h)} \approx 1-p_m o(h)$. The approximation being very good since p_m is very small.

The schema survival factor of crossover and mutation combined is well approximated with:

$$(1-p_c\cdot\frac{\delta(h)}{l-1})(1-p_m o(h)) \approx 1-p_c\cdot\frac{\delta(h)}{l-1}-p_m o(h)$$

where we have neglected the cross term. Note that we are once again overestimating the destructive effect of the genetic operator since we have not considered the fact that mutation also creates new schema instances.

Putting everything together we get the Fundamental Genetic Algorithm Theorem or Schema Theorem:

$$m(h,t+1) \leq m(h,t)\frac{\bar{f}(h,t)}{\bar{f}(t)}(1-p_c\frac{\delta(h)}{l-1}-p_m o(h)) \tag{1.2}$$

Proportionate selection and one-point crossover are not the only selection and recombination mechanisms. Therefore we write the schema theorem in a more general form. Call $\phi(h,t)$ the reproduction ratio and $\epsilon(h,t)$ the disruption factor of the schema h at population t.

The effect of selection only is then given as:

$$m(h,t+1) = m(h,t)\phi(h,t)$$

and adding the recombination step we get the generalized schema theorem:

$$m(h,t+1) > m(h,t)\phi(h,t)[1-\epsilon(h,t)] \tag{1.3}$$

The net growth factor $\gamma(h,t)$ is defined as

$$\gamma(h,t) = \phi(h,t)[1-\epsilon(h,t)]$$

so schemata with $\gamma(h,t) > 1$ increase in proportion, while schemata with $\gamma(h,t) < 1$ decrease in proportion.

For proportionate selection the reproduction ratio is $\phi(h,t) = \frac{f(h)}{\bar{f}(t)}$ and the disruption factor for one-point crossover and mutation is $\epsilon(h,t) = p_c\frac{\delta(h)}{l-1} - p_m o(h)$.

1.4.3 Optimal allocation of trials

The schema theorem states that above average fitness schemata with short defining length and of low order will receive exponentially increasing trials in subsequent generations. The question that immediately rises is why is this a good thing to do. Holland (1975) approached this problem from the viewpoint of statistical decision theory, more specifically he analyzed the two-armed bandit problem, computed the optimal allocation of trials and related this to the genetic algorithm processing of schemata.

Two-armed bandit problem: In the two-armed bandit problem we have two random variables ξ_1 and ξ_2 with mean and variance μ_1 , σ_1^2 and μ_2 , σ_2^2 respectively. Assume ξ_1 is the random variable with highest payoff or $\mu_1 > \mu_2$. An observer does not know the mean and variances and wants to maximize his cumulative payoff for the N trials he is allowed to do.

Call the observed best random variable after N trials $\hat{\xi}_h(N)$ and the observed second best random variable $\hat{\xi}_l(N)$. Assume n trials are allocated to the observed second best random variable $\hat{\xi}_l(N)$. The remaining $N-n$ trials are allocated to the observed best random variable $\hat{\xi}_h(N)$

There are two sources of loss:

1. A loss can occur when we make the wrong decision because the observed best is really the second best: $\hat{\xi}_h(N) = \xi_2$ or alternatively $\hat{\xi}_l(N) = \xi_1$. Call the probability this occurs q(N,n). $N - n$ trials are given to the random variable ξ_2 instead of to the best random variable ξ_1. The cumulative loss due to this decision error is $(N - n).|\mu_1 - \mu_2|$.

2. Another source of loss happens when we make the right decision, i.e.: $\hat{\xi}_h(N) = \xi_1$ or $\hat{\xi}_l(N) = \xi_2$. The probability this happens is $1 - q(N, n)$ and the loss is now caused by the trials spent to the worst random variable in the exploration phase or $n.|\mu_1 - \mu_2|$.

The total expected cumulative loss for any allocation of n trials to $\hat{\xi}_l(N)$ and $N - n$ trials to $\hat{\xi}_h(N)$ is:

$$L(N, n) = (q(N, n)(N - n) + (1 - q(N, n))n)|\mu_1 - \mu_2|$$

Optimal number of trials: The optimal number of trials $n_{opt}(N)$ that minimizes the loss $L(N, n)$ can be computed from $\frac{dL(N,n)}{dn}$ if we know $q(N, n)$ in function of n.

The estimated payoff is based on the sum of the observed payoffs during the trials, so Holland (1975) used the central limit theorem to compute $q(N, n)$ and after a number of simplifying approximations he obtained for the optimal number of trials to the observed worst arm in the 2-armed bandit problem:

$$n_{opt}(N) \approx b^2 \ln \left(\frac{N^2}{8\pi b^4 \ln N^2} \right)$$

with $b = \sigma_1/(\mu_1 - \mu_2)$ or after rearranging

$$N - n_{opt}(N) = \sqrt{8\pi b^4 \ln N^2} e^{n_{opt}(N)/(2b^2)}$$

Thus, in order to determine which of the two random variables in the two-armed bandit problem has a higher expected value while keeping the payoff losses to a minimum, the number of trials allocated to the random variable with the highest observed value has to increase exponentially relative to the number of trials allocated to the observed worst.

Based on this analysis and on the schema theorem that states that genetic algorithms allocate exponentially increasing samples to the observed best schemata, Holland concludes that genetic algorithms are a near optimal allocation procedure.

Holland also extended the analysis to the multi-armed bandit problem where k arms are competing. The conclusion is similar, namely to minimize the expected loss one should allocate an exponentially increasing number of trials to the observed best of the k arms.

In (Goldberg, 1989) the $k-$armed bandit problem is related to GA schema processing by defining schema partitions. A schema partition is the set of all schemata that have the same string positions as fixed alleles and consequently the same string positions as don't care symbols.

Schema partitions are basically binary strings $g = g_1 \ldots g_l$ with $g_i \in \{f, *\}$ where f indicates the fixed position of the schemata. The order and defining length of a schema partition is the same as the order and defining length of its schemata. For instance the schema partition $***ff*f*$ represents the 8 schemata:

***00*0*
***00*1*
***01*0*
***01*1*
***10*0*
***10*1*
***11*0*
***11*1*

Selection causes the population to converge which reduces the population diversity and ultimately only one string remains. The schemata in a schema partition are therefore in competition with each other to have member strings present in the population.

Viewing the schema partition as a $k-$armed bandit problem, the bandit analysis prescribes that the observed best schema within a partition should receive an exponentially increasing number of instances and this is basically what the schema theorem expresses.

For a string of length l there are 2^l different schema partitions that are all sampled and processed by the GA simultaneously. Naturally, not all of them are sampled sufficiently well, and some of them are

highly disrupted by the genetic operators. The low order, short defining length schema partitions however do have a significant number of member strings present in the population and their disruption probability is low. Thus by evaluating a population of size n during each generation, the GA is simultaneously processing a large number of schema partitions, a phenomenon called the implicit parallelism of genetic algorithms.

1.4.4 Building block hypothesis

The discussion on the schema theorem can be summarized as follows:

- low order, short, above average schemata (= building blocks) receive exponentially increasing trials in successive generations
- according to the k-armed bandit analogy this strategy is near optimal
- the building blocks processing happens in an implicit parallel way, so the evaluation of n strings in each generation gives us information on a much greater number of schema partitions.

When using the GA as an optimization algorithm the schema theorem analysis also makes clear on what kind of functions the GA is expected to perform well. Whenever the problem and the associated string coding is such that the building blocks can be recombined to form optimal or near optimal solutions, the genetic algorithm will give good results. This has been called the *building block hypothesis*. Viewing the generality of the assumptions made and the successful use of genetic algorithms on a variety of problems, it seems that the notion of sampling, recombining, and resampling building blocks to construct better and better solutions is a promising approach.

It should be emphasized that any search algorithm has to make some assumptions in order to be more efficient than pure random or enumerative search. The more specific these assumptions are, the more efficient the algorithm can be, but there is a trade off between specificity and applicability.

By making some fairly general assumptions, genetic algorithms can be widely applied and promise to find good solutions without demanding

extensive detailed domain knowledge. In addition it is straightforward to combine them with more specific search algorithms that have almost always a local view on the search space and the GA's global view often leads to hybrid algorithms with superior performance (Davis, 1991).

There are however at least three issues that might cause some concern about the building block hypothesis:

First, the low order schemata might give misleading information. Genetic algorithms increasingly sample the most highly fit low order schemata, but what if these lead away from the optimal solution ? To study this and to push the GA to its limits, deceptive functions are designed that are maximally misleading up to a certain order. In the next section we will discuss the deception problem and specify the class of fully deceptive trap functions and the multimodal, fully deceptive trap functions which will be used as ultimate test cases in this thesis.

A second potential problem with the schema theorem and the building block hypothesis is the fact that the GA relies on sampling information about the schemata. Since this is basically a statistical decision process it is important to consider the effect of noise . Section 1.6 talks about the schema variance and how this can be used to compute an estimate of the population size needed to allow reliable decision making.

Finally, a third aspect is the mixing problem. The schema theorem talks about increasing the number of instances of short, highly fit schemata, this is about the growth of the building blocks, and the building block hypothesis assumes that this will lead to good solutions. There is however an often unrealized implicit assumption in this, namely it should be possible to juxtapose the growing building blocks and this has to happen fast enough or else building blocks will start to cross-compete against each other, and some will be driven out of the population leading to premature convergence of the population. This mixing problem (Goldberg, Deb & Thierens, 1992; Thierens & Goldberg, 1993) has not received much attention yet and it will be the subject of Chapter 3.

1.5 Deception

1.5.1 Deceptive functions

The best way to understand a search algorithm is to create problems that mislead the algorithm and try to capture what it is that makes a problem hard or easy. Once this is done one can think of extending the algorithm so that it can solve the hard problems too.

In this section we will discuss the notion of deception and discuss two classes of functions that will be used throughout this text as test functions. First, we describe the fully deceptive trap function, and then extend this to the multimodal fully deceptive function.

The building block hypothesis assumes that short, low-order highly fit schemata lead to optimal or near optimal solutions. Let us assume that we have a problem where short, low-order highly fit schemata lead away from the optimal building block. Genetic algorithms concentrate on the most highly fit schemata by population sampling and redistribution, thereby decreasing the number of competing schemata. But what if the most highly fit schemata are instances of strings that are very different from the optimal solutions ? In (Goldberg, 1987) *deceptive* problems were introduced that are misleading in an average sense.

As an example let us consider the following order-3 schema partition and the relative schema fitness values:

$$\begin{aligned}
f(...0**...) &> f(...1**...)\\
f(...*0*...) &> f(...*1*...)\\
f(...**0...) &> f(...**1...)\\
f(...00*...) &> f(...11*...)\\
f(...0*0...) &> f(...1*1...)\\
f(...*00...) &> f(...*11...)\\
f(...000...) &< f(...111...)
\end{aligned}$$

Although the schema ...111... has a higher fitness than the schema ...000..., all the lower order schemata indicate just the opposite. Schemata where all the lower order schemata give misleading information are called *fully deceptive*. Since there are no lower order schemata that give reliable information about the optimal higher order schema, the GA has to process this higher order schema by itself, that is as one unit. A necessary

condition for this to happen is that the schema disruption factor is low. Recall that for one-point crossover the disruption factor is (neglecting the effect of mutation):

$$\epsilon = p_c \frac{\delta(h)}{l-1}$$

with p_c the probability of applying crossover to the parents, $\delta(h)$ the schema's defining length and l the string length. Schemata that are tightly linked have a low defining length and the disruption probability is thus also low. Schemata with high defining length however are very often disrupted and cannot be processed as one unit. For instance if the following schema partition is fully deceptive, then this would be no problem in the first case, but for the second case the GA would be misled because it cannot process this order-3 schema as one unit since crossover always disrupts this schema:

$$\begin{aligned}
\text{*****ff*f******} &: \delta(h) = 3\\
\text{f*****f*******f} &: \delta(h) = 14
\end{aligned}$$

Thus a basic genetic algorithm can only process deceptive schema partitions correctly when they are tightly linked. It is important to realize that no fitness function is inherently deceptive: it is the combination of string coding and fitness function that can make a certain problem deceptive. This is easy to see when we realize that any binary coded fitness function can be recoded as a simple bit counting problem. One simply has to sort all solutions according to fitness and allocate a number of bits '1' according to their rank. This way the best solution is simply the string with most bits '1'. Unfortunately finding such a code in general would be computationally more demanding than the optimization process itself so this is of little practical use. This also means that there is no specific coding in general superior to another. Discussions for instance whether gray coding is better than binary integer coding has no meaning unless one restricts the discussion to certain classes of functions and shows that the structure present in this class of functions is beneficial for a certain coding.

Fully deceptive trap function: To allow flexible experimentation with deception Deb and Goldberg defined a parameterized set of fully deceptive trap functions (Deb & Goldberg, 1991; Deb & Goldberg, 1992)

which will also be used in this work. A trap function is a function of unitation (= the function value is determined by the number of 1-bits) with one global optimum and one deceptive optimum located maximally far from the global optimum. It is defined as follows:

$$g(u) = \begin{cases} 1 & \text{if } u = k \\ (1-d)(1-\frac{u}{k-1}) & \text{if } u < k \end{cases}$$

where u is the number of 1-bits, k the string length, and d the signal, which is the fitness difference between the global and the deceptive optimum, A sufficient condition to be fully deceptive is:

$$d < \frac{k-2}{2k-3} \quad \text{for } k \geq 3$$

To obtain a function of arbitrary difficulty we concatenate a number of these deceptive subfunctions together on one string. The parameters to be chosen are thus the deception length k, the signal d, and the number of building blocks m concatenated in one string of length $l = km$.

Figure 1.1 and Table 1.1 show an example of a deceptive subfunction with deceptive length $k = 5$ and signal $d = 0.2$. Table 1.2 shows the schema fitness values: all schemata of order less than 5 are misleading.

unitation u	0	1	2	3	4	5
function f(u)	0.8	0.6	0.4	0.2	0.0	1.0

Table 1.1: *Fitness values in function of the number of bits '1' for the fully deceptive trap function of length $k = 5$ with fitness signal $d = 0.2$*

1.5.2 Multimodal deceptive functions

The fully deceptive trap function has one global optimum. Every subfunction has two optima thus a function with $m = 10$ building blocks has a total of $2^{10} = 1024$ local optima of which only one is the global optimum. By folding a fully deceptive trap function it is possible to create a multimodal, fully deceptive function with two global optima

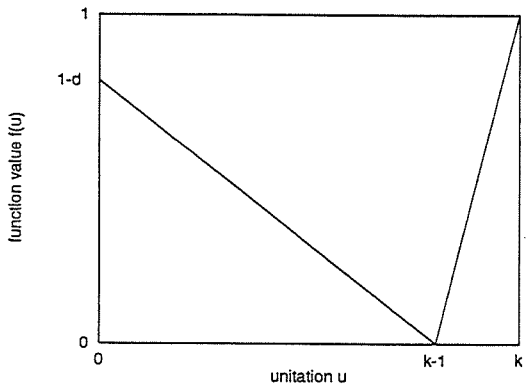


Figure 1.1: *A fully deceptive trap function with deception length k and fitness signal d. The unitation variable u is the number of 1-bits.*

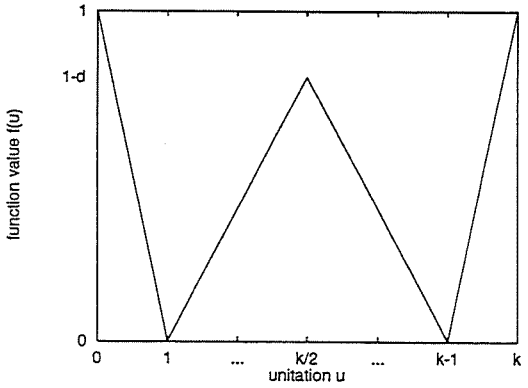


Figure 1.2: *A multimodal, fully deceptive trap function constructed by folding a uniglobal fully deceptive trap function with deception length $k/2$ and fitness signal d.*

schemata	*fitness*	*schemata*	*fitness*
0****	0.4	1****	0.275
00***	0.5	11***	0.25
000**	0.6	111**	0.3
0000*	0.7	1111*	0.5
00000	0.8	11111	1.0

Table 1.2: *Schemata fitness values for the fully deceptive trap function of length $k = 5$ and fitness signal $d = 0.2$*

unitation u	0	1	2	3	4	5	6
function f(u)	1.0	0.0	0.4	0.8	0.4	0.0	1.0

Table 1.3: *Fitness values in function of the number of bits '1' for the multimodal, fully deceptive trap function of length $k = 6$ with fitness signal $d = 0.2$*

and $\binom{k}{k/2}$ local and deceptive optima. Figure 1.2 shows how this folding looks like and Table 1.3 shows the fitness values for a $k = 6$ multimodal subfunction with signal $d = 0.2$. Note that these are very hard problems: concatenating for instance $m = 5$ of these 6-bit subfunctions we obtain a function with string length $l = 30$ that has $(\binom{6}{3} + 2)^5 = 22^5 = 5.153.632$ optima of which $2^5 = 32$ are global.

1.6 Schema Variance

1.6.1 Collateral noise

The schema theorem states that GAs guide their search by implicitly evaluating the fitness of a great amount of short, low-order schemata. The highly fit schemata receive increasingly more samples and are recombined such that - according to the building block hypothesis - better, higher-order schemata and finally near optimal strings are formed.

This implicit fitness evaluation is obtained through the sampling of

the schemata by the successive populations. Since we are basing our decisions on sampling information, it is very important to look at the noise in this process. In (Goldberg & Rudnick, 1991; Goldberg, Deb & Clark, 1992) the variance of schema fitness is identified as the major source of noise in properly evaluating the schema fitness and it is shown how this can be applied to derive a population-sizing equation. When the population size is large enough the genetic algorithm can reliably discriminate between good and bad schemata and the building blocks can be selected properly and grow in number.

The problem of choosing between competing building blocks thus becomes a basic statistical decision problem. For a particular schema the variance in estimating the mean fitness value is due to the fitness contribution of the bits that fill in the don't care positions of the schema. Since the fitness contributions of the other genes are acting as noise for the proper determination of a schema's fitness this variance has been called the *collateral noise*. In a random population of size n we expect to have $\frac{n}{2^{o(h)}}$ samples of a schema with order $o(h)$ on a total of $2^{l-o(h)}$ member strings. It is obvious that for any practical population size n and string length l only a small proportion of all possible member strings of the schema will be present in the population. According to the central limit theorem the observed schema fitness $\hat{f}(h)$ — i.e. the sampling distribution of the fitness mean — is normally distributed with mean $f(h)$ and variance $\frac{\sigma^2(h)}{n(h)}$ with $n(h)$ the number of schema member strings present in the population.

1.6.2 Population sizing

In (Goldberg, Deb & Clark, 1992) the schema variance is used to obtain an estimate of the necessary population size n such that there are enough samples of competing schemata to let the GA make a reliable decision from the initial population. Assume we have two competing schemata h_1 and h_2 with fitness values $f(h_1)$ and $f(h_2)$ such that $f(h_1) > f(h_2)$. The GA can only make decisions based on the observed fitness values $\hat{f}(h_1)$ and $\hat{f}(h_2)$ and from a statistically decision point of view a selection error will occur whenever the observed schema fitness of the best schema $\hat{f}(h_1)$ is lower than the observed fitness of the worst schema $\hat{f}(h_2)$, or $\hat{f}(h_1) < \hat{f}(h_2)$.

Given enough samples, the observed fitness values are normally distributed with mean $f(h)$ and variance $\frac{\sigma^2(h)}{n(h)}$ as guaranteed by the central limit theorem. A decision error can only take place when the two distributions are overlapping. The probability that such an error takes place can be computed by taking the difference of the two normal distributions which is also a normal distribution with mean the difference of the two individual means $(f(h_1) - f(h_2)$ and with variance the sum of the two individual variances $(\frac{\sigma^2_{h_1}}{m(h_1)} + \frac{\sigma^2_{h_2}}{m(h_2)})$, or

$$z^2(\alpha) = \frac{(f(h_1) - f(h_2))^2}{\frac{\sigma^2_{h_1}}{m(h_1)} + \frac{\sigma^2_{h_2}}{m(h_2)}}$$

with $z(\alpha)$ the ordinate of the unit, one-sided normal deviate and α the probability that $\hat{f}(h_1) < \hat{f}(h_2)$. For low values of α we can use the Gaussian tail approximation: $\alpha = \frac{\exp^{-z^2/2}}{(z\sqrt{2\pi})}$.

The above analysis leads to a population sizing equation. For an order-k schemata the number of instances in the initial random population is $m(h) = \frac{n}{2^k}$. Call the fitness difference or signal $d = f(h_1) - f(h_2)$ and the mean variance $\sigma^2_M = \frac{\sigma^2_{h_1} + \sigma^2_{h_2}}{2}$ then we obtain:

$$z^2(\alpha) = \frac{d^2}{\frac{2^k}{n} 2\sigma^2_M}$$

or

$$n = 2z^2(\alpha)2^k \left(\frac{\sigma_M}{d}\right)^2 \tag{1.4}$$

As an example let us calculate the population size needed for the fully deceptive trap function from the previous section with deception length $k = 5$, fitness signal $d = 0.2$ and $m = 10$ concatenated building blocks. The optimal string is thus the 50 bit long string with all bits '1' and has a fitness value of 10, while the deceptive attractor consists of all bits '0' and has a function value of $10.(1 - 0.2) = 8$.

The mean and variance of one subfunction is $\mu = 0.34$ and $\sigma^2_{rms} = 0.058$ respectively. The collateral noise for a subfunction is thus $\sigma^2_M =$

$(m-1)\sigma^2_{rms} = 0.527$. and the population size becomes:

$$\begin{aligned} n &= 2z^2(\alpha)2^5\frac{0.527}{0.2^2} && (1.5)\\ &= 843z^2(\alpha) && (1.6)\\ &\approx 1000 && (1.7) \end{aligned}$$

So for a population size $n = 1000$ the GA will reliably discriminate between the globally optimal and deceptive building blocks and converge to the overall global optimum.

1.7 Conclusion: Conditions for GA Optimization

An excellent summary of the key concepts of genetic algorithm theory is given by Goldberg's six conditions for reliable GA optimization (Goldberg, Deb & Clark, 1992; Goldberg, 1994):

1. Know what the GA is processing: building blocks.
2. Ensure there is an adequate initial supply of building blocks.
3. Ensure the growth of necessary building blocks.
4. Ensure the mixing of necessary building blocks.
5. Ensure that building blocks decisions are well made.
6. Solve problems that are of bounded building block difficulty.

The first condition to design effective GAs is to realize what is actually propagated from generation to generation. Holland (1975) identified building blocks as the quasi-invariant structures: highly fit similarities that are only marginally disrupted by the genetic operators when exploring new solutions. Combining this low disruption probability with the copying process of selection immediately leads us to the third condition that requires the necessary building block growth. Of course before building blocks can be copied and occasionally varied, they first have to be present - as stated by the second condition - so the population size

has to be large enough such that there is a sufficient number of them in a randomly initialized population.

Condition four emphasizes the fact that it is not enough that building blocks grow but they also have to be juxtaposed or mixed. This mixing problem is largely ignored in the GA literature and it will be the topic of Chapter 3. The fifth condition refers to the statistical decision problem one faces when deciding between competing schemata. The uncertainty due to collateral noise and the resulting population sizing equation was discussed in Section 1.6. Finally, the last condition states that we can only expect to solve problems reliably if the necessary building blocks are of low order. Assume we have a fully deceptive problem with deception length $k = 20$, thus the optimal building blocks have a length of 20 bits. Since all lower order schema partitions are misleading such a problem cannot be solved to optimality unless we use an extremely large population size of the order 2^{20}. Note however that this is basically a needle in a haystack problem so any search algorithm would do no better than pure random search.

Chapter 2

Bounds On GAs Convergence Time Complexity

In this chapter we analyze the dynamical properties of several genetic algorithm implementations by building convergence models for functions with independent gene fitness contributions.

In Section 2.2 we build a lower bound convergence model by using normal distribution analysis for the bit counting function which has equal gene fitness contributions. The model and experimental results show that for ranked based genetic algorithms - such as tournament selection - the number of generations towards population convergence is of order $O(\sqrt{\ell})$, with ℓ the length of the binary string.

Section 2.3 shows how the covariance build-up between the bit values, slows down the population convergence speed. This covariance is caused by the selection phase, and is only partly decorrelated by recombination.

In Section 2.4 we build an upper bound convergence model based on the domino convergence analysis for functions with exponentially scaled gene fitness contributions. The model and experimental results show that when using tournament selection, the number of generations needed for the population to converge is of order $O(\ell)$.

2.1 Introduction

Holland's original work on genetic algorithms made exclusive use of proportionate selection as reproduction procedure (Holland, 1975). When implementing the survival of the fittest principle there is an intuitive appeal to determine the number of offspring for each individual to be direct proportional to the fitness value of that individual compared to the average fitness of the entire population.

Unfortunately there are two basic problems with this approach:

- First, there is a considerable risk of premature convergence: suppose that during the initial generations we are 'lucky' and generate an individual which is significantly better than all other individuals in the current population. The number of descendants of this individual will increase very rapidly and take over a large proportion of the population in just a few generations, thereby inhibiting the GA to sample the search space thoroughly.

- A second problem is the lack of selection pressure when all individuals have a fitness of roughly the same value. This will most likely occur at latter generations when the population starts to converge and most individuals become similar. To see the problem let us assume there are only two individuals to choose from with respectively a fitness value of 99 and 101. The average fitness is thus 100 and the expected number of copies of the two individuals is 0.99 and 1.01 respectively. Clearly both individuals will receive with very high probability one copy, so there is almost no selective pressure favoring the best.

As a result of the problems with pure proportionate selection, a very diverse set of selection procedures has been proposed and used in different applications, ranging from proportionate selection with a scaling mechanism to rank based selection algorithms (Goldberg, 1989).

In (Goldberg & Deb, 1991) a comparative analysis of different selection schemes is given by modeling the change in proportion of the individual strings in the population, when using only selection. The strings are thus never recombined, and no new fitness values are generated during the entire convergence process. Convergence time is characterized

with the takeover time which is the number of generations needed for the best individual present in the initial population, to fill the population with copies of itself, and thus to take over the entire population.

To get an integrated selection recombination analysis we take an alternative view and we model different selection schemes when optimizing a normal distributed fitness function. Recombination is done with uniform crossover. The use of a linkage biased crossover operator, such as one-point crossover, will also be discussed. Restricting ourselves to normal distributed fitness functions allows simple yet accurate modeling of the convergence characteristics of selection and recombination together. This kind of analysis is also being used in quantitative genetics (Bulmer, 1985; Falconer, 1989) and used for some GA implementations by (Qi & Palmieri, 1994a; Qi & Palmieri, 1994b) and (Mühlenbein & Schlierkamp-Voosen, 1993).

2.2 A Lower Bound Through Normal Distribution Modeling

2.2.1 Introduction

To study the behavior of different selection schemes on a normal distributed fitness function, we simply take the number of bits '1' in the binary string as fitness value. The bit counting function is binomial distributed and is a good approximation of the normal distribution if the population size n and the string length l are not too small.

Call $p(t)$ the probability of having a 1-bit at a certain position in the string at generation t, then the fitness probability function is given by:

$$Pr[f = j] = \binom{l}{j} p^j(t)(1 - p(t))^{(l-j)}$$

The population mean fitness at generation t is then $\overline{f}(t) = l\ p(t)$ and the variance $\sigma^2(t) = l\ p(t)(1 - p(t))$.

To analyze the convergence characteristics the models try to predict the proportion $p(t)$ in function of the number of generations t. We will discuss a number of different selection schemes. First we will look at the unscaled fitness proportional selection since this was originally proposed

by Holland (1975) and continues to be used in some theoretical analyses of GA behavior. Next we will build a convergence model for tournament selection. Finally we will discuss results on block or truncation selection.

2.2.2 Proportionate Selection

Proportionate selection is certainly the most basic selection scheme in evolutionary computation as it implements the idea that reproduction rates should be proportional to the fitness value f (Holland, 1975; Goldberg, 1989). The probability $P_i(t^s)$ of selecting an individual string i with fitness f_i and a proportion $P_i(t)$ of the population at generation t is given by:

$$P_i(t^s) = P_i(t)\frac{f_i}{\overline{f(t)}}$$

with $\overline{f(t)}$ the mean fitness of the population at generation t.

The increment in population mean fitness after selection can easily be computed:

$$\begin{aligned}\overline{f(t^s)} - \overline{f(t)} &= \sum_{i=1}^{n} P_i(t^s) f_i - \overline{f(t)} \\ &= \sum_{i=1}^{n} \frac{P_i(t) f_i^2}{\overline{f(t)}} - \overline{f(t)} \\ &= \frac{1}{\overline{f(t)}}(\overline{f^2(t)} - \overline{f(t)}^2) \\ &= \frac{\sigma^2(t)}{\overline{f(t)}}\end{aligned}$$

Since the gene fitness contributions are independent of each other, the population mean fitness does not change after recombination, this is $\overline{f(t+1)} = \overline{f(t^s)}$. The increment of the population average fitness with proportionate selection is thus proportional to the fitness variance and inverse proportional to the mean fitness.

$$\overline{f(t+1)} - \overline{f(t)} = \frac{\sigma^2(t)}{\overline{f(t)}} \tag{2.1}$$

Since $\overline{f(t)} = l\ p(t)$ and $\sigma^2(t) = lp(t)(1-p(t))$ the increase of the proportion of optimal alleles p(t), is given by

$$l(p(t+1) - p(t)) = \frac{lp(t)(1-p(t))}{lp(t)} = 1 - p(t)$$

Approximating the difference equation with the corresponding differential equation we obtain a simple convergence model expressing the proportion $p(t)$ in function of the number of generations t

$$\frac{dp(t)}{dt} = \frac{1}{l}(1 - p(t))$$

which has as solution

$$p(t) = 1 - (1 - p(0))\ e^{-t/l}$$

Starting from a random initial population $p(0) = 0.5$ the convergence model becomes:

$$\boxed{p(t) = 1 - 0.5e^{-t/l}} \tag{2.2}$$

To calculate the convergence speed we compute the number of generations g_{conv} it takes to let the proportion $p(t)$ come arbitrarily close to 1 or $p(g_{conv}) = 1 - \delta$. Note that we need to take a δ-value because the exponential function $e^{-t/l}$ only goes asymptotically to zero.

$$g_{conv} = -\ell \ln 2\delta$$

For $\delta = 1/2l$ the convergence time is:

$$g_{conv} = \ell \ln \ell \tag{2.3}$$

In Figure 2.1 we have plotted the model for a string length $l = 100$ and compared it with experimental results when implementing proportionate selection with stochastic universal sampling (Baker, 1987).

The experimental results are obtained with string length $l = 100$ and population size $n = 200$. A population size $n = 2l$ allows reliable

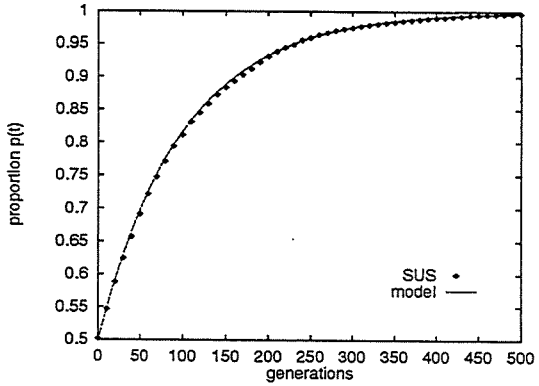


Figure 2.1: *Convergence model and experimental results of the proportion of optimal alleles when optimizing the Bit Counting function with proportionate selection (Stochastic Universal Sampling) and uniform crossover.*

decision making for the Bit Counting function as we have explained in the first chapter when discussing the fitness variance and population sizing (Goldberg, Deb & Clark, 1992). The experimental results shown are the average results of 25 independent runs starting from a different, randomly initialized population. The standard deviation is extremely small, so we have not plotted it.

Clearly the model predicts the experimental results very well. The rate of convergence with proportionate selection is extremely slow and it drastically slows down when approaching the solution. In order to obtain better convergence characteristics for proportionate selection one has to use some form of fitness scaling (Goldberg, 1989). Instead of scaling the fitness function explicitly it is also possible to use a selection scheme that does not look at the absolute function value but instead performs selection according to the number of individuals in the population that are better or worse. Examples of this rank based selection principle are tournament selection and truncation selection which we will discuss next.

2.2.3 Tournament Selection

Tournament selection randomly chooses a set of individuals and picks out the best for reproduction. The number of individuals in the set is mostly equal to two but larger tournament sizes can be used in order to increase the selection pressure. First we consider the case of optimizing the bit counting function with a tournament size $s = 2$. Under the assumption of a normally distributed function the fitness difference between two randomly sampled individuals in each tournament is also normally distributed with mean $\mu_{\Delta f}(t) = 0$ and variance $\sigma^2_{\Delta f}(t) = 2\sigma^2(t)$. Since we are only selecting the best of the two competing individuals we are actually only looking at the absolute value of the fitness difference. The average fitness difference between two randomly sampled individuals is thus given by the mean value of those differences that are greater than $\mu_{\Delta f} = 0$, which is equivalent to the mean value of one half of the normal distribution truncated at its mean value 0. The mean value of the right half of a standard normal distribution is given by $\sqrt{2/\pi} = 0.79...$.

Tournament selection selects the best out of every random pair of individuals so the population average fitness increase from one generation to the next is equal to half the mean value of the difference between two randomly sampled individuals:

$$\overline{f(t+1)} - \overline{f(t)} = \frac{1}{2}\, 0.7979\, \sigma_{\Delta f}(t) = \frac{1}{2}\, 0.7979\, \sqrt{2}\, \sigma(t) = \frac{1}{\sqrt{\pi}}\, \sigma(t)$$

or

$$\overline{f(t+1)} - \overline{f(t)} = 0.5642\, \sigma(t) \tag{2.4}$$

Since $f(t) = l\, p(t)$ and $\sigma^2(t) = l\, p(t)\, (1 - p(t))$ we get

$$p(t+1) - p(t) = \sqrt{\frac{p(t)(1-p(t))}{\pi l}}\ .$$

Approximating the difference equation with the differential equation

$$\frac{dp(t)}{dt} = \sqrt{\frac{p(t)(1-p(t))}{\pi l}}$$

gives us the solution:

$$p(t) = 0.5(1 + \sin(\frac{t}{\sqrt{\pi l}} + \arcsin(2p(0) - 1))) \ .$$

For a randomly initialized population $p(0) = 0.5$ the convergence model becomes:

$$\boxed{p(t) = 0.5(1 + \sin(\frac{t}{\sqrt{\pi l}}))} \tag{2.5}$$

The number of generations g_{conv} it takes to let the population converge is obtained by putting $p(g_{conv}) = 1$.

$$g_{conv} = \frac{\pi}{2}\sqrt{\pi l} \tag{2.6}$$

Note that we do not need a δ-value anymore as with proportionate selection, because the *sin*-function is not asymptotically converging. The convergence time complexity for tournament selection is thus $O(\sqrt{\ell})$ which compares very favorable to $O(\ell \ln \ell)$ for unscaled proportionate selection.

The proposed convergence model is compared with experimental results in Figure 2.2. The results shown are again averaged over 25 runs, and the standard deviation is again extremely small. The model slightly overestimates the proportion of optimal alleles for tournament selection and recombination with uniform crossover. In a second experiment we recombine the population twice at each generation, so after the usual procedure of tournament selection followed by recombination we randomly shuffle the population and again recombine the individuals (note that this does not increase the number of function evaluations since we are now evaluating the children only after the second recombination phase). By doing this the alleles of each individual are more decorrelated and the assumptions we made when building the convergence model are less violated. Figure 2.2 shows that the model now coincides very well with the experimental results, so the slight overestimation is due to the build-up of covariances between the alleles caused by selection. Due to the selection of the better strings in a population, gene values or alleles

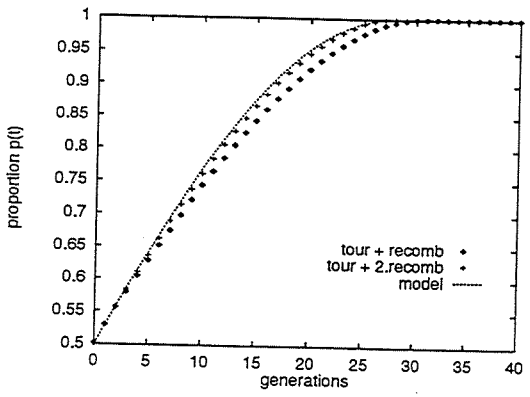


Figure 2.2: *Convergence model and experimental results of the proportion of optimal alleles when optimizing the Bit Counting function with tournament selection and uniform crossover.*

become correlated. Crossover only partly decorrelates them again, so the net effect is a build-up of covariance between the alleles. In Section 2.3 we will analyze this covariance build-up and its effects on the convergence time.

2.2.4 Truncation Selection

Truncation selection or block selection ranks all individuals according to their fitness and selects the best ones as parents. In truncation selection a threshold T is defined such that the $T\%$ best individuals are selected. This selection mechanism is often used in quantitative genetics where artificial selection performed by breeders is studied. In (Mühlenbein & Schlierkamp-Voosen, 1993) a specific genetic algorithm - the Breeder Genetic Algorithm - is proposed that incorporates ideas from breeders to perform parameter optimization tasks.

Block selection is equivalent to truncation selection since for a given population size n one simply gives s copies to the n/s best individuals. Both implementations are identical when $s = 100/T$.

If the fitness function is normally distributed, quantitative genetics defines the *selection intensity* i that expresses the selection differential

$S(t) = \overline{f(t^s)} - \overline{f(t)}$ in function of the standard deviation $\sigma(t)$:

$$S(t) = i\,\sigma(t)$$

A convergence model for truncation or block selection has been computed in (Mühlenbein & Schlierkamp-Voosen, 1993):

$$\overline{f(t+1)} - \overline{f(t)} = i\,\sigma(t) \tag{2.7}$$

Since $f(t) = l\,p(t)$ and $\sigma^2(t) = l\,p(t)\,(1-p(t))$ we get

$$p(t+1) - p(t) = \frac{i}{\sqrt{l}}\sqrt{p(t)(1-p(t))}.$$

Approximating the difference equation with the differential equation

$$\frac{dp(t)}{dt} = \frac{i}{\sqrt{l}}\sqrt{p(t)(1-p(t))}$$

the solution becomes

$$p(t) = 0.5(1 + \sin(\frac{i}{\sqrt{l}}t + \arcsin(2p(0)-1))).$$

For a randomly initialized population $p(0) = 0.5$, the solution becomes:

$$\boxed{p(t) = 0.5(1 + \sin(\tfrac{i}{\sqrt{l}}t))} \tag{2.8}$$

Calculating the number of generations g_{conv} to convergence ($p(g_{conv}) = 1$) is straightforward:

$$g_{conv} = \frac{\pi}{2}\cdot\frac{\sqrt{l}}{i} \tag{2.9}$$

It is interesting to see that the convergence models for tournament selection and truncation selection have the same functional form. The only difference is the magnitude of the selection intensity constant i, which can be changed by increasing the truncation threshold or tournament size of the respective algorithms.

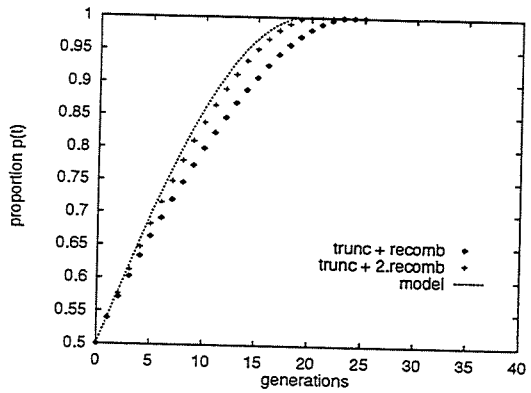


Figure 2.3: *Convergence model and experimental results of the proportion of optimal alleles when optimizing the Bit Counting function with truncation selection and uniform crossover.*

Experimental results are shown in Figure 2.3 for a block size $s = 2$ or equivalently a truncation threshold $T = 50\%$, which gives us a selection intensity $i = 0.8$ (Falconer, 1989). Again the model slightly overestimates the proportion $p(t)$ of optimal alleles and predictions become more accurate when the alleles are decorrelated by repeating the recombination phase.

2.2.5 Conclusion

Simple yet accurate convergence models for different selection schemes are derived. The models assume a normally distributed fitness function and predict the proportion of optimal alleles in function of the number of generations. Experimental results for optimizing the bit counting function show good agreement with the predictions. The experiments also suggest that the slight deviations are due to the build-up of covariances between the alleles – a phenomenon called linkage disequilibrium in quantitative genetics. In the next section we will discuss this linkage disequilibrium.

2.3 Effects Of Linkage Disequilibrium On Convergence Time

2.3.1 Introduction

The convergence models discussed in the previous section all assume that the fitness is normally distributed. The normal distribution assumption allows us to build simple analytical models but in practice there are a number of factors that violate this assumption. For instance we have neglected the effects of a finite population size and thus assumed that the population size is infinite. From the previous experiments it is clear that the most important factor is the build-up of correlations between the alleles. When the population size is finite, selection will introduce a covariance between the proportion of optimal alleles at each gene. Crossover on the other hand reduces the allele correlations, which explains the observation in the previous section that repeating the recombination phase during each generation made the model predictions more accurate.

In this section we will first quantify the covariance introduced by selection and the subsequent decorrelation by crossover. It is then shown how these results can be used to refine the convergence models based on the strict normal distribution assumption.

2.3.2 Genic Covariance Introduction through Selection

Selection picks out individuals that are mainly concentrated at the higher level of the fitness distribution. The variance of the population fitness is therefore reduced but this reduction is not only caused by the change in allele frequency but also by the creation of covariances between the genes.

For a population of size n we have a genic covariance matrix $\Theta(t)$ with the genic covariance between two arbitrary genes x and y defined as (alleles $x_u, y_v \in \{0,1\}$):

$$\begin{aligned}\Theta_{xy}(t) &= \frac{1}{n}\sum_{u=1}^{n}\sum_{v=1}^{n}(x_u(t)-\overline{x(t)})(y_v(t)-\overline{y(t)}) \\ &= \overline{x(t)y(t)} - \overline{x(t)}\;\overline{y(t)} \\ &= p_{xy}(t) - p_x(t)p_y(t)\end{aligned}$$

with $p_x(t), p_y(t), p_{xy}(t)$ the allele proportions for the corresponding genes. The population fitness variance can be computed by summing all elements from the genic covariance matrix $\Theta(t)$ (l = string length):

$$\sigma^2(t) = \sum_{i=1}^{l}\sum_{j=1}^{l}\Theta_{ij}(t)$$

or separating the genic variance and covariance components

$$\sigma^2_{var}(t) = \sum_{i=1}^{l}\Theta_{ii}(t)$$

$$\sigma_{cov}(t) = \sum_{i=1}^{l-1}\sum_{j=i+1}^{l}\Theta_{ij}(t)$$

the population fitness variance becomes:

$$\sigma^2(t) = \sigma^2_{var}(t) + 2\sigma_{cov}(t) \tag{2.10}$$

When the population is in linkage equilibrium - this is when $p_{ij}(t) = p_i(t)p_j(t)$ - the genic covariance is zero. Selection thus creates a linkage disequilibrium which makes the normal distribution assumption less valid.

In the previous section we have seen how the ranked based selection schemes have the same convergence characteristics. A general formula was obtained by using the selection intensity i for truncation selection. Truncation selection also creates a higher covariance than tournament selection, since it only copies the top $T\%$ strings. Therefore it is most instructive to model the genic covariance for truncation selection.

With truncation selection the selected parents represent one tail of the normal distribution. Truncating a normal distribution with variance σ^2 reduces this variance with a factor k such that

$$\sigma^2(t^s) = (1-k)\sigma^2(t) \tag{2.11}$$

with k given by

$$k = i(i - x)$$

where i is the selection intensity and x is the corresponding deviation of the truncation point from the population mean (Falconer, 1989). Table 2.1 shows numerical values of the parameters i, x, and k in function of the truncation threshold T.

The change in variance after selection is due to two phenomena: first there is an increase in the target or optimal allele frequency $p(t)$, and second there is a creation of genic covariance. The allele-1 frequency increase in one generation however is rather small so the change in variance is primarily caused by the creation of the genic covariance. To make the analysis tractable we assume that we can neglect the effect of the allele frequency increase.

Under this simplifying assumption the genic variance after selection $\sigma^2_{var}(t^s)$ is equal to the population fitness variance $\sigma^2(t)$ which allows us to calculate the genic covariance after selection $\sigma_{cov}(t^s)$ from Equations 2.10 and 2.11:

$$\begin{aligned} \sigma^2(t^s) &= \sigma^2_{var}(t^s) + 2\sigma_{cov}(t^s) \\ &= (1 - k)\sigma^2(t) \end{aligned}$$

or

$$\sigma_{cov}(t^s) = -\frac{k}{2}\sigma^2(t) \tag{2.12}$$

To check the validity of the above assumption we compare the theoretical predictions with experimental data. For the experiments we take a string length $l = 100$, a population size $n = 200$ and a truncation threshold $T = 50\%$, meaning that we select the top half of the strings and give them two copies. The selection intensity is thus $i = 0.8$, and since the truncation point coincides with the population mean ($x = 0$) the reduction factor k is $k = 0.64$.

The variance and covariance after selection are then:

$$\sigma^2(t^s) = 0.36\sigma^2(t)$$

$$\sigma_{cov}(t) = -0.32\sigma^2(t)$$

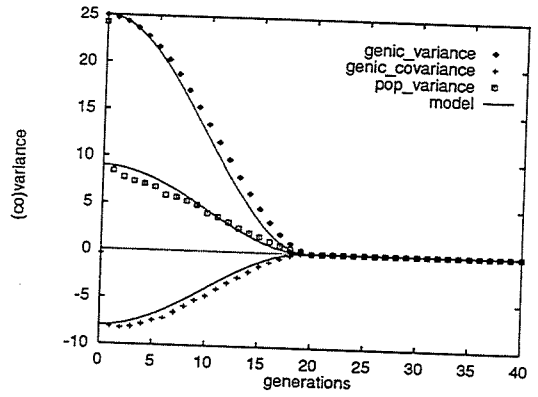


Figure 2.4: *Model and experimental results of the variance and covariance when optimizing the Bit Counting function with truncation selection and uniform crossover.*

Figure 2.4 plots the experimental data against the model predictions. The upper curve represents the variance without considering the covariances or linkage disequilibrium and compares this with the model $\sigma^2(t) = lp(t)(1 - p(t))$ with $p(t) = 0.5(1 + sin(0.08t))$. The lower curve shows the covariance after truncation selection but before recombination and the model $\sigma_{cov}(t) = -0.32lp(t)(1 - p(t))$. Finally the middle curve plots the population variance after selection - taking the linkage disequilibrium into consideration - and with model $\sigma^2(t^s) = 0.36lp(t)(1 - p(t))$. Clearly the model shows a very good agreement with the data so the approximations made are justified.

2.3.3 Genic Decorrelation through Crossover

Until now we have only looked at the effect of selection on the population fitness variance and showed how it reduces this variance by creating a negative genic covariance.

The effect of recombination on the covariance is just the opposite: crossover randomly exchanges alleles so the genic correlation will now be decreased. Under the repeated use of crossover the genic covariance or linkage disequilibrium rapidly approaches zero.

T%	x	i	k=i(i-x)
1	2.326	2.665	0.90
5	1.645	2.063	0.86
10	1.282	1.755	0.83
20	0.842	1.400	0.78
30	0.524	1.159	0.74
40	0.253	0.966	0.69
50	0	0.789	0.64

Table 2.1: *Parameter values of truncation selection.*

To quantify this decorrelation let us look at two arbitrary genes i and j with allele-1 proportion at generation t equal to $p_i(t)$ and $p_j(t)$ respectively. If we call the probability that crossover will disrupt the two alleles $p_r(ij)$ then the proportion of the combined alleles at the next generation $p_{ij}(t+1)$ is given by the proportion of $p_{ij}(t)$ that survives crossover plus the newly created combinations or:

$$p_{ij}(t+1) = (1 - p_c p_r(ij))p_{ij}(t) + p_c p_r(ij) p_i(t) p_j(t)$$

with p_c the probability of applying the crossover operator. The above equation can be rewritten as:

$$p_{ij}(t+1) = (1 - p_c p_r(ij))[p_{ij}(t) - p_i(t)p_j(t)] + p_i(t)p_j(t)$$

or since crossover does not change the proportion of alleles (and thus $p_i(t)p_j(t) = p_i(t+1)p_j(t+1)$) we have

$$p_{ij}(t+1) - p_i(t+1)p_j(t+1) = (1 - p_c p_r(ij))[p_{ij}(t) - p_i(t)p_j(t)]$$

or

$$\Theta_{ij}(t+1) = (1 - p_c p_r(ij))\Theta_{ij}(t) \qquad (2.13)$$

To calculate the disruption probability $p_r(ij)$ we have to consider the type of crossover that is used:

Uniform crossover: Uniform crossover exchanges individual alleles between two strings with probability p_x. The probability that two arbitrary alleles stay together is thus equal to the probability they are both swapped plus the probability they are both not swapped: $p_s = p_x^2 + (1 - p_x)^2$.

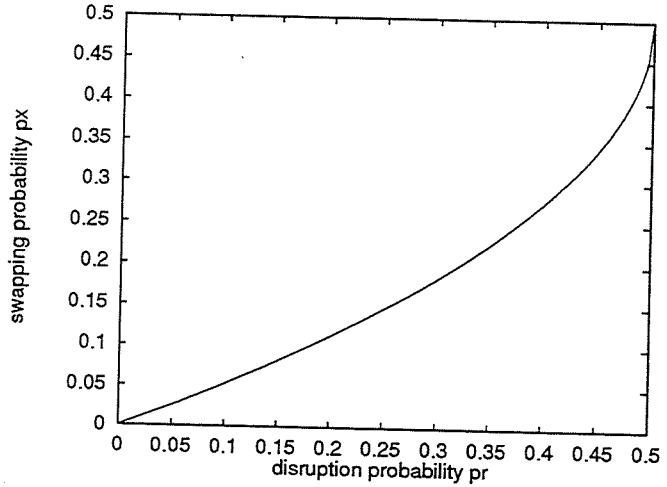


Figure 2.5: *Swapping probability in function of the disruption probability*

p_r	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
p_x	0.05	0.08	0.11	0.15	0.18	0.23	0.28	0.34	0.50

Table 2.2: *Disruption factor p_r vs. swapping probability p_x*

Obviously the disruption probability $p_r(ij)$ is the same for all gene pairs i and j thus we can drop the indices:

$$p_r = 1 - p_s = 1 - p_x^2 + (1 - p_x)^2$$

or

$$p_r = 2p_x(1 - p_x) \qquad (2.14)$$

Since the disruption probability p_r is equal for $p'_x = 1 - p_x$ we can limit the allele swapping probability p_x to $0 \leq p_x \leq 0.5$ which gives:

$$p_x = 0.5(1 - \sqrt{1 - 2p_r})$$

Note that p_x is the parameter we choose when running a GA, while p_r expresses the effect of uniform crossover on the GA performance. Figure 2.5 shows the allele swapping probability p_x in function of the disruption probability p_r. Table 2.2 lists a number of values that we will use in the next chapter.

One point crossover: One point crossover is a linkage biased recombination operator so the disruption probability is dependent on the defining length of the gene pair and is given by:

$$p_r(ij) = \frac{j-i}{l-1}$$

The average crossover disruption probability $\overline{p_r}$ can be computed as

$$\begin{aligned}
\overline{p_r} &= \frac{1}{0.5l(l-1)} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} \frac{j-i}{l-1} \\
&= \frac{1}{0.5l(l-1)^2}[(1+\ldots+(l-1))+(1+\ldots+(l-2))+\ldots+(1)] \\
&= \frac{1}{0.5l(l-1)^2}[1(l-1)+2(l-2)+\ldots+(l-1)1] \\
&= \frac{1}{0.5l(l-1)^2} \sum_{i=1}^{l-1} i(l-i)
\end{aligned}$$

Since $\sum_{i=1}^{l-1} i = \frac{1}{2}(l-1)l$ and $\sum_{i=1}^{l-1} i^2 = \frac{1}{6}l(l+1)(2l+1)$ we obtain after some calculation:

$$\overline{p_r} = \frac{1}{3}\frac{l+1}{l-1} \tag{2.15}$$

Equations 2.14 and 2.15 are both independent of the particular gene pair i and j. Since the genic covariance is given by $\sigma_{cov}(t) = \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} \Theta_{ij}(t)$ we can generalize the decorrelation Equation 2.13 to

$$\sigma_{cov}(t+1) = \delta\sigma_{cov}(t) \tag{2.16}$$

with δ defined as the decorrelation factor:

$$\delta = 1 - p_c p_r$$

Note that the two parameters p_c and p_r both characterize the use of crossover: the first is the probability of applying the crossover operator, the latter its disruptive power. Calling $r = p_c p_r$ the recombination factor we get as decorrelation factor:

$$\delta = 1 - r$$

When we apply recombination t times consecutively (like we did in the experiments to show the convergence dependence on the covariance build-up) we simply obtain:

$$\sigma_{cov}(t) = \delta^t \sigma_{cov}(0)$$

Recombination thus reduces the genic covariance or linkage disequilibri The largest decorrelation factor is obtained with the most disrupt e crossover which is uniform crossover with allele swapping probability $p_x = 0.5$ and a probability of applying crossover $p_c = 1$. The maximal decorrelation factor is thus $\delta = 0.5$, and the covariance between two genes is at most halved after a single recombination phase.

2.3.4 Selection and Recombination Combined

In Section 2.3.2 we have seen how selection creates a covariance between the genes. Equation 2.12 calculates this covariance from the population variance in the previous generation for truncation selection:

$$\sigma_{cov}(t^s) = -\frac{k}{2}\sigma^2(t)$$

Section 2.3.3 discussed the decorrelation by crossover. The amount of decorrelation is given by Equation 2.16:

$$\sigma_{cov}(t+1) = \delta\sigma_{cov}(t)$$

To compute the change in genic covariance from one generation to the next we have to combine the two results which gives:

$$\sigma_{cov}(t+1) = -\delta\frac{k}{2}\sigma^2(t) \tag{2.17}$$

The normal distribution analysis resulted in a general recursive expression for ranked based selection schemes:

$$\overline{f(t+1)} - \overline{f(t)} = i\ \sigma(t)$$

with $f(t) = lp(t)$ and $\sigma^2(t) = lp(t)(1-p(t))$.

To include the effect of covariance creation by selection we must compute the population variance from Equation 2.10: $\sigma^2(t^s) = \sigma^2(t) + 2\sigma_{cov}(t)$ so now we obtain:

$$p(t+1) - p(t) = \frac{i}{l}\sqrt{\sigma^2(t) + 2\sigma_{cov}(t)}$$

or from Equation 2.17

$$p(t+1) - p(t) = \frac{i}{l}\sqrt{lp(t)(1-p(t)) - 2\delta\frac{k}{2}lp(t)(1-p(t))}$$

and finally

$$p(t+1) - p(t) = \frac{i}{\sqrt{l}}\sqrt{(1-k\delta)p(t)(1-p(t))} \tag{2.18}$$

Approximating the difference equation with the differential equation

$$\frac{dp(t)}{dt} = \frac{i}{\sqrt{l}}\sqrt{(1-k\delta)p(t)(1-p(t))}$$

the solution becomes

$$p(t) = 0.5(1 + \sin(\sqrt{\frac{1-k\delta}{l}}\ it + \arcsin(2p(0)-1))).$$

For a randomly initialized population ($p(0) = 0.5$) the solution simplifies to:

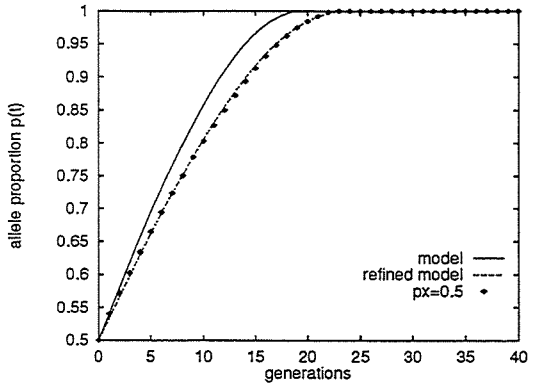


Figure 2.6: *Convergence model, refined convergence model and experimental results of the proportion of optimal alleles when optimizing the bit counting function (string length $l = 100$, population size $n = 200$) with truncation selection ($T = 50\%$) and uniform crossover. The allele swapping probability is $p_x = 0.5$.*

$$\boxed{p(t) = 0.5(1 + \sin(\sqrt{\tfrac{1-k\delta}{l}}\ it))} \tag{2.19}$$

The number of generations g_{conv} to convergence ($p(g_{conv}) = 1$) now becomes:

$$g_{conv} = \frac{\pi}{2i}\sqrt{\frac{l}{1-k\delta}} \tag{2.20}$$

To check the refined convergence model, we have compared it with a number of experimental data for the bit counting function: as in Section 2.2 we take a string length of $l = 100$ and population size $n = 200$. Figures 2.6, 2.7 and 2.8 show the proportion of optimal alleles-1 for truncation selection with threshold 50% and uniform crossover with swapping probability $p_x = 0.5$, 0.3 and 0.1 respectively.

For $p_x = 0.5$ and $p_x = 0.3$ the refined model agrees very well with the experimental data. For $p_x = 0.1$ however the experiments show a

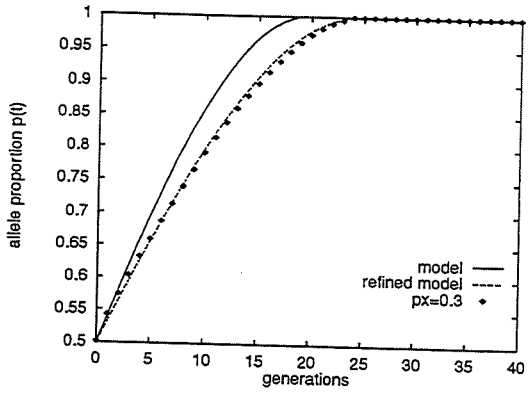


Figure 2.7: *Convergence model, refined convergence model and experimental results of the proportion of optimal alleles when optimizing the bit counting function (string length $l = 100$, population size $n = 200$) with truncation selection ($T = 50\%$) and uniform crossover. The allele swapping probability is $p_x = 0.3$.*

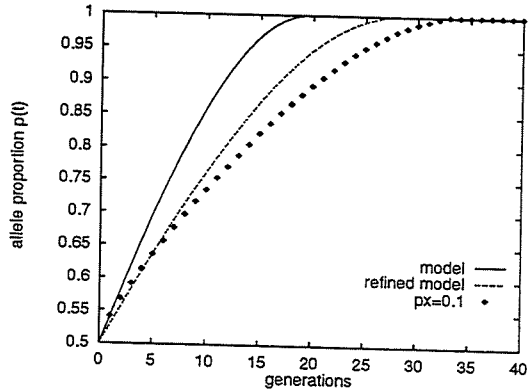


Figure 2.8: *Convergence model, refined convergence model and experimental results of the proportion of optimal alleles when optimizing the bit counting function (string length $l = 100$, population size $n = 200$) with truncation selection ($T = 50\%$) and uniform crossover. The allele swapping probability is $p_x = 0.1$.*

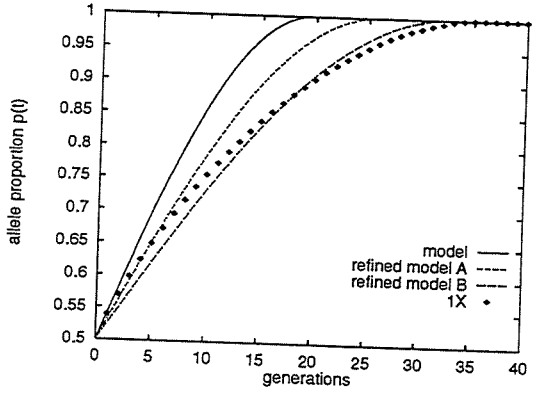


Figure 2.9: *Convergence model, refined convergence models and experimental results of the proportion of optimal alleles when optimizing the bit counting function (string length $l = 100$, population size $n = 200$) with truncation selection ($T = 50\%$) and one-point crossover. Model A takes the average crossover disruption probability, while model model B takes the lowest crossover disruption probability.*

slower convergence than predicted. A similar observation is made with one-point crossover (Figure 2.9).

The reason for this discrepancy is actually quite easy to understand: we have computed the creation of genic covariance when selecting the parents from a normal fitness distribution, but crossover only partially decorrelates the genes so some genic covariance remains. This residual genic covariance is accumulating during the subsequent generations so the normal distribution assumptions become less valid with increasing generations. Whenever the crossover operator has sufficiently high decorrelation power this covariance build-up is small enough to make the predictions of the refined model quite accurate. For recombination operators with low disruption probability however, the genic covariance build-up becomes too significant and the refined model overestimates the convergence speed although it does better than the pure normal distribution model.

2.3.5 Conclusion

We have discussed the creation of genic covariance or linkage disequilibrium by selection and the subsequent decorrelation by crossover. Combining these results gave us a refined model of the previous normal distribution convergence model.

Comparisons with experimental data showed that the predictive value of the refined model is quite accurate for disruptive crossover operators but breaks down when recombination is less disruptive.

2.4 An Upper Bound Through Domino Convergence Modeling

2.4.1 Introduction

In the two previous sections we made the assumption that the fitness contributions of all genes were simply additive and equal. The fitness values were therefore binomially distributed and the normal distribution theory could be used to build simple and accurate models of the convergence behavior of the most widely used genetic algorithm implementations. Since the fitness contributions of all genes are equal, the

population converges uniformly towards the optimum, i.e. the proportion of the optimal alleles is equal in expectation at all generations. In this section we take a look at the other extreme: instead of having all genes converge simultaneously we now assume that the fitness contributions of the genes are such that the genes converge sequentially. The most extreme case is when the fitness contribution of a particular gene at a certain position in the string is higher than all possible fitness contributions of all the following genes. Consider a binary string x of length l and define the fitness as follows :

$$f(x) = \sum_{i=1}^{l} x_i 2^{l-i} \qquad x_i \in \{1, 0\}$$

The fitness contribution of each gene is higher than the sum of the contributions of all following genes. The outcome of a competition between two arbitrary strings is thus uniquely determined at the most significant locus where the two strings have opposite alleles. For instance if we have two strings i1: 11$\underline{1}$10000 and i2: 11$\underline{0}$01100 and the fitness contributions decrease from left to right, then the competition is fully determined at the third position where i1 has allele 1 and i2 has allele 0.

Due to exponential scaling of the gene values the population no longer converges uniformly but instead the rate of convergence is different for all genes: the more significant the genes are, the faster they converge. The least significant genes can only start to converge when the population is already converged for the genes with higher fitness contributions. This causes the overall convergence to happen in a sequential order thus slowing down the convergence time. In (Rudnick, 1992) this problem was also studied and the sequential convergence phenomenon was called domino convergence due to its resemblance with a falling row of domino stones. Rudnick's study looked at two things: first he showed the existence of a convergence window. This window is a set of contiguous bits that are not fully converged yet but did start to converge already. The bits that have a higher fitness value (thus the more significant bits) are already fully converged while the bits at the other side of the convergence window (the less significant bits) did not start to convergence yet. The second result of Rudnick's study was the analysis of the so called convergence stall: when we are using mutation

at a fairly high rate the lower significant bits will not fully converge since the selective pressure acting on these bits is counterbalanced by the disruptive effect of the mutation operator. Rudnick gave a model that approximately predicted the position in the binary string where the bits no longer fully converged.

Here we will analyze the convergence time of the domino phenomenon. As in (Rudnick, 1992) we will look at tournament selection with size $s = 2$ and uniform crossover. The main reason for choosing tournament selection with size two is that this has the lowest selection pressure from all practically used ranked-based selection schemes. It will therefore have the longest convergence time of the rank-based methods and since we are looking for an upper bound this is exactly what we need. In addition a more practical reason is that we (and many others) always use tournament selection with tournament size two when applying a basic genetic algorithm because this selection scheme is extremely easy to implement, requires no population sorting operations and has a low selective pressure in comparison with other rank-based selection schemes and thus has a lower risk of premature convergence.

Before looking at tournament selection however, it may be instructive to see how badly unscaled proportionate selection performs on the exponential scaled fitness function.

2.4.2 Proportionate Selection

In Section 2.1 we already discussed the convergence problems with unscaled proportionate selection. One problem was the lack of selection pressure when all individuals have a fitness of roughly the same value and thus the fitness differences between the individual strings are very small in comparison with the fitness values. In the exponentially scaled fitness function this is certainly the case: the main contributions to the overall fitness is due to the most significant genes, while the impact of the least significant genes is very small. In a string of length l the most significant gene has a fitness contribution of 2^{l-1} while the fitness value of the least significant gene is only 1.

In Figure 2.10 we have plotted the proportion $p(x,t)$ of the optimal 1-bits of gene x at generation t for a typical run with proportionate

selection on a problem with string length $l = 50$ and population size $n = 5000$. Even after 5000 generations only the 10 most significant genes have converged while the rest barely undergoes any selective pressure.

2.4.3 Tournament Selection

In tournament selection with tournament size $s = 2$, individuals are selected by randomly pairing all the strings in the current population and picking the best of each pair. When the fitness values of the genes are exponentially scaled the outcome of this competition is fully determined at the most significant string position with opposing alleles.

Let us call $p(x,t)$ the proportion of optimal alleles '1' in the population at the string position x. If we assume that the genes are ordered in decreasing value from left to right then the probability that a competition between two strings is decided at the most significant gene is equal to the probability that the two competing strings have opposite allele values at the first gene which is equal to $2p(0,t)(1-p(0,t))$. The fraction of tournaments that are not decided at the most significant gene and th's remain to be decided at the following genes is simply $1 - 2p(0,t)(1-p(0,t)) = p^2(0,t) + (1-p(0,t))^2$.

A similar reasoning shows that the expected fraction of tournaments decided at the second most significant gene is equal to the fraction of undecided tournaments times the probability having opposite allele values at the second string position or:

$$[p^2(0,t) + (1-p(0,t))^2][2p(1,t)(1-p(1,t))]$$

The fraction of tournaments that now remain to be decided is given by:

$$[p^2(0,t) + (1-p(0,t))^2][p^2(1,t) + (1-p(1,t))^2]$$

Generalizing this to all gene positions gives us an expression to calculate the proportion of tournaments $g(x,t)$ that are available to be decided at position x:

$$g(x,t) = \prod_{i=0}^{x-1} \left[p^2(i,t) + (1-p(i,t))^2\right] \tag{2.21}$$

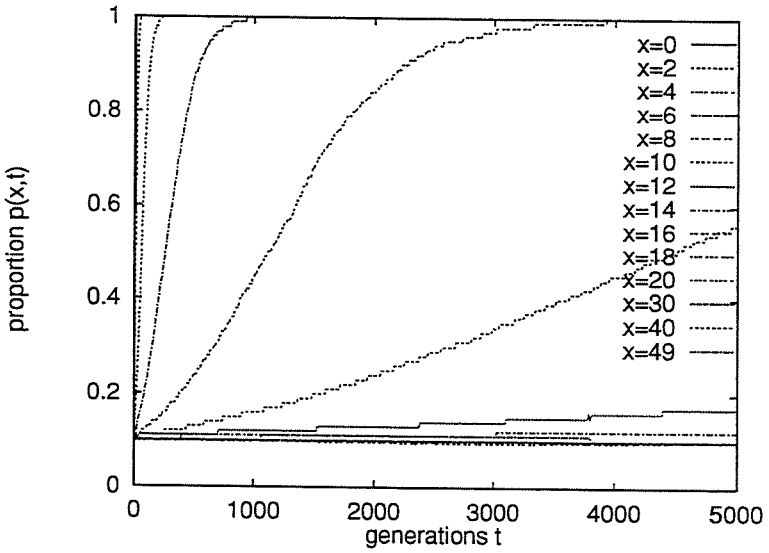


Figure 2.10: *The proportion of optimal alleles or 1-bits of different genes at position x when optimizing the function with exponentially scaled gene fitness contributions. The length of the string is $l = 50$, while the population size is $n = 5000$. The most significant genes (starting at $x = 0$) rapidly converge, while the gene at position $x = 10$ has hardly reached a proportion of about 0.55 even after 5000 generations. The proportion of 1-bits at the genes from position 12 to 49 show hardly any increase, indicating that proportionate selection is unable to exercise any selective pressure on these bits.*

with $x = 1, ..., l-1$ and $g(0,t) = 1$.

The expected fraction of tournaments that is decided at gene position x is equal to:

$$g(x,t)[2p(x,t)(1-p(x,t))]$$

Equation 2.21 has been derived in (Rudnick, 1992). There it is argued that the width of the convergence window can be approximated by calculating the width for a randomly initialized population (or $p(i,0) = 0.5$, $i = 0, \ldots, l-1$) during the first generation (or $t = 1$). The proportion of tournaments $g(x,1)$ that are available to be decided at position x now simplifies to $g(x,1) = 1/2^x$ and the expected fraction of tournaments that is decided at gene position x is equal to $1/2^{(x+1)}$.

As an example suppose we have a population of size $n = 128$. At the most significant gene we expect to decide 64 tournaments, at the next 32 while for the 7^{th} most significant bit we only expect 1 tournament to be decided. The genes from position 8 and more have no impact on the selection process at this generation, thus the convergence window's width is approximately $log_2 n$. Of course this is only the case when all genes have an optimal allele proportion equal to 0.5. For different values one could iterate the recursive Equation 2.21 but no results are given for this in (Rudnick, 1992).

Our goal here in using the domino convergence phenomenon is to obtain an upper bound on the convergence time complexity. Therefore we need to model the proportion of optimal alleles $p(x,t)$ at every position x and at each generation t. To build a closed form analytical model we start by computing the proportion of optimal alleles at the next generation $p(x,t+1)$. There are two different situations to consider:

1. for the fraction of tournaments $g(x,t)$ that are not already decided at an earlier gene position, the allele increase can easily be computed if we realize that the only way a non-optimal allele can survive a competition is when both strings have a non-optimal allele at the gene position considered:

$$1 - p(x,t+1) = (1-p(x,t))^2$$

or

$$p(x,t+1) = p(x,t)(2 - p(x,t))$$

2. for the fraction of tournaments $1 - g(x,t)$ that are already decided at an earlier gene position, the expected allele proportion simply stays the same:

$$p(x,t+1) = p(x,t)$$

Combining the two cases gives us a recurrent equation for the proportion of optimal alleles at each gene position x and generation t:

$$p(x,t+1) = p(x,t)(2 - p(x,t))g(x,t) + p(x,t)(1 - g(x,t))$$

or

$$p(x,t+1) - p(x,t) = g(x,t)p(x,t)(1 - p(x,t)) \qquad (2.22)$$

Approximating the difference equation with the differential equation

$$\frac{dp(x,t)}{dt} = g(x,t)p(x,t)(1 - p(x,t)) \qquad (2.23)$$

we see that the population convergence is modeled by the logistic equation with a variable growth rate $g(x,t)$, where $g(x,t)$ is equal to the expected fraction of tournaments that still have to be decided at the $x - th$ most significant gene.

The solution to the differential equation can be obtained by integrating the partial fractions:

$$\int_{p(x,0)}^{p(x,t)} \frac{dp(x,t)}{p(x,t)} + \int_{p(x,0)}^{p(x,t)} \frac{dp(x,t)}{1 - p(x,t)} = \int_0^t g(x,t)dt$$

which gives

$$\frac{p(x,t)}{1 - p(x,t)} = \frac{p(x,0)}{1 - p(x,0)} e^{\int_0^t g(x,y)dy}$$

or

$$p(x,t) = \frac{1}{1 + (\frac{1}{p(x,0)} - 1)\, e^{-\int_0^t g(x,y)dy}} \qquad (2.24)$$

Equations 2.21 and 2.24 describe the complete convergence dynamics of the population. In Figures 2.11 and 2.12 we have plotted experimental results for optimizing a string of length 50 and starting with an initial optimal allele proportion $p(x,0) = 0.1$. Figure 2.12 shows the proportion of optimal alleles '1' for 6 different genes ($x = 0, 9, 19, 29, 39$ and 49) in function of the number of generations t. On can clearly see the sequential convergence process: the most significant gene $x = 0$ converges in about 5 generations, while the least significant gene starts only to convergence after approximately 80 generations. Figure 2.11 shows the proportion of optimal alleles '1' for all genes x at different generations ($t = 0, 20, 40, 60, 80$ and 100). Initially ($t = 0$) all genes have a proportion $p(x,0) = 0.1$. After 20 generations we see that the 8 most significant genes are fully converged, while genes 16 and further did not start to convergence at all. In between we have the convergence window where the genes are partially converged.

Figures 2.13 and 2.14 show the corresponding model predictions by iterating the difference Equation 2.22. The modeling is thus quite accurate but although Equations 2.21 and 2.22 - or 2.21 and 2.24 - completely describe the convergence process they do not allow us to write down a closed form analytical expression, nor do they tell us much about the time it takes for the whole population to converge.

To get a better understanding of the whole convergence process we start modeling the convergence of the most significant gene. For $x = 0$ the logistic growth rate is $g(0,t) = 1$ so the proportion of optimal alleles is given by the logistic function (Equation 2.24):

$$p(0,t) = \frac{1}{1 + (\frac{1}{p(0,0)} - 1)\, e^{-t}}$$

In fact we do not need to use the differential equation approximation of the difference Equation 2.22. For $g(0,t) = 1$ the difference equation reduces to:

$$p(0,t+1) = p(0,t)(2 - p(0,t)) = 1 - (1 - p(0,t))^2$$

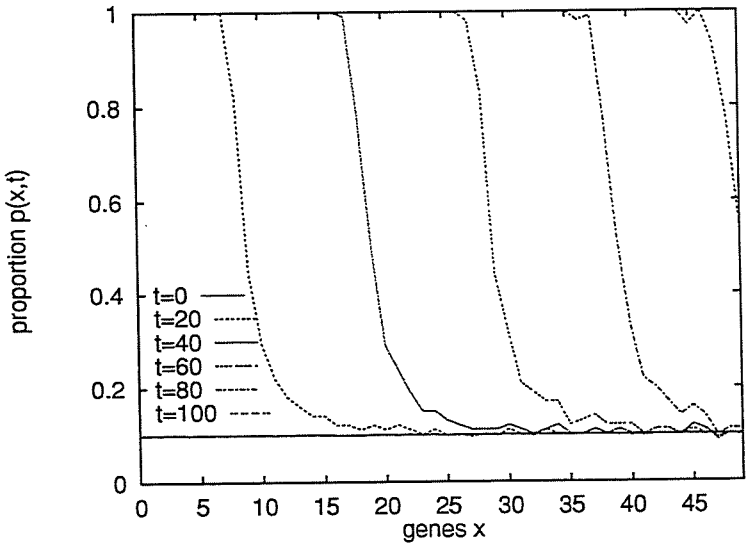


Figure 2.11: *Domino convergence experimental results: the optimal allele proportion $p(x,t)$ is shown at 6 different generations t for all genes with position x in a binary string of length 50.*

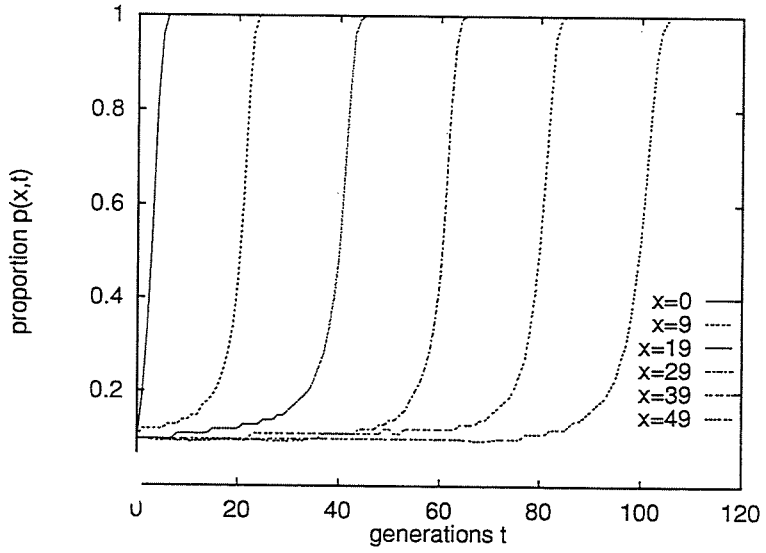


Figure 2.12: *Domino convergence experimental results: the optimal allele proportion $p(x,t)$ is shown for all generations t at 6 different genes with position x in a binary string of length 50.*

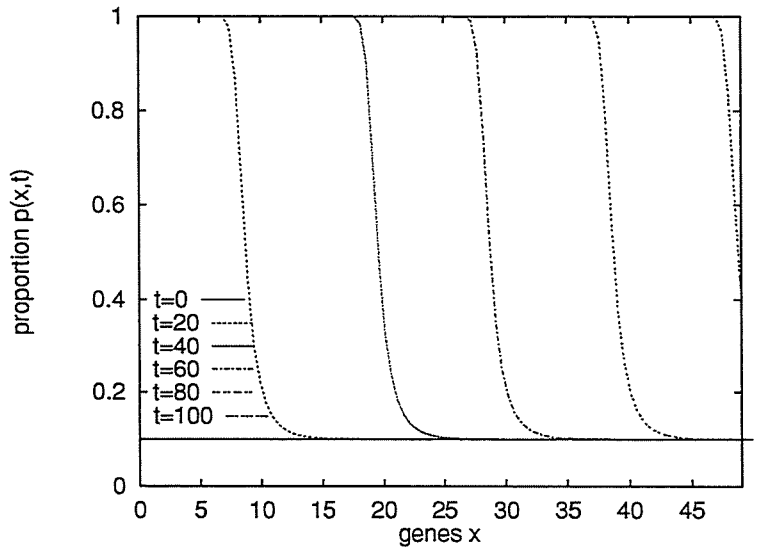


Figure 2.13: *Domino convergence modeled by iterating the difference equations: the optimal allele proportion $p(x,t)$ is shown at 6 different generations t for all genes with position x in a binary string of length 50.*

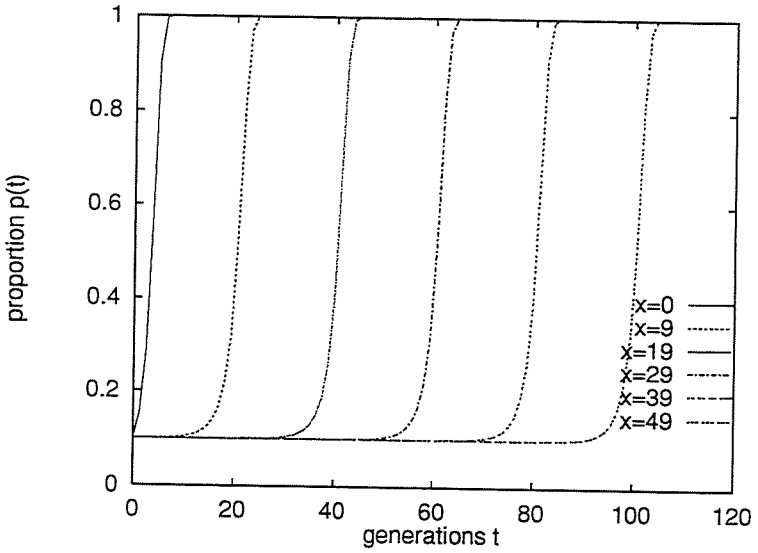


Figure 2.14: *Domino convergence modeled by iterating the difference equations: the optimal allele proportion $p(x,t)$ is shown for all generations t at 6 different genes with position x in a binary string of length 50.*

which has as solution

$$p(0,t) = 1 - (1 - p(0,0))^{2^t}$$

To calculate the number of generations g_{conv} it takes for the most significant gene to convergence completely we have to compute the time $t = g_{conv}$ at which the allele proportion comes arbitrarily close to 1:

$$p(g_{conv}) = 1 - \epsilon$$

or

$$g_{conv} = \frac{1}{\ln 2} \ln \left(\frac{\ln \epsilon}{\ln(1 - p(0,0))} \right) \tag{2.25}$$

For instance when $p(x,0) = 0.1$ and $p(g_{conv}) = 90\%$ (resp. 99%) or $\epsilon = 0.1$ (resp. 0.01) the time to convergence is 4.45 (resp. 5.45), thus after approximately 5 generations the first gene will be fully converged.

The analytical expression for the proportion of optimal alleles for the first gene also results in an analytical expression for the logistic growth rate of the second gene (Equation 2.21)

$$g(1,t) = p^2(0,t) + (1 - p(0,t))^2$$

Unfortunately plugging this value into Equation 2.24, analytically solving it and repeating the procedure for the other genes is not feasible.

Obviously we need to make some simplifications here. In Figure 2.15 we have plotted the growth function for the 10 most significant genes with variable growth rate (or the genes $x = 1, ..., 10$, since the growth rate of the first gene $x = 0$ is always 1). The data is obtained by taking the initial optimal allele proportion $p(x,0) = 0.1$ and iterating Equations 2.21 and 2.22. There are clearly two distinct phases in the convergence dynamics. The first phase shows a decrease of the logistic growth rate for all genes. This phase takes only a few generations: when the second most significant gene has a logistic growth rate equal to 1, the first phase has already ended a few generations ago. Since the growth rate of this gene will be 1 at the moment that the most significant gene ($x = 0$) is fully converged, the calculation of the convergence time of the most significant gene is an upper bound to the length of the first phase, and we have just calculated above that this goes very fast.

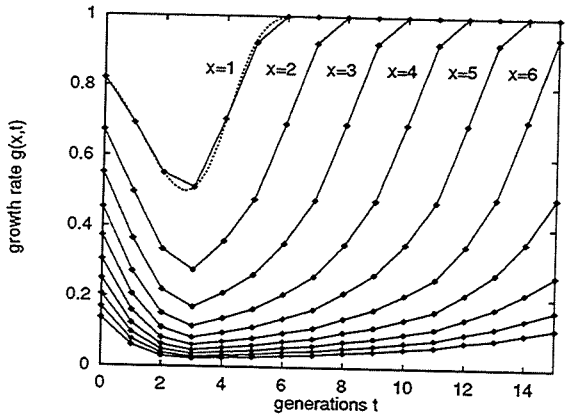


Figure 2.15: *The growth function $g(x,t)$ for the first 10 most significant genes when starting from an initial optimal allele proportion $p(x,0) = 0.1$.*

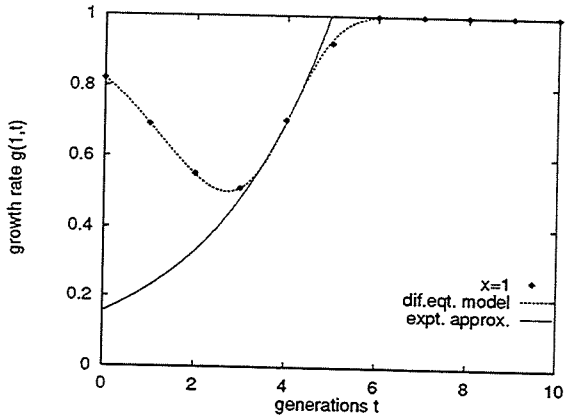


Figure 2.16: *The growth function $g(1,t)$ for the second most significant gene ($x = 1$) with the corresponding difference equation model and an exponential approximation to the second phase with increasing growth rate.*

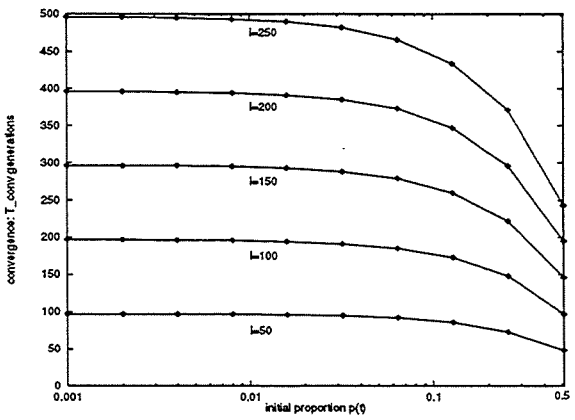


Figure 2.17: *The population convergence time minus the convergence time of the most significant gene for different initial allele proportions and different string lengths l.*

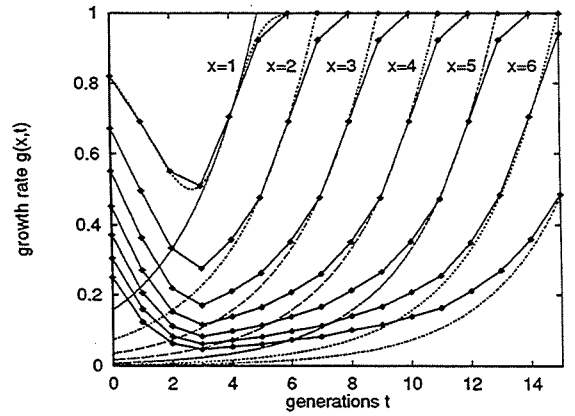


Figure 2.18: *The growth function $g(x,t)$ for the first 10 most significant genes with their corresponding exponential approximation.*

The first phase is thus insignificant compared to the second phase and the convergence time of the most significant gene can be used as a tight upper bound on the number of generations it takes for the first phase to finish. During the second phase the logistic growth rate of all genes is increasing, and it is this phase that basically determines the domino convergence time complexity. In Figure 2.15 we see how some of the genes have a rapidly increasing growth rate while others have a very low, hardly increasing growth rate and only start to increase rapidly after a number of generations have passed. This is of course what we expected from the convergence window discussion: genes that are currently in the convergence window rapidly converge while those that are not reached yet by the window stay more or less at the same allele proportion $p(x,t)$.

To build an approximate model we assume that the convergence rate is exponentially increasing. In Figure 2.18 we have plotted a set of exponential curves through the growth rates such that at high growth rates they fit well but at the lower growth rates the approximation becomes quite bad. However, since the low valued growth rates have only a marginal influence on the overall convergence we might expect that the approximation is good enough to model the domino process. In the next section we formalize this approximate model.

Domino convergence model

The approximate model for the domino convergence assumes that the allele proportions are growing logistically in time t with an exponentially increasing growth rate at each gene, and for a given generation t the growth rate is also exponentially decreasing in function of the gene's position x on the string. The model is thus given as $g(x,t) = e^{a't+b'x+c'}$ with $a' > 0$ and $b' < 0$ or in an equivalent form:

$$g(x,t) = ae^{b(x+ct)} \tag{2.26}$$

which is the general form of a wave equation with traveling speed c. To quantify the parameters a, b ($b < 0$) and c ($c < 0$) we first model the growth curve for the second most significant gene $x = 1$ (note that for the most significant gene $x = 0$ the growth rate is always 1). The growth

curve is given by

$$g(1,t) = p^2(0,t) + (1 - p(0,t))^2$$

with

$$p(0,t) = 1 - (1 - p_0)^{2^t}$$

Figure 2.16 shows the growth curve for gene $x = 1$ and the exponential approximation for the increasing part of the curve: $g(1,t) = ae^{b(1+ct)}$

We choose the exponential curve such that it goes through the inflexion point and has the same tangent as the growth curve. Therefore we need to calculate the first and second derivative of $g(1,t)$:

$$\frac{dg(1,t)}{dt} = [4p(0,t) - 2]\frac{dp(0,t)}{dt}$$

and

$$\frac{d^2g(1,t)}{dt^2} = 4p(0,t)\frac{d^2p(0,t)}{dt^2} - 2\frac{d^2p(0,t)}{dt^2} + 4[\frac{dp(0,t)}{dt}]^2$$

From the chain rule it follows that

$$\frac{da^{b^x}}{dx} = b^x a^{b^x} \ln a \ln b$$

and

$$\frac{d^2a^{b^x}}{dx^2} = b^x a^{b^x} \ln a \ (\ln b)^2(1 + b^x \ln a)$$

which gives us

$$\frac{dp(0,t)}{dt} = -\ln(1 - p_0)\ \ln 2\ 2^t(1 - p_0)^{2^t}$$

and

$$\frac{d^2p(0,t)}{dt^2} = \ln(1 - p_0)\ (\ln 2)^2\ 2^t(1 - p_0)^{2^t}[1 + \ln(1 - p_0)\ 2^t]$$

Given the initial allele proportion $p(x,0)$ we can now compute the exponential approximation to the growth curve $g(1,t)$. For example when $p(x,0) = 0.1$ the inflexion point is obtained by solving $\frac{d^2g(1,t)}{dt^2} = 0$

which gives us the coordinates $t_f = 4.13$ and $g(1,t_f) = 0.73$. The gradient of the tangent at the point of inflexion is: $\frac{dg(1,t_f)}{dt} = 0.276$.

Plugging these values into Equation 2.26 and its derivative $\frac{dg(x,t)}{dt} = abc\ e^{b(x+ct)}$ gives us the set of equations:

$$\begin{cases} a\ e^{b(1+4.13c)} &= 0.73 \\ abc\ e^{b(1+4.13c)} &= 0.276 \end{cases}$$

or

$$\begin{cases} bc &= 0.378 \\ a\ e^b &= 0.153 \end{cases}$$

The exponential curve approximating the growth rate of the second most significant gene $x = 1$ is thus $0.153e^{0.378t}$ and gives us two conditions for the three parameters a, b and c. To obtain a third condition we have to consider the exponential curves for the genes $x > 1$.

These curves can only be determined by iterating the difference equation model (Equations 2.21 and 2.22). In Figure 2.17 we have plotted the convergence time of the second phase for different initial optimal allele proportions $p(x,0)$ and for different string lengths l. Recall that the second phase was defined as the phase where the growth rate function $g(x,t)$ is increasing. The convergence time of the second phase is computed by subtracting the convergence time of the most significant gene from the total number of generations it takes for the optimal alleles to reach a proportion of 0.99. The total number of generations until convergence is obtained by iterating the difference equation model, and the convergence time of the most significant gene is computed by Equation 2.25.

Figure 2.18 clearly shows that for low initial proportions of the optimal allele the second phase convergence time is very close to twice the string length l. A close upper bound is thus $2l$ and consequently a good approximation of the speed of the traveling wave is $c = -0.5$.

The wave equation model now becomes

$$g(x,t) = ae^{b(x-0.5t)}$$

and for the initial optimal allele proportion $p(x,0) = 0.1$ the parameters a and b become $a = 0.33$ and $b = -0.75$, or

$$g(x,t) = 0.33\ e^{-0.75(x-0.5t)}$$

The wave equation approximation for the growth rate $g(x,t)$ can also be used in the logistic Equation 2.24 to compute an approximation of the proportion of optimal alleles at every gene for any generation:

$$p(x,t) = \frac{1}{1 + (\frac{1}{p_0} - 1)\ e^{-\int_0^t g(x,y)dy}}$$

Integrating the growth function gives us:

$$\int_0^t a\ e^{b(x+cy)}dy = \frac{a}{bc}[e^{b(x+ct)} - e^{bx}]$$

or

$$p(x,t) = \left[1 + (\frac{1}{p_0} - 1)\ e^{-\frac{a}{bc}[e^{b(x+ct)} - e^{bx}]}\right]^{-1} \qquad (2.27)$$

where the parameters a and b can be computed once the initial proportion p_0 is given and where the constant c is approximately equal to -0.5.

Figures 2.19 and 2.20 show the optimal allele proportion $p(x,t)$ as computed from the wave equation model 2.27. These figures should be compared with the experimental data shown in Figure 2.11 and Figure 2.12, and with the Figures 2.13 and 2.14 obtained by iterating the difference Equation 2.22.

The wave equation model gives an accurate closed form analytical model of the domino convergence. Since the wave speed is constant the convergence time complexity is of order $O(\ell)$.

2.4.4 Conclusion

The domino convergence for tournament selection can be modeled by a wave equation where the speed of the wave is upper bounded by $c = -0.5$. For low initial optimal allele proportions $p(x,0)$ the wave

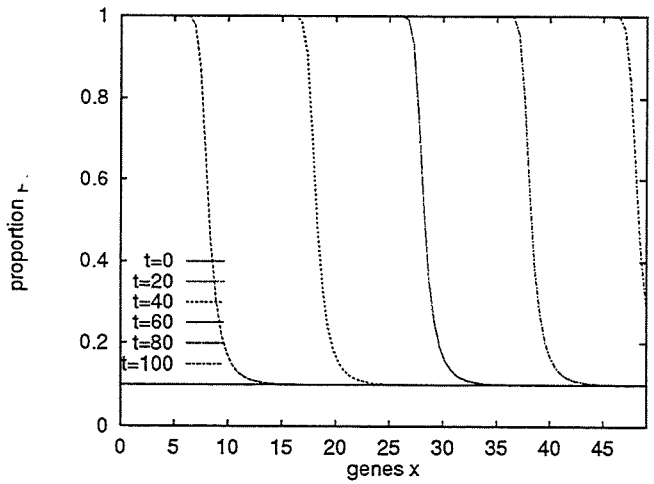


Figure 2.19: *Domino convergence modeled by the wave equation: the optimal allele proportion $p(t)$ is shown at 6 different generations t for all genes with position x in a binary string of length 50.*

with $\mu \leq 2$.

We now calculate the mixing time t_x as the expected number of generations to obtain one mixing event by extending the random assortive model to the building block level. Assuming a population size n, a crossover probability p_c, a building block length k, and 2 BBs, there are $\frac{n}{2}.p_c$ recombinations in one generation so the mixing time t_x is:

$$t_x = \frac{1}{\frac{n}{2} p_c p_{mix}} = \frac{2^{\mu k}}{n p_c}. \tag{3.8}$$

To interrelate recombination with selection we can use the takeover-time model for the selection time:

$$t_s = \frac{\ln n}{\ln s}. \tag{3.9}$$

It is obvious that when we increase the selection pressure s too much, or when our population size n is too small, selection will be too fast for the BBs to get exchanged well enough. So when the selection time t_s is smaller than the mixing time t_x we expect premature convergence to occur. On the other hand when the selection time t_s is larger than the mixing time t_x, we expect that the mixing process will have time enough to juxtapose all BBs in one single individual - the global optimum. Note that the selection and time equations are dimensional models and express exact relationships up to a constant factor c. Therefore a necessary condition for converging to the global optimum in the 2 BB case is thus:

$$t_s > c.t_x$$

or

$$n \ln n > c \frac{2^{\mu k} \ln s}{p_c}. \tag{3.10}$$

Recall that these dimensional models only say something about the functional interrelations between the GA parameters: for instance we can conclude from Equation 3.10 that the boundary between sufficient building block exchange and premature convergence will decrease linearly with the crossover probability p_c, and will grow logarithmically with the selection pressure and exponentially with the BB–length.

In Figures 3.10 and 3.11 we have plotted the predicted functional relations between the population size n and the crossover probability p_c, and between the selection pressure s and the crossover probability p_c. The enclosed region between the mixing and selection boundary is the region where the GA will reliably converge to the global optimum.

In the next paragraph we will check experimentally the dimensional relation.

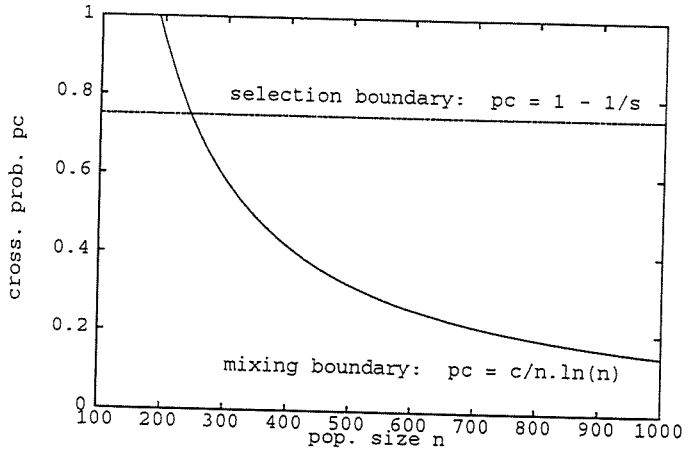


Figure 3.10: *Dimensional model predictions: population size versus crossover probability.*

Experimental Results

All experiments in this section and the next are performed with fully deceptive trap functions with signal value $d = \frac{1}{3}$ (Deb & Goldberg, 1991). For a single parameter combination 50 runs are tried and called successful when at least 49 of them converge to the global optimum. We use uniform crossover with allele-wise swapping probability $p_x = 0.5$, no mutation and block selection (i.e. the $\frac{n}{s}$ best strings in the current population get s copies in the next generation). For the two building block experiments we have to be careful with the initial population: for

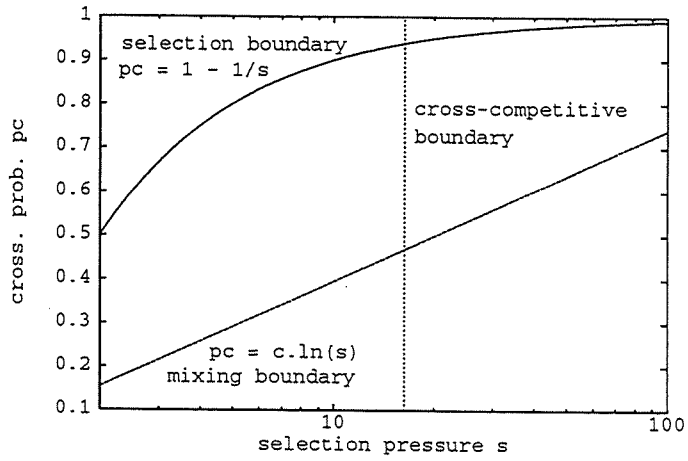


Figure 3.11: *Dimensional model predictions: selection pressure versus crossover probability.*

uniform crossover with fully deceptive functions to work well in a simple GA we need very large population sizes n. Since we are only interested in the mixing behavior of the two BB, all global optimal strings - those that already have the two BBs juxtaposed - are removed from the initial population.

In a first set of experiments (Table 3.1 and Figure 3.12) we look at the relation between the population size n and the building block length k. Results are shown for selection pressure $s = 4$, crossover probability $p_c = 0.5$, number of building blocks $m = 2$ and building block length k going from 3 to 10. The experiments confirm the exponential growth relation between $n \ln n$ and k. The μ (≈ 1.6) coefficient is a little less than 2 as expected.

Figure 3.13 gives the relation between the crossover probability p_c and the population size n for $s = 4$, $k = 4$ and $m = 2$. The bottom part of the curve represents the mixing boundary and has the functional form as expected by Equation 3.10. Below this curve the building blocks initially grow but do not get exchanged before the population converges.

BB-length k	*pop size n*
3	160
4	250
5	900
6	1400
7	9000
8	15000
9	60000
10	80000

Table 3.1: *Minimal population size n needed to exchange two building blocks of length k with uniform crossover for a fully deceptive function. Selection pressure $s = 4$ and crossover probability $p_c = 0.5$. The population sizes shown are the lowest values found for which at least 49 out of 50 runs converged to the global optimum.*

The top part represents the schema theorem (Equation 3.5): above this curve BBs do not grow. For any parameter combination falling in the enclosed area the GA will reliably converge to the global optimum.

Comparing Figure 3.10 and Figure 3.13 shows that the dimensional model captures the functional relations very well.

Finally, Figure 3.14 shows p_c versus $\log s$ ($k = 4$, $n = 600$ and $m = 2$). Again the upper curve represents the schema theorem. The part of the lower curve with small s–values is the mixing boundary indicating a linear relationship between p_c and $\log s$. The vertical boundary at $s = 2^k$ is the cross-competitive boundary where selection is too high (Goldberg, Deb & Thierens, 1992).

Comparing Figure 3.11 and Figure 3.14 shows the good fit between the dimensional model and the experimental results.

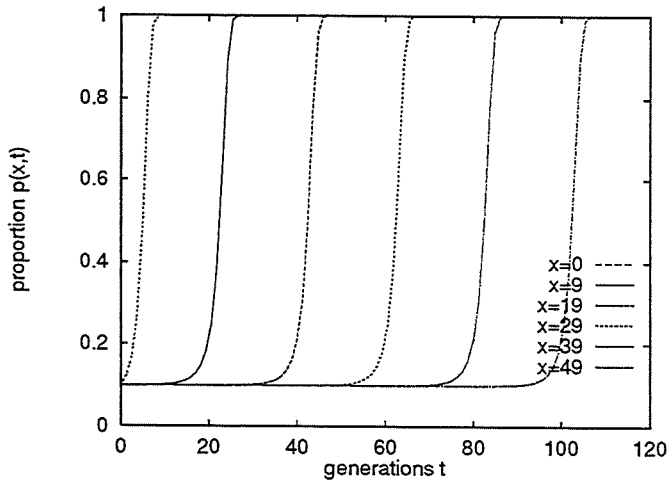


Figure 2.20: *Domino convergence modeled by the wave equation: the optimal allele proportion $p(t)$ is shown for all generations t at 6 different genes with position x in a binary string of length 50.*

speed is very close to -0.5, which means that with a string length l the population will be fully converged after approximately $2l$ generations plus the convergence time needed for the first growth phase. This first growth phase is closely upper bounded by the convergence time of the most significant gene which takes only a few generations.

The overall convergence time complexity for an exponentially scaled fitness function is thus approximately 2ℓ or of order $O(\ell)$.

2.5 Conclusion

We have developed a lower and upper bound for the convergence of different selection mechanisms on bitwise fitness functions. A lower bound is obtained by using normal distribution modeling on the bit counting function. Results show that unscaled proportionate selection is extremely slow, while ranked based selection mechanisms have a convergence time of ord $O(\sqrt{\ell})$.

The allele covariance build-up or linkage disequilibrium is shown to slow down the convergence. A rough refinement to the normal distribution models is proposed.

Finally in the last section we studied an upper convergence bound by analyzing domino convergence. It is shown that in domino convergence the genes are converging logistically with a time varying growth rate. A good approximation is obtained by taking an exponentially increasing growth rate which results in a wave equation model with convergence of order $O(\ell)$.

Chapter 3

Dimensional Analysis of Mixing in Basic GAs

In this chapter we study a largely ignored aspect of genetic optimization, namely the mixing or juxtapositioning of the alleles and building blocks, and its relation to selection. Although mixing has hardly received any attention in the literature, we will show that it is a key factor in order to determine when a genetic algorithm will converge to the global optimum. We argue that from a dynamical or convergence point of view, selection and recombination are involved in a kind of race against time: the number of instances of above average schemata or building blocks is quickly increased from generation to generation by the selection phase, however in order to create optimal solutions these building blocks have to be juxtaposed by the crossover operator and this also takes some time or generations to occur. If the selection of building blocks goes too fast - relative to the rate at which crossover can juxtapose or mix them - then the population will prematurely converge to a suboptimal solution. The GA will only reliably converge to the global optimum if the selection pressure is not too high such that crossover has enough time to recombine the necessary building blocks.

In this chapter we quantify this selection-recombination race by using the dimensional analysis methodology (Ipsen, 1960). Dimensional analysis gives us the functional relations between the key parameters, namely the selection pressure, the recombination rate, the population

size, the string length and the building block length.

In Section 3.2 we present a dimensional model of the bit-wise or allele-wise mixing. First we discuss the random assortive mixing model (Goldberg, Deb & Thierens, 1992) which is partly based on Holland's random assortive modeling of crossover (Holland, 1975). The random assortive mixing model gives the functional relation between the selection pressure and the recombination rate, but it is too crude to model the relation with the string length and the population size. Using our results of the normal distribution modeling in the previous chapter (Section 2.2), we present a complete allele-wise mixing model. The model agrees very well with experimental results, and indicates for what parameter combinations the genetic algorithm will fail to converge to the global optimum due to mixing problems.

In Section 3.3 we analyze the mixing at the level of building blocks. The traditional view on genetic optimization considers crossover to be a source of building block disruption and therefore very disruptive operators such as uniform crossover are supposed to be of no use for problems where one needs to process building blocks as a whole. The most pronounced problem where this is the case are the fully deceptive functions, and general GA wisdom says that a basic GA can only solve these problems to global optimality if the building blocks are tightly linked and one uses one-point crossover, which has only a small disruptive effect. Assuming that the building blocks are tightly linked is however an extremely strong requirement which certainly limits the applicability of the basic genetic algorithm.

Here we will argue that there is a way to solve - at least in principle - fully deceptive functions by using uniform crossover. Since uniform crossover is linkage independent, this means that there is no need to have the building blocks tightly linked. The key factor lies in the selection pressure: by increasing the selection intensity, more copies are made of the building blocks and thus we are allowed to use a more disruptive crossover operator. The schema theorem can be used to see how much we have to increase the selection pressure. Experimental results show indeed that fully deceptive functions are solved to global optimality by using uniform crossover and a high selection pressure. So do basic genetic

algorithms need tightly linked building blocks ? As far as the schema theorem is concerned they do not. There is however a caveat in all this, and it is here that the importance of the need for mixing analysis shows itself most profoundly. Having a high selective pressure ensures that the building blocks increase in number, but we show that in order to juxtapose or mix them we need a very large population size.

Using dimensional analysis we present the mixing ladder climbing model that tells us that the minimal population size needed grows exponentially in the number of building blocks. Therefore, in practice the basic genetic algorithm cannot be used to solve fully deceptive problems unless the building blocks are tightly linked.

3.1 Introduction

In the previous chapter we modeled the GA convergence dynamics under the assumption that the population size was large enough to allow reliable convergence to the global optimum. In theory the use of the normal distribution theory would require an infinite population size, although it was clear from the experiments that the value calculated from the population sizing equation (Goldberg, Deb & Clark, 1992) was sufficient. The modeling of the domino convergence did also need a very large population size because otherwise the least significant bits would be fixed due to genetic drift before the convergence wave reached their position in the string. The analysis of the effects of linkage disequilibrium on convergence speed showed the importance of the recombination rate. Crossover decorrelates the genes and when the build-up of genic covariances was relatively low, the normal distribution modeling was very accurate.

For the bit counting function a high recombination rate creates no problem since the GA can process the alleles independently from each other. However we do not want just to process non-interacting bits - any hillclimbing algorithm could do this more efficiently - but we want the GA to process high valued substrings or building blocks. The schema theorem (see Section 1.4) shows how to increase the proportion of building blocks:

$$m(h, t+1) > m(h, t)\phi(h, t)[1 - \epsilon(h, t)]$$

with $\phi(h,t)$ the reproduction ratio and $\epsilon(h,t)$ the disruption factor. This inequality is basically saying that we can grow building blocks if the number of instances created by selection is larger than the number of instances that are destroyed by recombination. Therefore as far as the schema theorem is concerned the recombination rate should be limited.

It is very important to realize that the schema theorem does not talk about the crucial issue of building block exchange: having the building blocks in large proportions in the population is not by itself a guarantee to find a good solution. Good building blocks of one string also have to be combined with good building blocks of another string to form a new string with a larger number of effective building blocks. Clearly the building block juxtapositioning or mixing is the main task of recombination. Unfortunately, the mixing process by recombination has received almost no attention in the literature and most studies of crossover are limited to its disruptive effects as expressed in the schema theorem.

There are two contradicting requirements on the recombination operator: on the one hand we wish to minimize the disruptive effect of crossover on the building blocks. On the other hand we want to maximize the building block exchange or mixing capability. Selecting building blocks and recombining building blocks are thus two opposing processes. In this chapter we will study the selection and recombination characteristics and focus on the interaction between the two. Contrary to the previous chapter we will particularly be concerned about the effect of low population sizes.

In the first section we will analyze the case with independent bits since this shows most clearly the interaction between selection and mixing for minimal sized populations. An important question to be answered is, when does the GA converge to the optimal solution and how and when does it fail to do so. The second section will then focus on the selection and mixing of building blocks, and shows that the mixing failure severely limits the basic genetic algorithm in solving hard problems if it has no linkage information about the building blocks.

3.2 Dimensional Analysis of Allele-Wise Mixing

3.2.1 Introduction

Since we want to study the GA's mixing limitations for functions that have independent genes it is necessary to look at the functions where the relative gene fitness contributions pose the greatest challenge to the genetic algorithm. In the previous chapter we considered the two extreme cases:

1. First we studied the case where all gene fitness contributions were equal and the proportion of the optimal alleles increased at the same rate for every gene. The normal distribution modeling showed that the convergence speed was of order $O(\sqrt{\ell})$.

2. The other case was when all gene fitness contributions dominated all following genes. The optimal alleles converged sequentially and at any time only a very limited number of genes was actually converging in the so called convergence window. Genes outside the convergence window were either fully converged already or they did not start to converge yet. The domino convergence analysis showed that the convergence was of order $O(\ell)$.

From the mixing point of view, the domino convergence is an easy problem: only a few genes are actually converging at the same time and juxtaposing them is a simple matter. When the gene fitness contributions are equal, all genes are converging simultaneously and thus they all have to be juxtaposed simultaneously. Considering also that the convergence speed is faster for the uniform fitness contributions, it is clear that this is the most difficult situation. Therefore we will analyze the mixing behavior when processing the bit counting function.

3.2.2 Recombination

The random assortive model: a first approximation

To predict the success or failure of gene exchange we first need to have a model that captures the recombinative capacity of crossover. A very

simple model is the so-called random assortive model (Goldberg, Deb & Thierens, 1992).

Assuming a population size n, a proportion of target alleles p, a probability of applying crossover p_c and a string length l, the model calculates the expected number of generations needed to create the optimal string when randomly picking the alleles from the population. For one generation each gene has thus np_c trials with the probability of successfully picking the target allele equal to p. To generate the optimal string we need l successes which gives us for the expected number of generations t_x:

$$t_x = \frac{1}{np_cp^l}$$

The effects of selection can be modeled with the takeover-time models for selection schemes (Goldberg, 1989), (Goldberg & Deb, 1991). The takeover-time models calculate the number of generations it takes for a string with highest fitness value to fill the entire population with copies from itself when starting from one single copy. Note that only selection is used thus there are no new strings created and the population converges to the best string of the starting population. For tournament selection with tournament size s the takeover time is

$$t_s = \frac{\ln n}{\ln s}$$

When selection is too rapid as compared to the mixing time - when $t_s < t_x$ - the population will prematurely converge because recombination did not have enough time to ensure proper mixing. Put it another way, if we want to avoid a mixing failure we need to have a mixing time t_x that is smaller than the selection time t_s or:

$$n \ln n > \frac{\ln s}{p_cp^l}$$

This very simple model gives a first approximation of the relation between some GA parameters, for instance between the selective pressure (s) and the recombination rate (p_c) (Goldberg, Deb & Thierens, 1992). The model is however too crude to capture the role of the string

length l and the proper functional form of the population size n. We will therefore develop a more elaborate model that makes use of the normal distribution modeling from the previous chapter.

Allele-wise mixing model

The random assortive model takes a very global view of the mixing process. To obtain a more accurate model we need to take a less global view and consider how crossover exchanges parts of the parent strings. There are two separate issues here: one is the rate at which two arbitrary genes are mixed, the other is the likelihood that the exchanged genes have a different value, since crossing over the same alleles does not have any effect.

- Calculating the gene exchange probability is straightforward and is basically the same as in the random assortive model: in one generation the expected number of exchanges between two genes i and j is equal to their disruption probability $p_r(ij)$, times the number of crossover operations performed. The number of crossover operations is just the probability of applying crossover p_c multiplied with the population size n:

  $$\Pr[\text{gene exchange}] = p_r(ij)p_cn$$

  The number of generations it takes in expectation to exchange two arbitrary genes is thus $(np_cp_r)^{-1}$.

  In the previous chapter we have already calculated the disruption probabilities when discussing the linkage disequilibrium phenomena. For uniform crossover the disruption probability is simply $p_r = 2p_x(1 - p_x)$ with p_x representing the allele swapping probability. For one-point crossover we can use the average disruption probability $\bar{p}_r(ij) = \frac{1}{3}\frac{l+1}{l-1}$ or the lowest disruption probability $p_r^{low} = \frac{1}{l-1}$

- The second factor we have to include is the probability that the exchanged genes have a different value. Let $p(t)$ be the proportion of the target bit value. Since the genes have an equal fitness contribution, the target bits are binomially distributed with mean

$lp(t)$ and variance $lp(t)(1-p(t))$. Selecting two parent strings at random, the expected number of genes that have a different value is equal to the mean of the absolute value of the fitness difference distribution. This is exactly what we have already computed when deriving the normal distribution convergence model for tournament selection and it was shown there that this is given by:

$$\Delta\bar{f} = \frac{2}{\sqrt{\pi}}\sqrt{lp(1-p)}$$

The exchange of different bit values is thus proportional with the square root of the string length no matter what the proportion of target bit values is.

Combining the similarity likelihood with the expected number of generations to exchange the genes, we obtain for the mixing time t_x:

$$t_x \propto \frac{\sqrt{l}}{np_cp_r}$$

Where $\propto$ stands for the proportionality relation. Note that the two parameters p_c and p_r both characterize the use of crossover: the first is the probability of applying the crossover operator, the latter its disruptive power. Recalling from Section 2.3.3 the definition of the recombination factor $r = p_cp_r$, the mixing equation becomes:

$$t_x \propto \frac{\sqrt{l}}{nr} \tag{3.1}$$

3.2.3 Selection

Incorporating selection in our dimensional modeling is straightforward if we consider the convergence time computed in the previous chapter:

$$t_{conv} = \frac{\pi}{2}\cdot\frac{\sqrt{l}}{i}$$

Since the selection intensity i is a logarithmic function of the tournament size s ($0.5\ln s < i < 0.6\ln s$, $i \propto \ln s$) we obtain for the selection time t_s :

$$t_s \propto \frac{1}{\ln s} \tag{3.2}$$

An alternative way to see this is by considering the selection takeover time equation ($t_s = \ln n/\ln s$) as was done before in the random assortive model. It is important to realize how this selection takeover time is obtained. In (Goldberg & Deb, 1991) the selection takeover time is computed as the number of generations it takes under selection only (this is without using recombination) for one single copy of the best string to get $n-1$ copies. For tournament selection with tournament size s this can be computed if we consider the complementary proportion of the best strings. Since any string that has a lower function value than the superior one, can only get a copy if it is not competing with it we get:

$$1 - p(t) = (1-p_0)^{s^t}$$

or

$$t\ln s = \ln\frac{\ln(1-p(t))}{\ln(1-p_0)}$$

The number of generations needed for the best string to go from an initial proportion of p_0 to a proportion p_t is thus:

$$t = \frac{1}{\ln s}\ln\frac{\ln(1-p_t)}{\ln(1-p_0)}$$

and it is clear that the factor $\ln n$ in the takeover equation comes from the boundary conditions $p_0 = 1/n$ and $p_t = (n-1)/n$, and should not be incorporated in our selection time t_s calculation, thus:

$$t_s \propto \frac{1}{\ln s}$$

3.2.4 Dimensional Model for Allele-Wise Mixing

Now that we have a dimensional model of both recombination and selection, we need to combine them. When selection acts more slowly than exchange - when $t_s \geq t_x$ - we expect the optimal alleles to have time to be juxtaposed and thus creating good strings which can then be given

more copies by selection. This juxtapositioning of target alleles can then proceed until all optimal alleles are put together.

When selection is too fast - when $t_s < t_x$ - too many copies are created of only a few good strings before recombination has had the time to exchange the target alleles. As a result some alleles will not be part of strings that receive extra copies and their proportion in the population will decline until they finally disappear. A lack of good mixing thus results in premature convergence.

To quantify this mixing failure we interrelate the previous dimensional equations (Equations 3.1 and 3.2) for the recombination and selection process. If we want to avoid a mixing failure we need $t_s \geq t_x$ or:

$$\frac{1}{\ln s} \geq c \frac{\sqrt{l}}{nr}$$

Note that dimensional analysis is only giving us the functional relation between the variables so the exact relation is only known up to a constant c which can be determined experimentally (Ipsen, 1960).

Rearranging the selection and recombination parameters at one side and the size parameters at the other side we finally get the mixing model:

$$\frac{\sqrt{l}}{n} \leq c \frac{r}{\ln s} \tag{3.3}$$

Experimental results

The use of dimensional modeling allows us to build simple analytical models that help us to gain insight into the GA convergence process. For instance the model tells us that the minimal population size needs to be doubled when the string length is increased by a factor 4. Dimensional modeling forces us to make some simplifying assumptions. It is therefore extremely important to compare the analytical predictions with experimental results. In addition the dimensional model gives us only a relation up to some constant factor. Again we need experimental results to obtain the value of this constant.

Since we have four independent variables (population size n, string length l, selection pressure s and the recombination factor $r = p_c p_r$) we

have $\binom{4}{2} = 6$ functional relations between two variables while keeping the other independent variables fixed. Exhaustively performing the 6 sets of experiments assures us of the correctness of the dimensional model and the value of the constant factor c.

In the following experiments the results are obtained by doing 20 independent runs for all the parameter combinations shown. We call the experiments a success whenever in at least 19 of these 20 runs the population converged to the optimal string.

1. **string length vs. selection pressure**

   The first experiment looks at the functional relation between the string length l and the selection pressure s. We fix the population size $n = 200$, the crossover probability $p_c = 1.0$ and the disruption probability $p_r = 0.3$ so the crossover swapping probability is $p_x = 0.18$.

   Plugging in these values in the mixing model (Equation 3.3) we predict the boundary between successful and failing convergence to have the following form:

$$\sqrt{l} = \frac{60\ c}{\ln s}$$

   Figure 3.1 shows the data and the corresponding fitted curve

$$l = \frac{320}{(\ln s)^2}$$

   The predicted functional relation and the data coincide very well and the corresponding value of the constant factor in the dimensional model is approximately:

$$c = 0.298$$

2. **disruption probability vs. string length**

   The second experiment checks the relation between the disruption probability p_r and the string length l. The population size $n =$

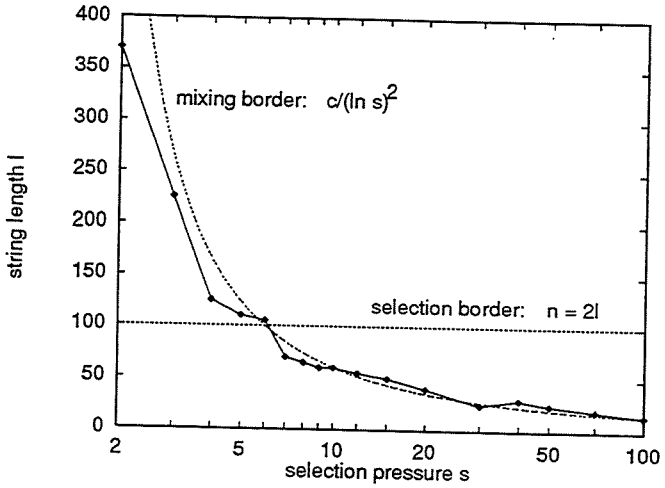


Figure 3.1: *Dimensional relation between the selection pressure and the string length.*

200, the selection pressure $s = 10$ and the crossover probability $p_c = 1.0$.

The predicted relation is now :

$$p_r = \frac{0.0115}{c}\sqrt{l}$$

Figure 3.2 shows a fitted curve equal to

$$p_r = 0.045\sqrt{l}$$

which gives us for the constant factor:

$$c = 0.256$$

The experimental data shows good agreement although for very low string lengths ($l < 30$) the necessary disruption probability is actually a little bit lower than predicted. This can be understood however if we realize that for small string lengths the problem becomes very easy.

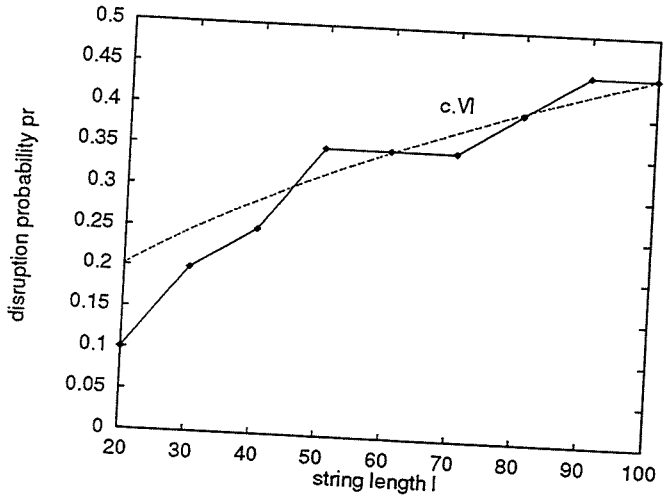


Figure 3.2: *Dimensional relation between the string length and the disruption factor.*

3. crossover probability vs. selection pressure

The relation between the selection pressure and the crossover probability was also determined by the random assortive model. Taking the string length $l = 100$, the population size $n = 200$ and the disruption probability $p_r = 0.5$, the predicted functional relation becomes:

$$p_c = \frac{\ln s}{10\ c}$$

Figure 3.3 shows the experimental results and the fitted curve:

$$p_c = 0.35 \ln s$$

which results in a constant factor :

$$c = 0.286$$

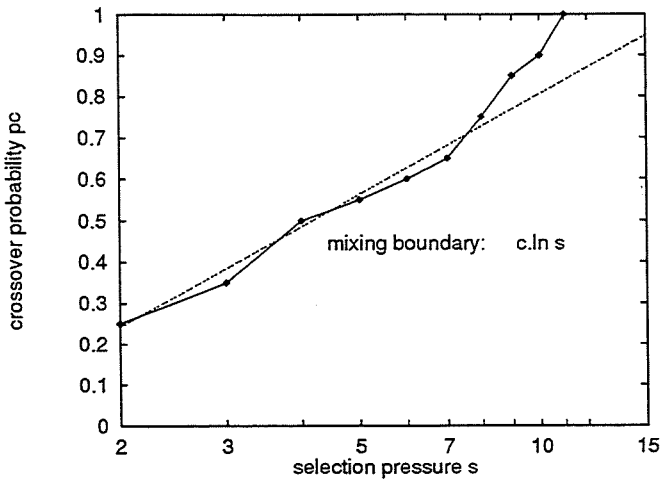


Figure 3.3: *Dimensional relation between the selection pressure and the crossover probability.*

For large values of the selection pressure s the GA behavior is no longer determined by the mixing but by the cross-competition between the genes as discussed in the random assortive model.

4. **population size vs. disruption probability**

The fourth set of experiments establishes the relation between the population size n and the disruption probability p_r. Taking for the string length $l = 100$, the selection pressure $s = 4$ and the crossover probability $p_c = 1$ the predicted relation is:

$$n = \frac{13.86}{c\, p_r}$$

Figure 3.4 shows the experimental results and the fitted curve:

$$n = \frac{50}{p_r}$$

and a corresponding constant factor:

$$c = 0.277$$

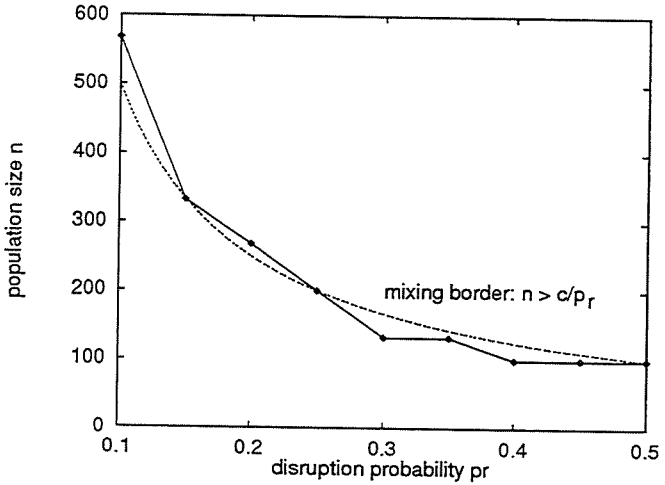


Figure 3.4: *Dimensional relation between the disruption factor and the population size.*

5. **population size vs. selection pressure**

For a string length $l = 100$, crossover probability $p_c = 0.75$ and disruption probability $p_r = 0.20$ ($\Rightarrow p_x = 0.11$) the relation between the population size n and the selection pressure s is:

$$n = \frac{66.67}{c} \ln s$$

Experimental results (Figure 3.5) are

$$n = 236 \ln s$$

or

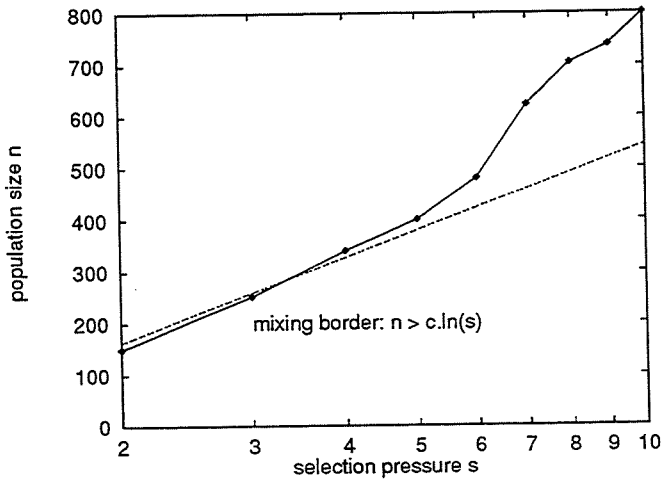


Figure 3.5: *Dimensional relation between the selection pressure and the population size.*

$$\Rightarrow c = 0.282$$

Again we notice that for very large values of the selection pressure s cross-competition starts to influence the convergence results.

6. **population size vs. string length**

Finally the last set of experiments looks at the relation between the population size and the string length. Fixing the selection pressure $s = 2$, the crossover probability $p_c = 1.0$ and the disruption probability $p_x = 0.1 (\Rightarrow p_r = 0.18)$ the dimensional model predicts:

$$n = \frac{7.7}{c}\sqrt{l}$$

Experimental results are shown in Figure 3.6. The fitted curve is:

$$n = 27.3\sqrt{l}$$

which gives us for the constant factor

$$c = 0.282$$

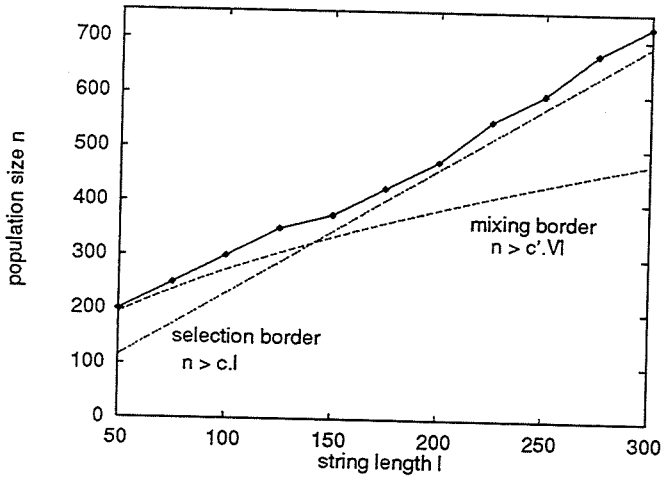


Figure 3.6: *Dimensional relation between the string length and the population size.*

Figure 3.7 shows the same experimental results but on a log-log scale to emphasize to switch between the mixing and selection boundary curves.

The 6 fitted curves gave an approximate value to the constant factor of respectively $c = \{0.298, 0.256, 0.286, 0.277, 0.282, 0.282\}$.

Considering the presence of the constant $\sqrt{\pi}$ in the normal distribution model of the previous chapter it is interesting to note that:

$$\frac{1}{2\sqrt{\pi}} = 0.282$$

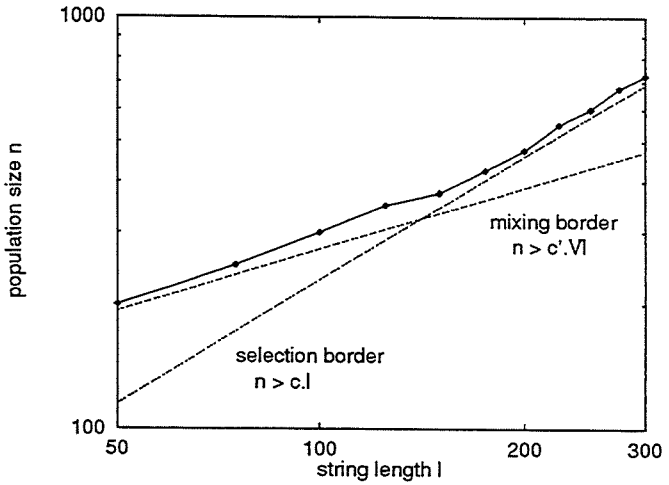


Figure 3.7: *Dimensional relation between the string length and the population size on logarithmic scales.*

Using this value we obtain the following expression for the mixing model:

$$\frac{\sqrt{l}}{n} = \frac{1}{2\sqrt{\pi}} \frac{r}{\ln s} \tag{3.4}$$

3.2.5 Mixing and Linkage Biased Crossover

The derived mixing model gives the boundary for reliable GA convergence for all possible GA parameter combinations. This model is derived for a recombination operator that is not biased toward the linkage between the genes such as uniform crossover. When using one-point crossover however the model is no longer valid. In the previous chapter we have discussed the effects of linkage disequilibrium on the convergence properties and showed how the poorer mixing capacity of one-point crossover causes a considerable covariance build-up. Quantifying the specific convergence properties of one-point crossover is not straightforward and would certainly lead us to far away.

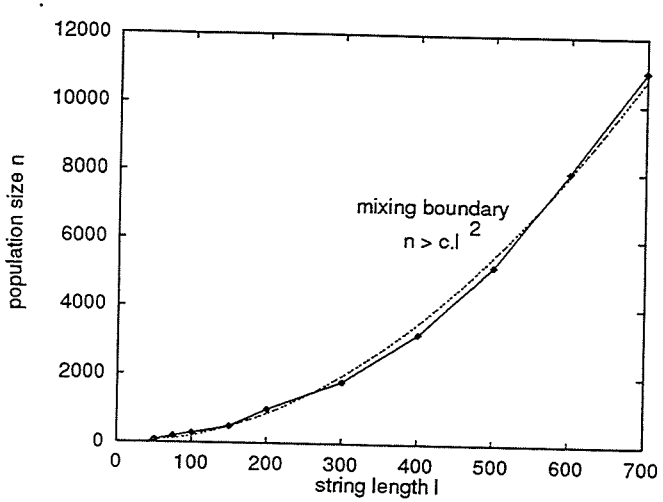


Figure 3.8: *Dimensional relation between the population size and the string length when using one-point crossover.*

In order to get an idea of the mixing capacity of one-point crossover we have determined experimentally the minimal population size needed when optimizing the bit counting function for string lengths with size up to 1000. Figure 3.8 shows the results and compares it with a fitted curve that varies quadratically with l. Obviously the mixing boundary for one-point crossover is at least of order $O(\ell^2)$ which is significantly worse than the order $O(\sqrt{\ell})$ of uniform crossover.

3.2.6 Conclusion

We have identified and shown the importance of mixing analysis to determine when a genetic algorithm will converge to the desired optimum or when it will fail to do so. It is argued that selection and recombination are involved in a race, and when selection goes to fast relative to the recombination or mixing time, then the GA will fail to converge to the global optimum.

We quantify this selection-recombination race by using the dimen-

sional analysis methodology. Dimensional analysis gives us the functional relations between the key parameters, namely the selection pressure, the recombination rate, the population size, and the string length. This resulted in the formalization of the mixing boundary in the genetic algorithm parameter space. The model agrees very well with experimental results, and indicates for what parameter combinations the genetic algorithm will fail to converge to the global optimum due to mixing problems.

3.3 Dimensional Analysis of Building-Block Mixing

3.3.1 Introduction

The previous section considered the allele-wise mixing in a standard GA. Here we extend the mixing analysis to the building-block level by focusing on bounded fully deceptive problems. In the first chapter we have already discussed how Holland identified building-blocks as the fundamental unit of GA processing primarily by examining the schema theorem (Holland, 1975). A somewhat generalized version of the theorem may be written as

$$\phi[1-\epsilon] > 1 \tag{3.5}$$

where ϕ is the growth rate under selection only and ϵ is a conservative estimate of the probability of a schema being disrupted by the genetic operators. The schema theorem is very important to understand GA performance, but that it is not the only factor may be established with the following reasoning. Since the schema theorem contains the product of two terms we may establish the necessary condition either by (1) increasing selection pressure ϕ to counterbalance some bounded loss ϵ or by (2) decreasing ϵ through parametric control of the disruption loss. Pushing harder does not necessarily mean running the risk of premature convergence because now we can also be more disruptive. Increasing the selection pressure ensures that we are able to grow the building blocks (*BBs*) without any need of linkage information, so why would we worry about tight codings as in the traditional GA approach, where one only looks at the disruption factor ϵ. Whenever GA difficult or deceptive

problems are faced, the emphasis is put on the use of low disruptive crossover operators and tight linkage so ϵ is kept small.

Increasing the selection pressure ϕ or decreasing the disruption factor ϵ however does have a cost. This can easily be seen if we take either of these approaches to their logical conclusions and end up with an absurdity. If a selection pressure is chosen to counterbalance a very high loss, all population diversity will be lost almost immediately resulting in no exchange and an extreme form of premature convergence. If disruption is made very small, exchange will never take place and despite meeting the schema theorem criterion, the growing building blocks will never exchange. This points out an often ignored fact of GA practise: growing building blocks is one thing, mixing them is another.

In the initial population we start with one or very few building blocks juxtaposed on certain individuals. Selection increases their proportion and by using the recombination operator we hope that eventually all *BBs* will come together in one optimal solution. In (Goldberg, Deb & Clark, 1992) the six pieces of the GA puzzle were identified as (1) knowing what GAs process - building blocks - (2) ensuring an adequate initial supply, (3) guaranteeing that *BBs* grow, (4) making *BB* decisions well, (5) solving problems that are not too *BB*-difficult, and (6) ensuring that *BBs* exchange to form better solutions. Although substantial progress has been made on the first five items, very little attention has been given to the important mixing issue, or how *BBs* are juxtaposed by the recombination operator.

In the previous section, an analytical model that considered the allele-wise mixing was developed and verified through computational experiments on the bit counting function. Here, we extend the mixing analysis to the building block level for GA-hard functions. In particular we look at simple GAs without linkage information, so no assumptions about tight *BB* linkage can be made. First, we discuss the notion and probability of mixing events. Next, we quantify the mixing of two building blocks, and finally this is generalized to m *BBs* by developing the *mixing ladder climbing* model. Mixing is interrelated with selection to obtain a dimensional GA model. All analytical predictions are verified with computational experiments on fully deceptive functions using

uniform crossover.

3.3.2 Exchanging Building Blocks

For fully deceptive functions with deception length k the building blocks are the optimal schemata of length k and they are in direct competition with the deceptive attractor of the same schema partition. A deceptive function constructed by concatenating m fully deceptive functions of length k has thus m *BBs*. Note that in this view the juxtaposition of two *BBs* at a particular string does not lead to a *BB* of length $2k$ but instead we keep looking to them as 2 separate *BBs* of length k that are mixed at 1 string.

There are basically two different ways for the crossover operator to increase the number of *BBs* on a particular string:

1. One possibility is that building blocks are *created* or *emerge* at a certain position. When both parents have for instance one half of the desired bit values of a particular building block, it is possible that crossover will combine them to create the *BB*. When the fitness function and coding is deceptive, most bit value combination that are not building blocks will differ only a few bits from the deceptive local optimum because they have a higher fitness contribution. The longer the deception length the more likely it is that less than half of the *BB* bit values are present, and therefore the less likely it becomes that building blocks can be created.

2. The alternative is that building blocks get *mixed*. The crossover operator now transfers complete building blocks from both parents to form an offspring that has a higher number of building blocks than either one of its parents. In fact this is really what a recombination operator is supposed to do: recombine basic entities in order to build up a better solution. Fortunately the proportion of these building blocks steadily increases by the work of the selection mechanism. Selection and building block mixing are the two fundamental mechanisms of Genetic Algorithms. The creation of *BBs* is just good luck: since all schemata of lower order than the *BB*-length k lead to the deceptive attractor there is no information

available to the GA that enables it to create *BBs* in a systematic way.

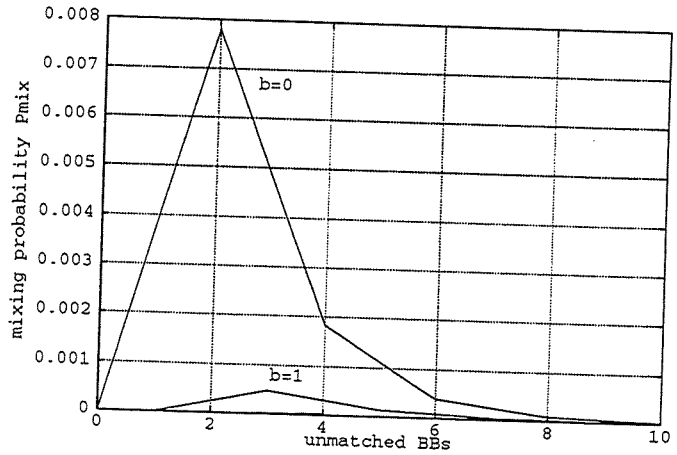


Figure 3.9: *Mixing probability p_{mix} with uniform crossover for deceptive functions ($l = 40$, $k = 4$) in function of the number of unmatched BBs d between the mating partners. Curves are drawn for recombining strings with equal number of BBs ($b = 0$) and with 1 BB difference ($b = 1$).*

To reach the global optimum we have to steadily increase the number of mixed building blocks until all of them are juxtaposed on one single individual. A particular crossover operation is successful if one of the offspring has more *BBs* than each of the parents: in the remainder we will call such a successful recombination a mixing event. Let us call p_{mix} the probability that this happens - p_{mix} is thus a measure of the crossover efficiency. To calculate p_{mix} we introduce the following notations: b_1 (resp. b_2) is the total number of *BBs* of the first (resp. second) parent, $b = |b_1 - b_2|$ is the different amount of *BBs* between the parents, d is the number of *BBs* that occur only in one of the parents (i.e. the unmatched *BBs*), and finally d_m is the number of matched *BBs*. It is easy to see that of the d unmatched *BBs* one parent has

$\frac{d-b}{2}$ *BBs* while the other has $\frac{d+b}{2}$. Therefore a mixing event takes place when one of the children has at least $1+\frac{d+b}{2}$ *BBs*.

A conservative view of mixing is obtained when we consider all non-BBs alleles to consist of all zeros versus the all ones *BBs*. So each individual string consists only of *BBs* and deceptive attractors. For uniform crossover with swapping probability $p_x = 0.5$ the probability that a *BB* is transferred to the other string is $\left(\frac{1}{2}\right)^k$ as is the probability for the *BB* be kept intact at the original string. The building block survival probability is therefore $p_{surv} = 2\left(\frac{1}{2}\right)^k$.

The probability to juxtapose i specific *BBs* and to disrupt the remaining $d-i$ *BBs* is given by $(\frac{1}{2^k})^i(1-\frac{1}{2^k})^{d-i}$. Computing all possible *BB* combinations that give one child a higher number of *BBs* than its parents, and realizing that a mixing success can happen at either one of the two children (which brings in a factor 2), we can calculate the probability p_{mix} that a mixing event between two parents will take place by:

$$p_{mix} = 2\sum_{i=1+\frac{d+b}{2}}^{d}\binom{d}{i}(\frac{1}{2^k})^i(1-\frac{1}{2^k})^{d-i} \tag{3.6}$$

As an example suppose we have two mating strings "111 000 111 000 000" and "111 111 000 111 000" with string length $l = 15$ and *BB*-length $k = 3$. The first string has 2 *BBs* ($b_1 = 2$) of which 1 is not matched by the second string and the second string has 3 *BBs* ($b_2 = 3$) of which 2 are unmatched. The different amount of *BBs* between the two parents is $b = |b_1 - b_2| = 1$ and the number of unmatched BBs is $d = 3$. Note that the number of unmatched *BBs* is indeed given by $\frac{d-b}{2} = 1$ and $\frac{d+b}{2} = 2$ for resp. the first and second string. The probability that one of the children will have more *BBs* than its parents is:

$$p_{mix} = 2\sum_{i=1+2}^{d}\binom{3}{3}(\frac{1}{2^3})^i(1-\frac{1}{2^3})^{3-3} = 2^{-8}.$$

Figure 3.9 shows the mixing probability for strings with 10 building blocks of length $k = 4$ in function of the number of unmatched BBs between the recombining strings. The two curves show p_{mix} when the parents have an equal amount of *BBs* ($b = 0$), and when one of the parents has 1 *BB* more than the other ($b = 1$). For higher values of b - when the different amount of *BBs* between the parents is higher - p_{mix} is too small to be drawn on the scale of the figure.

The chance for a mixing event is thus very low and has a pronounced maximum value when the two parents have the same number of *BBs* ($b = 0$) and only two unmatched *BBs* ($d = 2$). It is important to emphasize that this is of course only a lower bound on the actual GA mixing probability: in a real GA the *BB* survival probability p_{surv} is greater than or equal to $2\left(\frac{1}{2}\right)^k$ because non-*BBs* will still have some optimal alleles even with fully deceptive problems.

3.3.3 Mixing Two Building Blocks

Dimensional Model

Before considering the exchange of building blocks in its general form, we first look at the most elementary mixing problem, namely constructing the optimal solution in a problem with only two building blocks. In this case a single mixing event will juxtapose the two building blocks and create the global optimum. If the selection pressure is high enough, we expect that enough copies of the optimum will be made so that some of them survive the recombination phase and can start to take over the population. In the conservative view the recombining strings consist of a building block and a deceptive attractor (e.g. if "11111" and "00000" represent resp. the building block and deceptive attractor, the mates are for instance "11111 00000" and "00000 11111"). With uniform crossover the probability that an offspring will inherit the optimal allele is $\frac{1}{2}$. The mixing probability p_{mix} for juxtaposing 2 *BBs* of length k is thus $p_{mix} = 2(\frac{1}{2})^{2k}$, where the factor two comes in because the mixing event can happen at either one of the offspring. This is only a lower bound and in a less conservative view we might hope that the parents have a few optimal alleles in common so we can write the mixing probability as:

$$p_{mix} = \frac{2}{2^{\mu k}} \tag{3.7}$$

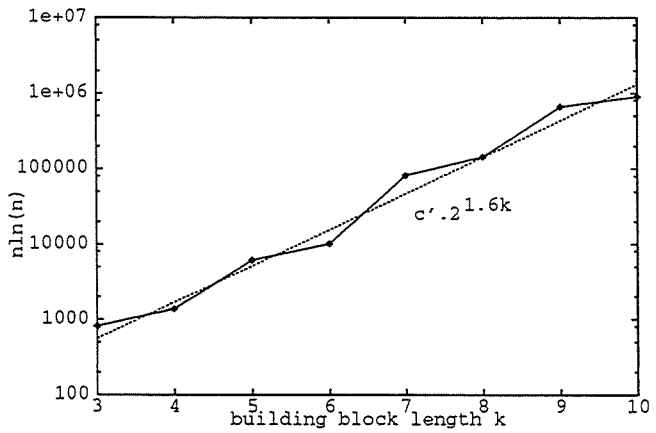


Figure 3.12: *Dimensional relation between BB-length k and population size n when mixing two building blocks with selection pressure $s = 4$ and crossover probability $p_c = 0.5$. The solid curve represents $n \ln n$ with n values shown in table 1. The dotted line $c' 2^{1.6k}$ ($c' = 20$) indicates the linear relationship between $n \ln n$ and $2^{\mu k}$ for mixing 2 building blocks.*

3.3.4 Mixing Several Building Blocks

Dimensional Model

In the previous paragraph the simplest mixing process was considered - the exchange of two building blocks. To extend this analysis to the general m building block case, we have to look explicitly how the GA juxtaposes all the *BBs*. From Equation 3.6 we know that the mixing probability p_{mix} between two parents rapidly decreases when the number of unmatched *BBs* ($= d$) increases and the different amount of *BBs* between the parents ($= b$) for a given d-value increases. By far the highest mixing probability p_{mix} is obtained when the parents have the

same number of *BBs* and they only have two unmatched *BBs* ($d = 2$, $b = 0$). In this case $p_{mix} = \frac{2}{2^{2k}}$ or when taking into account that non-*BBs* also have some optimal alleles $p_{mix} = \frac{2}{2^{\mu k}}$ with $\mu \leq 2$.

Let us call the number of building blocks present in an individual string its mixing level. The probability that a child will have two *BBs* more than one of its parents is negligible in comparison with the probability of having only 1 *BB* more. We can safely assume that after a mixing event has taken place one of the children will have one *BB* more juxtaposed and therefore will be at a mixing level that is only one higher than its parents: so the increase in mixing level goes step by step.

Another interesting consequence of the mixing probability calculation is that the level increase has to occur more or less simultaneously for all building blocks: to see this let us suppose that the mating parents are on a different mixing level. For instance the first parent might be at level two and have its 2 *BBs* at the first and second position ('bb***' with 'b' standing for the presence of a building block and '*' for a non-building block), while the other parent might be at level four ('*bbbb'). In addition let us assume that the first *BB* is covered until level 2, this means that there are no strings in the entire population that have the *BB* in the first position and are at the same time on a mixing level higher than 2. Now for the first *BB* to be covered at level four we need 2 mixing events: a first mixing event is needed to create a level three individual that has the first *BB* and two others juxtaposed. A second mixing event needs to mix the first *BB* with three other *BBs* to cover it at the fourth mixing level. However, while we are waiting on these two mixing events to occur, selection causes the strings at a higher mixing level to take over the population, hereby pushing the first *BB* to extinction. The greater the difference between the different *BB* coverage the more likely we will have premature convergence. To avoid this all *BBs* should more or less go simultaneously to the successive mixing levels and in a simple GA this can only be obtained by increasing the population size n.

To quantify this phenomenon we introduce the *mixing ladder climbing* model. We hypothesize that only the most probable mixing events actually determine the mixing dynamics: the mixing takes place between strings that are at the same mixing level and have only 2 unmatched

BBs ($d = 2$, $b = 0$). Although the model takes a conservative view of GA mixing, it becomes sufficiently accurate with increasing building block length k. Any serious deviation from it (for instance when the population size n is too small) will result in bad *BB* exchange and premature convergence.

Suppose that in the current population all *BBs* are covered up to the mixing level m_l and all strings are at the same mixing level: each string has thus m_l building blocks juxtaposed. Since mixing is only likely to occur when the two mating parents have only 2 unmatched *BBs* between them, we can calculate the mixing probability p_{mix} by realizing that for each string with m_l *BBs* there are $m_l(m - m_l)$ different strings with which it has 2 unmatched *BBs* (e.g. for 'bbb****' we have '*bb***b', 'b*b**b*',...). The mixing probability is therefore:

$$p_{mix} = \frac{m_l(m - m_l)}{\binom{m}{m_l}} \frac{2}{2^{\mu k}} \tag{3.11}$$

Now how do we reach mixing level m_l when we are at level $m_l - 1$? After n_x mixing events we have n_x individuals at level m_l, and the question to be answered is how many mixing events we need in order to have all *BBs* covered at this higher level. Since all of the n_x strings have m_l *BBs* we expect that $(1 - (1 - \frac{m_l}{m})^{n_x})m$ *BBs* are covered. By increasing the number of mixing events n_x the probability that all *BBs* are covered comes arbitrarily close to 1:

$$(1 - (1 - \frac{m_l}{m})^{n_x})m > (1 - \alpha)m \tag{3.12}$$

with $\alpha < \frac{1}{m}$.

After simplifying, approximating $\ln(1 - \frac{m_l}{m}) \approx -\frac{m_l}{m}$ and letting $\ln \alpha = -c$ we get:

$$n_x > c\frac{m}{m_l} \tag{3.13}$$

The number of mixing events needed to climb one step of the mixing ladder is thus proportional to the number of building blocks.

We now have all the necessary pieces to calculate the mixing time t_x - i.e. the number of generations needed for n_x mixing events to occur. This gives us:

$$t_x = c\frac{m}{m_l}\frac{1}{\frac{n}{2}p_c p_{mix}} \tag{3.14}$$

In order to allow the GA to climb each level of the mixing ladder step by step - and thus preventing premature convergence - the selection takeover time t_s has to be larger than the mixing time t_x at each level m_l. The worst case will take place when t_x is maximal and because of the combinatorial factor in Equation 3.11 this will happen at $m_l = \frac{m}{2}$. By using the Stirling approximation $m! \approx (\frac{m}{e})^m\sqrt{2\pi m}$, we see that:

$$\begin{aligned}\binom{m}{m/2} &= \frac{m!}{(m/2)!(m/2)!} \\ &= \sqrt{\frac{2}{\pi}}\frac{2^m}{\sqrt{m}}\end{aligned}$$

and thus

$$p_{mix} = \frac{\sqrt{2\pi}}{4}\frac{m^2\sqrt{m}}{2^m}\frac{1}{2^{\mu k}}$$

and Equation 3.14 can be simplified to:

$$t_x = c\frac{2^{\mu k}}{np_c}\frac{2^m}{m^{\frac{5}{2}}} \tag{3.15}$$

Interrelating the selection time t_s with the mixing time t_x we finally obtain:

$$n \ln n > c\frac{2^{\mu k}\ln s}{p_c}\frac{2^m}{m^{\frac{5}{2}}} \tag{3.16}$$

This dimensional relation shows the limitations of the basic genetic algorithm that has no linkage information and uses uniform crossover: by increasing the selection pressure fully deceptive building blocks can be grown but in order to mix them the population size has to increase exponentially with the number of building blocks !

Experimental Results

In Figure 3.15 we have plotted the population size n versus the number of building blocks m for different values of $BB-$length. Selection pressure s and crossover probability p_c are resp. 4 and 0.5. Clearly the exponential growth of n with m is confirmed. For low building block length values ($k = 3$ and $k = 4$) the growth is a little slower: obviously the influence of emerging building blocks on the pure mixing is more pronounced in these cases.

3.3.5 Conclusion

We have analyzed the exchange of building blocks - building block mixing - in simple GAs for GA-hard functions without the use of any prior linkage information. A dimensional mixing model was developed and interrelated with a selection model to identify regions of GA parameter combinations where the GA is predicted to reliably converge to the global optimum. Experiments with fully deceptive functions and uniform crossover confirm the derived dimensional relations. Attention has been drawn to the fact that by increasing the selection pressure and/or decreasing the crossover probability it is actually quite easy to obey the schema theorem - even for deceptive functions without linkage assumptions. However growing building blocks alone is not sufficient, they also have to be mixed and it was shown that in order to do this the population size for simple GAs grows exponentially with the building block number and length. Of course in practice we cannot work with exponentially increasing population sizes, thus due to the mixing problem basic genetic algorithms can only solve hard problems when the building blocks are tightly linked.

Until now we have restricted our attention to a building block mixing analysis for fully deceptive functions with uniform crossover in simple GAs. By increasing the selection pressure we can increase the proportion of BBs but unfortunately the BB mixing becomes intractable. A number of extensions to the simple GA can be brought in to improve the mixing complexity. Since the mixing probability p_{mix} is highest when the mating parents have only two unmatched BBs, a mating scheme

that restricts recombination to strings within a certain Hamming neighborhood should decrease the mixing time t_x. Niching techniques might help us to climb the mixing ladder in a more smooth way without the need of excessive population sizes. It has been shown here that mixing events are rare, therefore we should be careful not to destroy individuals at the highest mixing level and elitism might help us to achieve this. Finally, the use of blind recombination operators as uniform crossover can be questioned and more sensitive operators might be designed. These issues will be discussed in the next chapter.

3.4 Conclusion

The important aspect of mixing is identified and studied for bitwise and building block dependent functions. First a dimensional model is obtained for the bitwise mixing problem. Regions of GA parameter space are identified where the GA will converge reliably to the global optimum.

A dimensional analysis is also made of building block mixing. It is shown that the basic GA is severely limited in its mixing capacity and unless the building blocks are tightly linked the mixing problem becomes computationally too expensive.

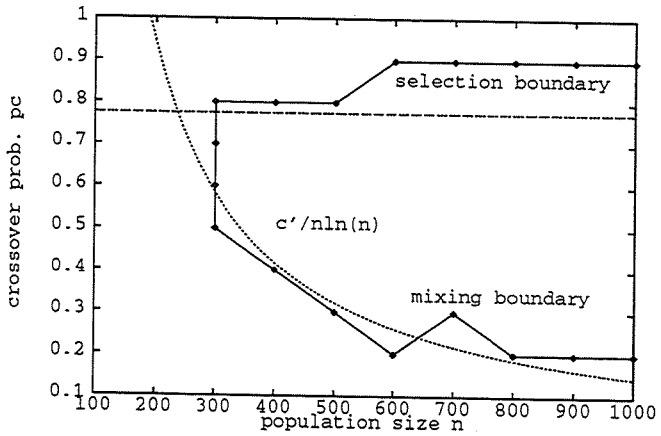


Figure 3.13: *Dimensional relation between population size n and crossover probability p_c when mixing two building blocks with selection pressure $s = 4$ and building block length $k = 4$. The region enclosed by the solid curve shows the n vs. p_c parameter combinations where at least 49 out of 50 runs converged to the global optimum. The lower part of this region is determined by the mixing boundary, which is well modeled by the dimensional relation $p_c > c'/n \ln n$ as indicated by the dotted line $c'/n \ln n$ ($c' = 1000$).*

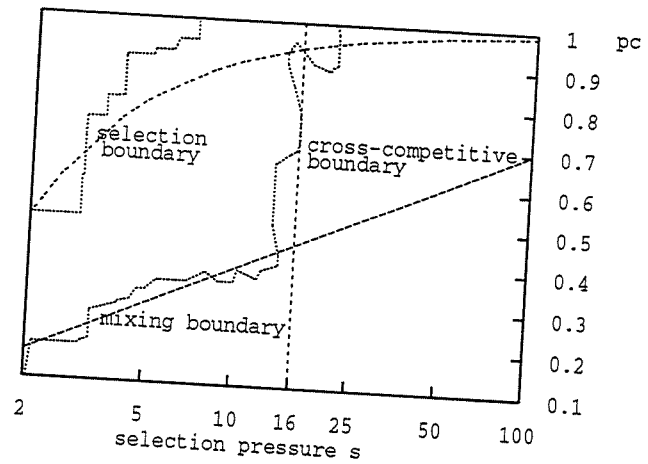


Figure 3.14: *Region of reliable convergence to the global optimum when mixing two building blocks with building block length $k = 4$ and population size $n = 600$ in function of the selection pressure s and the crossover probability p_c. The mixing boundary shows a linear relationship between $\ln s$ and p_c as predicted by the dimensional analysis.*

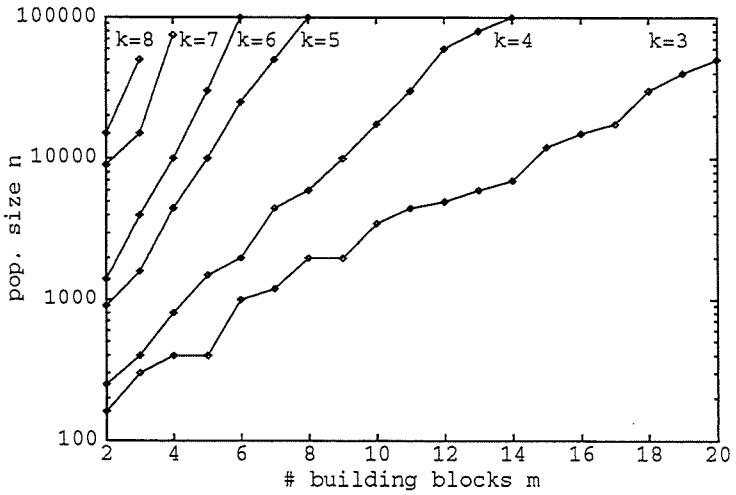


Figure 3.15: *Minimal population size n needed in function of the number of building blocks m in order to converge to the global optimum in at least 49 out of 50 runs. Results are shown for different building block lengths k, with selection pressure $s = 4$ and crossover probability $p_c = 0.5$.*

Chapter 4

Extensions to the Basic Genetic Algorithm

In this chapter we discuss extensions to the basic genetic algorithm and look at their impact in terms of convergence speed and mixing behavior. In Section 4.2 we propose a locally implementable genetic algorithm, where the selection and recombination phase are intertwined. Elitist recombination has a local elitist mechanism such that parent strings can only be replaced by children that have a better fitness value. We show that the selection pressure of the elitist recombination GA is equal to the selection pressure of tournament selection with tournament size two, which is widely used in practice. Due to the local elitism, elitist recombination has a number of additional benefits, and therefore promises to be a very useful GA implementation. Experimental results on a fully deceptive function with linked building blocks confirm the equivalence in convergence speed, and also show that elitist recombination is less sensitive to undersized populations than tournament selection.

Section 4.3 discusses the use of niching techniques in genetic optimization. After a short review of sharing and crowding techniques, a generalized crowding algorithm is outlined. The algorithm is tested on the difficult multimodal fully deceptive function with linked building blocks. Niching based genetic algorithms are usually only applied for multimodal function optimization. We argue that they have also an important role to play in tackling the selection-mixing race. To illustrate

this we apply the generalized crowding algorithm on the hierarchical Royal Road function, proposed by Holland as a challenge to all GA implementations.

In Section 4.4 we analyze the use of restricted mating for increasing the mixing probability. When the mating parents are close together in Hamming space, then most building blocks will be the same in both parents and thus cannot be disrupted by any crossover operator. The mixing probability is therefore increased, but we show that this only replaces one problem with another. For restricted mating to work well, most strings need to have a Hamming neighbor in the current population, and this requirement again makes the problem intractable.

In Section 4.5 we propose to adapt the crossover operator in function of the search trajectory. By making the crossover exchange distribution dependent on the state of the current population, the building blocks are more efficiently preserved and exchanged. We test the extended genetic algorithm using the adaptive crossover operator on a fully deceptive function without tightly linked building blocks and on a multimodal fully deceptive function also without linkage information. In both cases the optimal solutions were found and maintained.

4.1 Introduction

The three previous chapters all discussed the basic genetic algorithm. Chapter 1 gave a general review of the basic theoretical concepts as developed mainly by Holland (Holland, 1975) and by Goldberg and his students (Goldberg, 1987; Goldberg, Deb & Clark, 1992; Goldberg, Deb & Thierens, 1993; Goldberg & Rudnick, 1991; Deb & Goldberg, 1993; Deb, Horn & Goldberg, 1993).

In chapter 2 we discussed different selection mechanisms and developed lower and upper bounds for the convergence time complexity when using sufficiently large population sizes. Chapter 3 looked at the impact of the GA parameters (population size, string length, building block length, selection pressure, and recombination rate), and used dimensional analysis to see where we could expect reliable convergence to the global optimum, and how the failure boundaries scaled with the GA parameters. Great importance was given to what we have called the

mixing failure. An important conclusion was that due to this mixing failure, basic genetic algorithms are severely limited in their building block processing capability, unless these building blocks are tightly linked.

In this chapter we will go beyond the basic genetic algorithm and study some important extensions. First we will propose elitist recombination and analyze its convergence behavior. It will be shown that the elitist recombination GA is similar to tournament selection with regular recombination, but the local elitism mechanism has a number of additional benefits:

1. Good solutions once found are never lost during the search process.
2. Although elitist recombination has the same selection pressure as the widely used tournament selection with regular recombination, it is less sensitive to undersized populations.
3. The particular choice of the crossover probability parameter p_c is no longer an issue.
4. Elitist recombination reduces the risk of premature convergence, whereas other elitist mechanisms increase it significantly.
5. The elitist recombination GA is extremely simple to implement and uses only local computation which makes it ideal for parallel GA implementations.
6. Since solutions stay present in the population unless they are replaced by better solutions, elitist recombination is very well suited to be used in hybrid GAs where the global GA sampling is combined with local optimization algorithms. It has been shown for many different problems that hybrid GAs often give better optimization results than the pure GA or the local optimization algorithms by themselves (Davis, 1991).

A second extension we will discuss is the use of niching. After an overview of the existing niching techniques, the generalized crowding algorithm is outlined which is a natural extension of the elitist recombination GA. Niching has traditionally been used for optimizing multimodal

problems, but we will show that even for unimodal functions niching is a valuable technique since it allows us to side step the race between selecting and mixing the building blocks, which was shown in the previous chapter to be a serious problem for the basic genetic algorithm.

A third extension is the restriction of mate selection. By demanding that the parents are syntactically similar, a great number of alleles are the same in both parents and thus cannot be disrupted by crossover. It is determined to what extent this might help the mixing process.

Finally we investigate the benefit of adapting the crossover recombination distribution at run time. Traditional crossover operators do not change their recombination pattern during the search process: one point crossover for instance always selects a cut point at random with equal probability for all possible positions in the string. Uniform crossover exchanges each allele with a fixed swapping probability. We will show how the building block exchange can be made much more efficiently when the crossover operator is made adaptive such that its recombination probability distribution reflects the current state of the population.

4.2 Elitist Recombination

In the basic genetic algorithm individual strings are copied during the selection phase and are then recombined to create new strings. To reduce the schema disruption factor $\epsilon(h,t)$, the probability of applying crossover p_c is often less than one, so some of the string copies simply go to the next generation without further modifications. The best solutions from generation t however will not necessarily be present in the next generation, nor is there any guarantee that a better solution will be found at the next generation. To make sure that the best solution found so far is always present in the population we can simply copy the best string from the current population to the next generation. This approach - called elitism (De Jong, 1975) - has however a disadvantage: if the problem is very hard and recombination cannot create new, better strings in a small number of generations, then the amount of copies of the best solution will increase rapidly and will take over a large part of the population, driving out other good solutions and thus severely decreasing the search quality. For instance with tournament selection with

tournament size 2 the best solution gets always 2 copies unless it starts competing with itself, thus the growth is logistically and the population will rapidly converge without allowing proper schema processing. For proportionate selection this takeover can even go much faster depending on the relative fitness difference between the best solution and the average population fitness. An additional problem is that with this elitist method only the best is preserved while other strings that might have potentially good building blocks juxtaposed may be destroyed by recombination.

4.2.1 The Elitist Recombination Genetic Algorithm

In this section we introduce an elitist GA - called the elitist recombination GA - that has the same convergence characteristics as tournament selection and does not have the extreme premature convergence risks. In addition the algorithm is extremely simple to implement and does not use any global computations, which makes it ideally suited for parallel GA implementations.

In the basic genetic algorithm, selection and recombination are separated into two distinct phases. For every generation, selection first generates an intermediate population by making extra copies of the best individuals and by eliminating the worst performing individuals. Thereafter recombination is applied to generate the next generation. In order to promote building block growth one typically chooses a crossover probability value $p_c \leq 1.0$. There are however no guidelines - except from some empirical studies - on how to choose a value for p_c.

The elitist recombination GA is a much simpler implementation: no separate selection and recombination phases are needed and crossover is applied for every mating pair so there is no need to specify a value for p_c. Elitist_recombination works entirely at the level of each family: for every mating pair two offspring are created and the best two of these four individuals go to the next generation. Parents are replaced by their offspring when these have a higher fitness value, but this elitism is only working at the family level not at the population level. There is also no separate selection phase where copies are made of good strings. As a result elitist recombination does not suffer from the danger of extreme

premature convergence that often jeopardizes other elitist schemes (of course premature convergence might still occur, for instance when the population size is much too small).

Elitist Recombination algorithm:

1. initialize population
2. for every generation
   (a) random shuffle population
   (b) for every mating pair:
   - generate two offspring
   - keep best two of the four individuals

In order to compare the elitist recombination GA with more traditional GA implementations we analyze the convergence characteristics when optimizing the simple bit counting function with uniform crossover like we did in the normal distribution modeling in Section 2.2. Studying the bit counting function is useful to get a good insight in the speed of convergence, and it is also simple enough to allow building very accurate models. Since there is no interaction between the individual bits the GA has no need to process building blocks, so one should be careful not to overgeneralize the conclusions drawn from this modeling. Therefore we also compare the elitist recombination GA with tournament selection and standard crossover when processing a function composed of building blocks.

For the bit counting function the fitness is binomially distributed. Call $p_j(t)$ the probability of having a 1-bit at a certain position j in the string at generation t. For the bit counting function the gene fitness contributions are all equal, and starting from a randomly initialized population, the probability of having a 1-bit at any position is also equal, thus we can replace $p_j(t)$ simply by $p(t)$. The probability for a string i with length l to have a fitness value x is given by:

$$P[f(i) = x] = \binom{l}{x} p(t)^x (1 - p(t))^{(l-x)}$$

The mean and variance of the fitness function are resp.:

$$\mu = lp(t), \quad \sigma^2 = lp(t)(1 - p(t))$$

Applying crossover does not alter the number of alleles from the parent population to the offspring population, and the children are samples of the same binomial distribution as their parents.

To model the convergence of elitist recombination we need to know the expectation of the population average fitness increase between two successive generations: $\Delta f = f(t+1) - f(t) = l(p(t+1) - p(t))$. For a specific mating pair the sum of the fitness of the two parents is equal to the sum of the offspring's fitness.

$$f(p_1) + f(p_2) = f(c_1) + f(c_2)$$

Suppose that $f(p_1) > f(p_2)$ and $f(c_1) > f(c_2)$, then it is easy to show that $f(p_1) > f(c_2)$ and $f(c_1) > f(p_2)$

PROOF:

$$\begin{aligned} f(p_1) + f(c_1) &> f(p_2) + f(c_2) \\ f(p_1) + f(p_1) + f(p_2) - f(c_2) &> f(p_2) + f(c_2) \\ f(p_1) &> f(c_2) \qquad \square \end{aligned}$$

Proving that $f(c_1) > f(p_2)$ is analogous.

Thus the best parent is always better than the worst child and the best child is always better than the worst parent. For every mating pair the best parent will go to the next generation, and the worst parent will be replaced by the best of the two children. As a result the fitness increase Δf is equal to half the average difference between the worst parent and the best child. The fitness increase Δf can be approximately computed as half of the mean fitness difference between two random samples of a binomial distribution with mean $\mu(t) = lp(t)$ and variance $\sigma^2(t) = lp(t)(1 - p(t))$.

Obviously this is exactly what we have done when modeling tournament selection in Section 2.2, where we computed this by looking at the distribution of the difference $f(p_1) - f(p_2)$. We need to compute the absolute value of the average difference between two random samples

of the normal distribution with mean $\mu(t)$ and variance $\sigma^2(t)$, which is equal to the mean value of the right half of the normal distribution of the difference $f(p_1) - f(p_2)$. Since the difference of two normal distributions is again a normal distribution with mean the difference of the means ($\mu_{dif} = \mu_1 - \mu_2$) and variance the sum of the variances ($\sigma^2_{dif} = \sigma^2_1 + \sigma^2_2$) we have:

$$\mu_{\Delta f}(t) = 0, \quad \sigma^2_{\Delta f}(t) = 2\sigma^2(t)$$

The mean value of the right half of any normal distribution with mean 0 and variance σ^2 is given by $\sqrt{2/\pi}\sigma$. Since the fitness increase Δf is half of the mean of the fitness difference distribution, this becomes:

$$\Delta f(t) = \frac{1}{2}\sqrt{\frac{2}{\pi}}\,\sigma_{\Delta f}(t)$$

or

$$\Delta f(t) = \frac{1}{\sqrt{\pi}}\,\sigma(t)$$

and the increase in proportion of 1-bits is given by the difference equation:

$$p(t+1) - p(t) = \sqrt{\frac{p(t)(1-p(t))}{\pi l}}$$

which can be approximated with the differential equation:

$$\frac{dp(t)}{dt} = \sqrt{\frac{p(t)(1-p(t))}{\pi l}}$$

and solved as:

$$p(t) = 0.5(1 + \sin(\frac{t}{\sqrt{\pi l}} + \arcsin(2p(0) - 1))) \; .$$

For a randomly initialized population $p(0) = 0.5$ the convergence model becomes:

$$\boxed{p(t) = 0.5(1 + \sin(\tfrac{t}{\sqrt{\pi l}}))} \qquad (4.1)$$

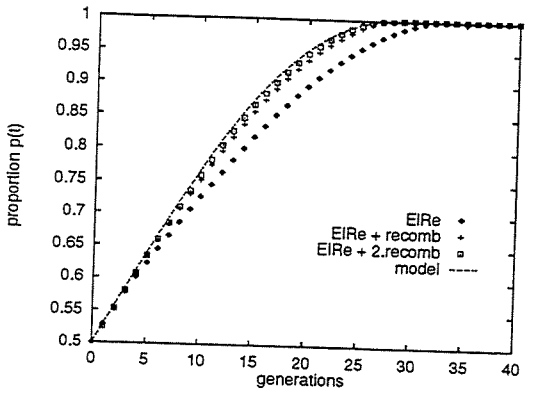


Figure 4.1: *Convergence model and experimental results of the proportion of optimal alleles when optimizing the bit counting function with elitist recombination and uniform crossover.*

The number of generations g_{conv} it takes to let the population fully converge is obtained by putting $p(g_{conv}) = 1$

$$g_{conv} = \frac{\pi}{2}\sqrt{\pi l} \qquad (4.2)$$

The convergence model for elitist recombination and tournament selection with tournament size $s = 2$ are thus the same. Experimental results are shown in Figure 4.1. These data are averages over 30 independent runs, and the standard deviation of these runs is extremely small. As with tournament selection the model slightly overestimates the proportion $p(t)$, and predictions become more accurate when the bit values or alleles are decorrelated by repeating the recombination phase.

Figure 4.2 compares elitist recombination with tournament selection and standard recombination for different crossover probabilities ($p_c = 1.0$, $p_c = 0.8$ and $p_c = 0.6$) and different string lengths ($l = 50$, $l = 100$ and $l = 200$).

For every string length tested the convergence speed for elitist recombination is a little bit slower than for tournament selection and standard recombination with $p_c = 0.8$ while it is a little faster than for tournament

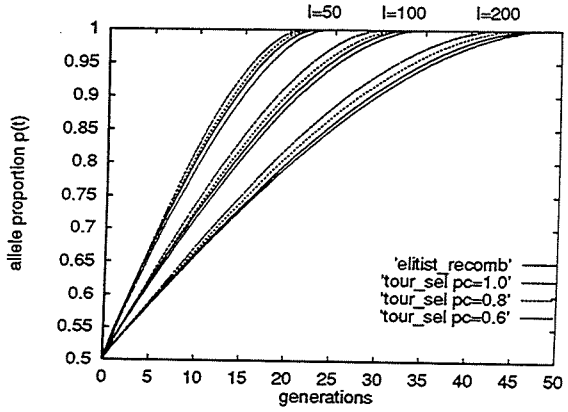


Figure 4.2: *Proportion of 1-bits or 1-alleles when optimizing the bit counting function (string length $l = 50, 100, 200$) with the elitist recombination GA compared to tournament selection with different crossover probabilities p_c.*

selection and standard recombination with $p_c = 0.6$.

More important however is to notice that the difference in convergence characteristics is very small, so that although the elitist recombination GA is implemented quite differently from tournament selection and standard recombination, both algorithms are in fact equivalent in selective pressure.

4.2.2 Example: Fully deceptive function with linkage

Studying the bit counting function is useful to obtain an insight in the speed of convergence of different GA implementations, but as we have already mentioned one should be careful not to overgeneralize the conclusions drawn from this modeling. The lack of interaction between the individual bits allows the GA to process each allele independently without need to preserve higher order building blocks.

To compare the elitist recombination GA with tournament selection and standard recombination when processing building block based functions we tested their performance on a fully deceptive trap function. The deception length is $k = 5$, the number of building blocks is $m = 20$, thus the string length is $l = 100$ and the signal is $d = 0.2$. To make sure that mixing is no problem we let the building blocks be tightly linked and use two-point crossover. The population sizing equation $n = 2z^2(\alpha)(m-1)\sigma^2 2^k/d^2$ reduces for this particular problem to $n = 1760z^2(\alpha)$.

#success/30	n=2000	n=1000	n=500
elitist recomb	30	30	16
tour sel pc=1.0	29	11	3
tour sel pc=0.8	30	23	8
tour sel pc=0.6	30	27	5

Table 4.1: *Number of times the population converges to the global optimum out of 30 independent runs when optimizing a fully deceptive function ($k = 5$, $m = 20$, $l = 100$, $d = 0.2$) using two point crossover.*

Figure 4.3 shows the growth of building blocks for a population of size $n = 2000$. As for the bit counting function there is very little difference between elitist recombination and tournament selection with standard recombination.

Figure 4.4 shows the growth of building blocks when the population is undersized ($n = 500$). Elitist recombination seems to perform only slightly better, but when we look at the actual number of times (out of 30 independent runs) the population converged to the global optimum (see Table 4.1), the difference becomes more pronounced. Clearly elitist recombination is less sensitive to undersized populations than tournament selection with standard recombination.

Conclusion An extremely simple GA implementation is discussed where selection and recombination take place simultaneously at the level of each family, which consist of two mating parents and their two offspring. The best two of each family survive and are included in the next generation. An analytical model is derived that accurately describes the convergence dynamics when optimizing the bit counting function. Analysis

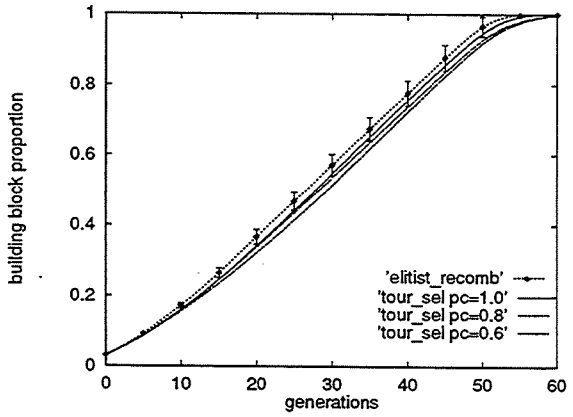


Figure 4.3: *Proportion of building blocks when optimizing a fully deceptive trap function with the elitist recombination GA compared to tournament selection with different crossover probabilities p_c (using two point crossover). Deception length $k = 5$, string length $l = 100$, signal $d = 0.2$ and population size $n = 2000$. Vertical bars show the $\mu \pm \sigma$ interval (average taken over 30 runs).*

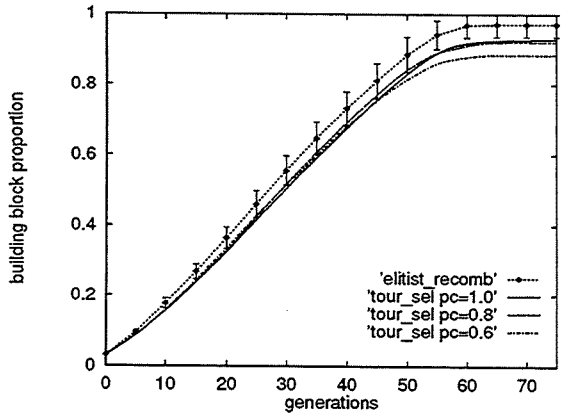


Figure 4.4: *Same as figure 3 but with a population size $n = 500$.*

and experiments show that the convergence characteristics of the elitist recombination GA are equivalent to those of the traditional tournament selection with tournament size $s = 2$. Experimental results also indicate that for functions where building block processing is needed, elitist recombination and tournament selection again behave very similar.

A number of advantages of elitist recombination were noted. First, good solutions are never lost during the search process. Second, elitist recombination is less sensitive to undersized populations than tournament selection with standard recombination. Third, there is no need to choose a particular value for the crossover probability p_c in order to minimize the disruption probability of the building blocks. Fourth, in contrast with other elitist mechanisms, the local elitist technique does not suffer from premature convergence problems. Fifth, elitist recombination is extremely simple to implement and well suited for parallel GA implementations. Finally, elitist recombination is ideal to be used in hybrid GAs where the genetic algorithm's global search is combined with the local search approach of most classical optimization algorithms to obtain the best of both worlds.

4.3 Niching

Many practical problems are multimodal function optimization problems and it is often desired to find multiple solutions simultaneously. The basic genetic algorithm however will ultimately converge to a population containing only one solution. Since GAs explore the search space from a population of points they also seem to be good candidates to search for multiple solutions simultaneously. The diversity of species that evolved in nature with each species adapted to its own specific environmental niche, can be looked upon as a multimodal optimization process. In this section we will argue that the use of niching is not only important for multimodal optimization but it is also useful in terms of the mixing problem.

In the previous chapter we have seen how in a basic GA, selection and mixing are actually involved in a race against time: in order for recombination to have enough time to juxtapose the optimal alleles or building blocks, the mixing time t_x has to be smaller than the selection

time t_s. When the problem is hard from a mixing point of view, the population size needs to be increased considerably to allow good mixing before the population converges due to the selective pressure. An alternative to the population size increase is the use of niching techniques. Niching basically keeps multiple good solutions in the population and prevents the GA from converging to a single string. Strings that are composed of different building blocks can thus coexist in the population and crossover has all the time it needs to mix or juxtapose the different building blocks. Put it another way, niching increases the selection time dramatically while leaving the mixing time untouched, and the condition for good mixing to occur - $t_x < t_s$ - is now easily fulfilled.

A number of mechanisms to induce niches in a genetic algorithm population have been proposed in the literature. These niching mechanisms can be classified into two classes: *sharing* schemes (Holland, 1975; Goldberg & Richardson, 1987; Deb, 1989; Oei, Goldberg & Chang, 1991) and *crowding* schemes (Cavicchio, 1970; De Jong, 1975; Mahfoud, 1992; Smith, Forest & Perelson, 1992; Mahfoud, 1993; Harik, 1994). Viewing the extensive literature on niching it is certainly outside the scope of this work to give a detailed discussion on it.

In the next section we will give a short description of the sharing and crowding techniques. Thereafter we will generalize the elitist recombination algorithm into a crowding algorithm called the generalized crowding algorithm. Finally we will look at the effect of the generalized crowding algorithm on the mixing process.

4.3.1 An overview of niching techniques

Niche formation through sharing

Sharing schemes induce niches by changing the fitness function according to the individuals present in the population. The idea is that similar individuals have to share their fitness value and thus discourage the presence of a large number of similar strings while encouraging the formation of individuals at places in the search space where no strings are present and where the fitness function is relatively high. In chapter 1 we have discussed Holland's analysis of the bandit problem and its relation to schema processing. Holland also introduced a modified bandit problem

which relates to niche formation through fitness sharing (Holland, 1975).

Consider a two-armed bandit with mean payoffs μ_1 and μ_2. In order for different niches to be created the payoffs have to be distributed over a set of players. Suppose we have a population of n players that each have to decide which arm they will play. In the original bandit problem the players would get the full payoff associated with the arm they selected. Those players that selected the arm with highest mean payoff μ receive in expectation the highest return. If the players reproduce according to their received payoff, more and more players would select the best arm until eventually the entire population would choose this arm.

To cause the creation of different niches the payoff scheme is modified as follows: each player in the population has to choose one of the arms. The payoff received however has to be shared with all the other players that have also selected that arm. For instance let $\mu_1 = 20$, $\mu_2 = 10$, and assume that 80 out of 100 players have chosen the first arm while the other 20 selected the second arm. Although the 80 players picked out the best arm, their expected fitness is now only $\frac{20}{80} = 0.25$ since they have to share with the others, while the fitness of the players that selected the worst arm have an expected fitness of $\frac{10}{20} = 0.5$ and thus will be more successfully reproducing to the next generation. An equilibrium will be obtained when:

$$\frac{\mu_1}{\bar{n}_1} = \frac{\mu_2}{\bar{n}_2}$$

with n_1 and n_2 the number of players that have chosen the first and second arm respectively.

Since $n_1 + n_2 = n$ the equilibrium equation can also be rewritten as:

$$\bar{n}_1 = \frac{\mu_1}{\mu_1 + \mu_2}.n$$

$$\bar{n}_2 = \frac{\mu_2}{\mu_1 + \mu_2}.n$$

The proportion of individuals clustered in a niche is thus proportional to the expected payoff at that niche.

Goldberg and Richardson (1987) introduced the notion of sharing functions to directly induce niches in practical genetic algorithm implementations. Sharing functions define the neighborhood and the degree

of sharing for each individual. Every string's fitness is degraded by an amount determined by the number of strings that are in its immediate neighborhood. The closer the individuals are, the stronger the degradation is:

$$f_s(i_k) = \frac{f(i_k)}{\sum_{j=1}^{n} sh(dst(i_k, i_j))}$$

with the sharing function defined as:

$$sh(d) = \begin{cases} 1 - \frac{d}{\sigma_{sh}} & \text{if } d \leq \sigma_{sh} \\ 0 & \text{otherwise} \end{cases}$$

and the distance function $dst(i_k, i_j)$ defined in the genotypic or phenotypic space. For binary genotypes the distance metric could be the Hamming distance, while for real valued phenotypes it could be the Euclidean distance.

Fitness sharing will allocate a number of individuals to a niche depending on the relative fitness of the niche. As proportionate selection, fitness sharing might thus have a need for scaling methods (see for instance (Goldberg, Deb & Horn, 1992)). An alternative method to let similar individuals compete with each other in order to create different niches are the crowding methods, which do not depend on the relative fitness of the individuals and are in that aspect similar to rank based selection methods.

Niche formation through crowding

De Jong (1975) originally proposed crowding by generalizing the preselection scheme of Cavicchio (1970). The idea is to maintain stable subpopulations or niches by replacing individuals by similar ones. By restricting the competition for limited resources to similar individuals, niches should emerge just like in nature different subspecies exist with their own environmental niche and without competition between the different niches.

In De Jong's crowding model a fixed percentage of the population reproduces and each new string replaces a similar individual. The string to be replaced is the closest string in Hamming distance chosen from

a small random subset of the population. In (Mahfoud, 1992) it is shown that this crowding model is sensitive to stochastic replacement errors which prevent the algorithm of maintaining stable subpopulations. Mahfoud proposes an improved crowding algorithm called *deterministic crowding*. All strings in the population are randomly paired and each pair is recombined to generate two children. A child can only enter the population if it is better than the parent with whom it has to compete. Since we have two parents and two children for each crossover operation there are two possible parent-child tournaments: parent 1 against child 1 and parent 2 against child 2 versus parent 1 against child 2 and parent 2 against child 1. The tournament with the closest parents and children is actually held.

Note that deterministic crowding is very similar to the elitist recombination GA we have introduced in the previous section. Both algorithms restrict competition to two randomly paired individuals and their two children. In elitist recombination the competition is held between all four individuals: the best two individuals of the two parents and the two children go to the next population. In deterministic crowding the competition is restricted to the pair of parent-child tournament with the closest distance to each other.

It is interesting to see that the two algorithms have such a common structure although they were both developed from totally different viewpoints. Deterministic crowding was developed as a hybrid of De Jong's crowding scheme and Cavicchio's preselection mechanism. De Jong and Cavicchio both wanted to design a GA that was able to induce and maintain stable niches to let the GA solve multimodal optimization problems. Their algorithms however showed some shortcomings (Mahfoud, 1992) and deterministic crowding was designed to overcome these problems.

Elitist recombination on the other hand was developed to make use of an elitist mechanism such that good solutions would never disappear from the population unless they were replaced with better ones. Originally the idea was simply to add the local elitist method to a basic GA using tournament selection. After selecting the parents with tournament selection they would generate their offspring by recombination and the children would then have to compete with their parents to be included in the population. The result of adding the local elitism to a basic GA was

a GA that had a very high selection pressure. Eliminating the separate selection phase led to the elitist recombination algorithm which turned out to have exactly the same selection pressure as regular tournament selection (with tournament size $s = 2$) but with the benefits of an elitist mechanism.

The choice of the particular competition scheme between the parents and their children thus has a dramatic effect: deterministic crowding is a GA that tries to maintain different niches, while elitist recombination converges to a single optimum as any basic GA, and more particularly it converges at the same rate as the widely used tournament selection.

4.3.2 The generalized crowding algorithm

De Jong's crowding algorithm selects randomly a subset of the population and replaces the string that matches the child most closely whether or not the child's fitness value is better. Deterministic crowding and elitist recombination let the children compete only with their parents but they need to have a better fitness value to enter the population.

Generalizing the subset selection mechanism and the local elitism principle we obtain a generalized crowding algorithm that captures the characteristics of both schemes:

Generalized Crowding algorithm:
1. randomly shuffle the population
2. for each pair of strings: create two children
3. for each child:
   a) randomly select $w-2$ strings from the population
   b) determine the closest string from these strings and the two parents
   c) let the child compete with this string

To understand how the generalized crowding algorithm maintains different solutions in the population it is instructive to compute how the growth of the highest valued string is limited and thus prevented from taking over a substantial part of the population and pushing other solutions out of the population.

Assume the best solution has a proportion of $p_i(t)$ strings, the population has size n and the crowding window size w. The number of copies of the best solution can only increase whenever a new copy competes with an element that is not a copy of the best string. This can only happen if none of the w strings in the crowding window are from the best set. Since the crowding window is randomly selected without replacement we have:

$$\begin{aligned} P[p_i(t+1) > p_i(t)] &= \frac{\binom{n(1-p_i(t))}{w}}{\binom{n}{w}} \\ &= (1-\frac{np_i(t)}{n})(1-\frac{np_i(t)}{n-1})\ldots(1-\frac{np_i(t)}{n-w+1}) \\ &< (1-p_i(t))^w \\ &< e^{-wp_i(t)} \end{aligned}$$

The first inequality is the probability of proportional increase with random selection with replacement and is almost an equality when the window size w is much smaller than the population size n. The second inequality is also almost an equality when $p_i(t) << 1$.

The probability that the number of copies of the best string increases is thus an exponentially decreasing function of the present proportion and the crowding window. Figure 4.5 plots the probability of a proportional increase of the best solutions when the crowding window is respectively $w = 10, 20$ and 30.

Recently Harik (Harik, 1994) introduced a very similar crowding algorithm called *Restricted Tournament Selection (RTS)*. RTS initially selects two elements from the population at random and generates two children from them. The two children are immediately tested for inclusion in the population. For each child a set of w elements is selected at random and the element closest to the child is replaced if its fitness is worse than the child's fitness.

RTS differs at two main points from the generalized crowding algorithm:

1. RTS does not include the parents in the crowding subset. The w elements selected for possible competition with the new string

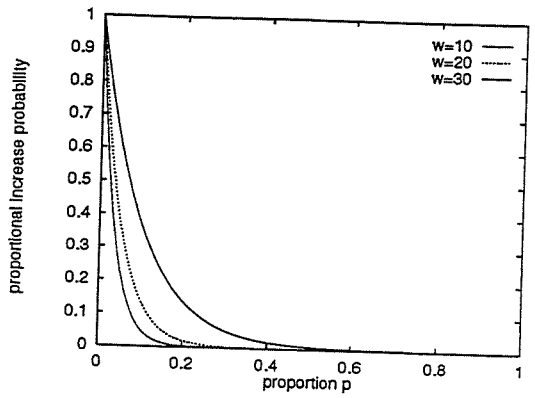


Figure 4.5: *Probability of a proportional increase of the best solutions when the crowding window is respectively $w = 10, 20$ and 30.*

are taken at random. Because the parents are not necessarily a member of this set there is no elitist mechanism between the parents and the offspring as in elitist recombination and deterministic crowding.

2. In the generalized crowding algorithm all strings create offspring and after that it is tested whether or not these new strings should be included in the population. The generalized crowding algorithm is thus a generational GA. RTS on the other hand does not work with separate generations but instead the children are immediately tested for inclusion. RTS is thus a member of the steady state GAs (Whitley, 1989; Eshelman, 1991). This makes that some strings can be replaced without ever having a chance to generate offspring and due to the stochastic sampling some strings will be selected more often than others to create children.

4.3.3 Example: multimodal deceptive function with linkage

To see the effect of the local elitism and the niching mechanism we optimize the multimodal deceptive function of Section 1.5.2 with the

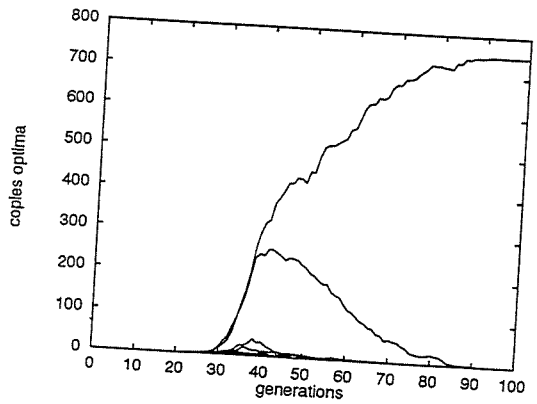


Figure 4.6: *Number of copies of each of the 32 global optima for the multimodal deceptive function that are present in the population when using the basic genetic algorithm with tournament selection.*

following GA implementations:

1. tournament selection
2. elitist recombination
3. generalized crowding

The specific function used had 5 concatenated building blocks of length $k = 6$ ($\Rightarrow$ string length $l = 30$) and a signal $d = 0.2$. The function has thus $(\binom{6}{3} + 2)^5 = 22^5 = 5.153.632$ optima of which $2^5 = 32$ are global. The population size used in the experiments was $n = 750$. We assume that the building blocks are tightly linked so we use one-point crossover.

Figure 4.6 shows the number of copies of each of the 32 global optima that are present in the population when using the basic genetic algorithm with tournament selection. The plot is obtained from a typical run, and other runs starting from a different random population gave equivalent results. After about 30 generations two global optima are found and they rapidly take over a large part of the population. Although the global optima have the same fitness value, they cannot be kept together

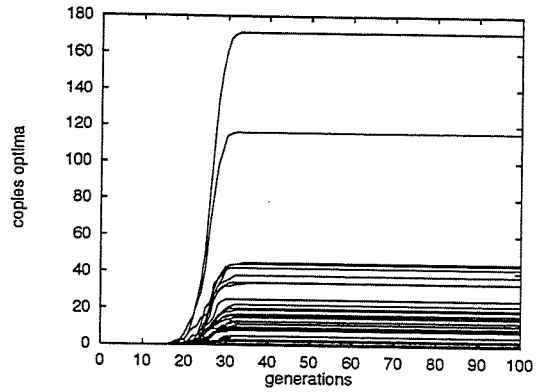


Figure 4.7: *Number of copies of each of the 32 global optima for the multimodal deceptive function that are present in the population when using the elitist recombination genetic algorithm.*

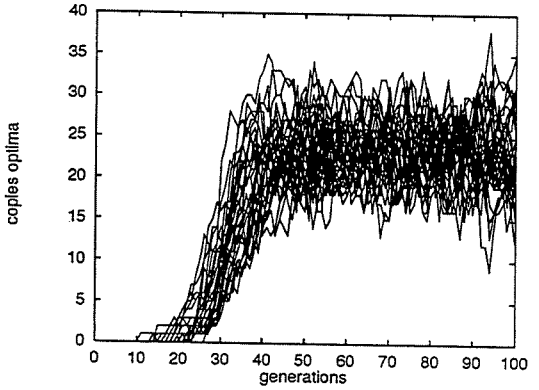


Figure 4.8: *Number of copies of each of the 32 global optima for the multimodal deceptive function that are present in the population when using the generalized crowding genetic algorithm.*

in the population due to the stochastic sampling of tournament selection which causes genetic drift (Goldberg & Segrest, 1987; Oei, Goldberg & Chang, 1991), and finally the population consists entirely of copies of one of the global optima.

Figure 4.7 shows the results of a typical run with elitist recombination. Most of the global optima have some copies present in the population, but in a very uneven proportion. Once the entire population consists of global optima no new children ever enter the population. This can easily be explained since elitist recombination requires the children to be better than their parents, and since all global optima have equal fitness value no further progress can be made. If the global optima had only a small fitness difference, then the population would also convergence to a single optimum, namely the one with the highest fitness.

Contrary to tournament selection and elitist recombination, the generalized crowding algorithm has a specific niching mechanism and thus is capable of sustaining all global optima in the population. Figure 4.8 shows how the number of copies of the global optima fluctuates around their expected number $n/2^k = 750/32 = 23.4$.

4.3.4 Example: Royal Road function

Holland (Holland, 1993; Jones, 1994) introduced an extended version of the Royal Road functions originally proposed in (Mitchell, Forrest & Holland, 1992) as a challenge to all GA implementations. It is claimed that many published GA implementations fail the test. The basic challenge is to (1) *discover* the elementary (lowest-level) building blocks used to define the function, and then (2) *combine* building blocks to form building blocks at progressively higher levels in a hierarchy of building blocks. The goal is to see how rapidly and consistently a particular GA implementation climbs successive levels of the hierarchy.

The Royal Road function assigns a real valued fitness to a binary string. The string is composed of 2^k contiguous regions, each of length $b+g$ bits. Each region is divided into two separate subregions: the first of length b is called the *block* while the second of length g is called the *gap*. The fitness calculation only looks at the blocks and is obtained by adding two independent fitness calculations: the PART fitness and

# 1-bits	0	1	2	3	4	5	6	7	8
PART	0.00	0.02	0.04	0.06	0.08	-0.02	-0.04	-0.06	0.00

Table 4.2: *The PART fitness for a block of length 8 in function of the number of target bits 1 present in the block.*

the BONUS fitness. The gap regions play no role in determining the fitness value but they facilitate the recombination of building blocks for one-point crossover since crossing somewhere in the gap region has no effect on the fitness calculation. The gaps are in analogy with introns in natural chromosomes (Levenick, 1991).

PART fitness

The PART fitness value is computed for each of the 2^k blocks individually and is determined only by the number of 1 bits it contains. A given block with i 1 bits has a PART fitness equal to:

$$\text{PART(i)} = \begin{cases} iv & \text{if } i \leq m^* \\ -(i-m^*)v & \text{if } m^* < i < b \\ 0 & \text{if } i = b \end{cases}$$

where b is the number of bits in the block, v is a fitness factor, and m^* specifies a threshold in the number of bits. Below the threshold value an increasing number of the target bits gives a higher payoff, while once above the threshold the fitness becomes negative. A simple hillclimbing algorithm would thus rapidly find m^* correct bits but would then get stuck.

Table 4.2 shows the PART fitness values for building blocks of length $b = 8$, with a fitness scaling factor $v = 0.02$ and a threshold $m^* = 4$.

BONUS fitness

The BONUS fitness rewards completed building blocks - this is with the number of 1 bits equal to the building block length b - and some combinations of completed blocks in a hierarchical way. There are a

total of $k+1$ levels ($0 \leq j \leq k$). At the lowest level $j = 0$ a BONUS fitness is given to each completed building block: the first completed block gets a fitness of u^* and each additional completed block receives a fitness of u.

At the next level $j = 1$, pairs of adjacent completed building blocks are rewarded. Call B_i a building block at position i ($0 \leq< i < 2^k$) then each block pair (B_{2i}, B_{2i+1}) receives a fitness calculated the same way as in level 0: the first completed pair gets u^* and any additional pair u.

The rewards for the higher levels are calculated in the same way. At each level j, groups of non-overlapping completed building blocks are rewarded with a fitness value u^* for the first completed group and u for each additional completed group. Each group consists of 2^j building blocks B_i and the BONUS depends on $t(j)$, the number of completed groups at level j.

$$\text{BONUS(j)} = \begin{cases} u^* + (t(j)-1)u & \text{if } t(j) > 0 \\ 0 & \text{if } t(j) = 0 \end{cases}$$

Note that the groups are non-overlapping and start at the leftmost bit of the binary string.

Royal Road function default values

Holland gave a set of default values for the Royal Road function:

- $k = 4$: The number of building blocks is $2^k = 16$ and the number of levels is $k + 1 = 5$.
- $b = 8$: The number of bits in each block is 8.
- $g = 7$: The number of bits in the gap is 7. The total length of the binary string is thus $(b+g)2^k = 240$.
- $m^* = 4$: The threshold in the fitness PART calculation is 4 bits.
- $v = 0.02$: The fitness PART scaling factor is 0.02.
- $u^* = 1.0$: The fitness BONUS for each completed group of blocks at each level is 1.

- $u = 0.3$: The fitness BONUS for each additional completed group is 0.3.

The global optimal string has all 16 blocks completely filled with 1 bits. The PART fitness for completed blocks is 0, while the BONUS fitness is calculated at each level:

1. level 0 has 16 blocks B_i $(0 \leq i < 16)$.
   The bonus fitness is $u^* + u(2^k - 1) = 1.0 + 0.3 * 15 = 5.5$.
2. level 1 has 8 groups of blocks $B_{2i}B_{2i+1}$ $(0 \leq i < 8)$.
   The bonus fitness is $u^* + u(2^{k-1} - 1) = 1.0 + 0.3 * 7 = 3.1$.
3. level 2 has 4 groups of blocks $B_{4i}B_{4i+1}B_{4i+2}B_{4i+3}$ $(0 \leq i < 4)$.
   The bonus fitness is $u^* + u(2^{k-2} - 1) = 1.0 + 0.3 * 3 = 1.9$.
4. level 3 has 2 groups of blocks $B_{8i}B_{8i+1} \ldots B_{8i+6}B_{8i+7}$ $(0 \leq i < 2)$.
   The bonus fitness is $u^* + u(2^{k-3} - 1) = 1.0 + 0.3 * 1 = 1.3$.
5. level 4 has 1 group of blocks consisting of all 16 blocks B_i.
   The bonus fitness is $u^* = 1.0$.

The total fitness of the best string is thus $5.5+3.1+1.9+1.3+1.0 = 12.8$

Table 4.3 and Figures 4.9 and 4.10 show the results for the elitist recombination GA and the generalized crowding GA for a population of size $n = 512$. Results are shown for 25 independent runs: both algorithms have no trouble finding the block groups at level 2. At levels 3 and 4 not all the block groups are always found, although the crowding algorithm performs better than elitist recombination.

Holland advocated the use of a population of size $n = 512$ because he wants to be certain that not all complete blocks at level 0 are present in the initial random population. Since the blocks are 8 bits long, the probability that a particular building block is not present in a random population of size 512 is:

$$P[\text{BB not present}] = (1 - \frac{1}{256})^{512} = 0.135$$

level	max. BBs	elitist recombination		generalized crowding	
		avg. BBs	# runs/25	avg. BBs	# runs/25
0	16	13.2	25	14.8	25
1	8	5.56	25	6.76	25
2	4	1.96	25	2.76	25
3	2	0.40	9	0.96	21
4	1	0.04	1	0.12	3

Table 4.3: *Results for the elitist recombination GA and the generalized crowding GA on Holland's Royal Road function with a population size $n = 512$. The # runs shows the number of runs from a total of 25 that reached the corresponding level in the building block hierarchy. A level is reached when one of the building blocks of that level is found. The avg. BBs shows the average of the number of building blocks that are found at each level for the 25 runs. This can be compared with the maximum number of building blocks that are present at each level.*

level	max. BBs	elitist recombination		generalized crowding	
		avg. BBs	# runs/25	avg. BBs	# runs/25
0	16	15.4	25	15.8	25
1	8	7.44	25	7.84	25
2	4	3.44	25	3.84	25
3	2	1.56	24	1.84	25
4	1	0.60	15	0.84	21

Table 4.4: *Results for the elitist recombination GA and the generalized crowding GA on Holland's Royal Road function with a population size $n = 1000$. The # runs shows the number of runs from a total of 25 that reached the corresponding level in the building block hierarchy. A level is reached when one of the building blocks of that level is found. The avg. BBs shows the average of the number of building blocks that are found at each level for the 25 runs. This can be compared with the maximum number of building blocks that are present at each level.*

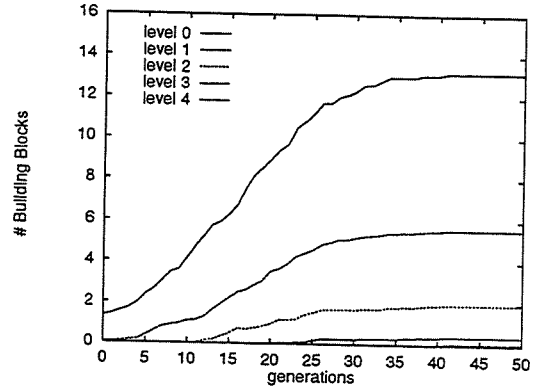


Figure 4.9: *The average number of building blocks (avg. BBs in Table 4.3) obtained at each level for the Royal Road function with the elitist recombination GA and a population size $n = 512$. The maximum number of blocks at level j is 2^{4-j}.*

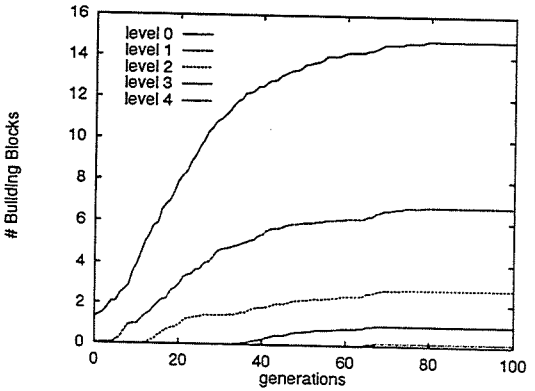


Figure 4.10: *The average number of building blocks (avg. BBs in Table 4.3) obtained at each level for the Royal Road function with the generalized crowding GA and a population size $n = 512$. The maximum number of blocks at level j is 2^{4-j}.*

or we can expect on average to have $0.135 * 16 = 2.16$ blocks missing.

This way Holland wanted to test whether a particular GA implementation is capable of not only recombining building blocks but also to discover them by using mutation. In our experiments however we have not used any mutation since we are only interested in the mixing capability the GA. Note also that one of the conditions for GA optimization as expressed by Goldberg (see also Section 1.7) was the adequate initial supply of building block, and if we use no mutation this supply can only come from a larger population.

We therefore did the same experiments with a population of size $n = 1000$. The probability that a particular building block is not present in a random population is now:

$$P[\text{BB not present}] = (1 - \frac{1}{256})^{1000} = 0.02$$

Results are shown in Table 4.4. The generalized crowding GA now finds the global optimum in most cases and has a better performance than elitist recombination.

4.4 Restricted Mating

Extending the genetic algorithm with a local elitist mechanism and with a niching technique helps the basic GA in overcoming some aspects of the mixing problem. Elitism makes sure that good solutions do not get lost unless they can be replaced by better ones. Niching eliminates the race against time that is going on in a basic GA between selection and recombination. Both mechanisms however do not change the mixing probability p_{mix}. The mixing ladder climbing analysis from the previous chapter showed that the mixing probability for bounded deceptive problems is extremely low:

$$p_{mix} = \frac{m_l(m - m_l)}{\binom{m}{m_l}} \frac{2}{2^{\mu k}}$$

where m is the total number of building blocks, m_l the number of building blocks that are already juxtaposed and k the deception length (μ is just a factor between 1 and 2).

In order to reduce the mixing complexity we have to increase the mixing probability p_{mix}. Recall that for a given deception length k the mixing probability p_{mix} was minimal when strings had half of the building blocks juxtaposed or $m_l = m/2$. The mixing probability was in this case :

$$p_{mix} = \frac{\sqrt{2\pi}}{4} \frac{m^2\sqrt{m}}{2^m} \frac{1}{2^{\mu k}}$$

The exponential term in the denominator causes the mixing to become intractable with increasing number of building blocks for a basic GA without a priori linkage information.

The mixing probability p_{mix} decreases exponentially with both the deception length k and with the number of building blocks m. The exponential dependence on the deception length however should not concern us: the use of bounded but fully deceptive problems was just intended to push the GA to its limits and each deceptive subfunction is basically a needle in a haystack so there is no information in the search space that might help any search algorithm of finding it. Therefore the population sizing equation as discussed in Section 1.6 also gave us an exponentially increasing complexity: $n \propto 2^k$.

The mixing probability for the basic GA without a priori linkage information, is also exponentially dependent on the number of building blocks. From the previous chapter (Section 3.3) we recall that the factor $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ expresses the probability that two randomly chosen strings with m_l building blocks are neighboring strings in Hamming space. This means that the two strings each have only one building block that is not matched by the other string, e.g.:

$p1$: 111 000 000 <u>111</u> 111 000 000 111
$p2$: 111 000 <u>111</u> 000 111 000 000 111

An apparently obvious way to increase the mixing probability is to replace the random selection of mating strings by a restricted mating mechanism. For every mating string we scan the population until we find a neighboring string, so the factor $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ disappears and the

mixing probability becomes $p_{mix} = 2^{1-\mu k}$. This assumes that the population is large enough so we can find a neighboring string. Suppose - as we did in the building block mixing analysis - that all strings have m_l building blocks juxtaposed. The probability that a string has a neighboring string in a population of size n is then:

$$P[neighbor] = 1 - \left(1 - \frac{m_l(m-m_l)}{\binom{m}{m_l}}\right)^n$$

This probability is lowest when the fraction $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ is smallest, or when $m_l = m/2$. Taking this value for m_l we can compute the necessary population size n to let the probability $P[neighbor]$ come arbitrary close to 1 as:

$$1 - \left(1 - \frac{\sqrt{2\pi}}{8} \frac{m^2\sqrt{m}}{2^m}\right)^n > 1 - \epsilon$$

or

$$n > \ln \epsilon^{-1} \, 4\sqrt{\frac{2}{\pi}} \, \frac{2^m}{m^2\sqrt{m}}$$

The value of ϵ depends on how close to 1 we want the neighboring probability to be. For ϵ resp. equal to $0.1, 0.05$ and 0.01 the constant factor $\ln \epsilon^{-1} \, 4\sqrt{2/\pi}$ becomes $7.0, 9.6$ and 14.7, so a reasonable value might be 10:

$$n > 10 \, \frac{2^m}{m^2\sqrt{m}}$$

Of course it is not necessary for every string in the population to have a neighboring string but even if we relax $P[neighbor]$ considerably the exponential increase in the number of building blocks remains overwhelming.

Extending the basic genetic algorithm with elitism, niching and restricted mating has only minor effects on the GA's mixing capability when no a priori linkage information is used. The three methods discussed did not alter the recombination operator itself and instead tried to

improve the GA's mixing performance by working on some side effects. In the next section we will introduce a method that directly changes the crossover exchange distribution in function of the current state of the search process as expressed implicitly by the current population.

4.5 Adaptive Crossover

4.5.1 Adaptive crossover algorithm

The previous three sections showed how to extent the basic GA to improve its search properties. Each of the three mechanisms helped to solve a separate subproblem of the overall optimization problem:

- **Elitism** made sure that once a good solution was found it would not be thrown away for some low value offspring. Good solutions can only be replaced by better ones, so the computational effort that went into finding good strings is maximally exploited.

- **Niching** made sure that the population kept all building blocks into the population by preventing the cross-competition between them. The times race between growing building blocks and mixing them - this is between selection and recombination - that characterized the basic genetic algorithm can thus be side stepped.

- **Restricted mating** made sure that the mixing probability was increased by restricting crossover to strings that are close together in Hamming space. This way most of the building blocks are matched in the two mating strings so they cannot be destroyed by the extremely disruptive uniform crossover operator.

Integrating these extensions into one algorithm gives us the following extended genetic algorithm:


```
EXTENDED GENETIC ALGORITHM
  A. create random population
  B. evaluate population
  C. for t: 1 → max_generation
    1. shuffle population
    2. select mates:
        for i: 1 → pop_size
          for j: (i+1) → (i+1+mating_window)
            mates(i) = closest_string(j)
    3. create offspring:
        for i: 1 → pop_size
          crossover(pop(i),mates(i))
          evaluate(offspring)
    4. insert offspring:
        for i: 1 → pop_size
          for j: i → (i+crowding_window)
            competitor(i) = closest_string(j)
            if fitness(competitor(i)) < fitness(child(i))
              replace competitor(i) with child(i)
```

Elitism, niching and restricted mating help to ameliorate the optimization performance of the genetic algorithm but the building block mixing probability p_{mix} is still very low due to two reasons:

1. The probability that two parents are neighboring strings in Hamming space is significantly higher when using restricted mating but it is still substantially low for problems with a large number of building blocks m.

2. The mixing probability p_{mix} is proportional to $2^{-\mu k}$ and thus the mixing of building blocks becomes very unlikely with increasing deception length k and with increasing μ-factor. Recall that when the strings in the population consist only of building blocks and deceptive attractors, the factor μ is equal to two.

To illustrate the remaining problem it is instructive to look at an example: we have run the extended GA as outlined above on the fully deceptive trap function with 10 building blocks of deception length $k = 5$,

a population size $n = 500$, mating window size $w_m = 50$ and crowding window size $w_c = 25$. We assume no a priori linkage information and use uniform crossover. Some typical strings after 25 generations were:

p_1 : 00000000000000011111000000000000000000000000000000
p_2 : 00000000000000000000000000000011111000000000000000
p_3 : 00000111110000000000000010000000000000000000011111
p_4 : 11111000001111100000000000000000000111111111100000
p_5 : 11111000000000000000000000000000000000100000000000

Due to the elitism and niching the entire population consists almost solely of building blocks and deceptive attractors. The μ–factor in the mixing probability is thus approximately equal to 2: $p_{mix} = \frac{2}{2^{2k}}$.

During the first generations the GA quickly identifies the locally optimal bit combinations but then its performance is severely hampered by the destructiveness of uniform crossover. Uniform crossover is a blind recombination operator and the probability of exchanging bit values is equal at all bit positions. Suppose we want to recombine the first two strings of the above example:

p_1 : 00000000000000011111000000000000000000000000000000
p_2 : 00000000000000000000000000000011111000000000000000

There are 10 bits where the two strings have a different value which gives us a total of $2^{10}/2 = 512$ possible recombinations. Just looking at the two mating partners gives us no information at all on how to exchange the bits and thus a crossover operator that only takes two strings as its input, and has no linkage information, can do no better than randomly swapping alleles with some probability p_x.

When looking at more strings together however, one realizes that there is an enormous amount of information in the population that the crossover operator simply ignores. The key point is that in order to exploit this information we have to change the swapping distribution in function of the current state of the search process which is expressed by the current population. Therefore if we make the probability for a bit value to be exchanged dependent on the current state of the population, we might exploit the information that is implicitly present.

Looking only at two mating strings, no information can be extracted to guide the crossover operation. However, if we simply include one more string, the picture changes drastically. Taking for instance the third string of the examples and considering only the bits where the parents differ we have:

p_1 : ...11111...00000...
p_2 : ...00000...11111...
p_3 : ...00000...00000...

If we now only exchange the bits where one parent differs with the randomly chosen string we can immediately juxtapose the two building blocks. This leads us to the following crossover operator:

ADAPTIVE CROSSOVER
1. p_1 = parent_1
2. p_2 = parent_2
3. p_3 = pop(rand)
4. $\forall$ bits $i = 0, ..., l-1$:
   if $(p_1(i) \neq p_2(i)$ and $p_1(i) \neq p_3(i))$
   exchange $p_1(i)$ and $p_2(i)$

When working at actual bit level an extremely fast implementation is obtained by creating a crossover mask, i.e. a binary string that has a bit value '1' whenever the alleles should be exchanged at that position. The children are then created by XOR-ing the parent with the mask.

ADAPTIVE CROSSOVER
1. p_1 = parent_1
2. p_2 = parent_2
3. p_3 = pop(rand)
4. mask = AND(XOR(p_1, p_2), XOR(p_1, p_3))
5. c_1 = XOR(p_1,mask)
5. c_2 = XOR(p_2,mask)

p_1	p_2	p_3	mask	c_1	c_2
1	1	1	0	1	1
1	1	0	0	1	1
1	0	1	0	1	0
1	**0**	**0**	**1**	**0**	**1**
0	**1**	**1**	**1**	**1**	**0**
0	1	0	0	0	1
0	0	1	0	0	0
0	0	0	0	0	0

Table 4.5: *All possible bit combinations and the result after applying adaptive crossover: p_1, p_2 are the parents, p_3 is the assisting string and c_1, c_2 are the children. The mask indicates when the parent alleles will be swapped.*

In Table 4.5 we have shown the outcome of all possible bit value combinations. The two rows in boldface are the two bit combinations where actual allele swapping takes place.

Note that the operator itself is not adaptive, but due to the changing population the allele swapping distribution is changed and reflects the current status of the search process.

We finally obtain for our extended genetic algorithm the same algorithm as outlined above but with the crossover operator replaced by the adaptive crossover operator:


```
EXTENDED GENETIC ALGORITHM
  A. create random population
  B. evaluate population
  C. for t: 1 → max_generation
    1. shuffle population
    2. select mates:
         for i: 1 → pop_size
           for j: (i+1) → (i+1+mating_window)
             mates(i) = closest_string(j)
    3. create offspring:
         for i: 1 → pop_size
           adaptive_crossover(pop(i),mates(i))
           evaluate(offspring)
    4. insert offspring:
         for i: 1 → pop_size
           for j: i → (i+crowding_window)
             competitor(i) = closest_string(j)
             if fitness(competitor(i)) < fitness(child(i))
               replace competitor(i) with child(i)
```

4.5.2 Adaptive crossover analysis

To analyze the adaptive crossover operator we compute a number of probabilities which can be compared to the results of the previous chapter for uniform crossover:

1. First we calculate the mixing probability when there are only building blocks and deceptive attractors in the population. Call the probability of having a building block p_{BB}. Two building blocks exchange successfully if the corresponding bits at the third string (p_3) are either two building blocks or two deceptive attractors:

p_1 : ...11111...00000...
p_2 : ...00000...11111...

p_1 : ...11111...00000...
p_2 : ...00000...11111...

p_3 : ...00000...00000... p_3 : ...11111...11111...
c_1 : ...00000...00000... c_1 : ...11111...11111...
c_2 : ...11111...11111... c_2 : ...00000...00000...

The probability of exchanging two building blocks is thus:

$$p_{mix} = p_{BB}^2 + (1 - p_{BB})^2$$

Since the building blocks and deceptive attractors are always processed as one unit, the building block mixing behavior is comparable to the allele mixing when optimizing the bit counting function with uniform crossover (see Section 2.2). For uniform crossover with swapping probability p_x the allele exchange probability is $2p_x(1 - p_x)$ which has a maximum value of 0.5 when $p_x = 0.5$. The building block exchange probability for adaptive crossover is at least 0.5 (this is when $p_{BB} = 0.5$), thus the mixing of the building blocks is even faster than the $O(\sqrt{\ell})$ mixing of single alleles with uniform crossover.

2. In general the population does not consist solely out of building blocks and deceptive attractors. For instance, certain parts of the string might not be deceptive or - even for those parts that are deceptive - during the initial generations there are not only building blocks and deceptive attractors present. Calling the probability of having an optimal bit '1' in a non-building block p, a number of important building block processing probabilities can be computed.

   Building block exchange probability: Building blocks can only transfer as one unit if the third string has a building block or a deceptive attractor at the corresponding position. If there is a bit '1' in the third string then there also has to be a bit '1' in the mating string at the same position in order for the building block to survive the crossover operation.

   p_1 : ...1$\underline{1}$111...0$\underline{1}$00$\underline{1}$...
   p_2 : ...0$\underline{1}$000...1$\underline{1}$111...
   p_3 : ...0$\underline{1}$000...0000$\underline{1}$...

c_1 : ...01000...01001...
c_2 : ...11111...11111...

The probability of exchanging two building blocks now becomes:

$$p_{mix} = p_{BB}^2 + [(1 - p + p^2)^k (1 - p_{BB})]^2$$

where $1 - p$ is the probability that there is no bit '1' in the third string and p^2 the probability that there is a bit '1' in the third string and in the mating string. For example if $k = 5$ and if on average one of the bits in the non-building blocks is '1' ($\Rightarrow p = 0.2$), then the building block exchange probability becomes: $p_{mix} = p_{BB}^2 + 0.175(1 - p_{BB})^2$.

Building block creation probability: When the two mating strings have no building block at the same position but they both have a number of optimal alleles '1' at the non-building blocks then a building block might be created. To calculate this probability a number of situations have to be considered:

- The probability the two parents have a non-building block is simply $(1 - p_{BB})^2$.
- The probability that both the parents have an optimal allele '1' at the same position is p^2.
- When the parents have opposing allele or bit values then we have to consider at which child the building block is created. If the building block is created at the first child c_1 then the alleles of the parents p_1 and p_2 and the third string p_3 have to be respectively: $1, 0$ and 1 or $0, 1$ and 1. The probability this occurs is $2p^2(1 - p)$.
  If the building block is created at the second child c_2 then the alleles of the parents p_1 and p_2 and the third string p_3 have to be respectively: $1, 0$ and 0 or $0, 1$ and 0. The probability this occurs is $2p(1 - p)^2$.

The probability a building block is created at some position thus becomes:

$$p_{crt} = (1 - p_{BB})^2 \left([p^2 + 2p^2(1 - p)]^k + [p^2 + 2p(1 - p)^2]^k \right)$$

4.5.3 Example: fully deceptive function without linkage

		w=10			w=30			w=50		
m	n^*	opt	gen	std	opt	gen	std	opt	gen	std
10	843	5	17	1.8	5	19	2.7	5	21	1.2
20	1760	5	28	3.5	5	38	4.4	5	33	6.0
30	2697	4	-	-	5	134	79.2	5	122	31.0
40	3627	0	-	-	3	-	-	4	-	-
50	4557	0	-	-	0	-	-	0	-	-

Table 4.6: *Extended GA with adaptive crossover on a fully deceptive function (deception length $k = 5$, signal $d = 0.2$, population size $n = 1000$). The table gives the results for 5 independent runs and shows the generations needed to find the global optimum. The m column is the number of building blocks that were used in the function, thus the string length is $l = 5m$. The n^* column shows the recommended population size computed by the sizing equation. Results are shown for three crowding window sizes $w = 10, 30$ and 50. The opt column is the number of runs that found the global optimum. The gen column is the average number of generations needed to find the global optimum. The std column shows the standard deviation of the number of generations needed.*

The extended GA with adaptive crossover is tested on a fully deceptive function with a deception length $k = 5$ and a signal $d = 0.2$. No linkage assumptions are made thus the building blocks can be distributed randomly on the binary string. Results are shown in Table 4.6 for different number of building blocks ($m = 10, \ldots, 50$) and for different crowding window sizes ($w = 10, 30, 50$).

The sizing equation gives a a recommended population size $n^* = 93(m-1)$ or approximately $n \approx 100m$. In the experiments we have used a population of size $n = 1000$. The results show that even for $m = 30$ building blocks the extended GA still finds the global optimum.

4.5.4 Example: multimodal deceptive function without linkage

A final test is the multimodal deceptive function discussed before , only now we make no linkage assumptions and replace the one-point crossover by the adaptive crossover operator. The function has 5 concatenated building blocks of length $k = 6$ ($\Rightarrow$ string length $l = 30$) and a signal $d = 0.2$. It has $(\binom{6}{3} + 2)^5 = 22^5 = 5.153.632$ optima of which $2^5 = 32$ are global. The population size used is $n = 750$, the crowding window $= 25$ and the mating window $=10$.

Results are plotted in Figure 4.11 and should be compared to the results in Figure 4.8). The extended GA is capable of finding all 32 global optima even without the tight linkage assumption.

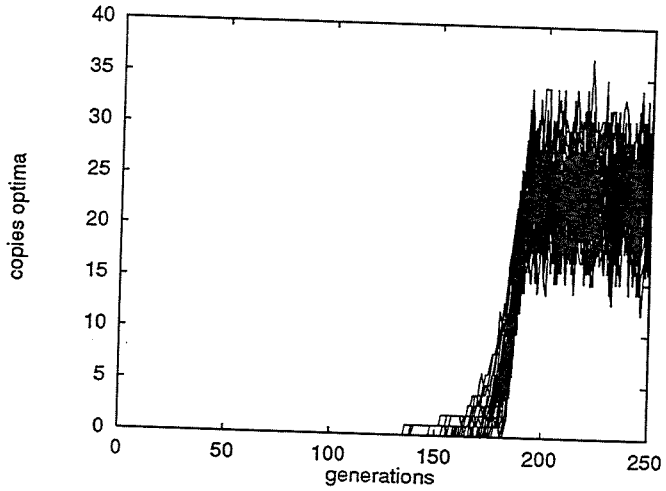


Figure 4.11: *Results of the extended GA with adaptive crossover on the multimodal deceptive function without linkage assumptions. The plot shows the number of copies of each of the 32 global optima that are present in the population for a typical run.*

4.6 Conclusion

We have proposed and analyzed some extensions to the basic genetic algorithm. First the elitist recombination GA was designed and shown to be similar to tournament selection with regular recombination, but the local elitism mechanism has a number of additional benefits. A second extension was the generalized crowding algorithm which is a natural extension of the elitist recombination GA. Niching has traditionally been used for optimizing multimodal problems, it is shown that even for unimodal functions niching is a valuable technique since it allows us to side step the race between selecting and mixing the building blocks, which was shown to be a serious problem for the basic genetic algorithm in the previous chapter.

Finally we investigated the benefit of adapting the crossover recombination distribution at run time. Traditional crossover operators do not change their recombination pattern during the search process. We have shown how the building block exchange can be made much more efficiently when the crossover operator is made adaptive such that its recombination probability distribution reflects the current state of the population.

From a mixing point of view, the elitism, niching and restricted mating extensions allow the building blocks to remain present in the population, but they only marginally help the building block mixing process itself.

An extremely important conclusion is that the extensions to the basic genetic algorithm are only marginally helping to solve the mixing problem. To solve mixing difficult problems, like the fully deceptive functions without linkage information, we have to design the genetic algorithm such that the building block mixing is explicitly taken care off.

We have shown one way how this could be done: by changing the recombination probability distribution such that the building blocks are properly exchanged, the mixing became very fast. The adaptive crossover operator was tested on fully deceptive functions without using linkage information, and was able to solve them. These results are very encouraging, but the adaptive crossover should be further studied and

modified when needed. For instance one open question is how the adaptive crossover will behave when used on functions that are not fully but partially deceptive.

An alternative is to use the messy genetic algorithm (mGA). Work on this non traditional GA has started in the late eighties and improvements to the initial version continues to be an active research topic today (Goldberg, Korb & Deb, 1989), (Goldberg, Deb & Korb, 1990, 1991), (Deb & Goldberg, 1991), (Deb 1991), (Goldberg, Deb, Kargupta & Harik, 1993), (Kargupta & Goldberg, 1994), (Harik, 1994). The primary goal to develop the mGA was to overcome the linkage problem of the basic genetic algorithm and to obtain a modified GA that could solve problems of bounded deceptive difficulty quickly and reliably.

It would certainly lead us to far to discuss the messy GA in detail, but results have proven that by first getting the linkage right and then solve the problem that has now become mixing easy, difficult deceptive problems can be solved without linkage information.

A very important conclusion to be drawn from this chapter is that, in order to increase the performance of genetic algorithms on hard problems, it is necessary to make sure that the mixing of building blocks is properly respected.

Chapter 5

General Conclusions

In this thesis we have tried to get an insight into the dynamical characteristics or convergence properties of genetic algorithms. To conclude, we review the main lessons that can be drawn from this work.

Bounds on convergence time complexity. A good understanding has been obtained of the convergence time complexity for problems with independent gene fitness contributions. More specifically, a lower and upper bound have been derived for the widely used tournament selection algorithm in combination with uniform crossover.

The lower bound was computed by normal distribution modeling, and resulted in an accurate model with convergence time complexity of order $O(\sqrt{\ell})$, where ℓ is the length of the binary string.

The upper bound was computed by domino convergence modeling, and resulted in a wave equation model with constant wave speed, and thus with convergence time complexity of order $O(\ell)$.

Clearly, for functions with independent gene fitness contributions, the convergence to the global optimum is very fast.

Previous analysis of tournament selection only looked at the convergence speed when using selection only - thus without using recombination - and basically computed how many generations it took for the initial best string in the starting population, to fill the population with copies of itself. By studying selection and recombination together, we

have derived models that can accurately predict the actual proportion in the population of the optimal bit values at each gene for every generation. Moreover, the obtained results give us important insights that are needed for building a mixing model using dimensional analysis.

Mixing analysis. The convergence time bounding models assumed that the population size was large enough to allow reliable convergence to the global optimum. In a real genetic algorithm application though, we want to minimize the number of function evaluations, so we want to use the minimal population size needed for optimal convergence.

When the population size is too small, the population will prematurely converge. The concept of premature convergence is well known in the GA literature, but it has hardly been studied in a formal way. We have argued that premature convergence is basically caused by a lack of sufficient time for recombination to juxtapose the necessary bit values. In the basic genetic algorithm, selection and recombination are involved in a race. Selection will cause the population to convergence, while in the meantime recombination has to exchange or mix the gene values, such that the optimal values are all juxtaposed.

If selection goes too fast relative to the recombination rate, then crossover cannot mix well enough, and the population will be converged before all necessary pieces are juxtaposed. We have called this a mixing failure, and used dimensional analysis to quantify this mixing in terms of the key parameters, namely the selection pressure s, the recombination rate r, the population size n, and string length l.

This resulted in the formalization of the mixing boundary in the genetic algorithm parameter space, that specifies whether for a certain parameter combination the GA will converge to the global optimum, or whether it will prematurely converge due to mixing failure.

The mixing law was derived for uniform crossover. We have also shown that for linkage-biased crossover operators like one-point crossover, the mixing becomes much slower, and as a consequence the risk of premature convergence due to mixing failure is much higher. This can easily be understood if one looks at the covariance build-up between the alleles for successive generations. Due to the selection of the better strings in a

population, the bit values or alleles become correlated. Crossover only partly decorrelates them again, so the net effect is a covariance build-up. Since one-point crossover has a much lower decorrelation rate than uniform crossover, the covariance build-up will be much higher, and the risk of a mixing failure increases.

Building block mixing. The mixing analysis for fitness functions with independent gene fitness contributions has led to a clear understanding of the quality and rate of convergence. These fitness functions are convenient to work with, since they are simple enough to allow accurate modeling. In a way they are to GA research, what ideal gases are to physics: a workable problem that is simple enough to analyze, yet still close enough to the overall problem such that the gained insights are representative for the bigger picture.

Holland's schema theorem showed that the prime reason for using genetic algorithms is their capacity to process building blocks. Therefore it is important to look at this building block processing from a mixing point of view, and the dimensional mixing analysis has also been done at the level of building blocks.

The traditional view on genetic optimization considers crossover to be a source of building block disruption and therefore very disruptive operators such as uniform crossover are supposed to be of no use for problems where one needs to process building blocks as a whole. The most pronounced problem where this is the case are the fully deceptive functions, and general GA wisdom says that a basic GA can only solve these problems to global optimality if the building blocks are tightly linked, and one uses a linkage biased crossover operator, such as one-point crossover.

We can question this wisdom however by looking at the schema theorem in another way: since the schema theorem basically contains the product of two terms - the selection pressure times the schema survival probability - it is perfectly possible to grow or increase the number of copies of building blocks that are not tightly linked. All we have to do is to increase the selection pressure such that the higher loss caused by the disruptive uniform crossover operator is counterbalanced. Thus as far as the schema theorem is concerned, the building blocks do not have

to be tightly linked.

Clearly, the schema theorem cannot be the only factor in determining GA success. It is very important to realize that the schema theorem does not talk about the crucial issue of building block exchange: having the building blocks in large proportions in the population is not by itself a guarantee to find a good solution. This building block juxtapositioning or mixing is the main task of recombination.

Using dimensional analysis we present the mixing ladder climbing model that shows that if we have no linkage information and therefore use uniform crossover with a high selection pressure, then the minimal population size needed grows exponentially in the number of building blocks. Therefore, in practice the basic genetic algorithm cannot be used to solve difficult problems unless the building blocks are tightly linked.

To summarize, the building block mixing analysis leads us to the conclusion that the basic genetic algorithm is severely limited in its mixing capacity, and one therefore has to respect proper building block mixing if we want fast convergence to the global optimum.

This respect for proper building block mixing can be obtained by tightly coding the building blocks. It is interesting to note that the schema theorem also emphasized tight building block coding. There, the underlying idea was to minimize the disruption probability of the building blocks, such that the number of copies would increase through selection. The mixing analysis however made clear that the basic GA can grow building blocks without tight linkage, but it cannot mix them.

Designing effective genetic algorithms. The basic genetic algorithm is thus severely mixing limited, and can only mix the building blocks when they are tightly linked. Demanding that the building blocks are tightly linked is however a strong requirement which certainly limits the applicability of the basic genetic algorithm. To increase the performance of genetic algorithms, one should therefore make some extensions that respect proper building block mixing, and at the same time do not require them to be tightly linked.

We have studied a number of GA extensions, namely elitism, niching, and restricted mating.

Elitism guarantees that once a good solution is found it will not be thrown away for some low value offspring. Good solutions can only be replaced by better ones, so the computational effort that went into mixing the building blocks is maximally exploited.

Niching eliminates the race between growing building blocks and mixing them - this is between selection and recombination - and therefore the mixing can be done at a slower rate, without running the risk of premature convergence.

Restricted mating increases the mixing probability by restricting crossover to strings that are close together in Hamming space. This way most of the building blocks are matched in the two mating strings so they cannot be destroyed by the extremely disruptive uniform crossover operator.

Although these extensions allow the building blocks to remain present in the population, they only marginally help the building block mixing process.

Finally, we have discussed an adaptive crossover operator where the gene exchange probability is changed such that it becomes tuned to the position of the building blocks. Once recombination respects the building blocks, the mixing becomes fast. Experimental results on fully deceptive functions without linkage assumptions confirmed the analysis. However, further study might be needed to see how well the mixing occurs for partially deceptive problems.

The important point is that the design of improved genetic algorithms should make sure that the mixing of building blocks is properly attended to. This result can be seen as a key engineering principle for designing effective genetic algorithms.

Changing the crossover swapping probability as in adaptive crossover is only one possible way to go. Another approach is taken by the fast messy GA where the building blocks are first filtered from the complete string, and then juxtaposed to form a complete solution (Goldberg, Deb, Kargupta & Harik, 1993).

Local elitist genetic algorithm. The work on elitism also led to an extremely simple, locally implementable genetic algorithm - called elitist recombination GA - that intertwines the selection and recombination

phase. The elitist recombination GA holds only a competition between each mating pair and their pair of offspring. Parent strings can thus only be replaced by children that have a better fitness value. We show that the selection pressure of the elitist recombination GA is equal to the selection pressure of tournament selection which is widely used in practice. Due to the local elitism, elitist recombination has a number of additional benefits, and therefore promises to be a very useful GA implementation.

Future work. With respect to future work, we would like to mention some open problems:

- Chapter 2 gave a lower and upper bound of the convergence time complexity for fitness functions with independent gene fitness contributions. This can be extended in a number of ways. One way is to look at functions where the gene fitness contributions are somehow interacting, for instance we might look at flat royal road functions where a group of bits are forming a block, and the same convergence characteristics will now appear at the block level, where they can be integrated with the effect of block disruption due to crossover.

  One might also want to look at intermediate cases of convergence time complexity, for instance when the gene fitness contributions are linearly scaled.

  We mainly focussed on tournament selection, but similar analyses could be made for other selection schemes. For instance deriving these bounds for steady-state GAs as GENITOR or CHC should be straightforward.

- In Section 2.3 we have shown how the gene covariance build-up, slows down the convergence speed. This occurs even when using the highly disruptive and thus highly decorrelating uniform crossover operator. When using a linkage biased crossover operator, such as one-point crossover, the covariance build-up will be a lot faster. This will not only cause a serious decrease in convergence speed, but it might also lead to the loss of some allele values and thus to premature convergence.

  This premature convergence is of course a mixing failure and in Section 3.2.5 we briefly discussed the effect of linkage biased crossover and mixing.

  It would certainly be interesting to have an analytical model that describes the covariance build-up for linkage biased crossover operators, and that can be used to describe the mixing failure boundary. This way the mixing law that we established for the uniform crossover case, would be generalized to include linkage-biased crossover operators also.

- Our proposal for changing the crossover exchange distribution in function of the current state of the population, is only a first attempt. The adaptive crossover operator quickly became tuned to the position of the building blocks, and thus could mix them very fast. However, we have only looked at fully deceptive problems, and a more detailed study is needed to see what will happen when the fitness function is only partially deceptive. The underlying idea however of adapting the crossover exchange distribution at run time, such that it reflects the specific characteristics of the current problem, promises to be very useful.

- The work on fast messy genetic algorithms (Goldberg, Deb, Kargupta & Harik, 1993) resulted in an algorithm that solved fully deceptive functions very efficiently without making any linkage assumptions. A potential problem is that the filtering of building blocks has to be carefully scheduled at the risk of loosing some of them. The opposing forces between selection and filtering is very analogous to the selection and recombination race we have studied in the mixing analysis.

  The extensions we have discussed might also be very useful in making the filtering process less sensitive to the selection-filtering schedule. Perhaps an adaptive filtering operator can be defined, that works in the same way as the adaptive crossover operator of Section 4.5.

- The mixing analysis led to the GA engineering principle, that effective GAs have to respect building block mixing. This principle is

not limited to the binary string based genotype representation that we considered throughout this work, but it applies to all crossover based algorithms.

For instance in the field of genetic programming one recombines tree structures by swapping subtrees. These trees represent Lisp programs, while the building blocks are represented by subtrees. The quality and rate of convergence of genetic programming will thus largely be dependent on proper subtree mixing. It would be most instructive to have a dimensional analysis of this subtree mixing process.

Appendix A

Genetic Weight Optimization of Feedforward Neural Networks

In this appendix we want to look at an application of genetic algorithms in optimizing the weights of feedforward neural networks. This problem is interesting because the neural network genotype representation causes some special crossover problems, that are basically building block mixing problems, and thus illustrative for the discussions in this work.

Structurally different feedforward neural networks can implement the same functional mapping due to representational symmetries. These redundancies however have caused major concern about the applicability of crossover to neural network optimization. A number of heuristic solutions have been proposed in the literature (Hancock, 1992; Radcliffe, 1993; Thierens, Suykens, Vandewalle, & De Moor, 1993).

Here we propose a non-heuristic and very simple genotype representation that eliminates the redundancies. It is shown that crossover transmits more information from the parents to the children when using the non-redundant representation than with the straightforward but redundant coding.

A.1 Introduction

Several authors have shown that single hidden layer feedforward neural networks (NNs) are universal approximators for any continuous mapping (Hornik, 1989; Funahashi, 1989; White, 1990). Unfortunately these existence proofs give very little guidance on how to construct a good network for a specific problem. The network design problem can be decomposed into two subtasks. First there is the problem of network architecture: how many hidden layers do we need, how many neurons in each hidden layer, and what connectivity will give us an optimal performance? Second there is the problem of weight determination: how do we get the connection weights once we have chosen a particular network topology? In this section we will look how genetic algorithms can help solving the latter problem. Using the GA for optimizing the weights of a feedforward neural network does not only have substantial practical value but it also points out some particular problems one can face when applying genetic algorithms to a specific problem.

The traditional approach in specifying the weights of a NN is to minimize an error function by using backpropagation, conjugate gradient methods, quasi-Newton methods, or some other optimization algorithm. Although there are considerable differences between these algorithms they do have one significant property in common: they all are local optimization algorithms. Starting from an initial random value, a sequence of neighboring points is generated by extracting local information of the search space. Depending on the particular algorithm used, this information consists of the function values, the first- and/or second derivative. For non-trivial problems however the error surface is high-dimensional and contains many local optima, so there is a significant risk that a local optimization algorithm will convergence to some bad local optimum, and in practice one simply does multiple runs with different random starting points. The amount of function evaluations for one single run is usually quite substantial so the ratio between the number of restarts and the number of local function evaluations is typically very low which in fact makes this approach a poor global optimization algorithm.

Here we discuss the use of genetic algorithms to search the neural network weight space in a global way. First we look at the representation problem: how to encode a feedforward neural network to allow efficient GA processing. It will be argued that the existence of functional equivalent symmetries in the neural network input-output mapping cause major problems for the recombination operator. Then a non-redundant neural network representation is discussed that is ideally suited for being processed by crossover. Finally we test the crossover friendly representation for a difficult classification problem.

A.2 Equivalence symmetries in neural networks

The functional mapping implemented by a feedforward neural network is not unique to one specific set of weights. For a single hidden layer NN the mapping from the m-dimensional input vector X to the p-dimensional output vector Y is defined as (using the hyperbolic tangent *tanh* as transfer function):

$$Y = tanh(Wtanh(VX))$$

where V is the (n,m)-dimensional matrix of weights from the input layer to the hidden layer, and W the (p,n)-dimensional matrix of weights from the hidden layer to the output layer.

Obviously we can generate a number of structurally different neural nets that have the same input-output mapping. What characterizes these networks is that they all are a member of a finite group of symmetries defined by two transformations. (Sussmann, 1992; Chen, Lu & Hecht-Nielsen, 1993). Any member of this group can be constructed from any other member by a sequence of these transformations. The first transformation is a permutation of hidden neurons. Interchanging the hidden neurons including their incoming and outgoing connection weights does not change the functional mapping of the network. The second transformation is obtained by flipping the weight signs of the incoming and outgoing connection weights of a hidden neuron. Since *tanh* is an odd symmetric function this sign flipping leaves the overall network mapping unchanged:

$$\begin{aligned} Y &= tanh(Wtanh(VX)) \\ &= tanh(-Wtanh(-VX)) \end{aligned}$$

For a network with a single hidden layer of n neurons there are a total of $n!$ permutations. Similarly we can choose any combination of the n hidden neurons to flip their weight signs so there are $\sum_{i=0}^{n}\binom{n}{i} = 2^n$ structurally different but functionally identical networks generated by this transformation. Since the two transformations are independent of each other, there are a total of $2^n n!$ functional equivalent but structurally different networks. In (Sussmann, 1992; Chen, Lu & Hecht-Nielsen, 1993) it is proven that all the functionally equivalent neural networks are compositions of hidden node permutations and sign flips.

For the traditional local weight optimization algorithms this redundancy poses no problem since they only look in the immediate neighborhood of the current point of the search space. Global optimization algorithms however will try to explore the whole connection weight search space and this is a factor $2^n n!$ bigger than it really ought to be for the network to function as a universal approximator. For the genetic algorithm the problem is not only one of scale but also of crossover efficiency: functional equivalent near optimal networks often give rise to totally inappropriate networks after straightforward recombination because their weight structure is only equivalent up to a certain amount of transformations.

In the next section we first take a closer look at the recombination problem. The consequences of the functional equivalent groups are discussed and a method is offered to eliminate these redundancies.

A.3 Genotype representation of neural networks

A redundant neural network genotype representation

The standard approach in genetic algorithm practice is to represent the search space simply by concatenating all the parameters in the genotype. Parameters close to each other on the genotype are more likely to be processed as a whole when using a linkage biased crossover such as one-point crossover. Whenever a set of parameters form a functional unit it is therefore better to encode them tightly at the genotype. In neural networks the incoming weights of a single hidden node form a high dimensional hyperplane, so we place them together on the genotype.

For instance for a single hidden layer network with m input neurons, n hidden neurons and p output neurons

$$Y = tanh(Wtanh(VX))$$

the weights can be represented on a string as

$$< v_{10}...v_{1m}...v_{n0}...v_{nm}w_{10}...w_{1n}...w_{p0}...w_{pn} >$$

where v_{ij} is the weight from the hidden neuron i to the input neuron j and w_{ij} is the weight from the output neuron i to the hidden neuron j. For $j = 0$ the weight represents the neuron's bias.

The order with which the hidden neurons are placed on the genotype is irrelevant for the mapping performed by the network. Unfortunately for the crossover operator this is not the case: suppose we have two networks with two hidden neurons

$$< v_{10}...v_{1m}v_{20}...v_{2m}... >$$
$$< v'_{10}...v'_{1m}v'_{20}...v'_{2m}... >$$

such that $v_{i1}...v_{im}$ and $v'_{i1}...v'_{im}$ represent similar hyperplanes. When the two hidden neurons are close to optimal - or the two hyperplanes are needed to solve the problem - then both the networks will have a low error value. If we recombine the two networks by crossing the two nodes then the children become

$$< v'_{10}...v'_{1m}v_{20}...v_{2m}... >$$
$$< v_{10}...v_{1m}v'_{20}...v'_{2m}... >$$

and most likely they will also be close to optimal.

Suppose on the contrary that the parents are represented as

$$< v_{10}...v_{1m}v_{20}...v_{2m}... >$$
$$< v'_{20}...v'_{2m}v'_{10}...v'_{1m}... >$$

Exchanging the two nodes gives us:

$$< v'_{20}...v'_{2m}v_{20}...v_{2m}... >$$
$$< v_{10}...v_{1m}v'_{10}...v'_{1m}... >$$

so the two offspring neural networks now have two similar hidden neurons thus representing almost the same hyperplane while they are lacking a hidden node around the other hyperplane. The new networks will almost certainly have a very high error value.

Although the example assumes that the crosspoint falls in between two hidden neurons it is clear that the same problem also exists when this is not the case. In addition the sign flip redundancy makes this problem even worse.

Ideally we want to exchange functional similar hidden neurons but due to the representation redundancy, crossover cannot preserve enough of the information present in the parent networks. In the following section we show how feedforward neural networks can be represented in a crossover friendly way.

A non-redundant neural network genotype representation

The goal of the crossover operator in GAs is to take partial solutions of two individuals and recombine them to form a better solution. In feedforward NNs with globally defined transfer functions (as opposed to locally defined transfer functions such as radial basis neural networks), the hidden nodes represent hyperplanes and we want to recombine the good hyperplanes of two NNs to create a better performing network. It is important however that the offspring inherit all the hidden nodes that are necessary to implement the desired mapping. To prevent that recombination would place the functional similar hidden neurons on the same offspring, we should rearrange the hidden nodes before crossover is applied such that similar neurons are in the same position. Many neural network rearrangements have been proposed in the literature but unfortunately they are all based on heuristic arguments.

A more principled neural network representation can be obtained if we recall that all functional equivalent neural networks form a finite group of networks that can be transformed into each other by a composition of hidden node permutations and sign flips. (Sussmann, 1992; Chen, Lu & Hecht-Nielsen, 1993). The functional redundancies can thus be eliminated if we transform each neural network to a non-redundant network that has a unique representation in each functional equivalence

class. This non-redundant representation can be achieved in a number of ways. For instance to eliminate the hidden node redundancy we can simply flip the signs of the bias weight, the incoming and the outgoing weights of each hidden neuron whenever the number of positive incoming and outgoing weights of a neuron in the hidden layer is less than the number of negative incoming and outgoing weights. Alternatively, we might flip the signs whenever the bias weight is negative, so only hidden neurons with a positive bias are allowed in the non-redundant neural network representation, thus reducing the 2^n functional equivalent neural networks to just one representative of the group. Because of its extreme simplicity we will use the last approach here.

The second redundancy can be found at the level of each hidden layer and is caused by the permutation of the hidden nodes in each layer. To eliminate this redundancy we rearrange all hidden neurons in each layer such that the bias weights are sorted in ascending order. This way the $n!$ functional equivalent networks are eliminated and since the sort process does not interfere with the previous sign flipping all the $2^n n!$ equivalent networks are transformed to a single group representative.

To summarize, the following transformations are made to each neural network before they are recombined:

Neural network transformation:

1. $\forall$ hidden neurons:
   if (bias < 0)
   flip signs of each node weight
2. $\forall$ hidden layer:
   sort neurons in increasing bias order

A.4 The hybrid GA+BPX algorithm

Genetic algorithms are global search algorithms and thus are less likely to converge to a bad local optimum. However they do not make use of

any gradient information which makes them very poor local optimizers. Combining the global and the local search in a hybrid optimization algorithm gives us the best of both approaches. Here we will use accelerated backpropagation (BPX) as local optimization algorithm.

The specific GA implementation used is the elitist recombination GA because it is ideally suited for hybrid GA algorithms and it has the same selective pressure as the well known tournament selection scheme (Thierens & Goldberg, 1994). The elitist recombination GA intertwines the selection and recombination phase by holding a competition between the mating parents and their offspring. This local elitist mechanism ensures that population members can only be replaced by better solutions.

Elitist Recombination GA:

1. initialize population
2. $\forall$ generation
   (a) randomly shuffle population
   (b) $\forall$ mating pair:
   - generate two offspring
   - keep best two of the four individuals

The elitist recombination GA allows us to integrate the local search algorithm in an elegant way. At every generation the population members are locally optimized for a few steps, and are then recombined to generate new offspring. Before the children compete with the parents to enter the population they also are locally optimized. Due to the elitism the local search is automatically focused on the current best solutions: good solutions are generally harder to replace by the children and will remain in the population for a number of generations thus receiving more local optimizations steps. Bad solutions are replaced very quickly so no expensive gradient search is wasted on them. In the worst case the problem, its coding and the crossover operator are totally incompatible, so recombination and the subsequently local search never produce any children better than their parents. As a result no new solutions will ever

enter the population and the hybrid algorithm reduces to a multistarted local search algorithm where the starting points are simply the initial random population.

Genetic algorithms work well when meaningful substructures can be recombined by the crossover operator. For neural networks these meaningful substructures are formed by the hidden neurons, and we restrict crossover to exchange only the incoming weights and bias of each neuron as a whole. One-point crossover is used as recombination operator. The GA is thus searching for a good combination of existing neurons in the parent networks, while the BPX algorithm changes the weights of the neurons. This way the hybrid GA+BPX algorithm decomposes the optimization task in two subtasks: a combinatorial search of meaningful structures by the GA and a local gradient search by the BPX algorithm. In the GA terminology we could say that each neuron with its incoming weights and bias is a gene and the specific gene value or allele is determined by the gradient algorithm.

Here we limit ourselves to the search of the weights for a fixed neural network architecture. The principle of genotype redundancy reduction before recombining two networks remains however valid in case when one is also searching for the optimal neural network topology.

Finally we have to specify how to switch between the global GA and the local BPX search. A straightforward approach is to optimize each generation all population members for a fixed number of epochs. Before competing with their parents the children are also optimized for a fixed number of epochs. This is of course a rather ad hoc procedure and a more intelligent approach should adapt itself to the behavior of the convergence process. Note that the hybrid GA+BPX algorithm is already adapting itself to the search dynamics by means of the adaptive learning rate of the backpropagation algorithm. Typically the learning rate changes in a cyclic way: periods of continuously increasing learning rate are suddenly stopped by a sharp decrease, and subsequently the increasing period slowly resumes. An adaptive global-local search scheduling mechanism can easily exploit this: whenever the learning rate is increasing the BPX search continues but if a sharp decline occurs the local optimization is stopped and the GA generates new networks.

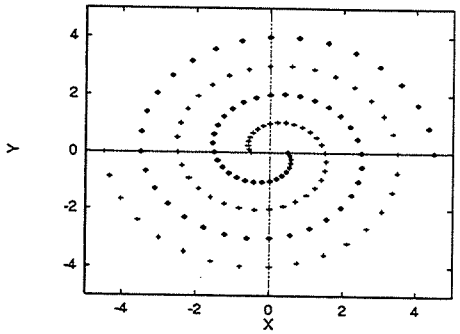


Figure A.1: *Training data for the two spirals classification problem.*

A.5 Example: two spirals classification problem

To compare the non-redundant neural network representation with the redundant representation we test both approaches on the two spirals classification problem. The two spirals problem is a widely used benchmark problem and is especially demanding for neural networks with non-local transfer functions such as multi-layer perceptrons. The task is to learn to discriminate between two sets of training points which lie on two distinct spirals in the x-y plane (so there are 2 input and 1 output neurons). In the classical benchmark the spirals coil three times around the origin and around one another. The networks typically used to solve the problem have two hidden layers and require a substantial amount of training time.

Since we merely want to compare the performance of the non-redundant and the redundant neural network representations for genetic recombination, we have reduced the computational complexity by letting the spirals coil two times around one another instead of three times (see Figure A.1). Each spiral has 65 training examples, specified by their (x, y) coordinate. The networks have 2 input neurons, and 1 output neuron indicating to which of the two spirals the (x, y) input pair belongs.

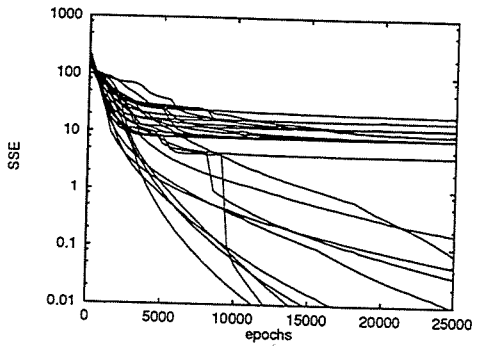


Figure A.2: *Sum-squared error of 30 neural networks with a hidden layer of 15 neurons when solving the two spirals classification problem with the accelerated backpropagation algorithm.*

Local search only

To have an idea of the task complexity we trained 30 neural networks with one hidden layer of 15 neurons. Training was done by the accelerated backpropagation algorithm (BPX) which uses a momentum and adaptive learning rate (Vogl, Mangis, Rigler, Zink & Alkon, 1988). The parameters were set to their default value: momentum = 0.9, initial learning rate = 0.01, learning rate increase factor = 1.05, learning rate decrease factor = 0.7 and maximum error ratio = 1.04. The neurons had a hyperbolic tangent transfer function and were initialized by the Nguyen-Widrow procedure to ensure that the linear region of the hyperbolic tangent function occurs within the input space region where training data are present (Nguyen & Widrow, 1990).

Figure A.2 plots the sum-squared error (SSE) in function of the number of epochs. From the total of 30 networks, 20 got stuck at a bad local optimum, 6 reached the training goal (SSE=0.01) in less than 25000 epochs, and 4 where still converging after 25000 epochs.

In the next section we will solve the same problem with a hybrid GA+BPX algorithm and compare the non-redundant with the redundant network representation.

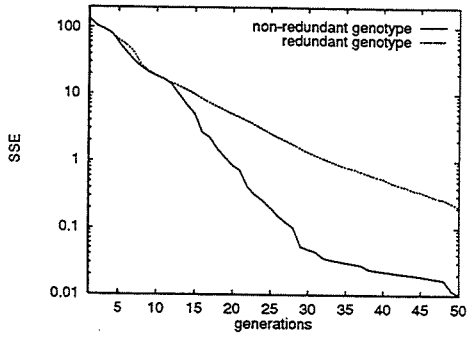


Figure A.3: *Sum-squared error of the best network in the population at each generation when solving the two spirals classification problem with the hybrid GA+BPX algorithm.*

Global + local search

To test the non-redundant neural network representation we compare it with the redundant representation starting from the same initial population of 30 neural networks as used above. At each generation the parents are optimized by the accelerated backpropagation algorithm for 100 epochs and the children for 200 epochs.

Figure A.3 shows the sum of the squared error of the best network at each generation. Clearly the non-redundant genotype representation outperforms the redundant genotype: after 25 generations the error of the best network in the non-redundant population is more than 10 times lower than the error of the best redundant neural net.

In Figure A.4 we have plotted the number of new neural nets using the non-redundant genotype that enter the population at each generation. After the first generations roughly 10% of the population is replaced each time, indicating that crossover is able to create better offspring during the entire optimization process.

Since the non-redundant genotype is more efficient than the redundant representation we expect it to transmit more information from the parents to the children. In the next section we test this by computing the crossover correlation coefficient for both network representations.

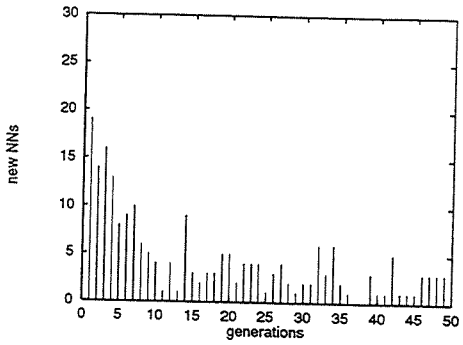


Figure A.4: *Number of non-redundant networks that enter the population at each generation.*

Crossover correlation coefficient

The crossover correlation coefficient is a statistical feature expressing how correlated the fitness landscape appears to the crossover operator (Manderick, de Weger & Spiessens, 1991). The fitness landscape is defined by the combination of the fitness function and the specific genotype coding. The idea is that the more correlated a landscape appears to be for a specific operator the more efficient the GA search will be because the higher the correlation coefficient the more information is transmitted from the parents to the children.

To calculate the crossover correlation coefficient ρ_X we randomly generate a large number of parents, apply the crossover operator to obtain the offspring and compute the correlation coefficient as:

$$\rho_X(F_p, F_c) = \frac{cov(F_p, F_c)}{\sigma(F_p)\sigma(F_c)}$$

where F_p and F_c are respectively the random variables representing the mean fitness of each parent and offspring pair, $cov(F_p, F_c)$ is the covariance between F_p and F_c, and $\sigma(F_p)$ and $\sigma(F_c)$ are the standard deviations.

In (Manderick, de Weger & Spiessens, 1991) the crossover correlation coefficient ρ_X was computed for four different crossover operators

applied to the traveling salesman problem. Experimental results showed that the relative order of the coefficient ρ_X agreed with the relative performance of the operators on the TSP problem. Other studies obtained similar results: the higher the crossover correlation coefficient ρ_X is on a specific problem the better this operator performs on the test problems (Mathias & Whitley, 1992; Inayoshi & Manderick, 1994; Dzubera & Whitley, 1994).

Here we take the opposite approach: instead of comparing different crossover operators on a certain problem coding, we fix the crossover operator (one-point crossover) but compare the genotype representation. Since the fitness landscape is determined by the combination of the fitness function and the specific genotype coding, the crossover correlation coefficient ρ_X can also be used to see how well the crossover operator can transmit information from parents to offspring for different genotype codings.

Table A.1 shows the crossover correlation coefficient ρ_X of the redundant and the non-redundant neural network representation. Each is computed by recombining 2500 randomly generated parent pairs. The same parents and the same crosspoint is taken for the two network representations, the only difference is that for the non-redundant genotype the parents are first transformed into non-redundant form before being crossed.

Clearly the crossover correlation coefficient ρ_X for the non-redundant neural network representation is much higher indicating that crossover transmits much more information from the parents to the children, and thus will lead to a more efficient GA search.

Note that the non-redundant genotype representation is also useful for networks with more than one hidden layer: Table A.1 compares the crossover correlation coefficients for networks with two hidden layers on the three times coiled spirals problem.

A.6 Conclusion

Genetic weight optimization of feedforward neural networks is hampered by the existence of two functional equivalence symmetries in these networks. A computationally efficient transformation is discussed that elim-

NNs	*redundant*	*non-redundant*
2-15-1	0.456	0.892
2-15-5-1	0.598	0.903

Table A.1: *Crossover correlation coefficient ρ_X for the redundant and non-redundant neural network genotype representation. Results are computed for networks with one hidden layer of 15 neurons, and for two hidden layers with respectively 15 and 10 hidden neurons.*

inates these redundancies: the weight signs of each hidden neuron with a negative bias are flipped and the neurons in each hidden layer are sorted according to their bias value. Experimental results on the two spirals classification problem show the better performance of the non-redundant network representation. Finally we have computed the crossover correlation coefficients of both genotype codings. As expected the non-redundant genotype representation has a significantly higher value than the straightforward coding, confirming that the non-redundant coding allows crossover to transmit more information from the parents to the children.

Appendix B

Nederlandse Samenvatting

Analyse en Ontwerp van Genetische Algoritmen

De interesse voor het ontwerp van algoritmen in analogie met de principes van natuurlijke evolutie en genetica, dateert reeds vanuit de jaren 60. De neo-darwinistische theorie toont aan hoe enkele eenvoudige organisatorische principes aanleiding kunnen geven tot het creëren van een volledig zelf-organizerend, en kontinu aanpasbaar systeem. Dit staat in scherp kontrast met de complexe, starre en moeizaam ontworpen artificiële systemen.

Heden ten dage worden ingenieurs gekonfronteerd met een steeds maar toenemende ontwerp complexiteit. Hierdoor wordt het steeds aanlokkelijker om te trachten een aantal probleem oplossingsprincipes die in de natuur voorkomen te vertalen naar algoritmen en zo het gamma probleem oplossingsmethoden van de ingenieur aan te vullen. Daarenboven vormt het praktisch gebruik van biologisch of fysisch geïnspireerde algoritmen geen probleem meer door de zeer uitgebreide beschikbaarheid van krachtige computers.

Genetische algoritmen (GAs) zijn niet de enige leden van de natuurlijk geïnspireerde algoritmen. Twee andere leden die intensief bestudeerd worden en met succes worden toegepast, zijn neurale netwerken en gesimuleerd uitgloeien (simulated annealing).

Niettegenstaande deze algoritmen gemodeleerd zijn volgens natuurlijke principes, is het belangrijk voor ogen te houden dat dit enkel maar een metafoor is, met als doel het ontwerpen van performante oplossingsmethoden. De kritiek die soms wordt geuit dat bepaalde uitbreidingen aan het basis algoritme "onnatuurlijk" en dus "niet interessant" zouden zijn, is dan ook niet terecht. Er is dan ook een hemelsbreed verschil tussen het modeleren van natuurlijke processen, en het gebruik van dezelfde processen als metafoor om nieuwe algoritmen te ontwerpen. Het eerste is ongetwijfeld een uitermate belangrijke wetenschappelijke onderneming, maar dit mag niet verward worden met de niet minder belangrijke ingenieurstaak om methoden te ontwerpen die hulp kunnen bieden bij het oplossen van complexe problemen.

In deze thesis nemen we het standpunt in van de ingenieur en bestuderen we genetische algoritmen met het doel problemen te kunnen oplossen op efficiënte wijze. Dit leidt ons tot het stellen van volgende vragen: "Wat zijn de beperkingen van het basis genetische algoritme ?", "Wat zijn de rekenkundige vereisten ?", "Hoe kunnen we verschillende genetische algoritmen evalueren ?", en "Hoe kunnen we het basis genetisch algoritme uitbreiden om de performantie te verhogen ?".

De eerste evolutionaire algoritmen maakten uitsluitend gebruik van selektie en mutatie, en hadden maar een beperkt succes in termen van rekenkundige efficiëntie. Een goed overzicht van dit vroege werk wordt gegeven in (Goldberg,1989). Het werk van John Holland van de Universiteit van Michigan over adaptieve systemen (startend in het begin van de jaren 60) leidde tot de ontwikkeling en mathematische analyse van genetische algoritmen. Hierbij werd zeer veel belang gehecht aan het populatie gebaseerd zoeken, dat verkregen werd door kombinatie van selektie en rekombinatie. De wiskundige basis werd gevormd door het schema theorema en het verband met het k-armige bandiet probleem uit de statische beslissingstheorie. Holland's invloedrijk boek *Adaptation in Natural and Artificial Systems* verscheen in 1975, en in hetzelfde jaar beëindigde ook De Jong zijn belangrijke doktoraatsthesis, waarin

Holland's theoriën getoetst werden aan uitgebreide experimentele resultaten.

Het onderzoek over genetische algoritmen werd vervolgens geleidelijk verspreid vanuit de Universiteit van Michigan, en in 1985 werd voor het eerst de tweejaarlijkse internationale conferentie over genetische algoritmen gehouden.

Pas in 1989 echter neemt de interesse in genetische algoritmen plotseling een hoge vlucht: in dit jaar publiceert David Goldberg zijn boek *Genetic Algorithms in Search, Optimization, and Machine Learning*, en dit werk verschaft een ideale mengeling van theoretische analyses, intuitief inzicht, praktische implementaties, historische omkadering, en toepassingen voor een uitgebreid publiek.

Sindsdien is er heel wat veranderd: de initieel kleine familie van GA onderzoekers groeide uit tot een wereldwijde wetenschappelijke discipline met verschilllende internationale conferenties, zoals het *International Conference on Genetic Algorithms*, *Parallel Problem Solving from Nature*, *IEEE Conference on Evolutionary Computation*, en *Foundations of Genetic Algorithms*. Onlangs startte MIT Press met een specifiek vaktijdschrift: *Evolutionary Computation*.

Ondanks de twintigste verjaardag van Holland's boek, en de wereldwijde interesse in genetische algoritmen, is er nog steeds een nood aan theoretische studies die ons helpen een beter begrip en intuitief inzicht te krijgen in deze niet-lineaire, populatie gebaseerde, stochastische algoritmen, die toegepast worden op de meest uiteenlopende, hoog dimensionele en complexe problemen.

Ik hoop van harte dat deze thesis hiertoe een kleine bijdrage kan leveren.

B.1 Overzicht

Het voornaamste doel van deze thesis is te trachten een inzicht te verkrijgen in de dynamische karakteristieken of konvergentie eigenschappen van genetische algoritmen. Een belangrijk concept hierin is de interaktie tussen selektie en rekombinatie in termen van hun relatieve snelheid. Meer algemeen kijken we naar de noodzakelijke balans tussen het exploiteren van wat we reeds weten van de zoekruimte, door het selekteren

van voorafgaande goede oplossingen, en het exploreren van nieuwe delen van de zoekruimte door de geselekteerde oplossingen te rekombineren.

Hoofdstuk 1: Genetisch algoritme overzicht

In dit hoofdstuk introduceren we het basis genetisch algoritme en geven een overzicht van de belangrijkste theoretische concepten. Eerst beschouwen we de filosofie om de neo-darwinistische theorie te gebruiken als metafoor voor het ontwerpen van optimalizatie algoritmen. Door de principes van het neo-darwinisme te abstraheren, bekomen we het basis genetisch algoritme.

Vervolgens bespreken we John Holland's fundamenteel werk over het schema theorema en de bijhorende bouwstenen hypothese (building block hypothesis). Het schema theorema geeft een ondergrens aan het verwachte aantal leden dat de schemata zullen hebben in de volgende generatie. Het schema theorema leidt tot de bouwstenen hypothese die stelt dat korte, lage-orde en waardevolle schemata kunen gerekombineerd worden tot hogere orde schemata en volledige strings met zeer hoge fitheid.

Drie potentiële problemen voor de bouwstenen hypothese kunnen geïdentificeerd worden.

Ten eerste kunnen de bouwstenen het zoekproces misleiden naar wat men noemt misleidende attraktoren. In dit hoofdstuk verklaren we het begrip misleiding en bespreken volledig misleidende funkties en multimodale, volledig misleidende funkties. Deze funkties werden specifiek ontworpen als moeilijke testgevallen, en worden ook in deze thesis veelvuldig gebruikt.

Een tweede probleem dat kan optreden is dat de grootte van de populatie te klein is om een betrouwbare selektie van de bouwstenen toe te laten. De minimale noodzakelijke populatiegrootte wordt berekend door de variantie van de funktiewaarde van de schemata in rekening te brengen.

Tenslotte, een derde potentieel probleem is de noodzaak om de bouwstenen snel te kunnen uitwisselen. Dit mengingsprobleem is tot nog toe weinig beschouwd in de literatuur, en het vormt dan ook het belangrijkste thema in hoofdstuk 3.

We besluiten het overzichtshoofdstuk met een korte bespreking van Goldberg's zes voorwaarden voor een suksesvolle toepassing van genetische algoritmen.

Hoofdstuk 2: Grenzen aan convergentie complexiteit van genetische algoritmen

In dit hoofdstuk analyseren we de dynamische eigenschappen van verschillende genetische algoritmen implementaties door het konstrueren van onder- en bovengrens konvergentie modellen voor funkties met onafhankelijke fitheid kontributie van de genen.

Eerst stellen we een ondergrens konvergentie model op door gebruik te maken van de normaal distributie theorie, voor funkties met eenzelfde fitheid kontributie van de genen. Het model en de experimentele resultaten tonen aan dat voor rang gebaseerde genetische algoritmen - zoals toernooi selektie - het aantal generaties nodig om de populatie volledig te laten konvergeren van de orde $O(\sqrt{\ell})$ is, met ℓ de lengte van de binaire string. Proportionele selektie daarentegen heeft een konvergentie complexiteit van $O(\ell \ln \ell)$.

De volgende sektie toont aan hoe de covariantie opbouw tussen de genen, de konvergentie snelheid vertraagt. Deze covariantie wordt veroorzaakt door het selektie proces, en wordt slechts gedeeltelijk gedecorreleerd door rekombinatie.

In de laatste sectie stellen we een bovengrens konvergentie model op, gebaseerd op de domino konvergentie analyse voor funkties met exponentieel geschaalde fitheid kontributie van de genen. Er wordt aangetoond dat de domino konvergentie logistisch verloopt met tijdsafhankelijke groeisnelheid. Een zeer goede benadering wordt bekomen door een exponentieel stijgende groeisnelheid te nemen in de tijd, en een exponentieel dalende groeisnelheid volgens de positie van de genen in de binaire string. Dit geeft aanleiding tot een golfvergelijking model met konstante golfsnelheid. Het model en de experimentele resultaten tonen aan dat de konvergentiesnelheid nu van de orde $O(l)$ is.

Hoofdstuk 3: Dimensionele analyse van menging in het basis genetische algoritme

In dit hoofdstuk bestuderen we een veelvuldig over het hoofd gezien aspekt van genetisch optimalizeren, met name het mengen of uitwisselen van de bits en de bouwstenen, en de relatie met selektie. Niettegenstaande menging nauwelijks enige aandacht heeft gekregen in de literatuur, tonen we aan dat het een doorslaggevende faktor is om te bepalen wanneer een genetisch algoritme zal konvergeren naar het gewenste optimum. De beperkte grootte van de populatie wordt expliciet in rekening gebracht.

We argumenteren dat vanuit dynamische of konvergentie overwegingen, selektie en rekombinatie in een tijdsrace gewikkeld zijn. Selektie zorgt ervoor dat het aantal copies van de bouwstenen snel toeneemt van generatie tot generatie, maar om optimale oplossingen te bekomen moeten alle bouwstenen samengebracht worden op één string door de kruisingsoperator, en dit vergt uiteraard een aantal generaties. Indien de selektie van bouwstenen te snel gaat - relatief ten opzichte van de snelheid waarbij rekombinatie ze kan uitwisselen - dan zal de populatie vroegtijdig konvergeren naar een suboptimale oplossing. Een genetisch algoritme zal enkel konvergeren naar de optimale oplossing indien de selektie druk niet te hoog is, zodat de kruisingsoperator voldoende tijd heeft om de bouwstenen te rekombineren.

We analyseren deze selektie-rekombinatie race met behulp van dimensionele analyse. Dit geeft ons de funktionele relaties tussen de determinerende parameters, met name de selektie druk, de rekombinatie snelheid, de populatie grootte, de lengte van de string, en tenslotte de lengte van de bouwstenen.

Hoofdstuk 4: Uitbreidingen van het basis genetisch algoritme

In dit hoofdstuk worden een aantal uitbreidingen van het basis genetisch algoritme besproken, en wordt hun impakt op de konvergentie eigenschappen en het mengingsproces bestudeerd.

Eerst stellen we het elitaire rekombinatie genetisch algoritme voor, waarbij de selektie en rekombinatie fase geïntegreerd zijn. Elitaire rekombinatie houdt een kompetitie tussen het ouderpaar en de twee nakomelingen. Hierdoor worden de oplossingen uit een vorige generatie enkel vervangen door nieuwe oplossingen die een betere funktiewaarde hebben. Er wordt aangetoond dat het elitaire rekombinatie GA dezelfde selektiedruk heeft als de veelvuldig gebruikte toernooi selektie. Het lokale elitaire mechanisme heeft echter een aantal bijkomende voordelen, bijvoorbeeld het elitaire rekombinatie GA blijkt minder gevoelig te zijn aan ondergedimensioneerde populaties. Eveneens is de moeilijke keuze van de parameters die de rekombinatie snelheid bepalen nu minder belangrijk voor het goed funktioneren van het genetisch algoritme. Daarenboven is het algoritme volledig lokaal implementeerbaar, en hoeft men geen aparte selektie en rekombinatie fase te houden.

Vervolgens wordt nagegaan hoe niche gebaseerde genetische algoritmen zich gedragen met betrekking tot het mengingsprobleem. Na een kort overzicht van verdeel- (sharing) en verdringingstechnieken (crowding), wordt het veralgemeend verdringingsalgoritme voorgesteld. Niche gebaseerde genetische algoritmen worden gebruikt voor het optimalizeren van multimodale funkties omdat ze in staat zijn om meerdere optima terzelfdertijd te vinden. We tonen hier aan dat ze echter ook een belangrijke rol kunnen vervullen bij het optimalizeren van funkties met slechts één globaal optimum. Niche gebaseerde genetische algoritmen zijn immers minder onderhevig aan de selektie-rekombinatie race. Om dit te illustreren passen we het veralgemeend verdringingsalgoritme toe op Holland's Royal Road funktie waarbij de bouwstenen in een specifieke hiërarchie zijn voorgesteld.

Daarna bestuderen we het gebruik van beperkte paarvorming om de mengingsprobabiliteit te verhogen. Door de paarvorming te beperken tot strings die slechts op korte Hamming afstand van elkaar verwijderd zijn, zullen de meeste bouwstenen in beide strings voorkomen, en kunnen ze dus niet vernietigd worden door de kruisingsoperator.

Tot slot wordt een adaptieve kruisingsoperator voorgesteld. Door de uitwisselingsdistributie van de individuele bits adaptief te wijzigen, kunnen we de mengingswaarschijnlijkheid van de bouwstenen aanzienlijk verhogen. Analyse en experimentele resultaten tonen aan dat volledig misleidende funkties (al dan niet multimodaal) hierdoor kunnnen opgelost

worden zonder dat de bouwstenen nauw gelinkt moeten zijn.

B.2 Belangrijkste bijdragen

De belangrijkste bijdragen van deze thesis kunnen als volgt worden samengevat. De referenties van resultaten die reeds gepubliceerd werden, worden eveneens vermeld.

- *Normaal distributie model van toernooi selektie:*

  Toernooi selektie is een veelvuldig gebruikt selektie mechanisme met een konstante en gematigde selektie druk. We hebben een konvergentie model voorgesteld dat de proporties van de optimale bitwaarden in de populatie geeft, in funktie van de tijd gemeten door het aantal generaties. Het model maakt gebruikt van normale distributie theorie en is geldig voor bijvoorbeeld funkties met een onafhankelijke en gelijke kontributie van de genen. Er werd aangetoond dat de konvergentie snelheid van de orde $O(\sqrt{\ell})$ is, met ℓ de lengte van de binaire string. Dit vormt tevens een ondergrens voor de konvergentie complexiteit voor funkties met onafhankelijke kontributie van de genen. Een vergelijking met proportionele en blokselektie werd eveneens gemaakt.

  Referenties zijn Thierens & Goldberg (1994b) and Thierens & Vandewalle (1994).

- *Effekt van gelinkt onevenwicht op GA konvergentie:*

  Door het selekteren van de betere strings in de populatie, worden de bit waarden gecorreleerd. De kruisingsoperator decorreleert deze slechts gedeeltelijk, zodat het netto effekt een opbouw van de covariantie is. Alhoewel dit fenomeen reeds lang erkend wordt in kwantitatieve genetica, werd het nog niet bestudeerd in de kontekst van genetische algoritmen. Helaas zijn de resultaten van de kwantitatieve genetica niet bruikbaar bij de analyse van genetische algoritmen.

  We hebben het effekt van deze covariantie opbouw op de konvergentie eigenschappen bestudeerd en het opgenomen in het normale

distributie model. Er is tevens geargumenteerd dat de covariantie opbouw een belangrijke faktor is voor het begrijpen van vroegtijdige konvergentie, waarbij het genetisch algoritme faalt te konvergeren naar het gewenste optimum.

Referenties zijn Thierens & Goldberg (1994d), en een korte bespreking kan eveneens gevonden worden in Thierens & Goldberg (1994b) en Thierens & Vandewalle (1994).

- *Domino konvergentie model van toernooi selektie:*

  Het normaal distributie model gaf een ondergrens voor de konvergentie complexiteit voor funkties met onafhankelijke kontributie van de genen. Een bovengrens wordt bekomen door de konvergentie te modeleren van funkties met onafhankelijke en exponentieel geschaalde kontributie van de genen. Dit exponentieel schalen veroorzaakt een konvergentie gedrag vergelijkbaar met een rij vallende dominostenen (Rudnick,1992). We hebben een model voor de domino konvergentie opgesteld waarbij de proportie van de optimale bit waarden logistisch toeneemt met een tijdsafhankelijke groeisnelheid. Er is aangetoond dat dit zeer goed benaderd kan worden door de logistische groeisnelheden exponentieel te laten toenemen in de tijd, en exponentieel laten afnemen in funktie van de positie in de binary string van de beschouwde bitwaarde. Dit leidt tot een golfvergelijking met konstante golfsnelheid, waaruit dus blijkt dat de konvergentiesnelheid van orde $O(l)$ is.

- *Dimensionele analyse van allele menging:*

  Er werd aangetoond dat de mengingsanalyse belangrijk is om te kunnen bepalen of een genetisch algoritme zal konvergeren naar het gewenste optimum. Er werd geargumenteerd dat selektie en rekombinatie betrokken zijn in een wederzijdse tijdsrace: indien de selektie snelheid te hoog is relatief ten opzichte van de rekombinatie snelheid, dan zal het genetisch algoritme vroegtijdig konvergeren.

  De menging werd bestudeerd op twee niveaus: eerst werd het niveau van de individuele bits beschouwd, daarna werd het niveau van de bouwstenen bestudeerd. Met behulp van dimensionele analyse werd een mengingsmodel opgesteld dat nauwkeurig beschrijft

waar de grens ligt in de GA parameter ruimte tussen optimaal en vroegtijdig konvergeren.

- *Dimensionele analyse van bouwsteen menging:*

  De dimenionsele analyse van het mengingproces werd eveneens uitgevoerd op het niveau van de bouwstenen. Hier werd aangetoond dat - in tegenstelling met de gangbare opinie - een basis genetisch algoritme in staat is om volledig misleidende funkties optimaal op te lossen met gebruik van uniform crossover, of met andere woorden zonder dat de bouwstenen nauw gelinkt moeten zijn. Dit wordt bekomen door de selektie druk te verhogen. Helaas bijkt ook dat dit in de praktijk slechts mogelijk is voor kleine problemen, vermits de populatie grootte die hiervoor nodig is exponentieel toeneemt met het aantal bouwstenen.

  Een belangrijke konklusie die hieruit volgt is dat het basis genetisch algoritme sterk beperkt is in zijn mengingscapaciteiten. De bouwstenen moeten dus wel nauw gelinkt worden maar dit omwille van de mengingproblematiek en niet om ze te kunnen laten toenemen in de populatie.

  Referenties zijn Thierens & Goldberg (1993).

- *Het elitaire rekombinatie genetisch algoritme:*

  Het elitaire rekombinatie GA is een uiterst eenvoudig genetisch algoritme waarbij de selektie fase en rekombinatie fase geïntegreerd zijn. Er wordt enkel een kompetitie gehouden tussen elk ouderpaar en hun twee nakomelingen. Het elitaire rekombinatie GA heeft dezelfde selektiedruk als toernooiselektie. Ouders kunnen enkel uit de populatie verdwijnen indien hun plaats wordt ingenomen door betere nakomelingen. Door dit lokaal elitaire mechanisme heeft het algoritme een aantal specifieke voordelen.

  Referenties zijn Thierens & Goldberg (1994a).

- *Niches en de selektie-rekombinatie race:*

  De mengingsanalyse leidde tot de identifikatie van de tijdsrace tussen selektie en rekombinatie. We hebben bestudeerd hoe het

gebruik van niches deze race kan vermijden. Een veralgemeend verdringingsalgoritme (generalized crowding algorithm) wordt gebruikt om dit aan te tonen. Niche gebaseerde GAs worden traditioneel enkel gebruikt voor multimodale funktieoptimalizatie, maar doordat ze de selektie-rekombinatie race vermijden zijn ze ook van belang bij funkties met hiërarchisch samengestelde bouwstenen zoals Holland's Royal Road functions.

Er werd eveneens nagegaan in welke mate uitbreidingen aan het basis genetisch algoritme, met name elitaire competitie, niches en beperkte paarvorming de mengingscapaciteit kan verbeteren.

Elitaire competitie verzekert dat goede oplossingen nooit verloren gaan, tenzij ze kunnen vervangen worden door nog betere. Niche technieken vermijden de selektie-rekombinatie race, en beperkte paarvorming zorgt ervoor dat de probabiliteit om bouwstenen te mengen verhoogt.

Samen zorgen deze drie uitbreidingen ervoor dat de bouwstenen steeds aanwezig blijven in de populatie, en dit zonder dat ze nauw gelinkt moeten zijn. Het mengen van de bouwstenen blijft echter inefficiënt indien ze niet nauw gelinkt zijn. Hierdoor is het dus noodzakelijk om ofwel de bouwstenen expliciet te linken, ofwel de kruisingsoperator adaptief aan te passen.

- *Adaptieve kruisingsoperator:*

  De adaptieve kruisingsoperator is ontworpen om de rekombinatie uitwisselingsprobabiliteit aan te passen, zodat deze specifiek afgesteld wordt op de posities van de bouwstenen in de binaire string. De hierboven vermelde uitbreidingen aan het basis genetisch algoritme zorgen ervoor dat de bouwstenen steeds aanwezig blijven in de populatie. De adaptieve kruisingsoperator maakt hiervan gebruik om de uitwisselingsprobabiliteit van de individuele bits te wijzigen, zodat de waarschijnlijkheid om volledige bouwstenen uit te wisselen enorm verhoogt. Hierdoor wordt het mengingsprobleem op efficiënte wijze opgelost. Analyse en experimentele resultaten voor volledig misleidende funkties zonder nauw gelinkte bouwstenen, en voor multimodale volledig misleidende funkties

zonder nauw gelinkte bouwstenen, tonen aan dat het algoritme in staat is deze zeer moeilijke problemen op te lossen.

- *Neurale netwerk representatie voor rekombinatie:*

  Een laatste bijdrage betreft het ontwerp van een niet-redundante neurale netwerk string representatie, die uitermate geschikt is voor het rekombineren van neurale netwerken. Door de aanwezigheid van symmetrieën, kunnen structureel verschillende neurale netwerken toch dezelfde funktionele mapping implementeren. Deze redundanties vergroten de globale gewichtsruimte, en verlagen de efficiëntie van de kruisingoperator. Een zeer eenvoudige neurale netwerk string representatie wordt voorgesteld die deze redundanties elimineert. Er wordt ook aangetoond dat de kruisingoperator meer informatie van de ouders naar de nakomelingen doorgeeft bij het gebruik van de niet-redundante representatie dan bij de redundante representatie. Hierdoor verhoogt de efficiëntie van de kruisingoperator, wat bevestigd wordt door experimentele resultaten.

  Referenties zijn Thierens & Vandewalle (1995).

B.3 Software Omgeving

Alle simulatie resultaten beschreven in deze tekst zijn bekomen door gebruik van zelf ontworpen C code, uitgezonderd de neurale netwerk experimenten in appendix A waar gebruik werd gemaakt van een kombinatie van C code met de Matlab Neural Network Toolbox. We zijn van mening dat het bediscussiëren van de specifieke implementatie details, de lezer onnodig zou afleiden van de belangrijkste argumentaties. Daarom hebben we dit achterwege gelaten. Niettemin worden alle algoritmen voldoende beschreven zodat de geïnteresseerde lezer ze zelf vlot kan implementeren.

De grafieken in deze thesis werden gemaakt met *GNUPLOT*, terwijl de tekst werd gegenereerd met LaTeX.

Appendix C

Notation

ℓ	genotype or string length
n	population size
k	building block length
m	number of building blocks in the genotype
d	fitness building block signal
s	selection pressure
r	recombination rate
p_c	crossover probability
p_x	gene or allele exchange probability
p_s	survival probability
p_r	disruption probability
p_m	mutation probability
t	generation t
t^s	generation t after selection, but before recombinatior
$p(t)$	proportion of optimal bit-values at generation t
t_s	selection time
t_x	mixing time
p_{mix}	mixing probability
p_{BB}	proportion of building blocks
f	fitness function
w	crowding window size
h	schema
$m(h,t)$	number of strings that are member of schema h

$\phi(h,t)$	reproduction ratio of schema h at generation t
$\epsilon(h,t)$	disruption factor of schema h at generation t
$o(h)$	order of schema h
$\delta(h)$	defining length of schema h
$P(t)$	population at generation t
$\bar{f}(t)$	population average fitness at generation t
i	selection intensity
$g(x,t)$	growth rate of gene x at generation t
$p(x,t)$	proportion of optimal bit-values of gene x at gen. t
$\Theta(t)$	genic covariance matrix
$\sigma^2(t)$	population fitness variance
$\sigma^2_{var}(t)$	genic variance
$\sigma_{cov}(t)$	genic covariance
δ	decorrelation factor

Appendix D

References

Baker J.E. (1987). Reducing bias and ineffiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms ICGA-87.* ed. J. Grefenstette. pp.14-21. Lawrence Erlbaum.

Bulmer M.G. (1985). *The Mathematical Theory of Quantitative Genetics.* Oxford University Press.

Burnet F.M. (1959). *The clonal selection theory of immunity.* Vanderbilt University Press, Nashville.

Calvin W.H. (1983). *The Throwing Madonna: Essays on the Brain.* McGraw-Hill, New York.

Cavicchio D.J. (1970). *Intelligent behavior as an adaptation to the task environment.* Doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 25-0199).

Chen A., Lu H., & Hecht-Nielsen R. (1993). On the Geometry of Feedforward Neural Network Error Surfaces. *Neural Computation, Vol.5,* pp.910-927.

CMU (1993). Carnegie-Mellon University Neural Network Benchmark Database, ftp.cs.cmu.edu.

Davis L.(Ed.) (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

De Jong K.A. (1975) *An analysis of the behavior of a class of genetic adaptive systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

Deb K. (1989). *Genetic algorithms in multimodal function optimization.* Master's thesis, TCGA Report No. 89002. University of Alabama, The Clearinghouse for Genetic Algorithms, Tuscaloosa.

Deb K., & Goldberg D.E. (1989). An Investigation of Niche and Species Formation in Genetic Function Optimization. *Proceedings of the Third International Conference on Genetic Algorithms ICGA-89.* ed. J.D. Schaffer. pp.42-50. Morgan Kaufmann.

Deb K., & Goldberg D.E. (1992). *Sufficient conditions for deceptive and binary functions.* IlliGAL Report No. 92001. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Deb K., & Goldberg D.E. (1993). Analyzing deception in trap functions. *Proceedings of Foundations of Genetic Algorithms FOGA-II.* ed. D. Whitley. Morgan Kaufmann.

Deb K., Horn J., & Goldberg D.E. (1993) Multimodal Deceptive Functions. *Complex Systems, Vol.7,* pp.131-153.

Dzubera J. & Whitley D. (1994). Advanced Correlation Analysis of Operators for the Traveling Salesman Problem. *Lecture Notes in Computer Science, Vol.866: Parallel Problem Solving from Nature PPSN-III.* eds. Y. Davidor, H.P. Schwefel, R. Männer. pp.68-77. Springer-Verlag.

Edelman G.M. (1987). *Neural Darwinism: The Theory of Neuronal Group Selection.* Basic Books, New York.

Eshelman L. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. *Proceedings of Foundations of Genetic Algorithms FOGA-I.* ed. G. Rawlings. pp.265-283. Morgan Kaufmann.

Falconer D.S. (1989). *Introduction to Quantitative Genetics.* Longman Scientific & Technical.

Forrest S., & Mitchell M.(1993). Relative building block fitness and the building block hypothesis. *Proceedings of Foundations of Genetic Algorithms FOGA-II.* ed. D. Whitley. Morgan Kaufmann.

Funahashi K.I. (1989). On the Approximate Realization of Continuous Mappings by Neural Networks. *Neural Networks, Vol.2,* pp.183-192.

Goldberg D.E.(1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley Publishing Company.

Goldberg D.E.(1992). Construction of higher-order deceptive functions from low-order Walsh coefficients. *Annals of Mathematics and Artificial Intelligence, Vol.5,* pp.35-48.

Goldberg D.E.(1993). Genetic and evolutionary algorithms come of age. *Communications of the Association for Computing Machinery, Vol.37,* pp.113-119.

Goldberg D.E.(1994). *First Flights at Genetic-Algorithm Kitty Hawk.* IlliGAL Report No. 94008. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Goldberg D.E., & Deb K.(1991). A comparative analysis of selection schemes used in genetic algorithms. *Proceedings of Foundations of Genetic Algorithms FOGA-I.* ed. G. Rawlings. pp.69-93. Morgan Kaufmann.

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

Goldberg D.E., Deb K., & Clark J.H. (1993). Accounting for noise in the sizing of populations. *Proceedings of Foundations of Genetic Algorithms FOGA-II.* ed. D. Whitley. Morgan Kaufmann.

Goldberg D.E., Deb K., & Horn J. (1992). Massive multimodality, deception and genetic algorithms. *Parallel Problem Solving from Nature PPSN-II.* eds. R. Männer & B. Manderick. pp.47-56. North-Holland.

Goldberg D.E., Deb K., Kargupta H., & Harik G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93.* ed. S. Forrest. pp.56-64. Morgan Kaufmann.

Goldberg D.E., Deb K., & Korb B. (1990). Messy genetic algorithms revisited: studies in mixed size and scale. *Complex Systems, Vol.4,* 415-444.

Goldberg D.E., Deb K., & Korb B. (1991). Don't worry, be messy. *Proceedings of the Fourth International Conference on Genetic Algorithms ICGA-91.* eds. R. Belew & L. Booker. pp.24-30. Morgan Kaufmann.

Goldberg D.E., Deb K., & Thierens D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society for Instrumentation and Control Engineers, SICE Vol.32,* No.1 pp.10-16.

Goldberg D.E., Kargupta H., Harik G., Thierens D., & Deb K. (1994). *Strategic Defense and Messy Genetic Algorithms.* IlliGAL Report No. 94009. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Goldberg D.E., Korb B., & Deb K. (1989). Messy genetic algorithms revisited: motivation,analysis, and first results. *Complex Systems, Vol.3,* pp.493-530.

Goldberg D.E., & Richardson J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the Second International Conference on Genetic Algorithms ICGA-87.* ed. J. Grefenstette. pp.41-49. Lawrence Erlbaum.

Goldberg D.E., & Rudnick M. (1991). Genetic Algorithms and the Variance of Fitness. *Complex Systems, Vol.5* pp.265-278.

Goldberg D.E., & Segrest P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms ICGA-87.* ed. J. Grefenstette. pp.1-8. Lawrence Erlbaum.

Hancock P. (1992). Recombination Operators for the Design of Neural Nets by Genetic Algorithm. *Parallel Problem Solving from Nature PPSN-II.* eds. R. Männer & B. Manderick. pp.441-450. North-Holland.

Harik G. (1994) *Finding Multiple Solutions in Problems of Bounded Difficulty.* IlliGAL Report No. 94002. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Holland J.H. (1973). Genetic Algorithms and the Optimal Allocation of Trials. *SIAM Journal of Computing, Vol.2(2),* pp. 88-105.

Holland J.H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Holland,J.H. (1993). Royal Road Functions. *Internet Genetic Algorithms Digest, Vol.6,* No.22.

Hornik K. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks, Vol.2,* pp.359-366.

Inayoshi H., & Manderick B. (1994). The Weighted Graph Bi-Partitioning Problem: A Look at GA Performance. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature PPSN-III.* eds. Y. Davidor, H.P. Schwefel, R. Männer. pp.617-625. Springer-Verlag.

Ipsen D.C.(1960). *Units, dimensions, and dimensionless numbers.* New York: McGraw-Hill.

Levenick J.R. (1991). Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue from Biology. *Proceedings of the Fourth International Conference on Genetic Algorithms ICGA-91.* eds. R. Belew & L. Booker. pp.123-127. Morgan Kaufmann.

Mahfoud S.W. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature PPSN-II.* eds. R. Männer & B. Manderick. pp.27-36. North-Holland.

Mahfoud S.W. (1993). *Simple Analytical Models of Genetic Algortihms for Multimodal Function Optimization.* IlliGAL Report No.93001. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Manderick B., de Weger M., & Spiessens P. (1991). The Genetic Algorithm and the Structure of the Fitness Landscape. *Proceedings of the Fourth International Conference on Genetic Algorithms ICGA-91.* eds. R. Belew & L. Booker. pp.143-150. Morgan Kaufmann.

Mathias K., & Whitley D. (1992). Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. *Parallel Problem Solving from Nature PPSN-II.* eds. R. Männer & B. Manderick. pp.219-228. North-Holland.

Mitchell M., Forrest S., & Holland J. (1992). The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. *Proceedings of the First European Conference on Artificial Life.* MIT Press/Bradford Books.

Mitchell M., & Holland J. (1993). When Will a Genetic Algorithm Outperform Hill-Climbing ? *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93.* ed. S. Forrest. pp.647. Morgan Kaufmann. (longer version: Sante Fe Institute Working Paper).

Mühlenbein H. & Schlierkamp-Voosen D. (1993). Predictive Models for the Breeder Genetic Algorithm. I. Continuous Parameter Optimization. *Evolutionary Computation 1(1):25-49,* MIT Press.

Nguyen D. & Widrow B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference of Neural Networks, Vol.3* pp.21-26.

Oei C., Goldberg D.E, & Chang S.J. (1991). *Tournament selection, niching, and the preservation of diversity.* IlliGAL Report No.91011. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Radcliffe N. (1993). Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications, Vol.1,* pp.67-90.

Riolo R.L. (1992). Survival of the Fittest Bits. *Scientific American, July 1992.*

Rudnick M. (1992). *Genetic Algorithms and Fitness Variance with an Application to the Automated Design of Artificial Neural Networks.* Unpublished doctoral disseration, Oregon Graduate Institute of Science and Technology, Beaverton.

Smith R.E., Forrest S., & Perelson A.S. (1993). Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation, Vol.1,* pp.127-149.

Srinivas M., & Patnaik L.M.(1993). Binomially Distributed Populations for modeling GAs. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93,* pp.138-145. Morgan Kaufmann.

Sussmann H.J. (1992). Uniqueness of the Weights for Minimal Feedforward Nets with a Given Input-Output Map. *Neural Networks, Vol.5,* pp.589-593.

Sveshnikov A.D. (1965). *Problems in probability theory, mathematical statistics and the theory of random functions.* Dover Publications (1978), New York.

Syswerda G. (1989). Uniform Crossover in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms ICGA-89.* ed. J. Schaffer. pp.2-9. Morgan Kaufmann.

Thierens D. (1991) Evolving Neural Networks and Constructive Induction. *Machine Learning Conference Benelearn-91.* Amsterdam (Nl), 1991.

Thierens D., & Goldberg D.E. (1993). Mixing in Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93.* ed. S. Forrest. pp.38-45. Morgan Kaufmann.

Thierens D., & Goldberg D.E. (1994a). Elitist Recombination: an integrated selection recombination GA. *Proceedings of the IEEE World Congress on Computational Intelligence,* Orlando (VS). Vol.1 pp.508-512.

Thierens D., & Goldberg D.E. (1994b). Convergence Models of Genetic Algorithm Selection Schemes. *Lecture Notes in Computer Science, Vol. 866: Parallel Problem Solving from Nature PPSN-III.* Jerusalem (Il). eds. Y. Davidor, H.P. Schwefel, R. Männer. pp.119-129. Springer-Verlag.

Thierens D., & Goldberg D.E. (1994c). *Building Block Processing, Local Elitism and Convergence Complexity.* Technical Report ESAT-SISTA/TR 1993-76. Presented at the International Conference Artificial Life-IV. Boston (VS).

Thierens D., & Goldberg D.E. (1994d). *Effects of Linkage Disequilibrium on GA convergence.* Technical Report ESAT-SISTA/TR 1994-41.

Thierens D., Suykens J., Vandewalle J., & De Moor B. (1993) Genetic Weight Optimization of a Feedforward Neural Network Controller. *International Conference on Artificial Neural Networks and Genetic Algorithms.* Innsbruck (A). eds. Albrecht R., Reeves C., & Steele N. pp.658-663. Springer-Verlag

Thierens D., & Vandewalle J. (1994) Normal Distribution Modeling of Genetic Algorithms. *Machine Learning Conference Benelearn-94.* Rotterdam (Nl), pp.29-38.

Thierens D. & Vandewalle J. (1995) *A Non-Redundant Neural Network Representation for Genetic Recombination.* Technical Report ESAT-SISTA/TR 1995-25.

Thierens D., & Vercauteren L.(1991). A Topology Exploiting Genetic Algorithm to Control Dynamic Systems. *Lecture Notes in Computer Science, Vol. 496: Parallel Problem Solving from Nature PPSN-I.* Dortmund (D). eds. Schwefel H.P., & Männer R. pp.104-108. Springer-Verlag.

Qi X., & Palmieri F.(1993). The Diversification Role of Crossover in the Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93. Urbana-Champaign (VS).* ed. S. Forrest. pp.132-145. Morgan Kaufmann.

Qi X., & Palmieri F. (1994a). Theoretical Analysis of Evolutionary Algorithms with an Infinite Population Size in Continuous Space, Part I:

Basic Properties of Selection and Mutation. *IEEE Transactions on Neural Networks, Vol.5,* No.1, pp.102-119.

Qi X., & Palmieri F. (1994b). Theoretical Analysis of Evolutionary Algorithms with an Infinite Population Size in Continuous Space, Part II: Analysis of the Diversification Role of Crossover. *IEEE Transactions on Neural Networks, Vol.5,* No.1, pp.120-129.

Vercauteren L, Sieben G, Thierens D., Otte,G., & Boullart L.(1991) Enhanced Mapping: an Extension of Topological Mapping to form Internal Representations and Spatial Mappings. *International Conference on Artificial Neural Networks.* Helsinki (Fi), pp.301-309.

Vogl T.P., Mangis J.K., Rigler A.K., Zink W.T., & Alkon D.L. (1988). Accelerating the Convergence of the Backpropagation Method. *Biological Cybernetics, Vol.59,* pp.257-263.

White H. (1990). Connectionist Nonparametric Regression: multilayer feedforward network can learn arbitrary mappings. *Neural Networks, Vol.3,* pp.535-549.

Whitley D. (1989). The Genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. *Proceedings of the Third International Conference on Genetic Algorithms ICGA-89.* ed. J.D. Schaffer. pp.116-123. Morgan Kaufmann.

Whitley D., Starkweather T., & Bogart C. (1990). Genetic Algorithms and Neural Networks: optimizing connections and connectivity. *Parallel Computing, Vol.14,* pp.347-361.