TKK Dissertations 55
Espoo 2006

METHODS FOR IMPROVING RELIABILITY OF EVOLUTIONARY COMPUTATION ALGORITHMS AND ACCELERATING PROBLEM SOLVING

Doctoral Dissertation

Jarno Martikainen

Helsinki University of Technology
Department of Electrical and Communications Engineering
Power Electronics Laboratory

TKK Dissertations 55
Espoo 2006

METHODS FOR IMPROVING RELIABILITY OF EVOLUTIONARY COMPUTATION ALGORITHMS AND ACCELERATING PROBLEM SOLVING

Doctoral Dissertation

Jarno Martikainen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Electrical and Communications Engineering for public examination and debate in Auditorium S3 at Helsinki University of Technology (Espoo, Finland) on the 8th of December, 2006, at 12 noon.

Helsinki University of Technology
Department of Electrical and Communications Engineering
Power Electronics Laboratory

Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Tehoelektroniikan laboratorio

Distribution:
Helsinki University of Technology
Department of Electrical and Communications Engineering
Power Electronics Laboratory
P.O. Box 3000
FI - 02015 TKK
FINLAND
URL: http://powerelectronics.tkk.fi/
Tel. +358-9-451 2431
Fax. +358-9-451 2432
E-mail: martikainen@iki.fi

ISBN-13 978-951-22-8523-5
ISBN-10 951-22-8523-1
ISBN-13 978-951-22-8524-2 (PDF)
ISBN-10 951-22-8524-X (PDF)
ISSN 1795-2239
ISSN 1795-4584 (PDF)
URL: http://lib.tkk.fi/Diss/2006/isbn951228524X/

TKK-DISS-2227

Otamedia Oy
Espoo 2006

HELSINKI UNIVERSITY OF TECHNOLOGY P. O. BOX 1000, FI-02015 TKK http://www.tkk.fi	ABSTRACT OF DOCTORAL DISSERTATION

Author Jarno Martikainen

Name of the dissertation

Methods for Improving Reliability of Evolutionary Computation Algorithms and Accelerating Problem Solving

Date of manuscript	Date of the dissertation 12/08/2006
Monograph	[✓ Article dissertation (summary + original articles)]

Department	Department of Electrical and Communications Engineering
Laboratory	Power Electronics Laboratory
Field of research	Electrical Engineering
Opponent(s)	Dr. Jonathan Timmis
Supervisor	Prof. Seppo J. Ovaska
(Instructor)	

Abstract

This dissertation deals with improving the reliability of evolutionary computation algorithms and accelerating problem-solving in optimization problems. Evolutionary algorithms have proven their value in difficult optimization problems that are not usually solvable in decent time using conventional optimization methods. However, evolutionary computation methods still suffer from problems related especially to premature convergence and the lengthy run times of the algorithms. In addition, the field of evolutionary computation does not commonly use the widely accepted practices for the comprehensive statistical comparison of two different evolutionary algorithms.

This dissertation aims at improving the process of using evolutionary computation in complex optimization problems from three perspectives. First, new algorithms are proposed for demanding optimization tasks. These algorithms rely on two perspectives, using a new multipopulation approach to enable appropriate conditions for candidate solutions to evolve and fusing evolutionary algorithms with other soft computing technologies, such as fuzzy logic, in a new way. Second, this dissertation discusses a method for reducing the computational time taken to evaluate a computationally demanding objective function value using neural network-based approximations. Third, a statistical method for comparing the results produced by two different evolutionary algorithms is illustrated. This method, relying on bootstrap resampling-based multiple hypothesis testing, is known outside the field of evolutionary computation, but has not been used within the evolutionary computing community. This dissertation illustrates the use of the statistical scheme and studies the parameters affecting the interpretation of its results.

The improvements to evolutionary algorithms this dissertation proposes have been proven to be beneficial by extensive testing. The proposed algorithms and the means to reduce the time required by the objective function evaluation have shown an increase in performance when compared to the reference algorithms. This dissertation also aims at awakening discussion related to the proper use of statistics in the field of evolutionary computation.

Keywords Evolutionary computation, hybrid algorithm, optimization, statistical comparison.

ISBN (printed)	951-22-8523-1	ISSN (printed)	1795-2239
ISBN (pdf)	951-22-8524-X	ISSN (pdf)	1795-4584
ISBN (others)		Number of pages	82 p. + app. 67 p.

Publisher Otamedia Oy

Print distribution Power Electronics Laboratory

[✓ The dissertation can be read at http://lib.tkk.fi/Diss/2006/isbn951228524X/]

TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi	VÄITÖSKIRJAN TIIVISTELMÄ

Tekijä Jarno Martikainen

Väitöskirjan nimi

Menetelmiä evoluutioalgoritmien luotettavuuden parantamiseksi ja ongelmanratkaisun nopeuttamiseksi

Käsikirjoituksen jättämispäivämäärä	Väitöstilaisuuden ajankohta 8.12.2006
☐ Monografia	[✓ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)]

Osasto	Sähkö- ja tietoliikennetekniikan osasto
Laboratorio	Tehoelektroniikan laboratorio
Tutkimusala	Sähkötekniikka
Vastaväittäjä(t)	Dr. Jonathan Timmis
Työn valvoja	Prof. Seppo J. Ovaska
(Työn ohjaaja)	

Tiivistelmä

Tämä väitöskirja käsittelee evoluutioalgoritmien luotettavuuden parantamista ja ongelmanratkaisun nopeuttamista optimointiongelmissa. Evoluutioalgoritmeja on käytetty menestyksekkäästi vaikeissa optimointiongelmissa, joita ei yleensä pystytä ratkaisemaan perinteisillä menetelmillä kohtuullisessa ajassa. Evoluutioalgoritmeilla on kuitenkin heikkouksia liittyen erityisesti ennenaikaiseen konvergoitumiseen ja algoritmien pitkiin suoritusaikoihin. Lisäksi evoluutiolaskennan alalla ei juurikaan käytetä yleisesti hyväksyttyjä menetelmiä kahden evoluutioalgoritmin perusteelliseen tilastolliseen vertailuun.

Tässä väitöskirjassa esitetään parannuksia evoluutioalgoritmien käyttämiseen vaikeissa optimointiongelmissa kolmesta eri näkökulmasta. Ensiksi, työssä esitellään uusia algoritmeja, joissa monen populaation avulla järjestetään ratkaisuehdokkaille sopivat olosuhteet kehittyä ja joissa evoluutiolaskentaan sulautetaan uudella tavalla eri pehmeän laskennan tekniikoita, kuten sumeaa logiikkaa. Toiseksi, tässä työssä esitetään menetelmä, jolla voidaan lyhentää laskennallisesti vaativan kustannusfunktion arvon laskemisen vaatimaa aikaa approksimoimalla kustannusfunktion osia neuroverkoilla. Kolmanneksi, väitöskirjassa esitellään tilastollinen menetelmä kahden evoluutioalgoritmin vertailemiseksi. Tämä bootstrap-näytteistämiseen perustuva usean hypoteesin testaamisen menetelmä on tunnettu monilla muilla tieteen aloilla, mutta sitä ei ole käytetty evoluutiolaskennan piirissä. Tässä työssä tutkitaan myös kyseisen tilastollisen menetelmän parametrien arvojen vaikutusta tulosten tulkittavuuteen.

Väitöskirjassa esitetyt parannukset on todettu hyödyllisiksi perinpohjaisella testaamisella. Sekä esitellyt algoritmit että kustannusfunktion laskemisen nopeuttamiseksi kehitetty menetelmä parantavat osoitetusti perusalgoritmien suorituskykyä. Tämän työn tarkoituksena on myös herättää keskustelua luotettavien tilastollisten menetelmien käytöstä evoluutiolaskennan piirissä.

Asiasanat Evoluutiolaskenta, hybridialgoritmi, optimointi, tilastollinen vertailu.

ISBN (painettu)	951-22-8523-1	ISSN (painettu)	1795-2239
ISBN (pdf)	951-22-8524-X	ISSN (pdf)	1795-4584
ISBN (muut)		Sivumäärä	82 s. + liit. 67 s.

Julkaisija Otamedia Oy

Painetun väitöskirjan jakelu Tehoelektroniikan laboratorio

[✓ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/2006/isbn951228524X/]

Preface

The process of learning how to write a Ph.D. dissertation (nothing more, nothing less) has been a spine-tingling experience for me. During this joyous journey, I've been fortunate enough to be accompanied by amazing people, to whom I here wish to express my deepest gratitude.

My supervisor, Prof. Seppo Ovaska, has shown enormous patience and dedication towards my work. During these first and, at times, heavily trembling steps in the academic community, Seppo has always offered his professional and supportive advice while simultaneously giving me quite a bit of freedom with the everyday routines. Seppo, it has truly been a pleasure and a privilege to work with you; thank you.

The staff of the Power Electronics Laboratory have created a pleasant atmosphere for me to work in. I'd like to thank Prof. Jorma Kyyrä and Prof. Jorma Luomi for their support and for giving me the opportunity to work there in the first place. Docent Xiao-Zhi Gao and secretary Anja Meuronen have always answered my tricky questions and saved my day more than a dozen times. I'd like to thank David Shilane from the University of California, Berkeley, for our excellent collaboration and wish him all the best for the future.

The phenomenal crew of the Signal Processing Laboratory (HUT) have shown me how to take a scientific approach to everything and have loads of fun in the meantime. I can only wish all of you the best of luck with your future projects!

Juho, Kalle, Maza, Sage, Tommi, Tuomo, and others; well, that's probably the finest mix of diversity, dynamics, and zenitism (if there is such a word) I've ever seen. Kiitos ja anteeksi.

Ilan, Jakke, Juha, Jussi, Late, Mauri, Sirén, Tirkkonen, and Ässät; thanks for the countless moments of glorious victories and bitter defeats.

My dear parents, my sister Nakke, and my grandparents have taught me the most important things in life. A heartfelt "thank you" for everything; I could not possibly have asked or hoped for anything more.

Niina, this work, among other more important things, would never have happened without your endless love and support.

I'm deeply grateful for the highly professional and insightful feedback from the pre-examiners of this work, Dr. David B. Fogel and Dr. Randy L. Haupt. Their valuable comments and constructive criticism helped to improve this work enormously.

I gratefully acknowledge the financial support of the Ulla Tuominen Foundation, Academy of Finland (Grant 214144), and Tekniikan edistämissäätiö. Ruth Wilmi and Simon Gill did a great job proofreading this work.

"And so castles made of sand fall into the sea, eventually..."

Jimi Hendrix

Otaniemi, December, 2006

Jarno Martikainen

Contents

List of Abbreviations

2PGA	Two-population genetic algorithm
2PES	Two-population evolution strategy
ACO	Ant colony optimization
AIS	Artificial immune system
cEC	Cellular evolutionary computation
CEC	IEEE Congress on Evolutionary Computation
CPU	Central processing unit
CSP	Clonal selection principle
DE	Differential evolution
dEC	Distributed evolutionary computation
EC	Evolutionary computation
EP	Evolutionary programming
ES	Evolution strategy
FIR	Finite impulse response
FL	Fuzzy logic
GA	Genetic algorithm
GECCO	Genetic and Evolutionary Computation Conference
GP	Genetic programming
HC	Hard computing
IEC	Interactive evolutionary computation
IEEE	Institute of Electrical and Electronics Engineers
LCS	Learning classifier system
MGP	Multiplicative general parameter
MLP	Multilayer perceptron
MPGA	Multipopulation genetic algorithm
NFL	No free lunch theorem
NN	Neural network
PEC	Parallel evolutionary computation
PSO	Particle swarm optimization
RNG	Random number generator
SC	Soft computing
TSP	Traveling salesman problem

List of Symbols

β	Blending crossover weighting factor
B	Number of bootstrap resamplings
b	Bias term in a neural network
δ	Weighting factor in gradient descent method
f_a	Activation function in a neural network
$f_{o,ave}(g)$	Average objective function value of individuals
$f_{o,max}(g)$	Maximum objective function value of individuals
$f_{o,min}(g)$	Minimum objective function value of individuals
f_o	Objective function
g	Generation
$\bar{\sigma}_i^2$	Variance of the data set S_i
$\bar{\sigma}_i^{2*}$	Variance of the resampled data set S_i^*
$\hat{\theta}$	Test statistic calculated from the actual data
$\hat{\theta}^*$	Test statistic calculated from the resampled data
H_0	Null hypothesis
k	Sample set size
m_{S12}	Common mean of data sets S_1 and S_2
n	Dimension of a problem
n_i	Number of individuals selected for the next generation
n_I	Number of individuals in the initial population
$n_{tournament}$	Number of individuals selected for a single tournament
n_{rank}	Number of individuals selected for the next generation (rank selection)
$\mathrm{p}_{\mathrm{boot}}$	Bootstrap-based estimate of the p-value
p_i	Linguistic definition in fuzzy logic
p_m	Mutation probability
p_r	Reproduction probability
S_i	Data set i
$\overline{S}_i$	Mean of data set i
$S_{i,adj}$	Data set S_i with adjusted mean
S_i^*	Resampled data set S_i
V_i	Variable in fuzzy logic
w_i	Weight of input i in a neural network
x_i	Solution vector i
$x_{nn,i}$	Input i of a neural network
$y_{nn,i}$	Output of neuron i in a neural network
y_{nn}	Output of a neural network

List of Publications

This dissertation consists of a summary and the following eight publications, which are referred to as [P1]-[P8] in the text:

[P1] J. Martikainen and S. J. Ovaska, “Designing multiplicative general parameter filters using adaptive genetic algorithms,” in *Proc. of the Genetic and Evolutionary Computation Conference*, Seattle, WA, 2004, pp. 1162-1167.

[P2] J. Martikainen and S. J. Ovaska, “Designing multiplicative general parameter filters using multipopulation genetic algorithms,” in *Proc. of the 6th Nordic Signal Processing Symposium*, Espoo, Finland, 2004, pp. 25-28.

[P3] J. Martikainen and S. J. Ovaska, “Fitness function approximation by neural networks in the optimization of MGP-FIR filters,” in *Proc. of the IEEE Mountain Workshop on Adaptive and Learning Systems*, Logan, UT, 2006, pp. 231-236.

[P4] J. Martikainen and S. J. Ovaska, “Hierarchical two-population genetic algorithm,” *International Journal of Computational Intelligence Research*, vol. 2, no. 4, 2006, in press.

[P5] J. Martikainen and S. J. Ovaska, “Optimizing dynamical fuzzy systems using aging evolution strategies,” in *Proc. of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing*, Benidorm, Spain, 2005, pp. 5-10.

[P6] J. Martikainen and S. J. Ovaska, “Using fuzzy evolutionary programming to solve traveling salesman problems,” in *Proc. of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing*, Benidorm, Spain, 2005, pp. 49-54.

[P7] D. Shilane, J. Martikainen, S. Dudoit, and S. J. Ovaska, “A general framework for statistical performance comparison of evolutionary computation algorithms,” in *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, Innsbruck, Austria, 2006, pp. 7-12.

[P8] J. Martikainen and S. J. Ovaska, “Comparison of a fuzzy EP algorithm and an AIS in dynamic optimization tasks,” in *Proc. of the IEEE Mountain Workshop on Adaptive and Learning Systems*, Logan, UT, 2006, pp. 7-12.

1. Introduction

This dissertation tackles some of the most challenging aspects of using evolutionary computation (EC) algorithms in solving complex application-specific optimization problems, namely, increasing the reliability of the algorithms and accelerating problem-solving. Reliability here refers to the algorithms' capability to produce competitive results despite the varying attributes of the optimization process. Evolutionary computation algorithms, a class of nature-inspired optimization methods, have been studied extensively during the last five decades. But using these algorithms for optimization is a multifaceted process, and a lot of research still remains to be done. First, the optimization problem at hand has to be mapped to the EC algorithm so that the optimization algorithm can tackle it effectively. Second, the optimization algorithm usually has to be tailored specifically to match the requirements of the present task to produce the required performance. Third, in many cases multiple techniques have to be compared before a decision is made as to which method is eventually used. The need for ever more powerful, i.e. more application-specific, optimization algorithms is clear when we consider the increasingly complicated optimization tasks the modern world offers us. In effect, this dissertation does not have a single purpose; rather, it aims in a versatile way at elaborating the process of using evolutionary computation algorithms for complex optimization problems.

The contribution of the work presented here consists of eight publications, [P1]-[P8], divided into three separate branches of research, as shown in Fig. 1: the modeling of an optimization problem in a computationally efficient form, the development of application-specific evolutionary algorithms, and the use of a comprehensive statistical scheme for evaluating the differences between two separate algorithms.

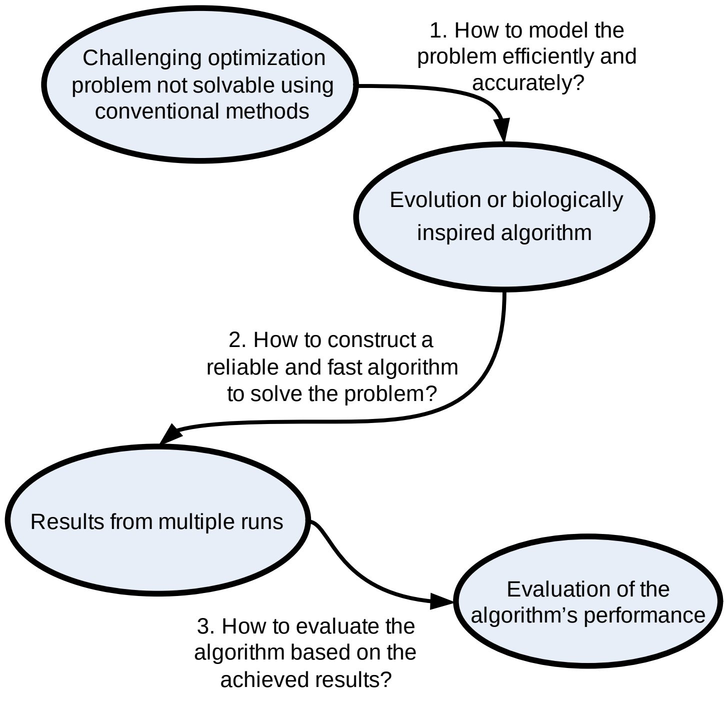

Challenging optimization problem not solvable using conventional methods
1. How to model the problem efficiently and accurately?
Evolution or biologically inspired algorithm
2. How to construct a reliable and fast algorithm to solve the problem?
Results from multiple runs
Evaluation of the algorithm's performance
3. How to evaluate the algorithm based on the achieved results?

Fig. 1. The three branches of EC research studied in this dissertation.

The efficient modeling of a system to be optimized is studied in [P3]. Especially in applications where single optimization runs can last for days, tailoring a high-performance algorithm to meet the requirements of that particular problem is highly desirable. However, even the best algorithms cannot alleviate the burden imposed by a computationally exhaustive objective function. The method proposed in this dissertation aims at reducing the time required by the objective function evaluation using neural network-based approximations. The problem of modeling an objective function efficiently can be interpreted as process identification, as, e.g., in [Kau67]. In that early paper a simple evolutionary scheme based on variation and selection was used to evolve a model for a single-input-single-output plant. The evolutionary scheme was able to choose a number of predefined transfer function blocks that were then concatenated to create the overall model of the plant. Computational complexity and the computing power available at that time (1967) made the proposed scheme slow, but the general concept of evolving models for system identification was found beneficial. Later, e.g. in [Wil03], Willmes et al. studied the use of neural networks (NN) to approximate the fitness function values of benchmark functions. That work consisted of trials in which a neural network was trained off-line and used for approximating the fitness function, or trained off-line and updated on-line. The paper concluded that updating the neural network on-line is capable of producing better solutions than is the case with its off-line counterpart. However, severe reliability issues were reported and additional research was called for. In [P3] a neural network is used to approximate certain components of a fitness function when designing a digital filter and the results show that the computational burden can be halved using this approach. Our work differs considerably from previous studies. First of all, the scheme introduced in [P3] uses a neural network for the multiple-input-multiple-output approximation of an objective function and the network is not trained off-line; rather, only the network topology is decided beforehand, but the training is done completely on-line. Clearly, the network in [P3] can freely evolve in a desired direction without the need to make the best of any pre-defined blocks. Publication [P3] concludes that the proposed method is capable of accelerating considerably the fitness function calculations in this application and, thus, adding to the performance of the entire method.

Publications [P1], [P2], [P4], [P5], and [P6] are studies on reliable evolutionary algorithms tailored for specific applications. Publication [P1] studies the effects of using adaptive parameters within a genetic algorithm. As a test case [P1] features a demanding digital filter design problem. Digital filters have been designed using evolutionary computation in numerous studies and various schemes have been proposed. Among others, in [Rao96] the coefficients of the filter could be selected freely from a discrete set, and in addition, no crossover operators were used to modify the candidate solutions. In [Dex95] digital filters are designed using a standard GA, but using parallel hardware. The research on filter design is usually focused on the application, and the evolutionary computation methods used for that task are rather simple and straightforward. More advanced schemes, such as using adaptive variation probabilities, as studied in [P1], are less common. In [P2] a multipopulation genetic algorithm (GA) is used to solve the same filter design problem as in [P1]. Using a GA consisting of multiple solution populations is the first step of this study towards controlling the whole solution population adaptively instead of adapting each solution separately. In general, many evolutionary computation methods try to enhance a single solution by adjusting its control parameters adaptively, e.g. the variation probabilities, but the approach proposed in this dissertation concentrates on entire populations. In other words, by adjusting the characteristics of the whole population instead of a single solution, advantageous conditions for creating competitive candidate solutions can be created. Publication [P4] is a study focused on the effect of the various parameters existing in the multipopulation genetic algorithm scheme proposed in [P3]. Publications [P5] and [P6] are studies of fusing together fuzzy logic (FL) and evolutionary algorithms. In [P5] and [P6] the parameters of a dynamic fuzzy system and a demanding combinatorial problem are solved using adaptive evolution strategies and evolutionary programming, respectively. The approach of using fuzzy logic to control the remaining lifetime of solutions in [P5] and the repetitive partitioning of a comprehensive problem into subproblems in [P6] are new.

Publications [P7] and [P8] discuss a methodology for a proper statistical comparison of two different evolutionary algorithms. In the field of evolutionary computation the widely accepted methods for statistical comparison are not commonly applied. This is not due to the scientists' lack of knowledge regarding such statistical methods; rather, the field of evolutionary computation has simply not placed enough stress on the need for proper statistical verification of results. This issue is, however, especially important, since most results reported in the EC community are based on empirical research rather than theoretical analysis. The work presented here illustrates a comprehensive scheme to compare statistically the performance of two candidate algorithms and hopes to awaken discussion on the matter. Publication [P7] explains a statistical scheme based on bootstrap resampling and multiple hypothesis testing for comparing two data-generating algorithms. Publication [P8] is an empirical study concerning the sensitivity of the parameters of the scheme in [P7], and [P8] also offers a comparison of evolutionary programming and artificial immune systems in a dynamic environment.

The methods proposed in this dissertation, among others, can be studied either theoretically, empirically, or both. Theoretical proof outpowers empirical results and it is naturally desirable, but such comprehensive theoretical results may occasionally be hard, if not impossible, to derive. Evolutionary algorithms are complex stochastic procedures, and often simplifications are required to make theoretical inspection of evolutionary algorithms feasible. Then again, the problem follows of how much the theoretical results achieved using simplifications correspond to reality. Through carefully designed experiments and objectively analyzing the results using proper statistical methods it is possible to draw valuable conclusions regarding the outcome of the research under certain conditions. However, one must be careful when generalizing such results, since empirical results only tell us about the exact process under study. The work presented in this dissertation concentrates mainly on empirical results. The methods proposed in this dissertation cannot be theoretically proven to be *generally* superior to other methods. However, the same applies to all optimization methods; important results achieved more than a decade ago, discussed in Chapter 2 of this work, state that no optimization algorithm is better than another over all the optimization problems. The methods proposed in this dissertation are not intended to yield a superior performance over a large collection of problems; rather, they offer increased performance over a specific problem or subset of problems. Thus, carefully planned and properly conducted empiricism and appropriate statistical analyses validates the results in this dissertation.

The rest of the dissertation is organized as follows. Chapter 2 discusses the role of soft computing and hard computing methods in optimization in general. Section 3 is a description of evolutionary computation methods for optimization and the operators related to them. Chapter 4 concentrates on some of the most popular methods used to improve the reliability and performance of evolutionary computation algorithms. Chapter 5 describes the process of statistically comparing the performance of evolutionary algorithms. In Chapter 6, the main results of the publications [P1]-[P8] and the contributions of the author are summarized. Finally, Chapter 7 contains conclusions, evaluates the dissertation's scientific value, and makes suggestions for future research and development.

2. Soft Computing Methods in Optimization

Optimization is a part of our everyday life. For most everyday problems, and many engineering ones, convenient deterministic solutions have been created. The current problem parameters are manipulated by a well-defined method, and we obtain an optimal or competitive solution. For example, when designing a digital filter we need to define the passband and stopband characteristics, and there exist various methods for solving the filter parameters in such a way that the design criteria are met [Ife93]. However, in many complex cases there are no methods to determine whether a certain solution is a global extremum unless all the possible solutions are evaluated. In fact, it may even be difficult to know whether a problem is difficult or not in the first place. The global extrema solutions of problems, although theoretically interesting, are usually not required for a practical system to produce satisfactory or even competitive results.

This chapter discusses the role of conventional optimization, as well as the emerging natural evolution-based optimization methods used in modern-day engineering problems. Both are definitely needed, but when should these new methods be used and why do the traditional methods not always perform as desired?

2.1 Soft Computing vs. Hard Computing

Optimization methods can generally be divided into conventional computing methods, *hard computing* (HC), and nature-inspired *soft computing* (SC) methods. HC methods are based on theoretically well-defined practices. Examples of popular hard computing methods in optimization are the steepest gradient approach, linear programming, and Newton's method. Then again, SC methods, such as evolutionary computation, are considered as a group of methods designed on the basis of the principles of natural phenomena, and HC methods are the opposite of this, namely, methods not directly mimicking natural models.

Traditional HC methods of optimization usually rely on derivative information, if such information exists, obtained from the *fitness landscape*. This term describes the value of different solutions in the solution space. Figures 2a and 2b illustrate two very different fitness landscapes. In Fig. 2a, a simple derivative method, such as Newton's method, is guaranteed to find the global maximum every time, regardless of the starting point. However, when considering the fitness landscape in Fig. 2b, it is by no means clear where the algorithm will end. The optimum that is found clearly depends on the starting point.

There is a vast reservoir of traditional optimization methods around and these can be summarized into three classes: calculus-based methods, enumeration methods, and random methods. Goldberg [Gol89] describes these classes as follows. Calculus-based methods have been studied extensively and they have been proven to work in a variety of practical problems. Calculus-based methods can be divided into indirect and direct methods. Indirect methods usually solve a set of nonlinear equations that result from the gradient of the objective function being set to zero. An extremum found using this method is usually local. Direct search methods, on the other hand, move towards the local gradient at a certain point. However, both of these methods lack robustness. Robustness in this case means the ability to find competitive solutions despite the difficulty, i.e., multimodality, of the fitness function. Unless some sort of random restart mechanism is added, the calculus-based methods get stuck at the first extremum they find, whether it is global or local. Another severe limitation of calculus-based methods is the fact that they require the objective function to be continuous and that there exists a first, or sometimes also a second, derivative at each point. In many challenging modern-day problems this is usually not the case. For example, when the setup of an electronic circuit is being optimized, there is only a discrete set of components available. Thus, no continuous domain exists and therefore neither does derivative information.

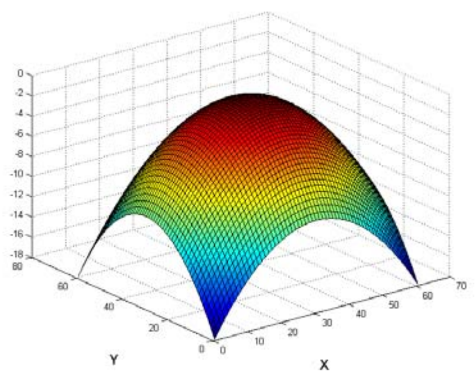


Fig. 2a. A simple fitness landscape.

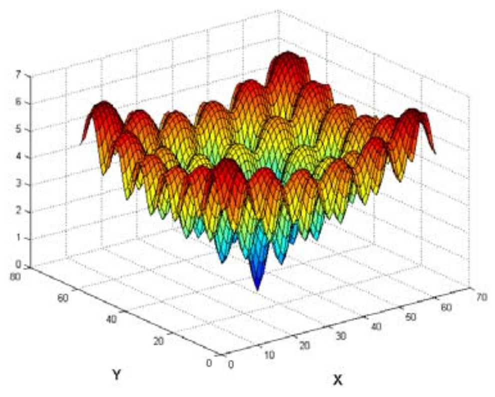


Fig. 2b. A complex fitness landscape.

Enumerative search is a so-called *brute force* method. This means that the algorithm calculates the objective function values at every point within the search space. Needless to say, this method is very convenient for very small search spaces, but for many problems it rapidly becomes extremely inefficient and basically useless as the dimensions of the problem increase. This is clear when a traveling salesman problem (TSP) is considered, for example. In short, a TSP is solved when the shortest path connecting a given number of cities is found. The problem is easy when the number of cities is small, e.g. of the order of 10. All the solutions can easily be evaluated and the shortest path can be selected. However, even with a moderately large number of cities, say 100, the evaluation process of all the paths becomes very time-consuming. The number of possible solutions is in this case 99!/2. Random search methods take *random walks* through the search space and they can save the best objective function value encountered. More information on some of the elementary HC-based optimization methods is found in [Rao78]. Additionally, HC-based optimization is an active research area, and new research results are frequently introduced at conferences and in journals.

The number of difficult optimization problems is increasing all the time in our everyday life, especially in the engineering sciences. During the last five decades the field of evolutionary computation has pursued the goal of offering nature-inspired solutions to ever more complex optimization tasks that are not reasonably solvable using conventional methods. Evolutionary computation methods are among the fundamental soft computing techniques, and, as Zadeh has pointed out in [Zad97], SC, also well justified theoretically, differs from HC in that, unlike HC, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In addition, the capability of SC methods to produce impressive results in difficult optimization tasks is due to the flexibility of these methods in conforming to the current problem. Popular soft computing methods include, in addition to EC, fuzzy logic and neural networks. These methods, although all emerging from nature, have different purposes: NN is used for learning and approximation, FL for approximate reasoning, and EC as a systematic random search.

An illustrative example of HC and SC methods' fundamentally different approaches to solving real-world control problems is given in [Sic98]. In that paper three different approaches were taken to a control problem. The HC- and SC-based methods were all capable of performing the task, but the SC methods were acknowledged for their flexibility towards varying circumstances, whereas the HC methods performed the given task computationally efficiently.

2.2 Soft Computing-Based Optimization: Nature as a Role Model

It is common for biological entities to consciously or unconsciously make the things they do better in order to cope with the requirements of the surrounding environment. In other words, they optimize their performance with respect to some measurable or immeasurable characteristics in order to survive or to succeed in some other way. After an optimization process, these optimized biological entities, or any systems in general, are improved in terms of speed, stamina, pleasure, or some other measure that is appropriate for those specific circumstances. In 1859 Charles Darwin described the evolution of species as an optimization process, the primary tools of which are variation and selection [Dar59]. Many of Darwin's findings have been elaborated by the achievements of modern-day science, such as genetics, and Steve Jones, a Professor of Genetics at University College, London, rewrites the work of Darwin in the light of early 21st-century knowledge in [Jon00].

The basic principles of evolution are easily explained using an example. An animal population evolves over time and reproduces, creating individuals ever better suited to meeting the demands of the environment. In this case the demands of the surrounding environment constitute the *objective function*, i.e. the value of the objective function tells us how well each animal survives in the current conditions. Good characteristics of an individual could be, for example, the ability to find food and shelter easily.

Good individuals may live longer and get a greater chance of creating offspring and thus passing their characteristics on to new generations. This phenomenon of the survival of the fittest can be described as a process of selection. Reproduction, or passing one's genes on to the next generation, can be carried out by means of mating, in which both the parents contribute to the *genetic* or *trait characteristics* of the offspring, or by means of asexual reproduction. In asexual reproduction descendants are created from a single parent and this kind of reproduction scheme is common, e.g., among plants and fungi. Variation is introduced to the animal population by mixing individuals' genetic codes in reproduction and, in addition, by a mutation that can randomly alter the genetic code of an individual. Variation is essential for evolution, for if no new genetic material or combinations are introduced into the population, the evolution of the population stops. This mechanism of variation and selection has shaped, e.g., animals from scratch to what they are today.

Natural evolution can be translated into a computational optimization technique, as follows. First, a *population* of random solutions is created. The population here refers to an animal species in the previous example, where a single solution corresponds to an individual animal. There also needs to be a way to calculate a value of the objective function, i.e. the function that is being optimized, for each and every individual. In nature, individuals struggle or survive, or both, and in EC methods we get an output produced by the objective function that can be arranged into an ordered list describing how good a solution this specific individual is. Reproduction is carried out in EC methods in such a way that a single solution is modified or multiple solutions are combined to create offspring solutions, the characteristics of which are a combination of the characteristics of a single parent or more parents. Additionally, some or all characteristics of the solution can be mutated so as to induce further variation in the solution population. Finally, those solutions that produce the best objective function values have the best chances in probabilistic terms of making it to the next generation to reproduce. This is how variation and selection are implemented in EC methods. The details of the operators used in evolutionary computation algorithms are discussed in Chapter 3.

Computational methods based on natural evolution can, for now, only be a pale shadow of what nature is implementing every day. Still, taking the basic elements of evolution, namely variation and selection, it has been shown that robust and powerful optimization methods can be created. In the context of using natural phenomena as models for optimization, novel paradigms are proposed and accepted, not necessarily for being faithful to their sources of inspiration, but for being useful and feasible. At this point it is necessary to bear in mind that evolutionary computation methods only model evolution and natural behavior from the genetic or trait point of view. The *phenotype*, the final outcome of an individual in nature, is an extremely complex combination of genetics and environment that is very difficult, if not impossible, to understand and model on the basis of current knowledge. Therefore, evolutionary computation in most cases neglects the effect of environmental factors.

The field of evolutionary computation is not fragmenting: rather, it is uniting from fragments based on different aspects and implementations of variation and selection. The main paths that should finally lead to a unified evolutionary computation field are *genetic algorithms* (GA), *evolution strategies* (ES), and *evolutionary programming* (EP). Additionally, other biologically inspired methods, such as *artificial immune systems* (AIS), are nowadays considered as a sub-group of evolutionary computation. Evolutionary computation has established its place in the fields of science and engineering, and an impressive but by no means complete list of recent application areas where evolutionary computation has been successfully applied is given below:

- Acoustics [Sat02]
- Aerospace engineering [Oba00]
- Astronomy and astrophysics [Cha95]
- Arts [Hau00]
- Chemistry [Ass98]

- Digital signal processing [Bri01]
- Electrical engineering [Dav97]
- Financial markets [Mah96]
- Gaming [Fog01]
- Geophysics [Sam93]
- Healthcare [Ji06]
- Materials engineering [Gir02]
- Mathematics and algorithms [Hau98]
- Medical engineering [Yon02]
- Military and law enforcement [Kew02]
- Molecular biology [Koz99]
- Music [Une03]
- Pattern recognition and data mining [Au03]
- Routing and scheduling [Bur99]
- Sports [Sch00]
- Systems engineering [Ben02]

An important moment for the EC community was the year 1997, when the Institute of Electrical and Electronics Engineers (IEEE) started publishing the *IEEE Transactions on Evolutionary Computation* [Tec06]. Each acclaimed research area within the electrical and electronics engineering community has its own publication in this series. Together with conferences, this publication is one of the most respected sources of research reporting in the field today. Among others, *Evolutionary Computation* [Evo06], a journal published by the MIT Press, is also a recognized publication in its field.

Although evolutionary computation algorithms have proven to be powerful optimization tools, there is still a great deal to be done in terms of the reliability and robustness of the algorithms and accelerating-problem solving. It has been shown, e.g. in [Fog06], that if the operators of the evolutionary algorithm fulfill certain requirements, the evolutionary algorithm is guaranteed to achieve the global optimum. The requirements are an elitist selection operator and the capability of a variation operator to transform a solution from any state to any other state (see Chapter 3 for details). Unfortunately, there are no guarantees regarding the time it takes the algorithm to arrive at such an optimum. Nowadays there is often no time to run an optimization algorithm dozens of times to get a satisfactory result: instead, competitive results should be achieved using as few runs as possible.

2.3 Capabilities of Evolutionary Computation

Evolutionary computation methods have proven their power in many demanding real-world optimization tasks. In many fields there exists a deterministic hard computing-based procedure to obtain optimal or near-optimal methods quite conveniently. If such a solution scheme exists for a particular problem, then such a method should definitely be used, for computationally they usually perform very efficiently. Sometimes, for the reasons described earlier, traditional methods are insufficient, and maybe then EC methods alone or EC methods implemented together with traditional hard computing methods can offer a competitive solution.

Wolpert and Macready stated in their seminal paper [Wol97] that there are no free lunches. The well-known *no free lunch* (NFL) theorem stated simply that all the imaginable optimization algorithms perform equally well *on the average* if all the world's optimization problems are considered. The NFL theorem has had a remarkable impact on the EC community, and thus readers are advised to read the literature published before the introduction of the NFL theorem cautiously. The reader may ask what the sense is of developing optimization algorithms if they all perform equally well in general. An optimization algorithm for all the world's optimization problems is naturally desirable, but eventually practitioners are only interested in algorithms that reliably produce good results for their specific problem or class of problems. This, in fact, is another way to understand the NFL theorem: it is certainly possible to create algorithms that perform better than others for some problem or problems.

There exist applications in which SC or HC methods excel alone. However, the fusion of the two offers a wide variety of methodologies from which powerful and robust optimization methods can be created. Kamiya gives an illustrative example of such a system in [Kam04], in which a general model for a control system for a large-scale plant, such as a chemical or electric power plant, is considered. The control system of such a plant can be divided into forecasting, scheduling, supervisory control, and local control. Scheduling is critical for the plant's efficient operation. Scheduling can be organized using traditional HC methods fused with SC. As an example, the user can describe the goals of the operation using fuzzy logic, and HC-assisted GA can be used to find satisfactory parameter settings to meet these goals. The main purpose of fusing HC and SC in such applications is to combine the advantages of the individual methods to compensate for their different weaknesses.

3. Evolutionary Computation Techniques in Optimization

Chapter 2 described briefly the primary elements of evolutionary computation, i.e., variation and selection. In this chapter, these and other important evolutionary computation-related concepts are discussed in more detail. Additionally, some prevailing evolutionary computation techniques, i.e., genetic algorithms, evolution strategies, evolutionary programming, and artificial immune systems are discussed.

Good and comprehensive introductions and lists of references in the field of evolutionary computation are given in [Fog94] [Bäc96a] [Whi01] [Jin05]. All these journal papers contain many seminal references and serve as a compact starting point for a reader interested in the area, [Jin05] concentrating in optimization in uncertain environments. Additionally, D. B. Fogel has collected some early papers from the field in [Fog98], and this edited volume offers interesting readings for anyone fascinated by the origins of evolutionary computation.

Genetic algorithms, created independently by Fraser [Fra57], Bremermann [Bre62], and Holland [Hol75], are clearly described and discussed in [Mit96] [Hau98]. Those books offer an introduction to GAs and their applications in a simple and easily readable form. The second edition of [Hau98], [Hau04], offers a revised introduction to the topic and also illustrates the latest cutting-edge technology. David Goldberg's book [Gol89] is one of the most cited publications in the area of evolutionary computation, and especially GAs, and also offers a convenient introduction to GAs and their basic operators. Although this book is famous, it is also rather old, and many new aspects, such as the no free lunch theorem, have been raised since its publication, something any reader should bear in mind.

Both [Bäc96b] and [Fog00] discuss all branches of evolutionary computation algorithms. However, [Fog00] could be considered as the next step from Goldberg's book, offering more insight in addition to the basic theory. Then, [Bäc96b] takes this a little further in terms of theory and foundations, but it is not the easiest book to start with.

Although all evolutionary computation algorithms rely roughly on the same principles of natural evolution, they are occasionally divided into two sub-categories: *genotypic algorithms* and *phenotypic algorithms.* Genetic algorithms are considered to be genotypic algorithms, i.e. in simulated evolution the genes, or the parameters of a solution candidate, are transferred through reproduction from generation to generation. Evolution strategies and evolutionary programming, on the other hand, are considered to be phenotypic algorithms. Evolutionary programming mimics the behavior of species and evolution strategies deal with individuals. This means that instead of the actual genes, or parameter values, the traits of the previous generation are passed to the offspring generation without mating.

Artificial Immune Systems [Cas02a] are a more recent approach to using nature as an example for an optimization algorithm. Instead of mimicking evolution, AIS imitates the mammalian immune system. In addition, apart from optimization, this bio-inspired algorithm can be used for various tasks, such as pattern recognition. The AIS concept as a whole is dealt with comprehensively in [Cas02a], whereas the optimization aspect of AIS, known as the Clonal Selection Principle (CSP) is more thoroughly explained in [Cas02b]. Other nature-inspired optimization schemes are discussed in Section 3.9.

The field of evolutionary computation is constantly developing. The latest information can be found published at the two main conferences in the field: the *Genetic and Evolutionary Computation Conference* (GECCO) [Gec06] and the *IEEE Congress on Evolutionary Computation* (CEC) [Cec06]. These conferences are among the best places to find out how natural systems are currently mimicked to construct innovative computational optimization schemes.

3.1 Generic Evolutionary Computation Algorithm

Biologically inspired algorithms vary remarkably in the way data are presented or which operators are used and how they work. However, eventually they all conform to the theory of Darwin, i.e. the algorithms search for the optimum, using variation and selection as the main operators. Thus, a generic biologically inspired optimization algorithm could be described as follows:

1. Generate an *initial population* of n_I individual solutions to the optimization problem.
2. Evaluate the *objective function* value to find out how good each solution is.
3. Introduce *variation* into the population.
4. *Select* solutions for the next generation on the basis of their objective function value so that better solutions have a higher probability of being selected.
5. Go to 2 if solution or run time requirements have not been met. Otherwise exit.

A single stage including everything from evaluating the solutions to selection is usually called a *generation*. There exist countless ways to introduce variation and carry out selection and new approaches can be tailored to conform to the requirements of a specific problem. In the following sections some key issues are discussed in relation to operators used in biologically inspired computation.

3.2 Solution Presentation

In most EC implementations every individual is a solution, better or worse, to the problem at hand and EC algorithm operators directly manipulate the structure of these candidate solutions. In [Ber00], however, a coevolutionary scheme is presented in which a population of solutions and a population of test cases are evolved in parallel. In that paper, the test case population evolves increasingly difficult test cases for the solution population as the algorithm proceeds. The study concluded that using this kind of method, better generalization of the results was achieved in a robot navigation problem. There are no restrictions on the format of the solutions, so the algorithm can easily be adapted to conform to a particular problem. Probably the most common form of solution presentation is the array presentation. Figure 3 shows example solutions to a generic problem. The objective function might have, e.g., 10 variables.

Then, each solution is an array of 10 variables, as in Fig. 3. Now the symbols in the solution array can be numbers, discrete or continuous, characters, symbols or almost anything, as long as we can somehow express how well it solves the problem to be optimized on the basis of the information contained in the solution. This type of solution presentation is most commonly used with genetic algorithms, evolution strategies, evolutionary programming, and the clonal selection principle.

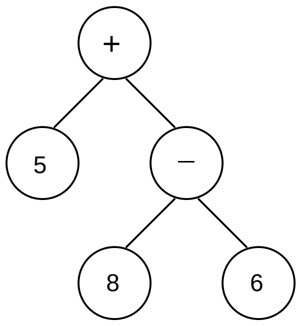
0	1	1	0	1	0	1	0	0	1

h	i	f	k	b	c	z	o	s	o

1.23	–4.34	5.51	2.10	–9.10	7.62	–8.32	6.65	–2.55	1.00

Fig. 3. Three different types of solution presentation. At the top is a binary array and in the middle is a character array. The bottom array is of a continuous type.

Another commonly used solution presentation format is the tree-like structure shown in Fig. 4. However, this format can also be transformed into an array. This type of individual presentation is most common in conjunction with genetic programming (GP) [Koz99] [Koz06], a subset of genetic algorithms. In genetic programming, the individual solutions are computer programs that are evolved using the principles of simulated evolution. Additionally, in evolutionary programming the solutions can be state machines that can be presented as trees.



5	+	(	8	–	6	)

Fig. 4. A solution presented as both a tree structure and an array.

Naturally, the form of the solution is not restricted, so one can combine different data types and presentations to solve the problem at hand in the best possible way.

The question easily arises as to how the solution should be formatted. Should binary or integer presentation be used? Binary presentation might have an advantage over integer presentation in some problems using certain evolutionary operators, such as single or multi-point crossover, and vice versa. A good guideline is to use such a presentation as the task at hand naturally exhibits. In other words, if it is natural to use binary solutions in a certain application, then there is no need to convert the solutions to some other format in the hope of a superior EC algorithm performance.

Solutions of evolutionary computation algorithms are sometimes called individuals, or chromosomes, referring to their biological counterparts. The term chromosome is especially used with genetic algorithms. Figure 5 explains a few of the terms regularly used in conjunction with evolutionary computation algorithms. Different variables in the genetic algorithm solutions are called *genes*. Other EC algorithms have adopted terms such as *variable value* or *trait value*. The genes can have different values, *alleles*. For example, the values 5 and 10 could be different alleles for a gene. A collection of individuals on which the EC algorithm operates is usually called a *population*, *solution pool*, or *selection pool*.

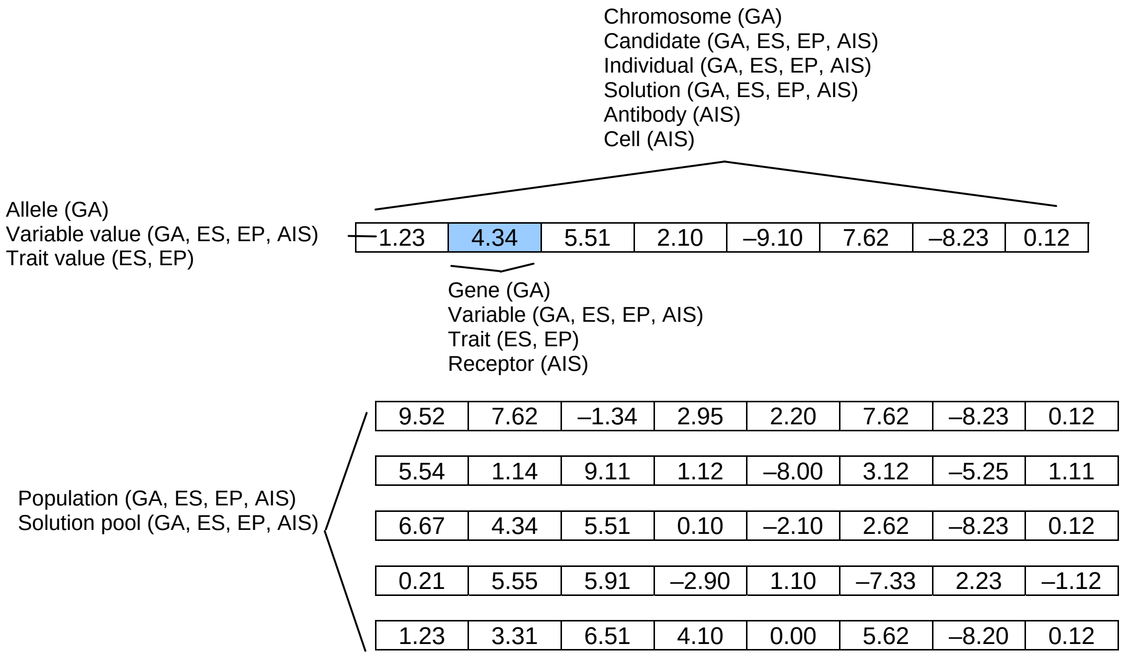


Fig. 5. Frequently used terms in evolutionary computation.

3.3 Initial Population

Generating the initial population is a process of creating n_I solutions for the EC algorithm to start working with. Solutions are usually created by assigning a random variable value to each parameter in each candidate solution. It is important that the mechanism creating random values is capable of covering the whole search space, and that it is not biased if no a priori information concerning a good solution is available. If there is some kind of information concerning particularly appropriate solutions, this information can be taken into account when creating the initial population. These kinds of mechanisms are called *seeding* [Gre87].

The size of the initial population is directly reflected in the run time of the algorithm, assuming that the population size stays constant throughout the run time of the algorithm. This is simply because the objective function usually has to be evaluated using each solution during each generation, so that the larger the population is, the more objective function evaluations there are, and thus the more time is required. A large number of solutions may give the population more diversity, a term discussed in more detail in Chapter 5. Wide diversity may help the algorithm to find better solutions, but then again, the cost lies in a longer execution time for the algorithm.

3.4 Operators of Evolutionary Computation Algorithms

This section discusses the operators used in evolutionary algorithms. The task of the operator is primarily to introduce variation into the solution population and to conduct selection. The section starts with discussion related to evaluating the objective function value.

3.4.1 Evaluating the Objective Function Value

Evolutionary computation algorithms, like many other optimization algorithms, do not need to know anything about the problem they are optimizing. The only requirement is that for each solution there has to be a way to determine how good it is. This evaluation is performed by calculating the objective function value using the variable values of the current solution. This objective function is the function that is being optimized. Depending on whether the objective function is minimized or maximized, it is called a *cost function* or *fitness function*, respectively. The terms cost function and fitness function are commonly used in context with GA, ES, and EP. For AIS optimization the value of the object function is called *affinity value*. Regardless of whether the task is to maximize or minimize, good solutions have *high affinity* and less fit solutions have *low affinity*. Despite the different names, all the values of the objective functions should be presented in numbers, or another directly comparable format.

Objective functions can basically be constrained or unconstrained, and continuous or discrete. If the objective function is constrained, then the parameter values lie at some specific interval, e.g. $-5 < x < 5$. In unconstrained problems such restrictions do not exist. A continuous objective function can basically produce any values, e.g. $-3 < y < 3$, whereas a discrete objective function can only produce certain values, e.g. $y \in \{-2, -1, 0, 1, 2\}$.

Additionally, the objective function can be dynamic, i.e. it changes as a function of time [Psa88]. In [Eng06] dynamic environments are divided into three types. In the first type the location of the optimum changes over time. In the second type, the optimum remains the same, but the value of the optimum changes. Finally, in the third type both the location and the value of the optimum are subject to change. The dynamic objective function can, for example, simulate a real-world situation in which the cheapest route by plane must be found in an environment where flight ticket prices change as a function of time.

Traditionally, objective functions have been functions evaluated by computers, but another very different and interesting approach has been proposed, in which the fitness of an individual is determined by a human [Tak01]. In other words, a human assigns a solution an objective function value depending on how it matches the characteristics of the solution that specific user is looking for. This kind of EC involving human-based fitness or cost function evaluation is called *Interactive EC* (IEC). Additionally, in most cases the objective function value of an individual is expressed as a numerical value, but other approaches also exist, such as the use of fuzzy logic to determine the quality of a candidate solution [Cha02].

3.4.2 Introducing Variation into the Solution Population

Introducing variation into the solution population is of crucial importance: without variation the evolution would stop, since no improvement in the candidate material would take place. There are multiple ways to introduce variation, and usually these methods are classified under the headings *mutation* and *reproduction*. These methods can be used separately or together. Furthermore, the variation operators can use one or two individuals or the whole population to produce individuals for the next generation. These kinds of methods are called *asexual*, *sexual*, or *panmictic* operators, respectively. Next, some variation schemes commonly used in evolutionary computation are discussed.

Reproduction

The basic principle of reproduction is that from one or more parent solutions the variation operator creates one or more new individuals. The characteristics of these new individuals are a combination of the parents' characteristics or an additional random variation on the parents' characteristics.

A popular form of reproduction is *crossover*, discussed e.g. in [Hau98], and this operator is used especially frequently in genetic algorithms. In Fig. 6, the principle of a simple *single-point crossover* between two parent solutions is shown. In this case two parents produce two offspring. First, a *crossover point* is randomly chosen. From this point, the chromosomes are separated and recombined using a part from the other parent. Although a single-point crossover is presented, the method can be implemented simultaneously at multiple points. This type of crossover efficiently passes on the characteristics of the parent solutions to the offspring. On the other hand, this type of reproduction operator is not able to produce new genetic or trait material unless the candidate solution is coded so that the crossover point can split a gene or trait variable. This can happen, for example, when the mantissa and the exponent of a variable are coded in consecutive genes and the crossover point is located between them. Various different types of crossover operators have been studied in the literature, depending on the problem that the specific algorithm is trying to solve. For example, Grefenstette proposed a crossover operator for traveling salesman problems in [Gre85]. The proposed operator avoids creating offspring that contain closed subpaths in the tour connecting the given cities. This is important, since the solutions to a TSP cannot contain closed subpaths, since such solutions violate the constraint stating that each city must be visited only once. However, Fogel points out in [Fog90] that a crossover operator performs poorly in TSP problems in general, whereas [Nag04] gives detailed instructions for designing efficient crossover operators for TSPs. Therefore, it is important to experiment with different operators when tuning an optimization algorithm for a specific problem.

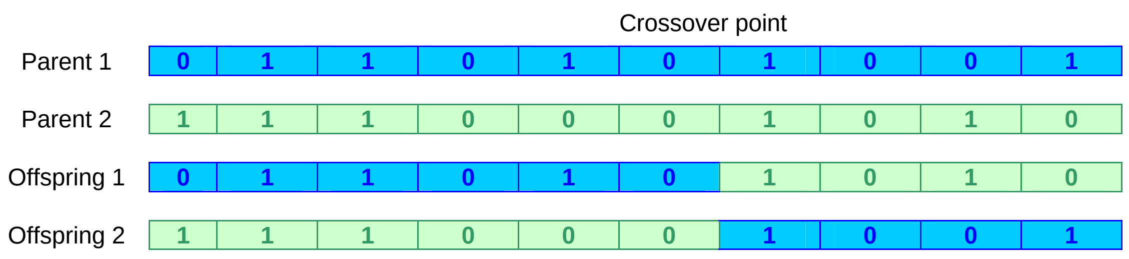


Fig. 6. Producing two offspring from two parents using a single-point crossover.

Blending crossover, described e.g. in [Hau98], is a convenient operator with integer and continuous parameter solutions in genetic algorithms. A random number, β, is first selected, and on the basis of this an offspring can be created, as shown in Fig. 7. β is a weighting factor that determines how much emphasis is given to each parent. If β is 0.5, then the offspring produced is the average of the parents.

$\beta = 0.4$
Offspring = $\beta \cdot$ Parent 1 + $(1 - \beta) \cdot$ Parent 2

Parent 1	–4.0	6.0	8.0	–6.0	0.0
Parent 2	8.0	0.0	8.0	0.0	6.0
Offspring	3.2	2.4	8.0	–2.4	3.6

Fig. 7. Producing a single offspring from two parents using blending crossover.

A single parent can also produce new individuals. In this case, a random variable with known mean and standard deviation can be added to the parent solution to create a new solution, as shown in Fig 8. This type of reproduction is usually used with evolution strategies and evolutionary programming. This type of creation of offspring constantly evolves new material, but, at the same time, is unable to maintain the existing individual characteristics intact. This phenomenon can be fought to some extent using *elitism*, a feature discussed later in this section.

Parent	3.0	5.0	–4.0	1.0	8.0
Random number	0.1	–0.2	0.0	0.1	–0.2
Offspring	3.1	4.8	–4.0	1.1	7.8

Fig. 8. Producing a single offspring from a single parent using a random variable.

The creation of new individuals does not need to happen every time. p_r is the probability of reproduction happening. For example, if reproduction happens, offspring are created according to the rules explained above. If reproduction does not happen, the offspring can be exact copies of the parents without any mixing of the characteristic values. The value of p_r is very problem-dependent and can usually be found only through trial and error by experimenting with different values and choosing the most suitable. In some applications it is enough to have the reproduction probability roughly in some region instead of a specific value. This is because the performance of the algorithm is not necessarily so strongly influenced by p_r, i.e. the algorithm behaves quite similarly, regardless of whether p_r is, e.g., either 0.80 or 0.85.

Genetic programming uses primarily a tree-like representation to describe the structure of a computer program and the dependencies of its different variables. To create offspring, a crossover between two or more trees can take place. In such a crossover the trees exchange sub-trees at random locations.

This section discussed some of the most commonly used operators for creating new individuals from existing ones. Some other operators also exist, both general-purpose and application-specific operators. There are few restrictions on the characteristics of the reproduction operator: the offspring can be created from the parents using many sorts of means. To select the parents, selection mechanisms similar to those used in selecting individuals for the next generation can be used. These methods are discussed in Section 3.4.3.

Mutation

Mutation is another way of producing variation in the solution pool, in addition to that offered by reproduction operators. Mutation operators essentially add a random change or changes to variables of a solution. The mutation operator should be able to produce any parameter value within the search space. Usually, the same kind of mechanism can be used to mutate solutions as is used to create parameter values in the initial population.

When dealing with binary solutions, the easiest and most commonly used mutation operator is the so-called *bit flip* operator. As shown in Fig 9, a single bit can be reversed to create variable diversity in the solution pool.

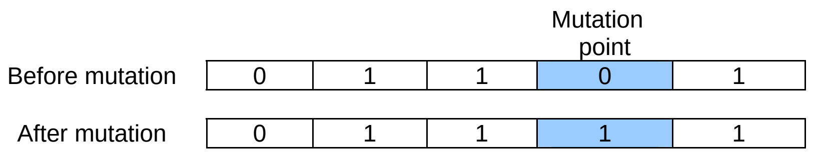


Fig. 9. Single point mutation flipping a bit.

A mutation operator can also be created by substituting a random parameter value for a single one, just as when creating an initial population. This procedure is shown in Fig. 10. Similarly, for example, a random variable can be subtracted or added to a parameter value in order to mutate it.

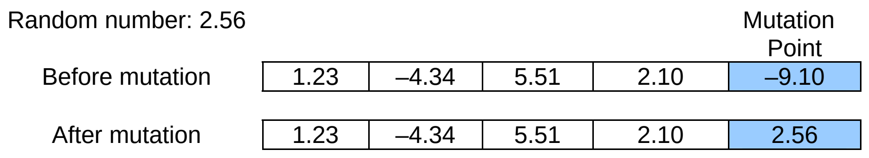


Fig. 10. Single point mutation using a random number.

Mutation can drastically change the fitness of an individual, regardless of whether the individual was fit or less fit before the mutation. This means that the best current solution can in some cases be so badly degenerated that the selection operator excludes it from the next generation. This kind of phenomenon can hinder or slow down the convergence of the evolutionary algorithm. To compensate this problem, elitism can be applied. Elitism can mean that what is currently the best solution is never mutated so that its fitness value would decrease.

Genetic programming primarily uses a mutation method similar to that shown in Fig 11. In mutation the tree-like structure encounters a variation at some node. At this node a random sub-tree is created. Other types of variations may include the addition of a node or sub-tree, removal of a node or sub-tree, or modification of a single variable-value character.

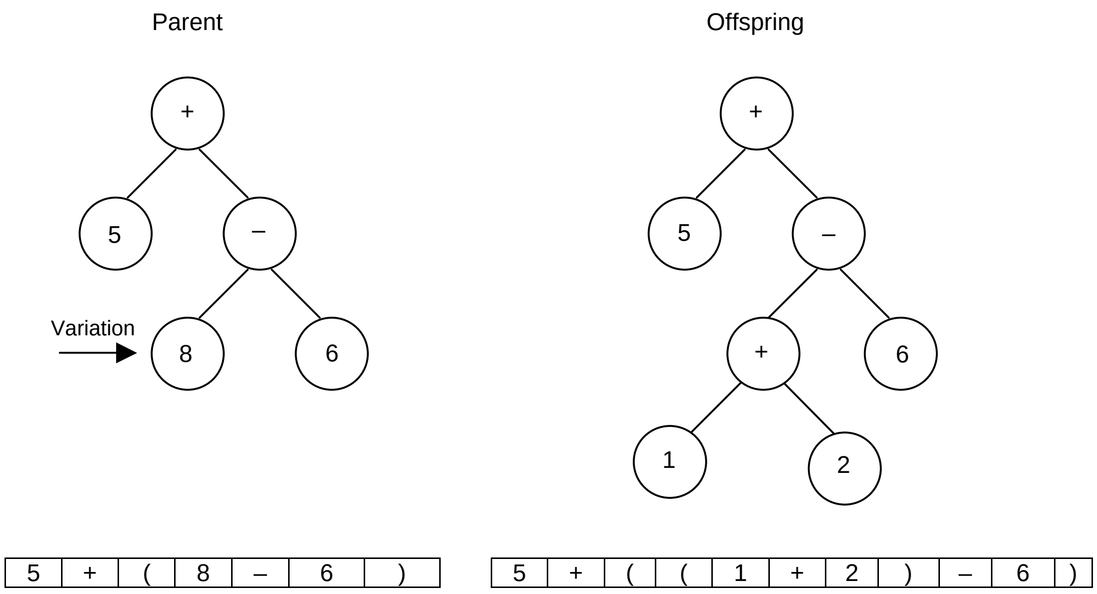


Fig. 11. Popular mutation method for tree-like solution structures.

Just like reproduction, mutation does not necessarily happen every time. p_m, the mutation probability, describes the probability that a single solution or a single variable of a solution is mutated. If mutation should take place, the algorithm proceeds as stated above; if not, the solution is left as it is. The mutation probability can also be determined experimentally.

In artificial immune systems mutation is used widely as a variation operator. However, the parent solution is not mutated directly; rather, a number of identical clones are created from the parent, and each clone undergoes a mutation, thus creating a collection of modified individuals related to the parent.

Discussion Concerning Variation

The significance of mutation operators has been discussed extensively. Goldberg states in [Gol89] that mutation is only a secondary operator in the path of evolution, meaning that the role of mutation is considerably less important than that of reproduction. On the other hand, Fogel explains in [Fog00] the nature of mutation as acting as the driving force of evolutionary algorithms, meaning that mutation is at least as important as reproduction. In most cases, crossover does not introduce new parameter values to the selection pool, unless the coding structure allows the crossover point to be located within a single variable. Then again, mutation is able to produce new material within the existing parameter values and, if it is applied appropriately, it is able to produce any possible solution combination within the search space. In this sense, crossover is just a special case of mutation and in most cases the mutation operator is capable of producing more diversity in the solution pool than crossover. Evolutionary algorithms have been proven to converge to a global optimum if the variation operator is capable of transforming a solution from any state to any other and an elitist selection method is applied (see [Fog06] for details). In most cases crossover does not fulfill this condition, but mutation does, and thus mutation should not be considered as a secondary operator, but rather a crucial one. Nevertheless, the no free lunch theorem still applies to all algorithms and operators, and so, eventually, the usability of any operator depends on the problem.

Mutation and reproduction can occasionally use similar methods, such as adding a random variable to a trait value. However, the fundamental difference between these two operators is that a reproduction operator produces one or more new individuals from an existing solution, whereas mutation only alters an already-existing solution.

It is clear that crossover alone is usually not capable of evolving indefinitely, but that it sooner or later faces stagnation. Then again, using a mere mutation operator promotes evolution, although the pace may be slow. The combination of these two operators may produce better results than using either one alone, since by using the operators side-by-side, the capability of the crossover to connect good combinations of parameter values and the capability of mutation to introduce new variations to the solution pool creates a potentially efficient algorithm. Still, according to the NFL theorem the individual and combined performance of the operators is the same over all possible problems.

The AIS optimization method, the clonal selection principle, is a straightforward way of adding variation to the solution pool. A small number of the worst individuals are replaced by randomly created new individuals. Such a simple method is powerful in exploring new areas of the search space.

3.4.3 Selection

A selection operator selects the individuals from the current generation to proceed to the next generation. A selection operator is also crucial to the operation of any evolution algorithm, since without selection the solution population would not be guided in any direction and would perform like a purely random search. Selection operators usually favor good individuals, thus directing the evolution of the population towards a better average value of the objective function.

Depending on the implementation, the selection can be carried out among offspring or parents and offspring, or whatever combination of solutions exists in the previous generations. The main idea in selection is that the better the objective function value of an individual, the better its chance of proceeding to the next generation. Usually, the size of the solution population is kept constant in order to keep the execution time for each generation the same. However, it is possible to increase or reduce the population sizes as the evolution proceeds, a procedure that directly affects the execution time of the algorithm. This kind of approach aims mainly at performing global searches with fewer individuals and using a larger population to refine the local search. The effect of variable population size is studied in [Kou06], where the population size is reduced by the selection operator and periodically increased by the addition of random individuals.

Numerous selection mechanisms exist, and some of the most widely used methods include *rank-based, roulette wheel*, and *tournament selection*, discussed, e.g., in [Hau98]. A rank-based selection mechanism simply selects the n_{rank} first solutions ordered according to their fitness values in descending order. n_{rank} equals n_I, the initial population size, if the population size is kept constant throughout the execution of the algorithm.

Rank-based selection is a somewhat deterministic selection method in the sense that the fit solutions are always selected and the less fit are not. This drives the EC algorithm relatively quickly towards an optimum, regardless of whether it is global or local. Less fit solutions can contain good ingredients when considering the global optimum, but especially in early generations of an EC algorithm local optima can distract the search and lead to a premature convergence of the algorithm. Premature convergence is a state in which the population of the EC is homogeneous. This condition can occur in local optima or at any other point within the search space. When using rank-based selection one should pay special attention to variation methods, i.e. reproduction and mutation, to ensure diversity within the solution population.

Roulette wheel selection, illustrated in Fig. 12, selects the individuals that will survive to the next generation on the basis of their proportional fitness. The selection simulates a roulette wheel, where each individual in the population has a selection sector proportional to its fitness. The better the individual is, the greater its chances of continuing to the next round. Roulette wheel selection makes it possible for each and every individual to make it to the next generation, regardless of its fitness. This may slow down or hinder the convergence of the EC algorithm, since the selection operator may disqualify good solutions. On the other hand, though, good ingredients within less fit individuals have a chance of being selected.

Solution	Fitness	Fitness % of total
A	200	52.63
B	100	26.32
C	50	13.16
D	25	6.58
E	5	1.32

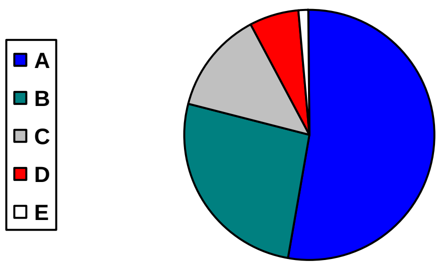


Fig. 12. The principle of the roulette wheel selection mechanism.

Figure 13 shows the general principle of a generic tournament selection operator. $n_{tournament}$ solutions are selected either randomly or using some kind of a selection operator for a tournament and each tournament produces a winner, just as in a sports competition. Solutions compete against each other and the best one, i.e. the one with the highest fitness value, moves on to the next round. This method can be implemented by assigning each individual an equal probability of getting selected for a tournament or the probabilities can be biased, as in roulette wheel selection. This method does not guarantee the survival of the fittest, which may lead to problems similar to those discussed with roulette wheel selection. This is the case when the fittest individual is not selected to compete in any of the tournaments. On the other hand, tournament selection allows each individual a chance to proceed to the next generation, thus preserving a possibly greater diversity of variable values in the selection pool than the rank-based selection method.

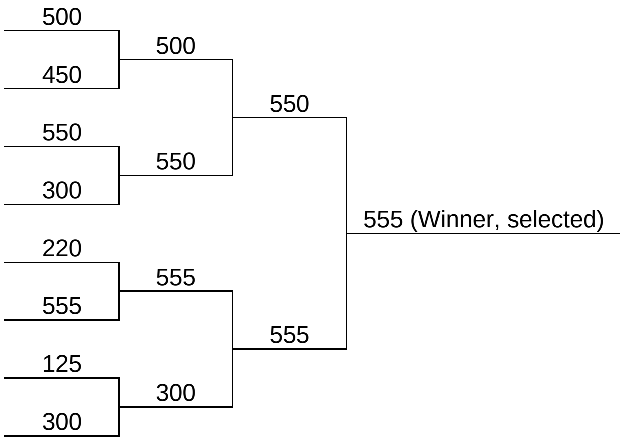


Fig. 13. The principle of the tournament selection method. The value describes the fitness function value of an individual in a maximization problem.

In the following sections, the evolutionary computation methods dealt with in this dissertation are discussed.

3.5 Genetic Algorithms

Genetic algorithms are probably the best known of all the evolutionary algorithms. GAs were invented independently at least three times by Fraser [Fra57], Bremermann [Bre62], and Holland [Hol75]. Fraser was a biologist who wanted to simulate evolution in order to get greater insight into the biological process and using the algorithms he developed for optimization was not his primary purpose. Then again, Bremermann and Holland clearly developed their algorithms for optimization purposes. The basics of simulated evolution based on genetics are compactly summarized in [Gol89]. Fogel describes the stages of plain GA in [Fog00] as follows:

1. The problem at hand is defined as an objective function.

2. A population of candidate solutions is initialized as vectors.

3. Each solution in the population is decoded into an appropriate form for objective function evaluation.

4. Each solution is assigned a probability of reproduction.

5. Reproduction is carried out according to the assigned probabilities. In addition, mutation can be applied to selected individuals on the basis of the assigned probabilities.

6. The process is stopped if a suitable solution is found or when run time requirements have been met.

GAs are suitable for many types of optimization tasks and they are widely used. For this reason, they have been widely studied and a variety of performance-tuning methods exists. Genetic algorithms could be described as the standard evolutionary algorithms. Genetic programming is a subset of GA. In genetic programming, the individuals are computer programs that are evolved to solve a specific problem. One type of problem in which GP is often used is where there are specific inputs and a desired target, and the problem is to find a program to use the input to achieve the target. The target can be, for example, to control a system based on input sensory values. In genetic programming the structure of the program can change drastically during evolution. Publications [P1]-[P4] and [P7] in this dissertation employ genetic algorithms.

3.6 Evolution Strategies

The concept of evolution strategies was created by Rechenberg and Schefel in 1964. Early papers that describe the first attempts to optimize practical problems using a crude predecessor of modern-day evolution strategies are [Sch65] and [Rec65]. In those papers, very simple evolution strategies (including only a few individuals) are outlined and implemented, mainly because of the insufficient computing power, computers becoming available for these scientists only a couple of years after the introduction of the original concept of the algorithm. Evolution strategies differ from genetic algorithms in that each individual, in addition to the parameter values, contains a set of *strategy parameters*, which were later introduced to the concept. These strategy parameters help to optimize the optimization process itself, as the strategy parameters are subject to crossover and mutation in a similar manner to the actual solution variables. Since ES individuals suffer mutation according to Gaussian distribution, a strategy parameter could contain the standard deviation of the distribution. So, this mutation-determining parameter evolves as the algorithm proceeds. The stages of ES as they are typically used at present are described in [Bäc96b] [Fog00] [Eng06], as follows:

1. The problem is defined as finding a real valued n-dimensional vector x that is associated with the extremum of the objective function.

2. An initial population of parent vectors, x_i, i=1, …, n_I, is selected randomly from the feasible search space.

3. An offspring vector is created from two or more parents by crossover. The crossover also includes the crossing over of the strategy parameters.

4. Mutation is introduced by adding a random variable with a zero mean and the standard deviation pointed in the individual’s strategy parameter to each component of x.

5. Selection determines which of these vectors (parent and offspring) are maintained by ranking the objective function values. The n_i best vectors act as parents for the next generation.

6. The process is continued until a satisfactory solution is reached or run time requirements have been met.

Evolution strategies are implemented in [P5] as a part of this dissertation.

3.7 Evolutionary Programming

The foundations for the EP research to come were laid in the early 1960s by L. J. Fogel in [Fog62]. Evolutionary programming is similar to evolution strategies in the sense that the mutation is controlled by a strategy parameter, i.e. a standard deviation value is conjugated to each individual. Mutation is usually the only operator used with EP; however, the possibility of multiparent recombination of the best solutions over various generations has also been considered in [Fog66]. D. B. Fogel describes the main phases of EP in [Fog00] as follows:

1. A population of individuals is created. These individuals can be presented, for example, as arrays.

2. The objective function value for each individual is evaluated.

3. Offspring individuals are created by mutating the parent individuals on the basis of the standard deviation parameter given in the strategy parameters.

4. The best individuals are selected to act as parents for the next generation.

5. The process continues until satisfactory results have been achieved or run time requirements have been met.

GP and EP share many similarities, the most significant difference being the use of crossover as an important variation operator in GP. EP relies more on a mutation operator. Genetic programming is able to modify the number of states through a crossover operator, whereas the number of states in evolutionary programming can be altered by the mutation operator.

In evolutionary programming, the individuals can be finite state machines, as described in the early experiments. Nowadays, however, the solution structure arises naturally from the problem specifications. In this dissertation, publications [P6] and [P8] employs evolutionary programming as a part of it.

3.8. Artificial Immune Systems

Artificial immune systems stand out as more recent contenders in the field of evolutionary computation when compared to GA, EP, and ES. AIS and its optimization scheme, the clonal selection principle (CSP) or the clonal selection theorem, are widely credited to, among others, de Castro and Timmis [Cas02a], dating back to the mid-'90s. Instead of mimicking natural evolution, CSP simplifies the mammalian immune system for optimization purposes. The major difference between the traditional EC methods, GA, ES, and EP, and CSP is that, instead of a single optimum, the basic configuration CSP is capable of finding multiple, and separate, good solutions. For GAs, ESs, and EPs, finding multiple good solutions is possible, but this requires the algorithms to be specifically designed that way. A common method for finding multiple good solutions is *niching*, discussed in more detail in Chapter 4.

In brief, the mammalian immune system operates as follows [Cas02b]. When an animal is exposed to an *antigen*, something that is normally not part of the animal's system, the bone marrow-derived B lymphocyte cells respond by producing antibodies. Antibodies are attached to the surface of the lymphocytes and they recognize and bind to antigens that match their structure. By binding to the antigens, and with the help of additional signals from the so-called T cells, the antigen stimulates the B cell to proliferate and mature into plasma cells or memory cells that can circulate for a long time within the animal. The whole scheme is naturally considerably more complicated, but the main aspects used in the optimization are quite straightforward. The objective function is the antigen. Antibodies are candidate solutions trying to maximize their affinity with respect to the objective function, i.e. the antigen. The candidate solutions are cloned proportionately to how good their affinities are. Thus the best matching candidate solutions are cloned in greater numbers. The optimization procedure using CSP, an algorithm called CLONALG, is explained in [Cas02b] as follows:

1. Create an initial population of n_I antibodies.

2. Determine the affinities of each individual in the solution pool.

3. Select m highest affinity antibodies for cloning.

4. The m selected antibodies are cloned independently and proportionately to their affinities: the higher the affinity, the higher number of clones created for each of the m selected antibodies.

5. The clones are mutated proportionately to their affinity: the better the affinity, the smaller the mutation rate. This stage is called *affinity maturation*.

6. The affinity of the clones is calculated.

7. If one of the clones has a higher affinity than its parent, then replace the parent with this higher-affinity mutated clone.

8. Replace the *w* worst individuals with random new antibodies.

9. The process is continued until a satisfactory solution is reached or run time requirements have been met.

CSP contains many of the operators of the traditional EC methods, such as variation and selection. CSP was studied in [P8] as a part of this dissertation.

3.9 Other Biologically Inspired Optimization Schemes

Nature and society are constantly being studied for new ideas to be refined as optimization algorithms. Some of the most popular biologically inspired optimization methods, apart from GA, ES, EP, and AIS, are discussed in the following section. These methods are not considered in publications [P1]-[P8], but they are introduced here to show the wide variety of different aspects used today to develop nature-inspired optimization methods.

Swarm intelligence takes advantage of the behavior of groups, i.e. a collection of individuals working for a common goal. As examples, nature provides an ants' nest or a flock of flying birds. Two forms of swarm intelligence, *Ant Colony Optimization* (ACO) and *Particle Swarm Optimization* (PSO), are briefly discussed below.

Ant colony optimization mimics the behavior of ants [Eng06][Hec01]. Ants lay down *pheromones* to signal the locations of food to other ants. Pheromones attract other ants to the same region, but they evaporate as time goes by. This notion is translated into a generic optimization scheme using artificial ants that lay artificial pheromones proportional to the objective function value at a visited point. Different pheromone levels at different points guide the artificial ants to change their direction towards the most promising areas. Thus, ACO optimization is constantly guided towards the current best solution. Common applications of ACO optimization include different routing problems, such as the traveling salesman problem [Dor97]. Additionally, other kinds of problems, such as data mining [Par02] and finding a charging pattern for lithium-ion batteries [Liu05] have been successfully solved using ACO.

Particle swarm optimization [Ken95] [Eng06] [Rey06] takes advantage of e.g. birds and fish, more generally a group of individuals moving collectively as a large group. The best location of food or protection is rapidly communicated through a group, and all the individuals seem to act nearly as one. As an optimization scheme these artificial individuals adjust their speed and direction in relation to the overall best solution, perhaps also influenced by the best solution in the surrounding neighborhood. PSO has been used to solve a wide variety of problems successfully. These include, among others, optimizing the design of a PID controller [Gai04] and the design of a microwave filter [Wan05].

Learning Classifier Systems (LCS) [Kov01] are a machine learning system suitable for optimization, among others. LCS uses a population of IF-THEN rules to make decisions. The rules are modified by another evolutionary computation technique, like genetic algorithms. The best rules are selected for the next generation, whereas the less-fit rules are discarded. LCSs have been implemented in various applications, such as studying the UK electricity market [Bag05].

Cultural evolutionary algorithms use human social evolution as their metaphor for optimization. In these algorithms culture is seen as a guidance system directing individuals towards a specific direction, determined by the best individuals in the population. According to [Cor05], a cultural evolutionary algorithm proceeds as follows. Candidate solutions are first evaluated and an *acceptance function* determines which individuals in the current population can make a difference to the current *beliefs.* The fitness, i.e. the experience, of these individuals is used to adjust the belief space. These group beliefs are then used to guide the evolution in a certain direction. In numerical optimization this would mean, for example, that the fittest individuals could be used to control the domain constraints, that is, the search intervals. Furthermore, only a certain percentage of the fittest individuals would be preserved for the next generation and thus act as parents for offspring.

Differential Evolution (DE) [Cor05] is a simple population-based optimization tool. A basic DE algorithm may proceed as follows. Select an individual from a created population. After this, select two other individuals and calculate their difference using subtraction. To this result, add a third randomly chosen individual. Then use crossover to combine the first selected individual and the individual built using the three other individuals. If the product of the crossover has a better fitness than the first selected individual, select the offspring for the next generation. Otherwise, use the individual selected first.

Memetic algorithms [Cor05] are population-based search methods that use the available knowledge about the problem. This knowledge is exploited using, for example, approximation algorithms, local search methods, or specialized operators tailored to meet the requirements of a specific problem.

Tabu search [Eng06] is an optimization technique concentrating on an iterative neighborhood search in which the neighborhood changes dynamically. Tabu search maintains a memory structure of points the search algorithm has visited previously and excludes these points from future searches.

3.10 About Nature-Inspired Optimization Schemes

In this chapter, some of the most commonly used evolutionary algorithms have been discussed. It is clear that all the algorithms take advantage of the two key operators to complete their task: variation and selection. In the previous years the field of evolutionary computation was more fragmented, and the algorithms were classified, for example, on the basis of the presentation of the data or the type of operators used or simply on different opinions of scientists. Fogel has remarked in [Fog06] that it is no longer possible to identify a certain approach as a genetic algorithm, evolution strategy, or evolutionary programming, simply by examining the representation chosen, the selection method, or the use of self-adaptation, recombination, or any other factor. At present, there are no implementational restrictions on what operators can be used with what operators and how to present data. All nature-inspired optimization algorithms are ultimately imaginative interpretations of variation and natural selection.

4. Improving the Performance of Evolutionary Algorithms

Evolutionary algorithms have proved their efficiency in complex optimization tasks, but their rudimentary versions usually suffer from severe problems that may prevent the algorithm from reaching optimal solutions in reasonable time. These problems are known as stagnation or premature convergence [Bäc96] [Fog00], but lengthy run times of the algorithms may also be an issue for the users. These problematic phenomena are important and interesting challenges for researchers and the possible competitive solutions are applied by practitioners working with complex optimization tasks. This chapter discusses some methods proposed to resolve these problems.

4.1 Behavior of Evolutionary Algorithms

Evolutionary algorithms operate on a population of individuals, improving the objective function value, f_o, of these individuals within the population gradually or in jumps. This development towards the final result is called *convergence*. The *maximum*, $f_{o,max}(g)$, *average*, $f_{o,ave}(g)$, and *minimum* objective function values, $f_{o,min}(g)$, are variables that describe the progress of an evolutionary algorithm. $f_{o,max}(g)$ and $f_{o,min}(g)$ describe the best and the worst solutions within a specific generation, whereas $f_{o,ave}(g)$ gives the average objective function value of all the solutions in the solution pool within a specific generation. This text discusses fitness values, i.e. maximization problems, without the loss of generality; all the concepts also apply to minimization problems, i.e. cost functions.

Figure 14 depicts the typical convergence characteristics of a generic EC algorithm. In the early generations of evolution, the *dynamic range* of the fitness values in a population is usually large. The dynamic range of a solution population is used to describe the difference between the fitness values of the best and the worst individual solutions, i.e. ($f_{o,max}(g) - f_{o,min}(g)$). Later on, as the solutions evolve, the average and minimum fitness values approach the maximum fitness value and the solution population converges.

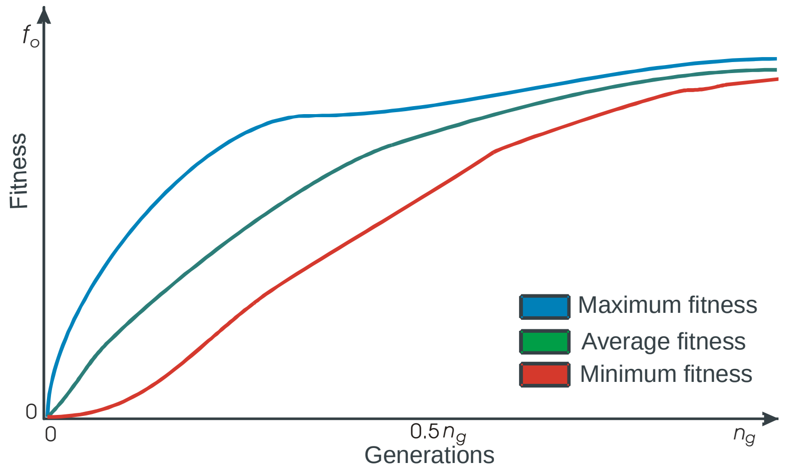


Fig. 14. Typical convergence behavior of an evolutionary algorithm.

At the beginning, there is usually significant *variable value diversity* within the solution pool, meaning that there may exist a large number of different values for a specific trait or a gene. The variable value diversity can be expressed, for example, as the difference between the minimum and maximum values of a single variable in the solution pool or using the variance of the values a variable has. Towards the end, this diversity tends to shrink and finally the evolutionary computation algorithm may end up with a solution pool consisting merely of a single solution or a few solutions. Problems with EC are usually related to a decrease in parameter value diversity occurring before the algorithm has found a competitive solution, if such a solution exists. It is easy to see that if the solutions in the solution pool are very similar, not much improvement or even variation can usually be achieved by the use of crossover-based reproduction operators. In this situation a mutation operator may be able to produce material with good new parameters.

The dynamic range of the fitness values and the diversity of the variable values are two different things and they should not be confused. The dynamic range of the fitness values describes the variety in the fitness values of the solutions in the current population, and the diversity of the variable values describes the variety of different values each variable of a solution has in the solution population. Both of these terms, alone or combined, tell us something about the population. For example, an algorithm may be close to convergence or stagnation if the diversity is low; however, the dynamics can still be large. This is because even a minor variation in a variable value can result in a drastic change in the fitness value. Then again, a small dynamic range of fitness implies that there is no considerable difference in the fitness values of the individuals, although the variable value diversity may be large, indicating the existence of multiple good solutions. Using this type of information, among others, it may be possible to design operators for evolutionary algorithms to overcome the obstacles in the optimization process, such as premature convergence.

Convergence and the decrease in the parameter value diversity are nothing to be alarmed about if the algorithm is centered on a global optimum or a good enough solution. This is basically what the algorithm is supposed to do. If, instead, the algorithm gets stuck around a local optimum or any other point, then there is a problem, either stagnation or premature convergence. This problem can be alleviated by using an appropriately devised variation operator that is able to break the algorithm free from such local optima, since there is usually no way of knowing whether a certain point is a local optimum, global optimum, or some other point.

The question arises of how one should choose the parameter values for operators in an EC algorithm. In fact, this is problematic, since the operator parameters are very application-specific and no guarantees can be given that some parameters will work for some specific problems. Another problem related to all optimization techniques is the sometimes very costly evaluation of the fitness function. The objective function can be a low-dimensional function or it can be a complicated simulation procedure for an application-specific digital filter.

The problems of evolutionary algorithms are thus related to the reliability and execution speed of the algorithm. Nowadays, the research and development cycles of products are so intensive that practicing engineers usually have no time to run a time-consuming optimization algorithm several times in the hope of getting a solution.

4.2 Means to Improve the Performance of Evolutionary Algorithms

Various methods to improve EC performance have been discussed, starting from the very first algorithms proposed. These modifications aim to tackle the bottlenecks of EC-based optimization processes and thus add to the performance of an algorithm. The performance of an algorithm is not an unambiguous concept, but, rather, it depends on the application and implementation. However, there are some characteristics that can describe the capabilities of an algorithm. The convergence rate of an algorithm describes how fast the algorithm is capable of finding the best attainable solution for that specific algorithm. So, the greater the convergence rate, the faster the algorithm is. The reliability of an algorithm to produce good results is also an important factor. An algorithm can be considered reliable if the variance of the results it produces is sufficiently small and there is no considerable number of outlier samples. The magnitude of variance and the number of outliers cannot be defined unambiguously; rather, these characteristics are problem-dependent. One of the most interesting measures of performance is the fitness of the solutions produced. Sometimes a single outstanding value produced by an algorithm after multiple runs is desirable, but it is more common to develop an algorithm that produces good results on the average. Additionally, sometimes multiple competitive solutions instead of just one are desirable. So, a high-performance EC algorithm should be able to produce quality solutions reliably in a short time. At times, designers have to decide on trade-offs between reliability, quality, and speed.

For complex problems, the addition of more computational power may improve the performance of the algorithm. The effect is twofold: with more computational resources to spare, an algorithm can run more generations in the same time. Then again, an algorithm with more computational power can have a larger solution population, and thus a more thorough search of the search space, than its counterpart with less computing power. Parallel computing architectures have been studied extensively in the context of EC. This is mainly due to the parallel nature of EC algorithms, i.e. the search for multiple solutions in parallel. Despite this inherent parallelism in evolution and evolutionary computation, many EC algorithms are serial in nature. This means that although evolutionary algorithms virtually evaluate multiple solutions in parallel, the actual execution of the program instructions is usually serial. It has also been noted that the mere existence of parallel populations adds to the performance of an EC, although no actual parallel hardware is available. Other schemes related to speed include the modification of operators to make them more efficient, thus making the operation of the whole algorithm faster.

Other than speed, modifications concentrate on the structure of the population and the EC operators. The main aim of such modifications is to avoid premature convergence, i.e. the stagnation of an algorithm, and to find new competitive solutions. Operators can be made adaptive, so that their characteristics change during the run time of an algorithm to compensate for the changes taking place in the solution population. This can mean, for example, that the operators produce more variation in the solution population when the solution population starts to lose variable diversity.

An important aspect of modifying EC algorithms is the fusion of evolutionary algorithms with other types of methods. These hybrid algorithms combine the strengths of multiple methods, most commonly evolutionary algorithms, neural networks, fuzzy logic, and conventional hard computing methods. A single algorithm can be superior alone in some cases, but a combination of techniques and their respective advantages can produce competitive algorithms for complex problems. In the following section, some modification techniques aimed at improving EC performance are discussed in detail.

4.3 Multiple-Population Evolutionary Algorithms

An evolutionary algorithm with multiple populations is a scheme in which, instead of a single solution population, multiple and separate but possibly interacting subpopulations are maintained and evolved in parallel. In the following, different approaches to multipopulation EC algorithms are discussed.

4.3.1 Parallel Processing and Evolutionary Algorithms

The use of parallel processing environments has been a traditional way of speeding up demanding computational experiments. Parallel hardware environments contain two or more processing units that can, theoretically, divide the execution time by the number of processing units. In practice, however, this is rarely the case, although significant speedup can usually be achieved [Cul99] [Cod93].

Because of the inherent parallelism in evolutionary algorithms, parallel implementations of evolutionary algorithms have been studied thoroughly. The advantages of using parallel processing in EC are listed in [Alb02] as follows:

- The ability to search for alternative solutions to the same problem in parallel
- Easy parallelization as island or neighborhood models (see below)
- Speedup resulting from the use of multiple central processing units (CPUs)

Early implementations of parallel EC algorithms (PEC) were carried out by, e.g., Grefenstette in the early 1980s [Gre81]. A comprehensive survey of parallel EC algorithms and parallel hardware environments, as well as the parallel programming tools used in parallel EC algorithms, is given in [Alb02]. In that paper PEC algorithms are divided into subsets. In the *panmictic* approach the solution population is dealt with as one, with all the operators affecting each individual: that is, in panmictic EC algorithms only the computational burden is divided using multiple processors. On the other hand, *structured* PEC algorithms take advantage of multiple subpopulations residing in different CPUs. Structured PEC algorithms can be further divided into *distributed* EC algorithms (dEC) [Bel95] and *cellular* EC algorithms (cEC) [Bal93]. The dEC approach is also known as the *island model*. In dEC, the population is divided into subpopulations: the exchange of individuals takes place on the basis of predetermined rules, and subpopulations reproduce among themselves. In cEC algorithms, a single individual usually has its own CPU, and it can reproduce only with the neighboring individuals in the processor architecture. In both the dEC and cEC schemes the operators acting on the populations are the same, i.e. the algorithms are *uniform*. In *nonuniform* structured PEC algorithms, the operators acting on different subpopulations may be different. This kind of approach was studied in [Tan87].

PEC has been used, for obvious reasons, to tackle computationally very challenging problems. These include, among others, demanding TSP in combinatorial optimization [Müh92], frequency assignment problems in telecommunications network design [Meu00], and trading models in the area of financial applications [Cho00]. Recently, parallel EC algorithms have been implemented in areas of increasing scientific interest, e.g., bioinformatics [Wie05] on the application side and peer-to-peer networks [Mel05] on the implementation side.

Parallel processing does not improve the robustness and reliability of an EC as such. When using parallel processing environments, the computational task at hand can be divided between all the processors. Evolutionary computation algorithms consist of a number of objective function evaluations per generation, and this is where parallel processing can speed up the EC. Since the objective function evaluations are separate computational tasks not requiring information from each other, the tasks can be spread across several processors. For example, if the EC algorithm contains 100 objective function evaluations per generation and there exists a parallel environment with five processing units, theoretically, each processor could conduct 20 objective function evaluations per generation instead of a single processor doing all the work. This means that an algorithm using parallel hardware would be five times faster than an application using only a single processing unit.

However, communication between the processing units takes time. The inter-processor communication time should be small in comparison to the time required by the objective function evaluation. If the evaluation of the objective function takes only a short time, then it could be that in fact the parallel implementation of EC is slower than that using only a single processing unit.

4.3.2 Multipopulation Approach

Improving the performance of EC algorithms with parallel hardware is beyond the scope of this dissertation. However, some parallel implementations of dEC algorithms use an interesting approach: in addition to distributing the computational load between separate processing units, each processing unit contains a separate EC algorithm that operates on its own population. These separate EC algorithms can exchange solutions with each other from time to time, and thus improve the performance of a standard EC. In [Gor93] the authors have proposed in their study that the mere existence of multiple populations without actual parallel hardware may add to the performance of an EC. These kinds of implementations take advantage of multiple subpopulations and different operators for different subpopulations, as well as different subpopulation characteristics. These kinds of multiple population algorithms are studied, for example, in [Sla99]. Kamiya and Makino [Kam05] have proposed that a multipopulation scheme can be used to improve the performance of an EC in dynamic environments, that is, when the objective function changes over time. In addition, all the approaches used in actual parallel hardware implementations of parallel EC algorithms are also applicable to single CPU EC algorithms.

Niching [Gol89] [Dar97], also known as *speciation*, is a popular multipopulation scheme in evolutionary computation. In nature, niches are subspaces of the environment that can support different kinds of life, i.e. different species, and, of course, different species cannot have offspring together. In addition, each niche has only a limited amount of resources, which have to be divided between the inhabitants of that particular niche, i.e., even a vast reservoir of resources cannot nurture an indefinite number of individuals. Then again, a niche with fewer resources can easily satisfy the needs of a couple of individuals. This metaphor turns into an optimization scheme quite conveniently. The main principle is to sustain diversity, not letting too many individuals search the same region of promising fitness. Furthermore, candidate solutions from two separate good regions cannot be combined to avoid the creation of inferior solutions. The two main mechanisms for niching are *fitness sharing* and *crowding* [Sar98]. A fitness sharing scheme divides the fitness of a region for all the individuals residing at that location, thus making it less attractive for masses of individuals. Crowding methods, on the other hand, insert new elements into the population, simultaneously replacing similar solutions.

Coevolution, as the name implies, is the evolution of multiple interactive populations in parallel. A coevolutionary scheme could include, for example, the solution population and a population of test cases [Wer00]. The fitness function can be a combination of different test cases. In this scheme, the desired property of the test case population is to gradually evolve into more challenging but appropriate test cases, thus enabling the solution population to be able to solve ever harder problems as the algorithm proceeds. In [Han97a], an optimization scheme in which two genetic algorithms work with the same population is discussed. The coevolution scheme can be either co-operative, in which the multiple components work on the same side, or a *predator-prey* scheme, in which the components fight against each other, as in [Ari96].

4.3.3 Problem Decomposition

Humans intuitively try to solve extensive problems by dividing the problem into smaller pieces, rather than trying to solve the whole problem at once. This method of slicing problems down into sub-problems, solving them, and recombining the partial solutions is referred to as a problem *decomposition* or *divide-and-conquer* approach. In [Val95] a TSP was divided into sub-problems and the combined solutions to the sub-problems constituted the solution to the overall problem. The partitioning of a TSP is a very illustrative example of the divide-and-conquer approach: the whole route connecting all the given cities can be optimized by optimizing separately optimized and then connected sub-paths. In [Jua05], another type of approach is selected, in which a recurrent fuzzy system is structurally divided into separate parts and these parts are optimized separately. In addition, a separate optimization process deals with the whole network.

4.4 Modifying Operators in Evolutionary Algorithms

The behavior of an EC algorithm during run time is mainly controlled by the variation probabilities. Of course, selection and other possible operators also play an important role in the course of the evolution of a solution population, but variation operators can essentially make the algorithm excel or stagnate. Selecting the appropriate values for reproduction and mutation probabilities is a problem-dependent task and it is usually carried out by means of trial and error through extensive testing of different control parameter values. As explained before, the variable value diversity of the solutions usually diminishes as the evolution of an EC algorithm proceeds. The location of the solution population changes stochastically during the evolution process and this basically means that fixed reproduction and mutation probability values may not produce the desired performance throughout the whole run time of an algorithm. To overcome this problem, *adaptive parameters* are used. Adaptive parameters adapt the EC control parameters, such as mutation and reproduction probabilities, to the changes in the population dynamics in the hope of improving the performance of the algorithm.

In [Hin97] and [Smi97] a compact survey of adaptation in evolutionary computation is offered, while [Eib99] is a more thorough presentation of the same topic. In [Hin97] EC is divided into *static* and *dynamic* EC. In static EC no adaptation takes place. In dynamic EC *environments*, *populations*, or *individuals* can be adapted. Dynamic adaptation is further divided into subcategories, including *deterministic*, *adaptive*, and *self-adaptive* methods. In deterministic adaptation the parameter is adapted according to some deterministic rule without feedback from the EC algorithm. For example, in [Fog89] the mutation probability was altered as a function of the number of generations the EC algorithm had run. In adaptive EC there is feedback from the algorithm, and this is used to control the parameters. In [Jul95], the ratio between mutation and crossover, based on their performance, was studied. In self-adaptation the control parameter to be adapted is placed within each individual and is thus subjected to mutation and reproduction. For example, in [Smi96] the mutation rates of individuals were controlled using self-adaptation.

A popular adaptive scheme for adaptively controlling the reproduction and mutation probabilities was proposed by Srinivas and Patnaik in [Sri94]. More recently, the use of adaptive fitness functions in the form of altering a penalty function has been studied in [Far03]. Theoretical considerations concerning the features of self-adaptation in EC algorithms are offered in [Bey01]. That publication offers guidance for selecting strategies to be used in self-adaptation. Operators for an EC algorithm are selected adaptively in [Mag00], i.e. different operators are used for the same purpose at different stages of the run time of the algorithm. Premature convergence is a common deficiency in EC, and it may result from poor selection of parameter values. Premature convergence, or stagnation, stops the evolution of the solutions and the algorithm is usually only able to output a non-satisfactory result. By modifying the control parameters of the algorithm, e.g. the mutation probability, adaptively, it is possible to overcome stagnation and continue the optimization process further.

The main purpose of the variation operators is to search for improvements, but at the same time they also introduce diversity within the population. How the operators should work depends on the state of the underlying population characteristics. Figure 15 shows how a generic EC algorithm could behave. Conventionally, the solution pool of an evolutionary algorithm starts with multiple candidates and potentially wide diversity of genetic or trait material. At the beginning the algorithm evaluates multiple promising solutions, instead of concentrating on what is currently the best solution. During the following generations, the solution variable diversity in the population tends to decrease and the individuals in the population may start increasingly to resemble one another. Usually, the variation probabilities are high at the beginning in order to allow new promising areas of search space to be searched through. Towards the end the population settles in an area in the search space through selection, and mutation is used to make minor local searches in the vicinity of the candidate solution. From this it follows that a variation operator is commonly used, so that the population starts with a high mutation probability and this probability is decreased as the algorithm proceeds towards its end. Figure 15 illustrates the behavior of the generic adaptive variation probability as a generic EC algorithm proceeds.

One must bear in mind that the situation described above is different when other types of operators or algorithms are used. For example, the clonal selection principle eventually produces multiple distinct good solutions instead of just the one the usually produced by other evolutionary algorithms.

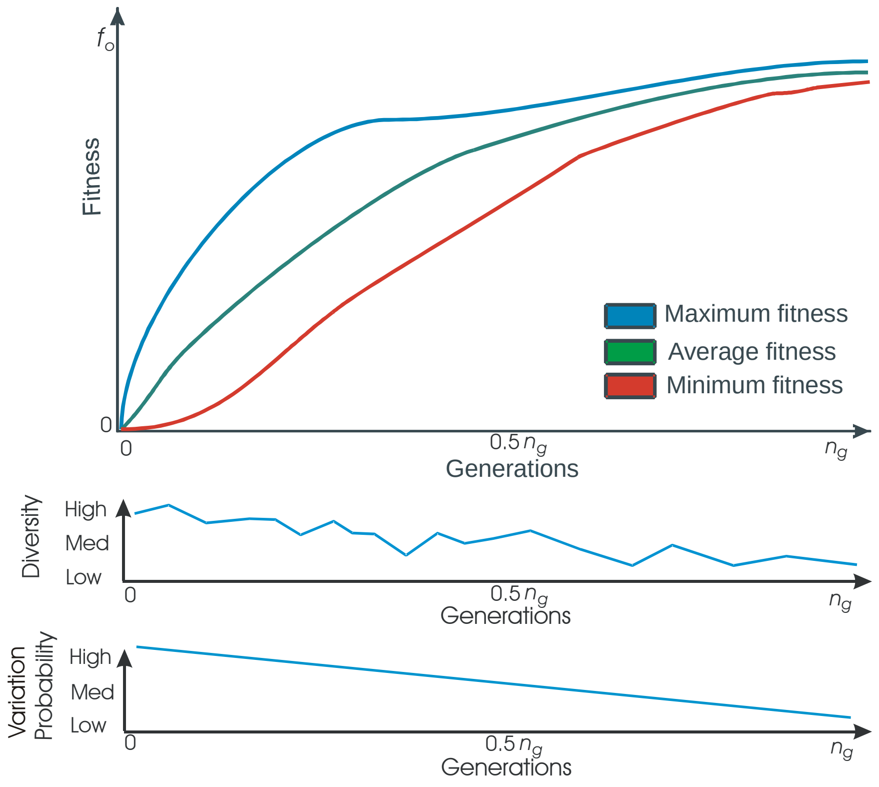


Fig. 15. The behavior of adaptive variation probability in evolutionary algorithms.

4.5 Hybrid Algorithms

Hybrid algorithms are a combination of two or more different techniques. Evolutionary algorithms have been successfully hybridized with other soft computing methods, such as fuzzy logic and neural networks. The combination of traditional hard computing methods and evolutionary algorithms is also common. The main reason for using these kinds of hybrid algorithms is to combine the strengths of different approaches in order to overcome the weaknesses of an individual method. Some hybridization schemes are discussed below.

4.5.1 Fuzzy Logic and Evolutionary Algorithms

One of the best-known areas of soft computing is fuzzy logic (FL) [Wan96a]. Developed in the 1960s by Lofti A. Zadeh [Zad02], FL has gained acceptance as an efficient way of transforming human knowledge into machine-understandable form. What is significant in FL is that it is very persistent with all kinds of uncertainties and lacks of information. Furthermore, it does not need very specific measurements to give good results. These kinds of properties are useful in adjusting the control parameters of EC. For instance, if we want to tune the mutation probability of a GA, we need to make decisions as to whether to increase or decrease the mutation probability on the basis of some population statistics, such as average fitness or the improvement of the maximum fitness over previous generations. There are no strict boundaries regarding our knowing when and how to manipulate the mutation probability.

In this dissertation, the emphasis is on applying fuzzy logic in order to improve the performance of evolutionary computation algorithms. However, on the other hand, evolutionary computation can also be used to improve the performance of fuzzy logic systems. A survey of EC enhancement of the performance of FL is presented in [Yuh99] and an application in the field of classification systems in [Mur95]. In [Mor05] EC is used to tune the if-then rules of a fuzzy function approximation system, and in general, it seems that the main focus of the research is on fusing EC and FL so that EC is used to enhance FL at present.

Takagi proposed a genetic algorithm employing FL to control population size, crossover probability, and mutation probability in [Tag93]. Wang describes in [Wan96b] the use of FL-tuned EC in a power economic dispatching problem in such a way that FL is used to control both the mutation and crossover probabilities. McClintock [Mcc97] implemented Takagi's scheme in the area of star pattern recognition to aid spacecraft navigation. In [Sub03] it is concluded that if FL is used to control EC control parameters, the variance of multiple search runs can be decreased in some cases.

Recently, FL has been used to boost EC performance in [Ahk04], in which a GA was used to configure an optimal electrical distribution network. In that paper FL controls both the mutation and crossover probabilities of the genetic algorithm and as inputs it uses the fitness averages of the two previous generations. In [Pyt04] a genetic algorithm is also modified, using an FL system, so that the selection probabilities, as well as the mutation and reproduction probabilities, are controlled by FL. As inputs that paper uses the relative fitness of individuals, as well as the fitness improvement over previous generations.

Figure 16 demonstrates the basic procedure of a Mandani-type fuzzy logic system. The core of such a fuzzy logic system is a *fuzzy rule base*, which defines the behavior of the fuzzy system. The fuzzy rule base contains the characteristics of the fuzzy system, and it is usually compiled by an expert in a particular application area. The fuzzy rules can be expressed as follows.

If V_1 is p_1 and V_2 is p_2, then V_3 is p_3.

V_1, V_2, and V_3 are variable names, e.g. *maximum fitness*, *minimum fitness*, and *mutation probability*, and p_1, p_2, and p_3 define the variables literally. These linguistic definitions can be, e.g. *low*, *medium*, *high* etc. In Fig. 16 each rule in the rule base defines an area, and all the areas defined by individual rules are combined together. From the combination of areas the final output of the fuzzy system is *defuzzified* using one of the various existing methods. For example, the *Center of Area* method defuzzifies the output by calculating the center of gravity of the composite area and the result is a crisp value.

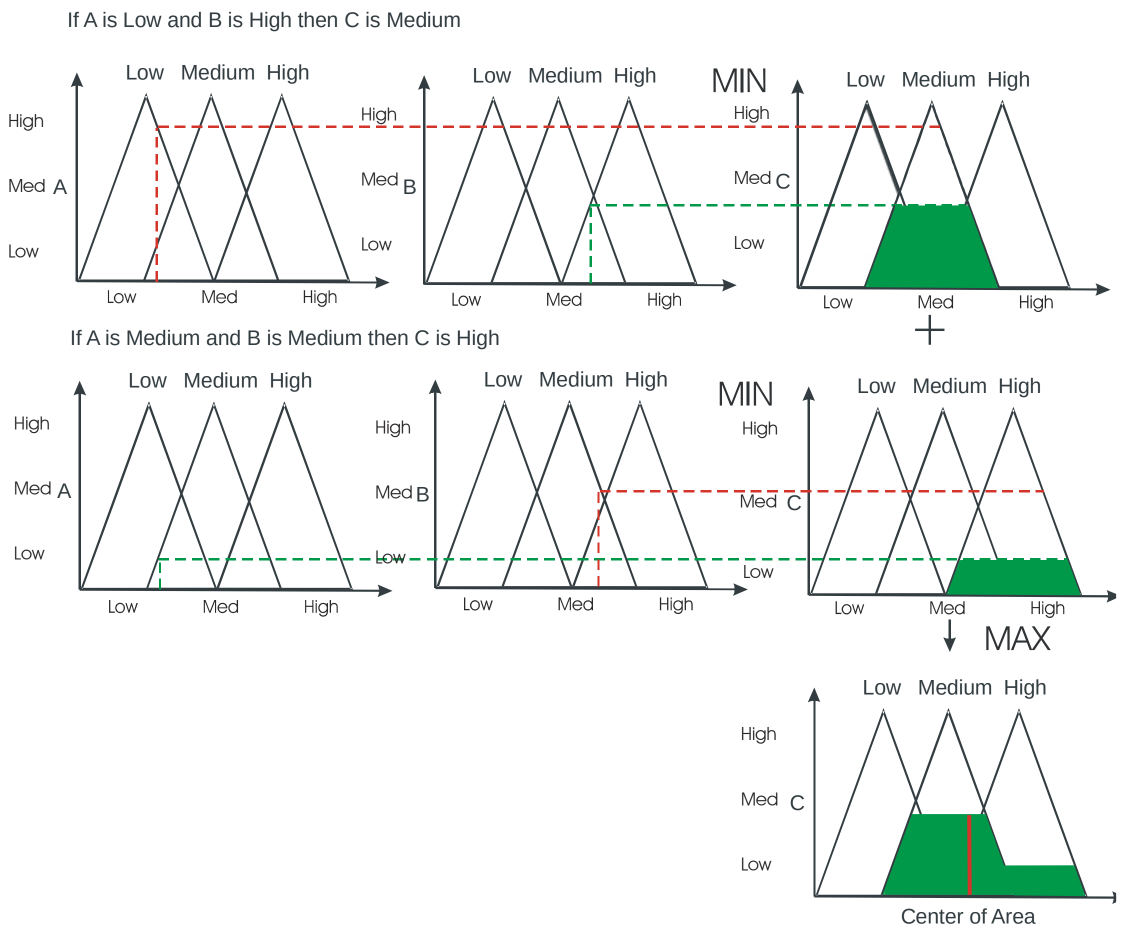


Fig. 16. The basic principle of a Mandani-type fuzzy logic system.

There exist several ways to perform the various stages of a fuzzy system and these stages are more thoroughly explained, for example, in [Wan96]. Fuzzy logic-controlled adaptive parameters were implemented in publications [P5] and [P6] in this dissertation.

4.5.2 Neural Networks and Evolutionary Algorithms

A neural network, also known as an *artificial neural network*, is a computational paradigm exhibiting a low-level resemblance to the human brain [Hay98]. Neural networks consist of units capable of learning, *neurons*. A single neuron is capable of performing only very limited tasks, but an interconnected network of multiple neurons is capable of handling difficult and complicated assignments, such as universal function approximation. Neural networks, like other soft computing techniques, are robust and tolerant towards errors in the input data. The structure of a neuron is depicted in Fig. 17.

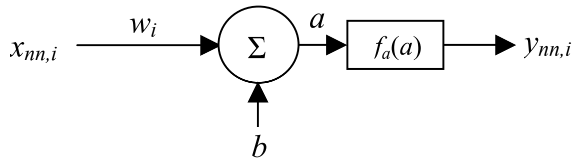


Fig. 17. A single neuron.

The neuron receives an input $x_{nn,i}$ through weight w_i. The weighted input is summed with a bias term b and this sum is the fed to an *activation function* f_a. The overall output of the neuron, $y_{nn,i}$, is thus

$$y_{nn,i} = f_a\,(w_i \cdot x_{nn,i} + b) \tag{1}$$

The weights w_i of the neural network are first *trained* using a training set of samples. There are various methods for training the neural network [Hay98]. After the training, the neural network can be used to approximate a function determined by the training set with new data.

There exist multiple types of neural networks, and the *multi-layer perceptron* (MLP) is one of the most commonly used. The single neuron shown in Fig. 17 is shown in simplified form in Fig. 18a. In Fig. 18b this model is used to display an MLP network with three input neurons and a single output neuron.

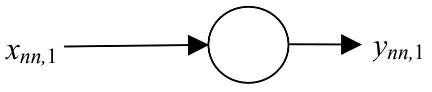


Fig. 18a. A simplified illustration of a neuron.

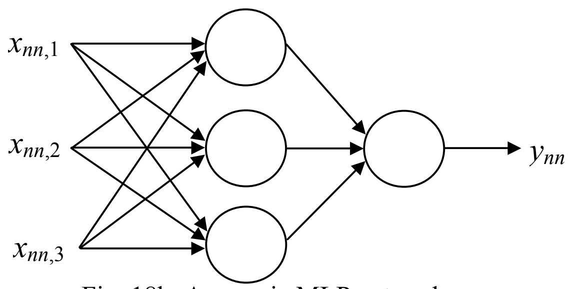


Fig. 18b. A generic MLP network.

Evolutionary algorithms and neural networks can be fused together in two main ways: either EC is used to improve NN performance, or vice versa. Kampfner et al. [Kam83] use an evolution-based computational method to evolve neural networks. In [Tsa06] a specific genetic algorithm is used to find the optimal parameter values and structure for a neural network. Similarly, in [Lam04] a genetic algorithm is used to devise the parameters of a neural network designed for digit and command interpretation for an electronics book. In this dissertation, however, the emphasis is on neural networks aiding evolutionary computation as in [P3]. In [Han97b] Han et al. discuss a method to produce silicone oxide films for integrated circuit fabrication using genetic algorithms with a neural network-based fitness function. In that paper, the calculation of fitness is a very complex task and a neural network is set up on the basis of sample data and this network is then used to evaluate the fitness of the solutions produced by the genetic algorithm. In [Pet98] a method whereby a neural network is used to produce the selection probabilities for individuals in a solution pool operated by a GA is discussed. That fusion of NN and GA is used to estimate the parameters of a model. In [Geo01] NN is used to calculate the fitness of the individuals in a GA in a transformer manufacturing problem. In [Miz00] Mizutani et al. discuss a fusion method involving NN, FL, and GA in the context of evolving color recipes. That paper uses both fuzzy logic and neural networks to produce the initial population for the genetic algorithm. Additionally, a neural network evaluates the fitness function of the genetic algorithm.

4.5.3 Hard Computing Methods and Evolutionary Algorithms

Hybrid algorithms fusing hard and soft computing methods are not dealt with in publications [P1]-[P8] in this dissertation, but they are widely used to improve the performance of evolutionary algorithms. These methods commonly take advantage of the good global search capabilities of evolutionary algorithms and the local search power of traditional methods. EC algorithms usually find the area of good fitness quite easily, but finding the final peak within that area may be time-consuming.

Lamarckian and *Baldwinian* strategies [Mag00] are often mentioned when these hybrid algorithms are being discussed. The Lamarckian approach means that an individual is subjected to HC-based local optimization at some stage of its evolution, and after this local search it is put back into the EC solution pool. If the HC optimization has resulted in a better fitness value, the structure of this individual is made to correspond to the new fitness value. Otherwise the individual is left intact. In the Baldwinian approach the structure of the individual is not changed, even though a better fitness may be achieved using HC, but only the fitness value is upgraded to the better one. As an example, evolutionary algorithms and gradient search in general are discussed in [Sal98]. In that paper an evolutionary optimization algorithm is executed conventionally for a specified interval of generations. At the end of the interval, solutions are subjected to a local search procedure, implemented using a gradient-based method. This kind of gradient descent approach relies on calculating the derivative for each parameter separately and adjusting the solution variable values accordingly. After the HC-based optimization procedure, the SC-based optimization methods resume.

It has been observed that using both hard computing methods and soft computing methods, including evolutionary computation, often result in more competitive solutions than using SC or HC alone. This fusion of hard and soft computing is discussed in detail in [Ova04] and applications implementing such techniques are surveyed in [Ova99].

4.6 Aging in Evolutionary Algorithms

Aging is a natural phenomenon for all animal species. The process of aging has been implemented in EC algorithms using many different approaches. [Gho98] presents the scheme of *effective fitness*. In that approach the age of an individual affects the fitness of the individual, younger and older individuals being less valuable than middle-aged individuals. In [Hub98] the age of an individual is used to direct the search of the algorithm. Each individual's age is expressed in terms of the number of generations it has managed to survive. Old individuals indicate good areas to search from, since it has been difficult for the selection operator to reject these individuals. In this scheme new individuals are generated through mutation from the best individuals. In [Iwa02] individuals' ages are used to control which operators are applied to any specific individual. For example, young individuals undergo only reproduction but no mutation. The main idea in aging in general is the prevention of stagnation, so that one average solution cannot dominate the population for too long a time. Aging is considered in [P5] in this dissertation.

5. Statistical Comparison of Evolutionary Algorithms

Evolutionary algorithms are simplifications of the phenomena of the surrounding natural systems. Natural and social environments offer an endless reservoir of ideas to be used in evolutionary computation. Clearly, not all natural phenomena can be transformed into an efficient optimization algorithm, if they can reasonably be modeled at all. Even if we could model all the phenomena of nature, for most of them we could only guess whether a particular occurrence is really pushing evolution forward or if it is just another blind alley of evolution that will die away after the next hundred generations. Usually, the value of an idea can only be measured through extensive testing. This leads us to an important question: when can one algorithm be considered to perform better than another in a certain optimization task?

To make a decision whether an algorithm performs well, the performance of the proposed algorithm needs to be compared to that of a reference algorithm or a reference performance value. In EC algorithms this means that the reference algorithm and the proposed algorithm differ only in the proposed new modification. For example, if we want to determine the effect of aging in a GA, GAs with and without aging need to be compared. Other parameters, such as population sizes, initial populations, and variation probabilities need to be the same. In particular, the number of objective function evaluations needs to be (at least roughly) the same in both the algorithms. If one algorithm has more opportunities to explore the search space than another, then it is probable that this algorithm will find a better solution than the other.

In general, the computational load presented by two algorithms under comparison should be roughly the same. Naturally, some control mechanisms of EC algorithms might take somewhat longer to execute than others, but the total run times should be approximately the same. If one algorithm uses far more computational resources, then it is probably likely to find better solutions than an algorithm employing less computation. In other words, both the algorithms may perform equally well if they are given an equal amount of iterations/generations. The insight the experiment brings depends on the settings of the experiment. This means, for example, that not lot of knowledge may be gained from an experiment in which the algorithms are given a different amount of computational time or resources, i.e. it is likely that one of the algorithms performs better because of the experimental setup.

A common reason for the comparison between two algorithms being less insightful is the extensive elaboration of the proposed algorithm and the negligible tuning of the reference algorithm. It is very likely that any modified EC algorithm will probably outperform a scratch-built EC algorithm without any parameter tuning. More insight is achievable only if the reference algorithm too is as good as it can be. Naturally, this is difficult to confirm in practice, but it is emphasized to say that a serious effort should be put into tuning the reference algorithm too.

EC-based optimization methods are stochastic by nature, so we cannot be absolutely sure how the algorithm is going to behave. Running two algorithms for a certain number of generations and comparing the final maximum objective function values tells us very little about the differences in the algorithms' performance. In general, as many runs as possible should be run for all the algorithms, and on the basis of the maximum fitness averages and variances some assumptions can be made as to whether one algorithm is better than another in a certain optimization problem. Suggestions on using proper statistics in EC are given in [Chr04]. In [Eib02] Eiben et al. discuss the experimental research methodology used in EC. That paper focuses on the most common mistakes made in EC research when considering common scientific research standards and it adds to the knowledge of most of the less experienced researchers working in the field of EC.

Statistical methods are not always properly used in SC-related papers. Flexer expresses his frustration towards inadequate statistics in neural network-related papers in [Fle96]. In particular, the documentation of the statistical methods used leaves a lot of things unexplained. That paper offers various important principles that are also applicable to the field of evolutionary computation.

5.1 Methods for Comparing Evolutionary Algorithms

Evolutionary optimization algorithms are stochastic in nature. This means that the results they produce are, in a sense, random and uncertain. This randomness follows from the use of a *random number generator* (RNG) as a part of most of the operators, e.g. an RNG can be used to decide the crossover point of two individuals, the location of a mutation, or the individuals to be selected for the next generation. Thus, runs of evolutionary algorithms with exact parameter settings produce different results if the RNG is not reset. Since evolutionary algorithms behave differently each time, the only acceptable way to compare the results of different algorithms is to use data collected over several runs. Below, some very basic tools for statistical comparison are discussed [Mil90].

The average of result samples is a commonly used measure for the performance of evolutionary algorithms. The average has the problem of letting so-called outliers affect the result; that is, a few exceptionally good or bad results can distort the average, especially if small sample sets are used. A median operator, similar to the average, conveniently reduces the effect of outliers.

However, comparing the average and median values alone is insufficient, for these do not explain the distribution of the data. Variance, then, is used to describe how much a single value usually differs from the expected value. The expected value, in this case, can be expressed as either an average or the median. Variance also tells us something about the reliability of an algorithm: the smaller the variance, the more reliably the algorithm produces results around the expected value. Then again, large variance denotes that the results of an algorithm can vary significantly from the expected value.

A convenient and illustrative tool for comparing two data sets is the *box plot*, also known as the *box-and-whisker diagram*. A box plot for the two data sets, A and B, shown in Table 1 is shown in Fig. 19.

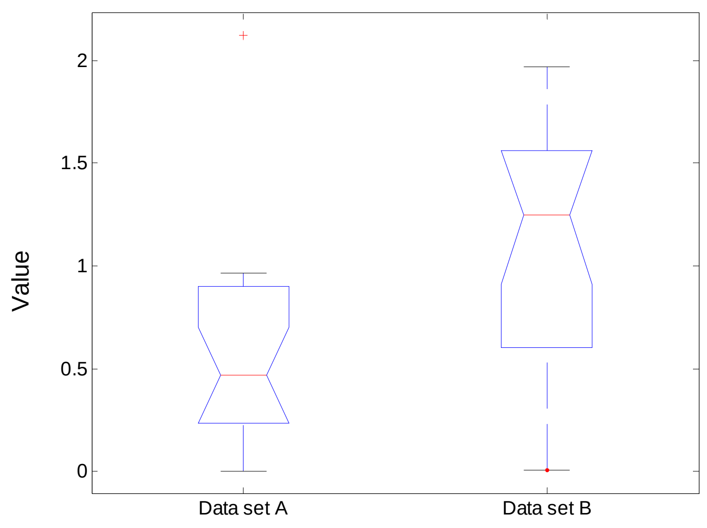


Fig. 19. A box plot for data sets A and B.

Table 1. Data sets A and B

A	B
0.4493	1.8953
0.2088	0.8462
0.9641	1.3599
0.0765	1.0444
2.1234	1.5413
0.0015	0.0486
0.2609	1.9685
0.3469	0.0905
0.9105	1.5831
0.1164	1.2426
0.5779	0.5689
0.4946	1.3714
0.0736	1.3818
0.9285	0.0093
0.4436	1.7458
0.3281	1.2561
0.8889	0.6411
0.4931	0.0724
0.9509	0.7127
0.6566	1.6466

The box in the box plot consists of three horizontal lines: the lower quartile, the median, and the upper quartile, from bottom to top. The lower quartile cuts off the lowest 25% of the data, whereas the median and upper quartile cut off 50% and 75% of the data, respectively. The whiskers are the lines extending from each end of the box denoting the whole scope of the observations, i.e. the lowest line indicates the smallest sample value and the highest line denotes the highest observed value. A "+" sign denotes an outlier that is not considered a part of the sample set. For example, the value 2.1234 is considered as an outlier in sample set A. In Fig. 20 the boxes have notches on both sides. These notches represent a robust estimate of the uncertainty of the medians when a box is compared with another box. In Fig. 19, data set B has a greater median value and the notches of the sets do not overlap. In this light, it is relatively safe to state that set B has a higher median value than set A. Averages, medians, variances, and box plots are standard tools for statistical comparison and these routines are automated in many of the sophisticated mathematical software packages available, such as Matlab [Mat06a].

The Student's t-test [Chr04] is occasionally used to compare the outputs of two data sets produced by evolutionary algorithms. The t-test is a convenient tool, but using it correctly requires the data set to be normally distributed. This is difficult, if not impossible, to show in practice in the context of EC and thus one should be very careful when using the t-test to compare the results of two different evolutionary algorithms. Using the t-test when the requirement for the normal distribution of the input data sets is not met may lead to erroneous results. However, the reliability of the t-test depends on a few things, such as sample size and the degree of normality.

5.2 Bootstrap Resampling-Based Multiple Hypothesis Testing

The problem of comparing two data sets for equality or difference in a parameter of interest is a common problem encountered in many fields of science. Thus, alternative statistical methods for comparing parameters, other than averages and standard deviations, have been developed. One interesting approach to this problem of comparison is the use of bootstrap resampling-based multiple hypothesis testing, discussed in [Efr98]. This approach allows the user to compare the performance of the two algorithms and it requires no distributional assumptions. This makes resampling-based multiple hypothesis testing an attractive tool for comparing the results produced by evolutionary algorithms. The resampling-based multiple hypothesis testing framework is explained in [Efr98] as follows.

To start with, there are two sets of data, S_1 and S_2, generated by, for example, two different evolutionary algorithms and they are possibly from two different probability distributions, F and G. We will consider the mean of the data distribution to be a suitable *parameter of interest* to estimate as a measure of the algorithm's performance. The task is to determine whether the two distributions have equal means or one of the two algorithms performs better in this particular problem. A *null hypothesis*, H_0, of no difference between the means of F and G is created.

$$H_0 : \overline{S}_1 = \overline{S}_2 \tag{2}$$

This null hypothesis states that the means of both the data sets are equal. If we cannot reject the null hypothesis then there is insufficient evidence to conclude that they are unequal; otherwise, there exists a difference between them, and this is the *alternative hypothesis,* $\overline{S}_1 \neq \overline{S}_2$, to the null hypothesis. The null hypothesis and the alternative hypothesis may be defined as complements of each other, so that eventually one of them is true. Since the initial population may have an effect on the performance of the algorithm, in this comparison scenario multiple initial populations are created and both the algorithms are run a number of times using the same initial population. From this it follows that there is a null hypothesis for each initial population created and that altogether there are as many null hypotheses as there are initial populations created. Conclusions on the performance differences can be drawn on the basis of the ratio between the rejected null hypothesis and the number of initial populations.

A hypothesis test begins with a selection of test statistics, $\hat{\theta}$, and in this case it is convenient to choose the Studentized mean difference as the test statistics. So,

$$\hat{\theta} = \frac{\overline{S}_1 - \overline{S}_2}{\sqrt{\overline{\sigma}_1^2 / k_1 + \overline{\sigma}_2^2 / k_2}} \tag{3}$$

$\overline{\sigma}_1^2$ and $\overline{\sigma}_2^2$ represent the variances of each point the original data sets 1 and 2, respectively. k_1 and k_2 denote the sample sizes of S_1 an S_2, respectively.

Having the data sets S_1 and S_2, the achieved significance level, the *p-value*, is defined as the probability of observing at least that large a value when the null hypothesis is true.

$$\mathrm{p} = \mathrm{prob}_{H_0} \left\{ \left| \hat{\theta}^* \right| \geq \left| \hat{\theta} \right| \right\} \tag{4}$$

The smaller the p-value, the stronger the evidence against the null hypothesis is. $\hat{\theta}$ is calculated directly from the data sets S_1 and S_2 sized k_1 and k_2 data points, respectively, and $\hat{\theta}^*$ is the test statistic calculated from the resampled data.

Finally, the approximation for the $\mathrm{p}_{\mathrm{boot}}$, the bootstrap-based estimate for the p-value, is calculated as follows:

1. Modify both the data sets S_1 and S_2 by subtracting the set mean and adding the common mean, m_{S12} of S_1 and S_2:

$$\begin{aligned} S_{1,adj} &= S_1 - \overline{S}_1 + m_{s12} \\ S_{2,adj} &= S_2 - \overline{S}_2 + m_{s12} \end{aligned} \tag{5}$$

2. Draw B (e.g. B>10000) samples of size k_1+k_2 with replacement from a pool containing all the individual samples of the sets $S_{1,adj}$ and $S_{2,adj}$. The probability for sampling a single observation is $1/(k_1+k_2)$. The first k_1 resampled values are denoted by S_1^* and the last k_2 values are called S_2^*, the asterisk indicating resampled values rather than observed ones.

3. The test statistic for each of the B sampled sets is calculated as

$$\hat{\theta}^*(b) = \frac{\overline{S}_1^* - \overline{S}_2^*}{\sqrt{\overline{\sigma}_1^{2*}/k_1 + \overline{\sigma}_2^{2*}/k_2}}, \quad b = 1,2,...,B \tag{6}$$

   and

$$\begin{aligned} \overline{\sigma}_1^{2*} &= \frac{\sum_{i=1}^{k}(S_{1,i}^* - \overline{S}_1^*)^2}{k_1} \\ \overline{\sigma}_2^{2*} &= \frac{\sum_{i=1}^{k}(S_{2,i}^* - \overline{S}_2^*)^2}{k_2} \end{aligned} \tag{7}$$

4. Finally, we have an estimate for the p-value as

$$\mathrm{p}_{\mathrm{boot}} = \#\left\{\left|\hat{\theta}^*(b)\right| \geq \left|\hat{\theta}\right|\right\}/B, \quad b = 1,2,...,B. \tag{8}$$

where $\hat{\theta}$ is the observed value of the test statistic. Equation 10 means that $\mathrm{p}_{\mathrm{boot}}$ equals the number of resampled sets that have a larger test statistic value (in magnitude) than the observed value and this number is then divided by B to get an approximation for the achieved significance level. There are no strict guidelines as to the values of p showing when to reject the null hypothesis and support the alternative hypothesis, but the p-value is usually selected before the test; Efron et al. [Efr98] suggest the conventions shown in Table 2.

Table 2. Rough guidelines for selecting the p-value [Erf98].

p-value	Evidence against H_o
< 0.10	Borderline
< 0.05	Reasonably strong
< 0.025	Strong
< 0.01	Very strong

The field of soft computing does not generally use the commonly accepted procedure for comparing the data sets produced by two different algorithms. Resampling-based multiple hypothesis testing is an interesting option that should be seriously considered. The entire procedure is explained in [Efr98] and more recent results concerning multiple hypothesis testing can be found in [Pol05].

The resampling-based multiple hypothesis testing method for statistical comparison is a *non-parametric* test, meaning that it does not require assumptions regarding the underlying distribution of the sample sets. *Parametric* tests, then, rely on some assumptions regarding the distribution of the data.

5.3 About Statistical Comparison of Evolutionary Algorithms

When developing evolutionary algorithms, similar attention must be paid to the evaluation of the results as to the development process itself. Evolutionary algorithms produce stochastic results, and therefore as much data as possible should be collected before the algorithm is evaluated. The amount of data to be collected is usually determined by the available time and computational resources and the computational requirements of the algorithm under study. Additionally, attention must be paid to the reference algorithms. It is comparatively easy to produce an algorithm that outperforms, e.g., the basic genetic algorithm. The comparison of two algorithms is only insightful if a serious effort is made to elaborate both the reference and the proposed algorithm. Preferably, if we are studying a particular modification to an existing algorithm, the reference and the proposed algorithm should only differ in respect of the modification being studied. Thus, it is possible to conclude if the new mechanism adds to the performance of the algorithm. Each algorithm under comparison should be given an equal amount of computational resources or time in order to ensure an insightful comparison. In addition, initial populations may introduce bias to the results, so the use of the same initial populations for the algorithms being studied is preferable.

Averages and medians give some indication of the performance of the algorithm. However, a minimum requirement is to report the standard deviation of these results, since average and median values alone are poor estimates of the reliability of the results. More sophisticated comparison methods, such as the use of resampling-based multiple hypothesis testing, are strongly encouraged. The field of evolutionary computation does not commonly use the accepted methods for statistical evaluation. This dissertation hopes to raise discussion related to this important matter.

6. Summary of the Publications

This dissertation is divided into three parts. Publications [P1], [P2], and [P4]-[P6] discuss the use of enhanced evolutionary algorithms for complex optimization problems. Publication [P3] concentrates on reducing the computational requirements imposed by the objective function evaluation by means of neural networks. Finally, publications [P7] and [P8] illustrate the use of a comprehensive statistical scheme for comparing two different evolutionary algorithms. In the following sections, the main results of the publications and the contribution of the author are discussed.

6.1 [P1]

J. Martikainen and S. J. Ovaska, "Designing multiplicative general parameter filters using adaptive genetic algorithms," in *Proc. of the Genetic and Evolutionary Computation Conference*, Seattle, WA, 2004, pp. 1162-1167.

In [P1] an adaptive genetic algorithm is used to optimize a Multiplicative General Parameter (MGP) basis filter, a Finite Impulse Response (FIR) filter design problem. MGP-FIRs have been designed before using evolutionary computation, but this paper introduces an effective adaptive GA-based approach to produce competitive design performance. Additionally, an applicable structure for the solutions was discovered and seeding the initial population with this structure was found to accelerate the convergence of the algorithm.

The main result of this paper is the successful implementation of the adaptive variation probabilities of the EC algorithm. Modifying the original adaptive probabilities of [Sri94], the results showed better performance in terms of increased average fitness when compared to a reference algorithm without adaptive parameters and the original adaptive parameters in [Sri94]. Other important results include the optimization of the adaptation gain factor of the MGP-FIR and the proper structure for seeding the initial population. Altogether, this paper proposed the most powerful design method for MGP-FIR basis filters that existed at the time of its publication.

The author was responsible for implementing and modifying the adaptive scheme used in this paper. S. J. Ovaska introduced the problem to the author and suggested the use of genetic algorithms as the optimization tool.

6.2 [P2]

J. Martikainen and S. J. Ovaska, "Designing multiplicative general parameter filters using multipopulation genetic algorithms," in *Proc. of the 6th Nordic Signal Processing Symposium*, Espoo, Finland, 2004, pp. 25-28.

In [P2], the work already started in [P1] is continued, namely, the efficient optimization of MGP-FIR basis filters. In this study, a two-population GA (2PGA) scheme is introduced, in which a large *plain population* conducts a global search while a small *elite population* searches through local optima. The two populations are allowed to exchange solutions under predefined rules. The proposed two-population

GA is straightforward to implement and, apart from adding to the performance of its single-population counterpart, it does not add much to the complexity of the basic algorithm.

The main result of the paper is the introduction of a straightforward two-population genetic algorithm. Multiple-population EC algorithms have been studied extensively before, but the proposed approach, contrary to the common trend, uses only a single CPU instead of actual parallel hardware. Similar idea to this scheme have also been studied before, but what makes this implementation stand out is the fact that the individuals are placed and maintained in subpopulations on the basis of their fitness characteristics, whereas allocating individuals to subpopulations has been more or less a random process in the past.

This proposed algorithm is convenient to implement and it can be applied with every EC algorithm, thus making it a very general-purpose enhancement. The computational overhead resulting from the multipopulation extension is negligible. The results in the paper clearly show that the proposed scheme improves even further the design process of the MGP-FIR basis filter.

The author implemented and tuned the proposed 2PGA scheme and was also responsible for conducting the experiments. S. J. Ovaska proposed the idea of hierarchical populations, a concept which was later refined in cooperation with the author.

6.3 [P3]

J. Martikainen and S. J. Ovaska, "Fitness function approximation by neural networks in the optimization of MGP-FIR filters," in *Proc. of the IEEE Mountain Workshop on Adaptive and Learning Systems*, Logan, UT, 2006, pp. 231-236.

Previous work in [P1] and [P2] developed efficient algorithms for the MGP-FIR basis filter optimization problem. Still, the evaluation of the MGP-FIR objective function was time-consuming, regardless of the efficient optimization algorithms. This study concentrated on reducing the time required for the evaluation of the objective function by means of neural network approximation.

The main result of the paper is the fusion of neural networks and genetic algorithms to improve the optimization process of MGP-FIR basis filters. An appropriate structure for neural networks is defined and the training of the neural network is embedded in the on-line optimization process. Furthermore, the fitness function was redefined so as to conform better to the requirements of the application. The proposed scheme is capable of reducing significantly the computational effort required by the fitness function evaluations and this also includes the time required to train the network. An important notion was that it was difficult to approximate the different parameters of the objective function accurately with separate neural networks. Instead, using a single network seems to bind the approximations together, so that the results are more accurate than those obtained using separate networks.

S. J. Ovaska proposed the use of neural networks as part of the fitness function. The author and S. J. Ovaska reformulated the fitness function together. The author was responsible for elaborating and tuning the NN-assisted fitness function, as well as the general optimization scheme.

6.4 [P4]

J. Martikainen and S. J. Ovaska, "Hierarchical two-population genetic algorithm," *International Journal of Computational Intelligence Research*, vol. 2, no. 4, 2006, in press.

Publication [P4] is a comprehensive study that concentrates on the two-population genetic algorithm proposed in [P2]. The 2PGA scheme proposed in [P2] contains many tunable parameters and the purpose of this work was to understand the effect of these parameters on the optimization process. This work is important, since the tuning of any algorithms' parameters can be a time-consuming process.

The main result of this paper is the instructions that could be given concerning 2PGA parameter settings. On the basis of the paper, a user can implement the 2PGA scheme and start searching for suitable parameter values for a particular application from reasonable initial values. The proposed 2PGA scheme is intended to improve a basic population-based optimization scheme and cannot replace more sophisticated methods. However, this enhanced basic algorithm too can benefit from any advantageous modification applicable to evolutionary computation.

The author was fully responsible for carrying out the research concerning the affect of different parameter settings in the 2PGA scheme. S. J. Ovaska originally proposed the idea of investigating the effect of different parameter settings.

6.5 [P5]

J. Martikainen and S. J. Ovaska, "Optimizing dynamical fuzzy systems using aging evolution strategies," in *Proc. of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing*, Benidorm, Spain, 2005, pp. 5-10.

In [P5], the 2PGA scheme proposed in [P2] is implemented using evolution strategies (2PES). Apart from this, an adaptive aging parameter is introduced in order to further improve the algorithms' performance. The aging parameter controls the remaining lifetime of each solution on the basis of the value of the offspring they have produced. Different aspects of aging have been implemented before in EC, but in this study the remaining lifetime of an individual is controlled by fuzzy logic based on the offspring the individuals produce, as well as the population dynamics. Such an approach (combining multiple populations and a fuzzy aging parameter) has not been used before. As a test case, this paper uses a very demanding optimization task dealing with the parameters of a dynamical fuzzy system with linguistic information feedback.

The main result of this paper is the successful implementation of the 2PGA scheme with evolution strategies. Additionally, the proposed fuzzy logic-controlled aging parameter was found to add to the performance of the 2PGA scheme used for comparison.

S. J. Ovaska proposed the use of aging as a part of the MPGA scheme. The author designed and implemented the fuzzy logic-controlled aging parameter.

6.6 [P6]

J. Martikainen and S. J. Ovaska, "Using fuzzy evolutionary programming to solve traveling salesman problems," in *Proc. of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing*, Benidorm, Spain, 2005, pp. 49-54.

Publication [P6] introduces a new way to implement problem decomposition to improve evolutionary programming performance. As a demanding test case, a 500-city Traveling Salesman Problem (TSP) was used. It was found that the best way to decompose a problem varies as the algorithm proceeds. Studying the behavior of the algorithm with different parameter settings, we were able to present fuzzy adaptive control for partitioning the problem as the algorithm is executed. These divide-and-conquer-type approaches have been used before, but no fuzzy control for this kind of partitioning seems to be available. In particular, in the proposed method the partitioning of the problem into sub-problems takes place repetitively instead of just once.

The main result of this paper was the design and implementation of the fuzzy logic-controlled problem repetitive decomposition scheme. The proposed adaptive partitioning scheme was able to outperform the static approach used as a reference.

S. J. Ovaska proposed the idea of divide-and-conquer for large-scale optimization problems and the author designed and implemented the fuzzy logic-controlled divide-and-conquer scheme used to partition the TSP.

6.7 [P7]

D. Shilane, J. Martikainen, S. Dudoit, and S. J. Ovaska, "A general framework for statistical performance comparison of evolutionary computation algorithms," in *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, Innsbruck, Austria, 2006, pp. 7-12.

The field of evolutionary computation does not commonly take advantage of the accepted procedures for sound statistical comparison of two algorithms. Publication [P7] illustrates the use of such a statistical comparison scheme based on bootstrap resampling and multiple hypothesis testing.

The proposed statistical scheme is explained thoroughly in [P7] and the scheme is also shown to work in a case involving two generic genetic algorithms. This publication should enable all practitioners of evolutionary algorithms to implement these methods in their studies for improved statistical analysis.

The problem of not applying the unified standards for statistical comparison within the EC field was recognized by S. J. Ovaska and the author. The actual problem of properly comparing two data sets produced by different evolutionary algorithms was formulated by D. Shilane, the author, and S. J. Ovaska. D. Shilane proposed the statistical scheme, including bootstrap resampling-based multiple hypothesis testing.

The author was responsible for implementing the evolutionary algorithms and D. Shilane implemented the statistical testing procedure. The publication was a joint effort by the author and D. Shilane. S. J. Ovaska and S. Dudoit offered advice and comments during the research process.

6.8 [P8]

J. Martikainen and S. J. Ovaska, "Comparison of a fuzzy EP algorithm and an AIS in dynamic optimization tasks," in *Proc. of the IEEE Mountain Workshop on Adaptive and Learning Systems*, Logan, UT, 2006, pp. 7-12.

New optimization methods making use of natural phenomena are frequently introduced. Recently, artificial immune systems have been one such approach. Publication [P8] studies the optimization scheme of AIS, the clonal selection principle, in a dynamic environment. This study was conducted in order to survey the capabilities of a standard CSP and a specialized EC algorithm in a dynamic environment. CSP is known to be capable of finding multiple good solutions, whereas evolutionary algorithms usually find only a single optimum.

The main result of this publication is the notion of CSP outperforming the evolutionary algorithm in the studied dynamic environments. The evolutionary algorithm that was implemented does well in a static problem, but, because of the preserved diversity, the CSP does better when the objective function varies in time. Publication [P8] uses the statistical scheme proposed in [P7] to study the actual difference of the algorithms. The statistical comparison scheme has some adjustable parameters and [P8] studies the effects of some of them. The conclusion regarding the parameter setting indicates that too few bootstrap resamplings may cause erroneous interpretations of the results of the statistical comparison.

The author and S. J. Ovaska jointly proposed the evaluation of AIS and an evolutionary algorithm in a dynamic environment. The comparison of the effect of different parameter values was proposed by the author. All the implementation and analysis was done by the author.

7. Conclusion

This dissertation proposes methods aimed at improving the reliability of evolutionary algorithms in complex optimization tasks and accelerating problem-solving. The dissertation takes three distinct viewpoints on evolutionary computation-based optimization and offers improved methods for tackling these commonly agreed demerits. These key areas for additional improvement are the reliability of the algorithms, reducing the time required to evaluate the complicated objective function, and the use of a commonly agreed framework for the comprehensive statistical comparison of evolutionary algorithms.

7.1 Main Results

The main results of this dissertation can be divided into three main categories as defined by the goals of the work. This work proposes new algorithms to increase the reliability of evolutionary computation in complex optimization problems. The 2PGA scheme discussed in publications [P2] and [P4] is a new approach to conducting a local and global search in parallel, using only genetic algorithms and with negligible additional computational costs. The 2PGA scheme is not intended to replace more sophisticated algorithms, such as hybrid methods with evolutionary algorithms and hard computing-based local search mechanisms. Rather, the proposed scheme is intended to act as a powerful basic platform that can be enhanced using the same means as any other population-based optimization scheme. The 2PGA has multiple tunable parameters and their effect is studied in [P4], a work that also suggests a reasonable parameter set from which to start the search for a competitive parameter set for a specific application.

In addition to the 2PGA scheme, fuzzy logic is used to improve the reliability of evolutionary algorithms. Publications [P5] and [P6] use fuzzy logic to adapt the parameters of evolutionary algorithms on the basis of the changing characteristics of the solution population. Publication [P5] introduces a fuzzy aging strategy that determines the remaining lifetime of an individual on the basis of the offspring it produces. In [P6] fuzzy logic is used to repeatedly partition a traveling salesman problem into smaller sub-problems as the algorithm proceeds.

Publication [P3] studies a scheme in which a part of an objective function of an evolutionary algorithm is approximated using neural networks. The proposed scheme approximately halved the computational time required to evaluate the objective function under study. The trade-off for faster computation is the approximation error. However, the proposed method succeeds in minimizing the approximation error so that the proposed method eventually outperforms the reference algorithm. The study of the NN approximated fitness function of the MGP-FIR basis filter suggested that it is possible to model complex systems accurately enough to speed up the calculation of the fitness function. An important notion resulting from extensive experimenting was that seemingly unconnected variables could be approximated using only a single network rather than an individual network for each variable. This phenomenon is caused by the fact that a single neural network binds the approximated variables together, not allowing for large errors for individual components. Then again, when using an individual network for each component, separate networks have no way to relate to each other and the approximation of the overall fitness function performs poorly.

Finally, publications [P7] and [P8] discuss a proper statistical framework suitable for comparing two evolutionary algorithms. These publications offer instructions for implementing the scheme and give advice on the respective parameter selection.

The proposed modifications to evolutionary algorithms are applicable to many evolutionary computation schemes. According to the no free lunch theorem, the average performance of all optimization algorithms is similar over all possible optimization problems. Thus, the usefulness of the proposed schemes depends on the application. In addition, all the proposed techniques could be combined into a single algorithm, but then again, the addition to the performance of the algorithm would depend on the problem.

7.2 Scientific Importance of Author's Work

The scientific importance of the work proposed in this dissertation is threefold. First, the methods intended for improving the reliability of evolutionary algorithms take a different viewpoint to evolutionary algorithms from that which common in the field in general. The proposed 2PGA scheme does not try to optimize a single solution; rather, it creates a good environment for both fit and less fit solutions to evolve in. The proposed scheme bears similarities to niching and coevolution, but, unlike niching schemes, it does not require excessive additional computation and the solutions can migrate from one niche to another. This kind of low-cost improvement of the basic algorithm has not been studied before. Additionally, when compared to coevolution frameworks, in the proposed scheme the individuals are placed in populations on the basis of their fitness characteristics rather than randomly. The fusion of evolutionary algorithms and fuzzy logic has been studied extensively before. However, the fuzzy aging parameter that controls the remaining lifetime of solutions on the basis of the offspring it produces is unique. Similarly, the repetitive partitioning procedure controlled by fuzzy logic has not been studied previously.

The concept of using a neural network to approximate the MGP-FIR basis filter is completely new and has never been studied before. The proposed method sets new standards for optimizing the MGP-FIR basis filter, since the time required to evaluate a candidate filter is nearly halved.

The field of evolutionary computation usually neglects the commonly accepted procedure for statistical comparison. The work presented in this dissertation sets an example and raises discussion of the important topic of the comprehensive statistical comparison of evolutionary algorithms.

To conclude, the methods proposed in this dissertation will enable increasingly challenging applications to be tackled more reliably and efficiently by means of evolutionary computation.

7.3 Topics for Future Research and Development

Topics for future research could include the process of adapting the 2PGA parameters on-line using fuzzy logic. Static parameter values are usually not optimal throughout the run time of the algorithm, so adaptation could further increase the performance of the 2PGA scheme. In addition, the concept of using more than two hierarchical populations at the same time should be studied more extensively.

The approximation of the objective function using neural networks could be studied further. In particular, the repeated training of the network as the algorithm proceeds should be studied more extensively. At present, the approximation error increases towards the end, but it is possible to reduce this error by retraining the network again during the run time of the algorithm.

To further speed up time-consuming optimization tasks, the feasibility of assigning probabilities on the basis of the individual's fitness values for objective function evaluations should be studied. In addition, an effort should be made to see if it is possible to gain more insight into the proposed algorithms through theoretical examination. In particular, this would mean that accurate models of individual operators, such as aging or fuzzy logic-controlled variation operators, should be constructed and subjected to theoretical studies. These theoretical inspections could bring more insight, e.g., into the operators' capabilities of maintaining diversity within a solution population.

The proposed twofold statistical comparison scheme could be built into a Matlab toolbox to offer ready-made easy-to-use functions for practitioners of evolutionary computation. Additionally, a comparison procedure for more than two evolutionary algorithms simultaneously could be explored.

References

[Ahk04] R. T. F. Ah King, B. Radha, and H. C. S. Rughooputh, “A fuzzy logic controlled genetic algorithm for optimal electrical distribution network reconfiguration,” in *Proc. of the IEEE International Conference on Networking, Sensing, and Control*, vol. 1, Taipei, Taiwan, 2004, pp. 577-528.

[Alb02] E. Alba and M. Tomassini, “Parallelism and evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, 2002, pp. 443-462.

[Ari96] T. Arita and A. Ojika, “Generation of color patterns based on the interactions between predators and prey,” in *Proc. of the IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, pp. 291-294.

[Ass98] A. Assion, T. Baumert, M. Bergt, T. Brixner, B. Kiefer, V. Seyfried, M. Strehle, and G. Gerber, “Control of chemical reactions by feedback-optimized phase-shaped femtosecond laser pulses,” *Science*, vol. 282, 1998, pp. 919-922.

[Au03] W.-H. Au, K. Chan, and X. Yao, “A novel evolutionary data mining algorithm with applications to churn prediction,” *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, 2003, pp. 532-545.

[Bag05] A. J. Bagnall and G. D. Smith, “A multiagent model of the UK market in electricity generation,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, 2005, pp. 522-536.

[Bal93] S. Baluja, “Structure and performance of fine-grain parallelism in genetic search,” in *Proc. of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, 1993, pp. 155-162.

[Bel95] T. C. Belding, “The distributed genetic algorithm revisited,” in *Proc. of the 6th International Conference of Genetic Algorithms*, Pittsburg, PA, 1995, pp. 114-121.

[Ben02] E. Benini and A. Toffolo, “Optimal design of horizontal-axis wind turbines using blade-element theory and evolutionary computation,” *Journal of Solar Energy Engineering*, vol. 124, no. 4, 2002, pp. 357-363.

[Ber00] A. Berlanga, P. Isasi, A. Sanchis, and J. R. Molina, “Coevolutive adaptation of fitness landscape for solving the testing problem,” in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, Nashville, TN, 2000, pp. 3846-3851.

[Bey01] H.-G. Beyer and K. Deb, "On self-adaptive features in real-parameter evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, 2001, pp. 250-270.

[Bre62] H. J. Bremermann, "Optimization through Evolution and Recombination," in *Self-Organizing Systems*, M. C. Yovits, G. T. Jacobi, and G. D. Goldstine (eds.). Washington, DC: Spartan Books, 1962, pp. 93-106.

[Bri01] M. S. Bright and T. Arslan, "Synthesis of low-power DSP systems using a genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 1, 2001, pp. 27-40.

[Bur99] E. K. Burke and J. P. Newall, "A multistage evolutionary algorithm for the timetable problem," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 1, 1999, pp. 63-74.

[Bäc96a] T. Bäck and H.-P. Schwefel, "Evolutionary computation: an overview," in *Proc. of IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, pp. 20-29.

[Bäc96b] T. Bäck, *Evolutionary Algorithms in Theory and Practice, Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY: Oxford University Press, 1996.

[Bäc97] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 3-17.

[Can02] E. Cantú-Paz, "Markov chain models of parallel genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no 3, 2000, pp. 216-226.

[Cas02a] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*. London, UK: Springer Verlag, 2002.

[Cas02b] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, 2002, pp. 239-251.

[Cec06] 2006 IEEE World Congress on Computational Intelligence, [WWW page]. [cited 20 June 2006]. Available at <http://www.wcci2006.org/>.

[Cha95] P. Charbonneau, "Genetic algorithms in astronomy and astrophysics," *The Astrophysical Journal Supplement Series*, vol. 101, 1995, pp. 309-334.

[Cha02] B. Chackraborty, “Genetic algorithm with fuzzy fitness function for feature selection,” in *Proc. of the IEEE International Symposium on Industrial Electronics*, L’Aguila, Italy, 2002, pp. 315-319.

[Cho00] B. Chopard, O. Pictet, and M. Tomassini, “Parallel and distributed evolutionary computation for financial applications,” *Parallel Algorithms Applications*, vol. 15, 2000, pp. 15-36.

[Chr04] S. Christensen and M. Wineberg, “Using appropriate statistics – statistics for artificial intelligence,” in *Tutorial Program of the Genetic and Evolutionary Computation Conference*, Seattle, WA, 2004, pp. 544-564.

[Cod93] B. Codenotti and M. Leoncini, *Introduction to Parallel Processing*. Pisa, Italy: Addison-Wesley, 1993.

[Cor05] D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. Maidenhead, Berkshire: McGraw-Hill, 1999.

[Cul99] D. E. Culler and J. P. Singh, *Parallel Processing Architecture: a Hardware/Software Approach*. San Francisco, CA: Morgan Kaufmann, 1999.

[Dar59] C. Darwin, *On the Origin of Species by Means of Natural Selection or the Preservations of Favored Races in the Struggle for Life*. London, UK: John Murray, 1859.

[Dar97] P. J. Darwen and X. Yao, “Speciation as automatic categorical modularization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, 1997, pp. 101-108.

[Dav97] C. Davidson, “Creatures from primordial silicon,” *New Scientist*, vol. 156, no. 2108, 1997, pp. 30-35.

[Dor97] M. Dorigo and L. M. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 53-66.

[Eib99] A. E. Eiben, R. Hinterding, and Z. Michalewicz, “Parameter control in evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, 1999, pp. 124-141.

[Eib02] A. Eiben and M. Jelasity, “A critical note on experimental research methodology in EC,” in *Proc. of the Congress on Evolutionary Computation*, Honolulu, HI, 2002, pp. 582-587.

[Eng06] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. West Sussex, UK: John Wiley & Sons, 2006.

[Evo06] Evolutionary Computation, [WWW-page]. [cited 21 June 2006]. Available at < http://www.mitpressjournals.org/loi/evco>.

[Far03] R. Farmani and J. A. Wright, "Self adaptive formulation for constrained optimization," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, 2003, pp. 445-455.

[Fog62] L. J. Fogel, "Autonomous automata," *Industrial Research*, no. 4, 1962, pp. 14-19.

[Fog66] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York, NY: John Wiley, 1966.

[Fog89] T. Fogarty, "Varying the probability of mutation in the genetic algorithm," in J. D. Schaffer (ed.), *Proc. of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989, pp. 104-109.

[Fog90] D. B. Fogel, "Simulated evolution: a 30-year perspective," in *Proc. of 24th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, 1990, vol. 2, pp. 1009-1014.

[Fog94] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, 1994, pp. 3-14.

[Fog98] D. B. Fogel (ed.), *Evolutionary Computation: The Fossil Record.* Piscataway, NJ: IEEE Press, 1998.

[Fog00] D. B. Fogel, *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence*. 2nd edition. Piscataway, NJ: IEEE Press, 2000.

[Fog01] D. B. Fogel, *Blondie24: Playing at the Edge of AI.* San Francisco, CA: Morgan Kaufmann, 2001.

[Fog06] D. B. Fogel, *Evolutionary Computation – Toward a New Philosophy of Machine Intelligence.* 3rd edition, Hoboken, NJ: Wiley - IEEE Press, 2006.

[Fra57] A. S. Fraser, "Simulation of genetic systems by automatic digital computers. I. Introduction," *Australian Journal of Biological Sciences*, vol. 10, 1957, pp. 484-491.

[Gai04] Z.-L. Gaing, "A particle swarm optimization approach for optimum design of PID controller in AVR system," *IEEE Transactions on Energy Conversion*, vol. 19, no. 2, 2004, pp. 384-391.

[Gec06] International Society for Genetic and Evolutionary Computation, [WWW page]. [cited 20 June 2006]. Available at <http://isgec.org/>.

[Geo01] P. S. Georgilakis, N. D. Doulamis, A. D. Doulamis, N. D. Hatziargyriou, and S. D. Kollias, “A novel iron loss reduction technique for distribution transformers based on a combined genetic algorithm - neural network approach,” *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 31, no. 1, 2001, pp. 16-34.

[Gho98] A. Ghosh, S. Tsutsui, and H. Tanaka, “Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals,” in *Proc. of the IEEE International Conference on Evolutionary Computation*, Anchorage, AK, 1998, pp. 666-671.

[Gir02] R. Giro, M. Cyrillo, and D. S. Galvão, “Designing conducting polymers using genetic algorithms,” *Chemical Physics Letters*, vol. 366, no. 1-2, 2002, pp. 170-175.

[Gol89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA: Kluwer Academic Publishers, 1989.

[Gor93] V. S. Gordon and D. Whitley, “Serial and parallel genetic algorithms as function optimizers,” in *Proc. of the 5th International Conference on Genetic Algorithms*, San Mateo, CA, 1993, pp. 177-183.

[Gre81] J. J. Grefenstette, “Parallel adaptive algorithms for function optimization,” *Technical report CS-81-19*, Vanderbilt University, Nashville, TN, 1981.

[Gre85] J. J. Grefenstette, R. Gopal, R. Rosmaita, and D. Gucht, “Genetic algorithms for the traveling salesman problem,” in *Proc. of the 2nd International Conference on Genetic Algorithms*, Lawrence Eribaum Associates, Mahwah, NJ, 1985, pp. 160-168.

[Gre87] J. J. Grefenstette, “Incorporating problem-specific knowledge into genetic algorithms,” in L. D. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, London, UK: Pitman, 1987, pp. 42-60.

[Han97a] H. Handa, N. Baba, O. Katai, T. Sawaragi, and T. Horiuchi, “Genetic algorithm involving coevolution mechanism to search for effective genetic information,” in *Proc. of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, 1997, pp. 709-714.

[Han97b] S.-S. Han and G. S. May, “Using neural network process models to perform PECVD silicon dioxide recipe synthesis via genetic algorithms,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, no. 2, 1997, pp. 279-287.

[Hau98] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. New York, NY: John Wiley & Sons, 1998.

[Hau00] R. L. Haupt and S. E. Haupt, “The creative use of genetic algorithms. Computers evolve into the artistic realm,” *IEEE Potentials*, vol. 19, no. 2, 2000, pp. 26-29.

[Hau04] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. 2nd edition. New York, NY: John Wiley & Sons, 2004.

[Hay98] S. Haykin, *Neural Networks: A Comprehensive Foundation*. 2nd edition. Upper Saddle River, NJ: Prentice Hall, 1998.

[Hec01] P. S. Heck and S. Ghosh, “A study of synthetic creativity through behavior modeling and simulation of an ant colony,” in *Proc. of the 5th International Symposium on Autonomous Decentralized Systems*, Dallas, TX, 2001, pp. 391-397.

[Her00] F. Herrera and M. Lozano, “Gradual distributed real-coded genetic algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, 2000, pp. 43-63.

[Hin97] R. Hinterding, Z. Michalewicz, and A. E. Eiben, “Adaptation in evolutionary computation: a survey,” in *Proc. of the 4th International Conference on Evolutionary Computation*, Indianapolis, IN, 1997, pp. 65-69.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press, 1975.

[Hub98] A. Huber and D. A. Mlynski, “An age-controlled evolutionary algorithm for optimization problems in physical layout,” in *Proc. of 1998 IEEE International Symposium on Circuits and Systems*, vol. 6, 1998, pp. 262-265.

[Ife93] E. C. Ifeacor and B. W. Jervis, *Digital Signal Processing – a Practical Approach*. Boston, MA: Addison–Wesley Publishers, 1993.

[Iwa02] M. Iwashita and H. Iba, “Island model GP with immigrants aging and depth-dependent crossover,” in *Proc. of the Congress on Evolutionary Computation*, vol. 1, Hawaii, HI, 2002, pp. 267-272.

[Jin05] Y. Jin and J. Branke, “Evolutionary optimization in uncertain environments – a survey,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, 2005, pp. 303-317.

[Ji06] Z. Ji, D. Dasgupta, Z. Yang, and H. Teng, “Analysis of dental images using artificial immune systems,” in *Proc. of the IEEE Congress on Evolutionary Computation*, Vancouver, BC, 2006, pp. 1635-1642.

[Jon00] S. Jones, *Almost Like a Whale*. London, UK: Anchor, 2000.

[Jua05] C.-F. Juang, “Genetic recurrent fuzzy system by coevolutionary computation with divide-and-conquer techniques,” *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 35, no. 2, 2005, pp. 249-254.

[Jul95] B. A. Julstrom, “What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm,” in *Proc. of the 6th International conference on Genetic Algorithms*, Pittsburgh, PA, 1995, pp. 81-87.

[Kam83] R. R. Kampfner and M. Conrad, “Computational modeling of evolutionary learning process in the brain,” *Bulletin of Mathematical Biology*, vol. 45, no. 6, 1983, pp. 931-968.

[Kam04] A. Kamiya, “General model for a large-scale plant application,” in S. J. Ovaska (ed.), *Computationally Intelligent Hybrid Systems: The Fusion of Soft Computing and Hard Computing.* Hoboken, NJ: Wiley-IEEE Press, 2004, pp. 35-51.

[Kam05] A. Kamiya, F. Makino, and S. Kobayashi, “Worker ants’ rule-based genetic algorithms dealing with changing environments,” in *Proc. of the IEEE Midnight-Sun Workshop on Soft Computing in Industrial Applications*, Espoo, Finland, 2005, pp. 117-121.

[Kau67] H. Kaufman, “An experimental investigation of process identification by competitive evolution,” *IEEE Transactions on Systems Science and Cybernetics*, vol. SSC-3, no. 1, 1967, pp. 11-16.

[Ken95] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proc. of the IEEE International Conference on Neural Networks*, Perth, Australia, vol. 4, 1995, pp. 1942-1948.

[Kew02] R. H. Kewley and M. J. Embrechts, “Computational military tactical planning system,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C – Applications and Reviews*, vol. 32, no. 2, 2002, pp. 161-171.

[Kou06] V. K. Koumousis, C. P. Katsaras, “A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, 2006, pp. 19-28.

[Kov01] T. Kovacs, “What should a classifier system learn?,” in *Proc. of the Congress on Evolutionary Computation*, vol. 2, Seoul, Korea, 2001, pp. 775-782.

[Koz99] J. Koza, F. Bennett, D. Andre, and M. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[Koz06] Genetic Programming Inc, [WWW page]. Maint. J. R. Koza, [cited 21 June 2006]. Available at <http://www.genetic-programming.com/>.

[Lam04] H. K. Lam and F. H. F. Leung, "Digit and command interpretation for electronic book using neural network and genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 34, no. 6, 2004, pp. 2273-2283.

[Leu03] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, 2003, pp. 79-88.

[Liu05] Y.-H. Liu, J.-H. Teng, and Y.-C. Lin, "Search for an optimal rapid charging pattern for lithium-ion batteries using ant colony system algorithm," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 5, 2005, pp. 1328-1336.

[Mag00] G. Magyar, M. Johansson, and O. Nevalainen, "An adaptive hybrid genetic algorithm for the three-matching problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, 2000, pp. 135-146.

[Mah96] S. Mahfoud and G. Mani, "Financial forecasting using genetic algorithms," *Applied Artificial Intelligence*, vol. 10, no. 6, 1996, pp. 543-565.

[Mat06a] Mathworks Inc., "The homepage of Matlab® Software," [WWW page]. [cited 20 June 2006]. Available at <http://www.matlab.com/>.

[Mcc97] S. McClintock, T. Lunney, and A. Hashim, "A fuzzy logic controlled genetic algorithm environment," in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, Orlando, FL, 1997, pp. 2181-2186.

[Mel05] N. Melab, M. Mezmaz, and E.-G. Talbi, "Parallel hybrid multi-objective island model in peer-to-peer environment," in *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO, 2005, pp. 1-9.

[Mil90] J. S. Milton and J. C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Science*. New York, NY: McGraw-Hill, 1990.

[Meu00] H. Meunier, E. G. Talbi, and P. Reininger, "A multiobjective genetic algorithm for radio network optimization," in *Proc. of the IEEE Congress on Evolutionary Computation*, San Diego, CA, 2000, pp. 317-324.

[Mic96] Z. Michalewicz, *Datastructures + Genetic Algorithms = Evolution Programs*. Berlin: Springer, 1996.

[Mit96] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[Miz00] E. Mizutani, H. Takagi, D. M. Auslander, and J.-S. R. Jang, "Evolving color recipes," *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 30, no. 4, 2000, pp. 537-550.

[Mor95] C. Moraga and R. Salas, "A new aspect for the optimization of fuzzy if-then rules," in *Proc. of the 35th International Symposium on Multiple-Valued Logic*, Galgary, Canada, 2005, pp. 160-165.

[Mur95] T. Murata and H. Ishibuchi, "Adjusting membership functions of fuzzy classification rules by genetic algorithms," in *Proc. of the International Joint Conference of the 4th IEEE International Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium*, vol. 4, Yokohama, Japan, 1995, pp. 1819-1824.

[Müh92] H. Mühlenbein, "Parallel genetic algorithms in combinatorial optimization," *Computer Science and Operations Research*, O. Balchi, R. Sharda, and S. Zenios (eds.), New York, NY: Pergamon, 1992, pp. 441-456.

[Nag04] Y. Nagata, "Criteria for designing crossovers for TSP," in *Proc. of the Congress on Evolutionary Computation*, vol. 2, Portland, OR, 2004, pp. 1465-1472.

[Oba00] S. Obayashi, D. Sasaki, Y. Takeguchi, and N. Hirose, "Multiobjective evolutionary computation for supersonic wing-shape optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, 2000, pp. 182-187.

[Ong04] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, 2004.

[Ova99] S. J. Ovaska, Y. Dote, T. Furuhashi, A. Kamiya, and H. F. VanLandingham, "Fusion of soft computing and hard computing techniques: a review of applications," in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, Tokyo, Japan, 1999, pp. 370-375.

[Ova04] S. J. Ovaska (ed.), *Computationally Intelligent Hybrid Systems: The Fusion of Soft Computing and Hard Computing*. Hoboken, NJ: Wiley-IEEE Press, 2004.

[Par02] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, 2002, pp. 321-332.

[Pet98] V. Petridis, E. Paterakis, and A. Kehagias, “A hybrid neural-genetic multimodel parameter estimation algorithm,” *IEEE Transactions on Neural Networks*, vol. 9, no. 5, 1998, pp. 862-876.

[Pol05] K. S. Pollard, S. Dudoit, and M. J. van der Laan, “Multiple testing procedures: The Multtest package and applications to genomics,” in *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, R. C. Gentleman, V. J. Carey, W. Huber, R. Irizarry, and S. Duduoit (eds.), Springer-Verlag, New York, NY, 2005, pp. 249-271.

[Psa88] H. N. Psaraftis, “Dynamic vehicle routing problems,” in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds.), Elsevier Science Publishers, Amsterdam, Holland, 1988, pp. 223-248.

[Pyt04] K. Pytel, G. Kluka, and A. Szymonik, “Fuzzy methods of driving genetic algorithms,” in *Proc. of the 4th International Workshop on Robot Motion and Control*, Puszczykowo, Poland, 2004, pp. 339-343.

[Rao78] S. S. Rao, *Optimization Theory and Applications*. New Delhi, India: Wiley Eastern Limited, 1978.

[Rao96] S. S. Rao and K. Chellapilla, “Design of discrete coefficient FIR filters using fast simulated evolutionary optimization,” in *Proc. of the International Conference on Neural Networks*, vol. 2, Washington, DC, 1996, pp. 1185-1190.

[Rec65] I. Rechenberg, “Cybernetic solution path of an experimental problem,” *Royal Aircraft Establishment*, Library translation no. 1122, Farnborough Hants, UK, 1965.

[Rey06] M. Reyes-Sierra and C. A. Coello, “Multi-objective particle swarm optimizers: a survey of the state-of-the-art,” *International Journal of Computational Intelligence Research*, vol. 2, no. 3, 2006, pp. 287-308.

[Sal98] R. Salomon, “Evolutionary Algorithms and Gradient search: Similarities and Differences,” *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 2, 1998, pp. 45-55.

[Sam93] M. Sambridge and K. Gallagher, “Earthquake hypocenter location using genetic algorithms,” *Bulletin of the Seismological Society of America*, vol. 83, no. 5, 1993, pp. 1467-1491.

[Sar98] B. Sareni and L. Krahenbuhl, “Fitness sharing and niching methods revisited,” *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 2, 1998, pp. 97-106.

[Sat00] S. Sato, K. Otori, A. Takizawa, H. Sakai, Y. Ando, and H. Kawamura, “Applying genetic algorithms to the optimum design of a concert hall,” *Journal of Sound and Vibration*, vol. 258, no. 3, 2002 pp. 517-526.

[Sch65] H.-P. Schwefel, “Evolutionsstrategie und numerische Optimierung,” *dissertation,* Technische Universität Berlin, Germany, 1975.

[Sch00] J. Schonberger, D. C. Mattfeld, and H. Kopfer, “Automated timetable generation for rounds of a table-tennis league,” in *Proc. of the 2000 Congress on Evolutionary Computation*, vol. 1, Vancouver, BC, 2000, pp. 277-284.

[Sic98] B. Sick, M. Keidl, M. Ramsauer, and S. Seltzsam, “A comparison of traditional and soft-computing methods in a real-time control application,” in *Proc. of the 8th International Conference on Artificial Neural Networks*, 1998, Skövde, Sweden, pp. 725-730.

[Sla99] V. Slavov and N. Nikolaev, “Genetic algorithms, sublandscapes and subpopulations,” in W. Banzhaf and C. Reeves (eds.), *Foundations of Genetic Algorithms 5*, Morgan Kaufmann, San Francisco, CA, 1999.

[Smi96] J. Smith and T. Fogarty, “Self-adaptation of mutation rates in a steady-state genetic algorithm,” in *Proc. of the 3rd IEEE International Conference on Evolutionary Computation*, 1996, pp. 318-323.

[Smi97] J. Smith and T. Fogarty, “Operator and parameter adaptation in genetic algorithms,” *Soft Computing*, vol. 1, no. 2, 1997, pp. 81-87.

[Sri94] M. Srinivas and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, 1994, pp. 656-667.

[Sub03] R. Subbu and P. P. Bonissone, “A retrospective view of fuzzy control of evolutionary algorithm resources,” in *Proc. of the 12th IEEE International Conference on Fuzzy Systems*, vol. 1, St. Louis, MO, 2003, pp. 143-148.

[Tak93] H. Takagi, “Fusion techniques of fuzzy systems and neural networks, and fuzzy systems and genetic algorithms,” in *Proc. of the SPIE’s International Symposium on Optical Tools for Manufacturing and Advanced Automation*, vol. 2061, Boston, MA, 1993, pp. 402-413.

[Tak01] H. Takagi, “Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation,” in *Proc. of the IEEE*, vol. 89, no. 9, 2001, pp. 1275-1296.

[Tan87] R. Tanese, “Parallel genetic algorithms for a hypercube,” in *Proc. of the 2nd International Conference on Genetic Algorithms*, Cambridge, MA, 1987, pp. 177-183.

[Tec06] IEEE Computational Intelligence Society, “IEEE Transactions on Evolutionary Computation,” [WWW page]. [cited 20 June 2006]. Available at <http://www.ieee-nns.org/pubs/tec/>.

[Tsa06] J.-T. Tsai, J.-H. Chou, and T.-K. Liu, “Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm,” *IEEE Transactions on Neural Networks*, vol. 17, no. 1, 2006, pp. 69-80.

[Une03] M. Unehara and T. Onisawa, “Music composition system based on subjective evaluation,” in *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, Washington, DC, 2003, pp. 980-986.

[Val95] C. L. Valenzuela and A. J. Jones, “A parallel implementation of evolutionary divide and conquer for the TSP,” in *Proc. of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sheffield, UK, 1995, pp. 499-504.

[Wan96a] L.-X. Wang, *A Course in Fuzzy Systems and Control*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[Wan96b] P. Y. Wang, G. S. Wang, Y. H. Song, and A. T. Johns, “Fuzzy logic controlled genetic algorithms,” in *Proc. of the 5th IEEE International Conference on Fuzzy Systems*, vol. 2, New Orleans, LA, 1996, pp. 972-979.

[Wan03] W.-Y. Wang and Y.-H. Li, “Evolutionary learning of BMF fuzzy-neural networks using a reduced-form genetic algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, vol. 33, no. 6, 2003, pp. 966-976.

[Wan05] W. Wang, L. Yilong, J. S. Fu, and Y. Z. Xiong, “Particle swarm optimization and finite-element based approach for microwave filter design,” *IEEE Transactions on Magnetics*, vol. 41, no. 5, 2005, pp. 1800-1803.

[Wer00] J. Werfel, M. Mitchell, and J. P. Crutchfield, “Resource sharing and coevolution in evolving cellular automata,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, 2000, pp. 388-393.

[Wil03] L. Willmes, T. Bäck, J. Yaochu, and B. Sendhoff, “Comparing neural networks and Kringing for fitness approximation in evolutionary optimization,” in *Proc. of the Congress on Evolutionary Computation*, vol. 1, Canberra, Australia, 2003, pp. 663-670.

[Whi01] D. Whitley, “An overview of evolutionary algorithms: practical issues and common pitfalls,” *Information and Software Technology*, vol. 43, 2001, pp. 817-831.

[Wie05] K. C. Wiese, A. Hendriks, A. Deschenes, and B. B. Youssef, “P-RnaPredict-a parallel evolutionary algorithm for RNA folding: effects of pseudorandom number quality," *IEEE Transactions on NanoBioscience*, vol. 4, no. 3, 2005, pp. 219-227.

[Wol97] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 67-82.

[Yon02] F. Yong, J. Tianzi, and D. J. Evans, "Volumetric segmentation of brain images using parallel genetic algorithms," *IEEE Transactions on Medical Imaging*, vol. 21, no. 8, 2002, pp. 904-909.

[Yuh99] S. Yuhui, "Combinations of evolutionary algorithms and fuzzy systems: a survey," in *Proc. of the 18th International Conference of the North American Fuzzy Information Processing Society*, 1999, New York, NY, pp. 610-614.

[Zad97] L. A. Zadeh, "What is soft computing?," *Soft Computing*, vol. 1, no. 1, 1997, pp. 1.

[Zad02] L. A. Zadeh, "From computing with numbers to computing with words: from manipulation of measurements to manipulation of perceptions," in *International Journal of Applied Mathematics and Computer Science*, vol. 12, no. 3, 2002, pp. 307-324.

ISBN-13 978-951-22-8523-5
ISBN-10 951-22-8523-1
ISBN-13 978-951-22-8524-2 (PDF)
ISBN-10 951-22-8524-X (PDF)
ISSN 1795-2239
ISSN 1795-4584 (PDF)