EVOLUTIONARY PROBLEM SOLVING

Thomas Philip Runarsson

Faculty of Engineering
Reykjavik 2000

ISBN 9979-9494-1-4

Háskólaútgáfan
University of Iceland

This thesis has been accepted by the Faculty of Engineering of the University of Iceland for public defence in fulfilment of the requirements for the degree of doctor scientiarum ingeniarius.

Faculty of Engineering
Reykjavik, 29th January 2001
Valdimar K. Jonsson, Dean

Thesis Committee:

Professor Magnus Thor Jonsson
Professor Xin Yao
Professor Hans-Paul Schwefel

Preface

This thesis on evolutionary problem solving is a summation of work accomplished over a period of four years. My interest in this field was aroused when unintentionally attending an IEEE workshop on genetic algorithms in 1994.

There are numerous people to thank for making this work possible: First of all Magnus Thor Jonsson for his encouragement, and creating the liberty to pursue this area of research. Xin Yao for sharing his expertise and guidance. Einar Arnason for endless discussions on evolutionary processes and natural systems. John Stuart for his patience in discussing the nature of representations. Gudmundur R. Jonsson for helping with difficult statistics. Jörgen Pind for renewing my interest in connectionist models. Mikael Karlson for making me doubt everything. Ruhul Sarker for generating interest in constrained problems. Hans-Paul Schwefel and David B. Fogel for sharing their insight into the working principles of evolutionary computation. Hans-George Beyer for discussions on theories for evolution strategies. Richard Sutton for pointing out the problems with evolutionary computation. Finally, I thank the Department of Mechanical Engineering and my many friends there for creating an invigorating academic environment.

Financial support for this work was provided by the Research Council of Iceland and the Research Fund of the University of Iceland. Travel grants were awarded by the Engineering Labour Union and the University of Iceland. Many thanks to the Department of Computer Science at the Australian Defence Force Academy for hosting me in Canberra, in particular Bob McKay and Charles Newton.

Thomas Philip Runarsson

Faculty of Engineering, University of Iceland
Reykjavik, December 2000.

Abstract

The work presented is a study of a problem solving technique inspired by the principles of biological evolution. Solving a problem is the transformation of a given situation into a desired one. The work may be divided into three main interdependent areas of research: (1) *describing a situation*, (2) operator design for *changing a situation*, and (3) *evaluation* by selection. Evolutionary problem solvers are inspired from a model of natural evolutionary processes where organisms are 'solutions' to 'problems' set up by the environment. Therefore, how an organism and its environment are represented on a computational device is of central interest in the work.

Experimental studies and examples are given illustrating the use of symbolic and connectionist representations in evolutionary problem solving. The plausibility that the syntactical manipulation of symbols is an effective and efficient approach to evolutionary search is questioned. For 'intelligent' search operators must essentially infer the location of better solutions and for this the symbols must be transformed based on their meaning.

It is often difficult to interpret how evolutionary search methods locate good solutions. If a landscape is described by the parent position and the expected fitness of the best offspring, then its topology will determine when evolutionary search locates on average the desired goal. An interpretation of this landscape assists in the design of variation operators and in determining the number of offspring required. The analysis is also useful for the understanding of a new rank-based selection method presented for the treatment of infeasible solutions.

Ágrip

Í verkefni þessu er rannsökuð aðferð við lausn vandamála, sem byggir á lögmálum líffræðilegrar þróunar. Lausn vandamála jafngildir því að varpa gefnu ástandi yfir í óskað ástand. Skipta má verkefninu í þrjú háð rannsóknarsvið: (1) *lýsing á ástandi*, (2) gerð vörpunarvirkja fyrir *breytingu ástands* og (3) *mat* með vali. Lausnaraðferðir byggðar á þróun eru innblásnar af líkani fyrir náttúruleg þróunarferli þar sem lífverur eru 'lausnir' á 'vandamálum' sem verða til í umhverfinu. Túlkun lífveru og umhverfis hennar með aðstoð tölvu er því eitt af viðfangsefnum ritgerðarinnar.

Gerð er grein fyrir prófunum á aðferðum og leyst eru dæmi, með það að markmiði að sýna notkun táknrænna og netrænna útsetninga í þróunaralgrímum. Einnig eru dregnir í efa möguleikar táknbundinnar framsetningar sem alhliða lausnaraðferðar. Því ef um vitræna leitun er að ræða, þá eiga virkjar að leiða af sér ályktun um betri stöðu lausna og þess vegna verður vörpun tákna að byggja á þýðingu þeirra.

Oft er óljóst á hvern hátt þróunarleitaraðferðir finna góðar lausnir. Ef landslag er ákvarðað út frá stöðu foreldris ásamt væntanlegri hæfni valinna afkvæma, þá mun lögun landslagsins ákvarða hvenær leitaraðferð mun að meðaltali finna æskilega lausn. Túlkun landslagsins gefur upplýsingar sem nýtast við hönnun á virkjum ásamt ákvörðun um nauðsynlegan fjölda afkvæma. Þetta er einnig hægt að nota til að auka skilning á nýrri valaðferð, sem hefur verið þróuð hér fyrir bæði ólínuleg og skorðuð bestunarvandamál.

Contents

Chapter 1

Introduction

Solving a problem is the transformation of a given situation into a desired situation or goal [Hay81]. A solution exists in the form of a sequence of actions necessary to achieve this goal and the term 'problem' refers to a task specified by a set of actions, a goal, an initial state, and a set of reachable states. The work presented is committed to the design of efficient probabilistic problem solving strategies. The technique is referred to as 'evolutionary problem solving' because its strategy is inspired from principles of biological evolution. Nature has solved many technical problems and mankind has imitated some of its solutions. By harnessing these principles, additional technical problems may potentially be solved.

If a problem domain has strong mathematical structure, strategies may exist that consistently locate an optimal solution in acceptable computation time. The development of novel strategies is challenging but often time consuming. When traditional techniques are applied, the conditions are simplified and may be a far way off from the real problem. In real world problems, objectives and constraints may not be analytically treatable, noisy, non-stationary, etc. For these reasons, evolutionary problem solving may be an attractive alternative. All that is needed is a data structure to represent a solution, a means of generating a variation of that structure, and the ability to determine whether one data structure is better than another. This minimalist approach, although inefficient at times, may just be the sensible all round method. Nature has solved numerous 'problems' and one may be inclined to infer that its problem solving strategy is in some sense optimal. However, these hopes are dashed by the so called *no free lunch theorems* [WM97]. According to the

theorems, the average time taken to seek a goal for any pair of search algorithms, which do not resample points, across all possible problems, is identical. Essentially, if nothing is known about the problem, there is no justification for selecting any particular search algorithm [MF00, p. 161].

1.1 Motivation

If the principles of biological evolution are to be used as a problem solving paradigm, its relation to human problem solving must be understood. It is in the end the human designer who sets up the description, operators, and evaluation criteria. Furthermore, the implementation of evolutionary problem solving on a computational device may also introduce limitations. Evolutionary algorithms are inspired from a model of natural evolutionary processes where organisms are 'solutions' to 'problems' set up by the environment. It is fair to say that most evolutionary algorithms assume that the environment, the problem, is the criterion for survival, which is constant over time [Sch95, p. 106]. For living systems, however, there is a *coevolution* of organism and environment [Lew00, p. 101] in which both are acting as both, the problem and solution. By isolating the problem from the solution, useful structure is lost that may have made problem solving more efficient and effective. The situation description, common to dynamic programming [Ber95] and reinforcement learning [SB98, BT96], is perhaps in this case a better model of natural processes. These are discrete-time dynamic systems whose state transition (environment) depends on a control (organism), which again depends on the current state. The fitness is then an accumulated cost that depends on the states visited.

Solving problems is not about random search in the problem-solution space. The space is a frame in which the designer brings knowledge and when sufficient the search is unnecessary. An 'intelligent' problem solver will not limit itself to direct search but also to knowledge search [New90, p. 98]. This is the search for knowledge that guides the search. A living system must also necessarily know something in so far as it can act in the world. In other words, organisms are in some sense *cognitive*. Similarly, problem solvers may be made 'cognitive' by the insertion of specific knowledge, either by hand or as a result of automatic learning. In evolutionary problem solving, the knowledge of *how to solve* a problem is optimized. The evolutionary algorithm that

allows for the greatest amount of automatic learning, will be the more 'intelligent' problem solver. Domain knowledge may be optimized not only for a single problem instance, but also for the problem class. The search for domain knowledge is also subject to the no free lunch theorems and therefore, it becomes necessary to know something about the cognitive architecture chosen. What is ultimately of interest is efficiently finding effective solutions to specific problem classes. An evolutionary problem solver may require a longer time optimizing the knowledge needed for a given problem class and when found any case may be solved instantaneously. An example of this is where learning rules are evolved for the training of neural networks [RJ00]. For natural systems, however, the search for knowledge is never ending because 'the problem' is forever changing. Understanding the nature of cognition, and hence how to search a knowledge space, is the key to designing universal evolutionary problem solvers.

1.2 Research Objectives and Contributions

The overall objective of the work is to gain a greater understanding of what makes an evolutionary problem solver effective and efficient. The problem solver is effective in the sense that it will on average locate near optimal solutions and doing so efficiently implies with little computational effort. The investigation may be broken down to three main interdependent areas of research: *Describing a situation*, operator design for *changing a situation*, and *evaluation* by selection. An attempt will be made to contribute an increased understanding in each of the respective areas. The following list describes the primary objectives of the research work presented and corresponding contributions.

1. 'Intelligent' problem-solution description

The object is to develop a more rigorous methodology for setting up problem-solution architectures for evolutionary algorithms. Specific knowledge must be utilized for effective and efficient search. How this knowledge is embodied will depend on the architecture chosen. The main contribution here lies in the recognition that a problem-solution description is equivalent to *knowledge representation*. The most commonly cited architectures for knowledge repre-

sentation are: *symbolic* and *connectionist*. The most suitable architecture is, however, dependent on the task at hand. It must be easy and transparent for the designer to specify the domain knowledge. The representation must be adequate, e.g. span the entire portion of the world that is of interest. The search for alternative solutions depends on what is represented. The efficiency of the search for alternatives relies on how inferential the representation is. The most desirable search operator will have the ability to infer the most probable location of better solutions.

The domain knowledge should be simple to modify to allow for automatic learning. A problem-solution description, which permits learning, will generate the more 'intelligent' problem solver. There will be a price paid in computation time to acquire domain knowledge, but as long as this knowledge is reused, the problem and further instances thereof may be solved in a shorter time. It will be argued that neuro-evolutionary methods are well suited for this task. The benefits of using connectionist models for knowledge representation is that there exists some idea of how to search the knowledge space. This is not the case for a symbolic representation, unless the designer supplies this knowledge. Two new neuro-evolutionary models are presented to illustrate how evolutionary problem solving can be made more 'intelligent'. This is an attempt to break free from the conventional approaches to evolutionary computation and where all knowledge must be pre-specified by the designer. This is the case for *any* symbolic representation. The first architecture presented is a *choice network* that deliberates over a number of plausible actions to be taken for a given environmental state. An application of this model is presented where dispatch scheduling rules are evolved. The second architecture is an example of a reactive connectionist model where the model's output is its action. This model is used to evolve training rules for neural networks. In this case, the learning rule becomes part of the neural network architecture.

2. **Efficient and effective evolutionary search**

It is important to understand when evolutionary problem solving may locate an optimum in reasonable time. Every data structure can be assigned a 'fitness'. This is analogous to the idea in biology that every genotype has a fitness. In evolutionary biology fitness is often thought of as a function of the genotype frequency in the population [Wri31]. The distribution of fitness over

the space of genotypes constitutes a *fitness landscape*. In natural systems, this landscape deforms in response to changes in the environment and in response to coevolution [Kau93]. In evolutionary problem solving, the problem-solution space is predefined and, therefore, the fitness landscape is fixed in structure. It is intuitive to assume that the topology of the fitness landscape will determine problem difficulty. This means, landscapes that are multipeaked and rugged are difficult and those with few smooth peaks will be easy to search. Here it is implied that evolutionary search is a hill climbing strategy. The climbing is performed by the search operators. This leads to the notion of *operator landscapes* [JF95] where neighbours are considered in *operator space*. Any fitness landscape topology depends ultimately on how the distances between genotypes are defined. This is known as the *distance metric* space. As long as the search operators work in this metric space, the evolutionary search may perhaps be visualized as a hill climber. However, to complete the landscape description, the influence of *selection* must be considered. The result is yet another landscape concept, which will be called the *expected fitness landscape*. In this case, neighbourhoods are determined by the parent individuals, and the fitness by the expected fitness of their best offspring.

It will be argued that the efficiency and effectiveness of the evolutionary search is dependent on the topology of the expected fitness landscape. This is the case where the fitness landscape, which may be rugged and multipeaked, is transformed to a smooth singlepeaked expected fitness landscape. Conditions are established for when at least linear convergence rates to the global optimum may be expected in general. The approach taken gives a greater insight to the general working mechanisms of evolutionary search, which is otherwise a difficult theoretical problem.

3. **Constraint handling technique**

All real world problems have constraints. If solutions are set up so that constraints are not 'naturally' satisfied by the description and operators, they must be treated by selection. This introduces the dilemma of finding a proper fitness measure for feasible and infeasible individuals. Currently this is an open question. Due to the frequent appearance of constraints in real world problem, it is vital to investigate their handling further.

Constraints will often transform a relatively well behaved fitness landscape

to a rugged and even disjointed landscape. Functions, which may have been easy to optimize, become difficult. By establishing a proper fitness measure for feasible and infeasible individuals it may be possible to maintain a smoother landscape and make the traversing of disjointed regions easier. A new approach, called *stochastic ranking*, is presented, which attempts to accomplish this goal. The method is the result of a new way of interpreting penalty function methods.

1.3 Overview

Chapter 2 presents a literature review and necessary background in the area of problem solving and evolutionary computation. Stepwise methodologies to human problem solving are related to the stepwise approach to designing evolutionary algorithms. Common evolutionary algorithms for nonlinear programming are described and in this context, constraint handling, local convergence rates, and global convergence are discussed. Furthermore, a brief introduction to connectionist models is given as they will become the prime means of problem-solution representation.

In chapter 3 procedures for abstracting problems and their solutions for evolutionary search are established. This is discussed in terms of an organism and its environment. Two world models are presented, in one the organism is isolated from its environment, and in the other they are treated as a whole. These worlds may be represented using symbolic or connectionist models. A detailed example is given to emphasize the difference between symbolic and connectionist representations.

Further developments of connectionist models for problem and solution representation are discussed in chapter 4. These models are separated into two distinct classes: deliberative and reactive connectionist architectures. The design of a choice network illustrates the application of a deliberative connectionist problem solving architecture. The model is used to evolve adaptive solutions to scheduling problems. The technique is compared with an evolutionary approach using a symbolic representation. An attempt is made to make the symbolic approach adaptive by introducing learnable alleles. For their implementation two learning techniques are examined: Lamarckian learning and the Baldwin effect. The reactive connectionist problem solving architecture is illustrated by evolving training rules for the optimization of linear neural

networks.

In chapter 5 an effort is made to gain a greater understanding of how evolutionary search works. This is aided by the examination of expected fitness landscapes. It will be argued that this new landscape concept is more suitable for the determination of problem difficulty than conventional fitness landscapes. The role of recombination in mutative step size control is examined empirically. Directed and hopeful monster mutations are also discussed and investigated. Twenty-four commonly used benchmark functions for unconstrained nonlinear programming are studied.

The problem-solution abstracted may sometimes be unable to 'naturally' satisfy all of the constraints set and illegals can only be weeded out by selection. This introduces a dilemma of how to assign fitness to illegal individuals. The ranking procedure developed in chapter 6 illustrates how this may be accomplished. In order to understand how constraint handling techniques function, it is important to know how evolutionary search works. The method presented may be understood in terms of the concepts developed in the previous chapter 5.

Finally, conclusions are presented in chapter 7. This includes a discussion of contributions and directions for future research.

Chapter 2

Background and Literature

In this chapter, theoretical and practical background literature to evolutionary algorithms and its relation to problem solving is described. A detailed account of the evolutionary algorithm applied in the work for numerical optimization problems is given. Local convergence rate theory for these problems is presented but may be generalized to other problem classes, provided a distance metric is defined. Global convergence theorems for evolutionary algorithms are also presented. Finally, a brief overview of connectionist models is provided, as they are frequently used for knowledge representation in this work.

2.1 Problem Solving

To solve a problem there must preexist a *description* of the situation, *operators* for changing the situation, and an *evaluation* to determine whether the goal has been achieved. The prescriptive of how a problem solver should proceed as described by Polya [Pol57], is given by four phases: (1) *understanding the problem*; to see clearly what is required, (2) *devising a plan*; to see how the various items are connected and how the unknowns are linked to the data in order to obtain an idea of the solution, (3) *carrying out the plan*, and (4) *looking back*; review the completed solution. Similar steps have been proposed by other authors [Hay81, BS84] but they usually include an explicit step for finding a representation of the problem. The characteristic sequence of problem solving noticed by Hayes [Hay81] is: (1) *finding the problem*; recognizing that there is a problem to be solved, (2) *representing the problem*;

understanding the nature of the gap to be crossed, (3) *planning the solution*; choosing a method for crossing the gap, (4) *carrying out the plan*, (5) *evaluating the solution*; asking "how good is the result?" once the plan is carried out, and (6) *consolidating gains*; learning from the experience of solving. Here Hayes breaks Polya's *looking back* down into two components, the one focused on evaluating the immediate problem-solving effort and the other on learning something that may be useful in the future.

Similar problem solving strategies may also be identified in engineering design [Hub87]. Furthermore, one may recognize the typical steps of an evolutionary algorithm [Hol75, Sch95, Fog95]. In particular, Hayes' steps may be reinterpreted as shown below in figure 2.1.

problem-solution abstraction

1. *identifying the problem*; understanding and finding the problem
2. *representation*; design a data structure for the problem-solution

evolutionary search

3. *variation (initialization)*; devise a plan or modify an existing one
4. *genotype to phenotype mapping*; carry out the plan
5. *selection (fitness evaluation)*; evaluating its utility
6. *self-adaptation*; learning from the experience of solving

Figure 2.1: A stepwise approach to evolutionary problem solving

If the desired solution is not found, the process is repeated from step 3. In some cases it may be necessary to review steps 1 and 2. Perhaps the difference between human problem solving and evolutionary problem solving is essentially that a probabilistic operator is used to modify a given situation. Being a probabilistic process, it must necessarily be repeated a number of times for success. The best modified situation is kept and the whole process repeated. Evolutionary search, therefore, relies on multiple individuals engaged in processing the steps presented where the worst situations, relative to

the ones currently available, are abandoned and the better ones are replicated more than once. The probabilistic operator for changing the situation is predefined and will reflect the designer's understanding of where better solutions are most likely to be found for any given situation.

In the standard text on the application of evolutionary algorithms, see for example [GC97, MF00], it is assumed that, the problem-solution is abstracted and its representation is left to the designer's 'intuition'. This exercise is probably the most challenging and creative aspect of problem solving. The implementation and theory of an evolutionary algorithm is therefore only concerned with the last four steps of problem solving illustrated in figure 2.1, i.e. that of search. There exists, however, numerous approaches to describing a problem-solution. The description chosen will clearly influence the search for alternatives. It is reasonable to inquire how the search for alternatives relates to the problem-solution description. Indeed the situation description and the operator for changing the situation should be co-determined [MF00]. In chapter 3 an attempt will be made to generate a better understanding of the properties of problem-solution descriptions and the probabilistic operators that variate them. The following section describes some theory and application of evolutionary algorithms for nonlinear programming.

2.2 Nonlinear Programming

The general nonlinear programming problem can be formulated as solving the *objective function*

$$\text{minimize} \quad f(\mathbf{x}), \quad \mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{R}^n \tag{2.1}$$

where $\mathbf{x} \in \mathcal{S} \cap \mathcal{F}$, $\mathcal{S} \subseteq \mathcal{R}^n$ defines the *search space* which is a n-dimensional space bounded by the *parametric constraints*

$$\underline{x}_j \leq x_j \leq \overline{x}_j, \quad j \in \{1, \ldots, n\} \tag{2.2}$$

and the *feasible region* $\mathcal{F}$ is defined by

$$\mathcal{F} = \{\mathbf{x} \in \mathcal{R}^n \mid \mathrm{g}_k(\mathbf{x}) \leq 0 \; \forall \; k \in \{1, \ldots, m\}\}, \tag{2.3}$$

where $\mathrm{g}_k(\mathbf{x}), k \in \{1, \ldots, m\}$, are the inequality *constraints*. Only minimization problems need be considered without a loss of generality since $\max\{f(\mathbf{x})\} = -\min\{-f(\mathbf{x})\}$.

2.2.1 Evolutionary Algorithm

The most commonly known evolutionary algorithms for optimization are *genetic algorithms* (GA) [Hol75, Gol89], *evolutionary programming* (EP) [Fog95], and *evolution strategies* (ES) [Sch95, Rec94]. The typical evolutionary algorithm may be outlined as follows:

0. *Initialization* of a population of individuals composed of n objective variables and a number of algorithm control parameters known as strategy parameters. Usually this is done at random and may also be generated from known points. The population size is λ.
1. *Recombination* of individuals in the population. The averaging or combining of traits among individuals.
2. *Mutation* of individuals by random variations in the representation.
3. *Selection* of $\mu < \lambda$ individuals based on their fitness and replicated at least once such that a constant population size of λ is maintained in the next generation pool.
4. *Termination* when some criteria have been met, typically when a maximum number of generation is reached. The best found individual(s) for the entire evolution is returned. Otherwise, go to step 1.

Each step of the evolutionary algorithm for numerical optimization will now be discussed in greater detail. Since many possible variations exists, emphasis will be placed on the version most frequently applied in this work. This algorithm is based on the evolution strategy [Sch95, Rec94]. References to modern evolutionary programming and genetic algorithms are mentioned when appropriate.

Initialization

The objective variables are typically initialized randomly as follows

$$x_{i,j}^{(0)} = \underline{x}_j + (\overline{x}_j - \underline{x}_j)u_j \quad i \in \{1, \ldots, \lambda\}, j \in \{1, \ldots, n\}, \tag{2.4}$$

where u_j is a random number uniformly distributed over $[0, 1]$. Here $x_{i,j}$ denotes the j-th component of a vector $\mathbf{x}_i$. Unlike the evolution strategy and

evolutionary programming, the canonical genetic algorithm works on a binary string (bitstring) representation. This means that an individual is a binary vector $\mathbf{b} = (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$ of a fixed length ℓ. For continuous parametric optimization problems a means of decoding the space $\{0,1\}^\ell$ to $\mathcal{R}^n$ is needed. Each variable x_j is encoded by a bitstring of length ℓ_j and decoded by

$$x_j = \underline{x}_j + \Delta x_j \sum_{k=0}^{\ell_j - 1} b_{(\ell_j - k)} 2^k \tag{2.5}$$

where Δx_j is the resolution of the now discretized interval $[\underline{x}_j, \overline{x}_j]$,

$$\Delta x_j = \frac{\overline{x}_j - \underline{x}_j}{2^{\ell_j} - 1}. \tag{2.6}$$

The greater the precision required, the larger the string length ℓ_j needed. These substrings are then concatenated to form a string of length $\ell = \sum_{j=1}^{n} \ell_j$.

Strategy parameters, using random mutative control, are typically implemented in evolution strategies and evolutionary programming. These parameters are usually predetermined in genetic algorithms but this approach may be changing [EHM99]. The most common strategy parameter is the standard deviation of the mutation distribution, also known as the 'mean step size'. Let Δx be a rough measure of the expected distance to the optimum then the initial setting for the 'mean step sizes' is [Sch95, p. 117]:

$$\sigma_{i,j}^{(0)} = \Delta x_j / \sqrt{n} \;\propto\; (\overline{x}_j - \underline{x}_j)/\sqrt{n}, \quad i \in \{1, \ldots, \lambda\}, j \in \{1, \ldots, n\}. \tag{2.7}$$

Here each objective variable in the population has a standard deviation associated with it. Sometimes a common strategy parameter is used for a set of objective variables. From practical experience, it has been suggested that this initial value should be smaller, otherwise, there is a possibility that the algorithm may diverge [Bäc96, p. 80]. A larger initial step size is, however, more reliable as it guarantees initial coverage over the search space. By using the initial values in (2.7) as upper bounds for the step sizes, divergence was avoided in this work. When each dimension has its own mean step size associated with it, the search is aligned to the coordinate system. To overcome this, another strategy parameter is required. This is the rotation angle $\alpha \in [-\pi, \pi]$, which is used to correlate the mean step sizes. The coordinate transformation for the two dimensional case may be read from figure 2.2. The rotation angle

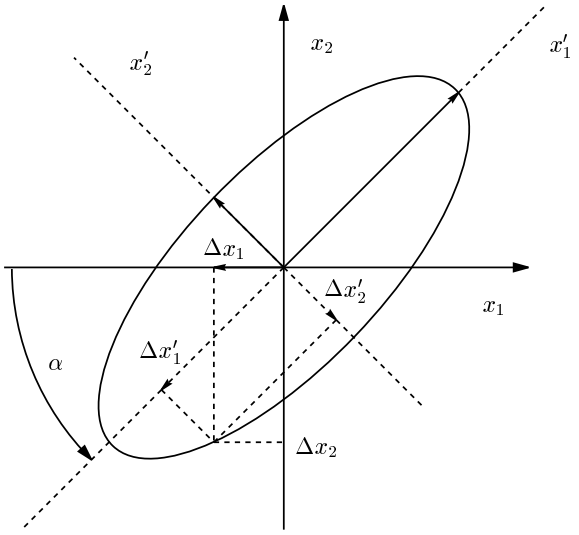


Figure 2.2: Correlated mutation generated using rotation angle α. The ellipsoid represents the place of equal probability density.

is initialized uniform randomly over $[-\pi, \pi]$. The angles are related to the covariances and variances by [Bäc96, p. 70]

$$\tan(2\alpha_{ij}) = \frac{2c_{ij}}{\sigma_i^2 - \sigma_j^2}. \tag{2.8}$$

As many as $n(n-1)/2$ rotation angles are needed when n different standard deviations are used. When n_σ standard deviations are used then the number of angles is $n_\alpha = (2n - n_\sigma)(n_\sigma - 1)/2$.

Recombination

Recombination is the exchange or averaging of traits between one or more individuals in a population. A recombination is called global when all parents are involved in building a single offspring. The traditional approaches are: *discrete*, the exchange of variables, and *intermediary*, the averaging of variables. For the genetic algorithm this operator, also known as crossover, combines low-order, highly fit structures to form even more fit higher-order schema. This argument follows from the controversial *building block hypothesis*

[Gol89]. The role of crossover is frequently debated. In evolution strategies, recombination acts as a statistical error correction which diminishes the harmful effects of mutation [Bey95a]. According to Beyer [Bey95a], the benefits of recombination are *genetic variety* and *genetic repair*. For the evolutionary strategy, a linear speed up in the convergence rate may be shown when recombination is used on the hypersphere model [Bey95a]. This is useful for local optimization but may be less beneficial to global optimization. Evolutionary programming simply ignores recombination.

The number of different recombination operators available grows with each new implementation of an evolutionary algorithm. The most commonly used operators, in numerical optimization, associated with evolution strategies are $(\forall j \in \{1, \ldots, n\})$ [Bäc96]:

$$\hat{x}_{h,j}^{(g)} = \begin{cases} x_{i,j}^{(g)} & \text{no recombination} \\ x_{i,j}^{(g)} \text{ or } x_{k,j}^{(g)} & \text{discrete} \\ x_{i,j}^{(g)} \text{ or } x_{k_j,j}^{(g)} & \text{global discrete} \\ (x_{i,j}^{(g)} + x_{k,j}^{(g)})/2 & \text{intermediate} \\ (x_{i,j}^{(g)} + x_{k_j,j}^{(g)})/2 & \text{global intermediate} \\ x_{i,j}^{(g)} + \chi(x_{k,j}^{(g)} - x_{i,j}^{(g)}) & \text{generalized intermediate} \\ x_{i,j}^{(g)} + \chi_j(x_{k_j,j}^{(g)} - x_{i,j}^{(g)}) & \text{global generalized intermediate} \end{cases} \tag{2.9}$$

where $i,\ k\ \in \{1, \ldots, \mu\}$ are the two parents selected, g denotes the generation counter, and the index j in k_j indicates that k is sampled anew for each value of j. Genetic algorithms typically use a discrete recombination, where sections of the parent strings are concatenated to form offspring. The traditional operator is *one point crossover* [Hol75], where a random position is chosen $\in \{1, \ldots, \ell - 1\}$, and all bits to the right of this point are exchanged between two stings. This operation may be generalized to an n_x-point crossover by repeating the one point crossover n_x times.

In order to keep the algorithm as simple as possible it is decided to ignore the recombination of objective variables and consider only using recombination on the mean step sizes. In particular global intermediate recombination is recommended [Sch95, p. 148] and implemented as follows

$$\hat{\sigma}_{h,j}^{(g)} = (\sigma_{i,j}^{(g)} + \sigma_{k_j,j}^{(g)})/2, \quad k_j \in \{1, \ldots, \mu\}. \tag{2.10}$$

An experimental investigation into the effect of recombination on the 'mean step size' is conducted in section 5.2. Following recombination, the strategy parameters and objective variables are mutated.

Mutation

Variation of strategy parameters is performed *before* the modification of objective variables. The new λ strategy parameters are generated from the μ parents from the previous generation. These strategy parameters are then used to variate the corresponding objective variables later.

The mutation of the 'mean step size' is a *multiplicative* variation. The variation should preserve positive values, the median of the modification should be one, and small changes should occur more frequently [Bäc96, p. 72]. The median of one guarantees an average neutrality of the process in the absence of selection. These conditions may be satisfied by a number of different rules. The 'mean step sizes' in both evolution strategy and modern evolutionary programming are updated according to the lognormal update rule [Sch95]: $i = 1, \ldots, \mu$, and $h = 1, \ldots, \lambda$,

$$\sigma_{h,j}^{(g+1)} = \hat{\sigma}_{h,j}^{(g)} \exp(\tau' N(0,1) + \tau N_j(0,1))) \quad j \in \{1, \ldots, n_\sigma\}, \tag{2.11}$$

where $N(0,1)$ is a normally distributed one-dimensional random variable with an expectation of zero and variance one. The subscript j in $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The *learning rates* τ and τ' are set equal to $\varphi^*/\sqrt{2\sqrt{n}}$ and $\varphi^*/\sqrt{2n}$ respectively, where φ^* is the expected rate of convergence [Sch95, p. 144] and is usually set to one [Bäc96, p. 72].

Most of the theoretical results in evolution strategies are derived from using a single global step size. This guarantees spherical symmetry of the search distribution. For a single mean step size, $n_\sigma = 1$, the modification reduces to

$$\sigma_h^{(g+1)} = \hat{\sigma}_h^{(g)} \exp(\tau_o N(0,1)), \tag{2.12}$$

where $\tau_o = \varphi^*/\sqrt{n}$. Lower bounds on the mean step sizes are commonly implemented. This parameter sets the desired precision of the objective variable and may be quite useful when known. However, it has also been shown empirically that its choice may influence the performance of the algorithm [LYL+98].

The rotation angles are updated as follows,

$$\alpha_{h,j}^{(g+1)} = \hat{\alpha}_{h,j}^{(g)} + \beta N_j(0,1) \quad j \in \{1,\ldots,n_\alpha\} \tag{2.13}$$

where $\beta \approx 5\pi/180$. So that the angles remain within the range $[-\pi, \pi]$ they are mapped as follows

$$|\alpha_j| > \pi \Rightarrow \alpha_j = \alpha_j - 2\pi \text{ sign}(\alpha_j). \tag{2.14}$$

Having varied the strategy parameters, each individual $(\mathbf{x}_i, \sigma_i)$, $\forall\ i \in \{1,\ldots,\mu\}$, creates λ/μ offspring on average, so that a total of λ offspring are generated by

$$x_{h,j}^{(g+1)} = x_{i,j}^{(g)} + \Delta x_{i,j} = x_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1), \tag{2.15}$$

where the rotation angles are ignored. When rotations are used the transformation of Δx_i, as shown in figure 2.2, is required. Rotations introduce up to $n(n-1)/2$ additional parameters to the search. This increases the complexity of the search and computational overhead. Rotation angles are not applied in the work presented and the reader is referred to [Bäc96, p. 71] for their implementation in an n dimensional search space. The adaptation is therefore not independent of the chosen coordinate system. The influence of the coordinate system is discussed further in section 5.4.

When the parametric bounds are also the parametric constraints for the problem, the variation of the objective variable may be retried until the variable is within its bounds. In order to save computation time, the mutation is retried a number of times and then ignored, leaving the object variable in its original state within the parameter bounds.

Selection

Selection works solely on the fitness values. In the canonical genetic algorithm, selection is probabilistic, using scaled absolute fitness values to determine the probability of mating. The more common approach to selection is based entirely on ranking a number of individuals λ' according to some fitness function ψ,

$$\psi(\mathbf{x}_{1:\lambda'}) \leq \psi(\mathbf{x}_{2:\lambda'}) \leq \ldots \leq \psi(\mathbf{x}_{\mu:\lambda'}) \ldots \leq \psi(\mathbf{x}_{\lambda':\lambda'}), \quad \mu < \lambda' \tag{2.16}$$

where $\mathbf{x}_{i:\lambda'}$ is the ith *order statistic* ($i = 1, \ldots, \lambda'$). The fitness of an individual may be due to the objective function, multiple objectives, constraints, or even a combination of these. The handling of constraints by selection is discussed in the following section. The best $\mathbf{x}_{1:\lambda'}, \ldots, \mathbf{x}_{\mu:\lambda'}$ individuals are then replicated (and varied) at least once for the next generation pool. The evolution strategy commonly ranks the entire population, i.e. $\lambda' = \lambda$, and replicates each of the best μ individuals an average of λ/μ times, this selection strategy is called (μ, λ). Another selection strategy is the $(\mu + \lambda)$. This is an elitist selection strategy where parents are ranked along with their offspring, i.e. $\lambda' = \mu + \lambda$. The optimal setting of μ/λ will be discussed in the following section. Evolutionary programming uses also an elitist selection strategy, but it is probabilistic. In one version the individuals are selected from a subpopulation of size λ', sampled randomly with replacement from the pool of $\lambda + \lambda$ parents and offspring. The number of parents is equal to the number of offspring. Typically $\lambda' = 10$ and only the best of these is replicated. This tournament style selection is repeated until λ individuals have been generated.

Termination

An evolutionary algorithm is commonly terminated after a number of generations, G. Alternatives involve determining the difference in fitness values over a number of generation and whether it goes to zero or to a prescribed limit [Sch95, p. 114]. It is similarly possible to use the fitness difference between the best and worst individual current population.

2.2.2 Constraint Handling

One common approach to deal with constrained optimization problems is to introduce a penalty term into the objective function to penalize constraint violations [FM68]. The introduction of the penalty term enables the transformation of a constrained optimization problem into a series of an unconstrained one such as,

$$\psi(\mathbf{x}) = f(\mathbf{x}) + r_g \; \phi(\mathrm{g}_j(\mathbf{x}); j = 1, \ldots, m) \tag{2.17}$$

where $\phi \geq 0$ is a real valued function which imposes a 'penalty' controlled by a sequence of *penalty coefficients* $\{r_g\}_0^G$. The general form of function ϕ includes both the generation counter g (for dynamic penalty) and the population (for adaptive penalty). In the current notation, this is reflected in

the penalty coefficient r_g. The function ψ is the transformed *fitness function*. This transformation, i.e. equation (2.17), has been used widely in evolutionary constrained optimization [KP98, SS89]. In particular, the following quadratic loss function [FM68], whose decrease in value represents an approach to the feasible region, has often been used as the *penalty function* [MA94, JH94]:

$$\phi(\mathrm{g}_j(\mathbf{x}); j = 1, \ldots, m) = \sum_{j=1}^{m} \max\{0, \mathrm{g}_j(\mathbf{x})\}^2. \tag{2.18}$$

The penalty function method may work quite well for some problems, however, deciding an optimal (or near-optimal) value for r_g turns out to be a difficult optimization problem itself! If r_g is too small, an infeasible solution may not be penalized enough. Hence an infeasible solution may be evolved by an evolutionary algorithm. If r_g is too large then a feasible solution is very likely to be found but could be of very poor quality. A large r_g discourages the exploration of infeasible regions even in the early stages of evolution. This is particularly inefficient for problems where feasible regions in the whole search space are disjoint. In this case, it may be difficult for an evolutionary algorithm to move from one feasible region to another unless they are very close to each other. Reasonable exploration of infeasible regions may act as bridges connecting two or more different feasible regions. The critical issue here is how much exploration of infeasible regions (i.e., how large r_g is) should be considered as reasonable. The answer to this question is problem dependent. Even for the same problem, different stages of evolutionary search may require different r_g values.

There has been some work on the dynamic setting of r_g values in evolutionary constrained optimization [JH94, KP98, MA94]. Such work usually relies on a predefined monotonically nondecreasing sequence of r_g values. This approach worked well for some simple problems but failed for more difficult ones because the optimal setting of r_g values is problem dependent [Ree97]. A fixed and predefined sequence cannot treat a variety of different problems satisfactorily. A trial-and-error process has to be used in this situation in order to find a proper function for r_g, as is done in [JH94, KP98]. An adaptive approach, where r_g values are adjusted dynamically and automatically by an evolutionary algorithm itself, appears to be most promising in tackling different constrained optimization problems. For example, population information can be used to adjust r_g values adaptively [SC97]. Different problems lead

to different populations in evolutionary search and thus lead to different r_g values. The advantage of such an adaptive approach is that it can be applied to problems where little prior knowledge is available because there is no need to find a predefined r_g value, or a sequence of r_g values, which is 'optimal' for the problem.

According to (2.17), different r_g values define different fitness functions. A fit individual under one fitness function may not be fit under a different fitness function. Finding a near-optimal r_g, adaptively, is equivalent to ranking individuals in a population adaptively. Hence, the issue becomes how to rank individuals according to their objective and penalty values. Rank-based selection is assumed and a novel method for ranking individuals without specifying an r_g value is proposed in chapter 6. Experimental studies test the effectiveness and efficiency of the method, which can be regarded as an exterior penalty approach.

2.2.3 Rate of Convergence

In optimization, the convergence rate to an optimum point is of major interest. This information can be used to estimate the time complexity of an algorithm. In this section, the formulation for convergence rates for the $(1, \lambda)$ evolution strategy is presented. It is shown how the optimal number of offspring may be computed and the expected running time to reach the vicinity of the optimum.

Let s' be the useful distance covered in the sense of approaching the optimum point. The progress rate is then the expectation of s':

$$\varphi = \mathrm{E}(s') = \int_{s'=s_u}^{\infty} s' w(s') \mathrm{d}s' \tag{2.19}$$

where for a *comma* strategy $s_u = -\infty$ and the *plus* strategy $s_u = 0$. If the function $w(s')$ is symmetrical and unimodal then the expected value may be found as the value of s' at which the probability density $w(s')$ reaches its maximum.

Let $w(s = s')$ denote the probability density that a particular descendant gets exactly s' closer to the objective. The probability that a descendant advances less than s' is $p(s < s')$. For a single parent $(1 \overset{+}{,} \lambda)$ strategy[1], the probability of the best descendant covering the distance s' is [Sch95, p. 123]:

$$w(s') = \lambda w(s = s') \big[p(s < s')\big]^{\lambda - 1} \tag{2.20}$$

[1]The $\overset{+}{,}$ is shorthand for both "," and "+" strategies.

where

$$p(s < s') = \int_{s=-\infty}^{s'} w(s = s')\mathrm{d}s \tag{2.21}$$

The exact analytical solution of the progress rate for a number of objective function has been found for $(1 + 1)$. Other strategies have been solved using approximate theory and simulation. Even for the incline plane (a simple linear model), the analytical solution for $(1 \overset{+}{,} \lambda)$ is difficult to find for all λ. For illustrative purposes, a uniform distribution is used, instead of normal distribution, to obtain an analytical solution. This example illustrates how the progress rates and optimal λ may be computed. Imagine the function to be a terrain in the $(n + 1)$-dimensional space, it appears as an inclined plane [Sch95, p. 124]

$$f(\mathbf{x}) = x_1. \tag{2.22}$$

Orient the coordinate system so that the plane only slopes in the direction of one axis x_1 and the starting point under consideration lies on the origin. The useful distance s towards the objective is just part of the random vector lying along the x_1 axis. For the uniform distribution in the hypercube

$$w(s = s') = \begin{cases} \frac{1}{2\eta} & \text{for } |s'| \leq \eta, \\ 0 & \text{otherwise,} \end{cases} \tag{2.23}$$

the result will depend on the rotation of the coordinate system since the mutation distribution is not spherically symmetric. The probability that a descendant advances less than s' is

$$p(s < s') = \int_{s=-\eta}^{s'} \frac{1}{2\eta}\mathrm{d}s = \frac{1}{2\eta}(s' + \eta), \;\; s' \leq \eta \tag{2.24}$$

and hence the progress rate is

$$\varphi = \lambda\left(\tfrac{1}{2\eta}\right)^{\lambda} \int_{s'=s_u}^{\eta} s'\left[s' + \eta\right]^{\lambda-1}\mathrm{d}s'. \tag{2.25}$$

Then for $s_u = 0$, $(1 + \lambda)$ strategy:

$$\varphi = \eta\left[\frac{\lambda - 1}{\lambda + 1} + \frac{1}{2^{\lambda}(\lambda + 1)}\right] \tag{2.26}$$

and for $s_u = -\infty(\equiv -\eta)$, $(1, \lambda)$ strategy:

$$\varphi = \eta\left[\frac{\lambda - 1}{\lambda + 1}\right] \tag{2.27}$$

For the plus strategy a positive progress rate is maintained when $\lambda = 1$, but for the comma strategy the progress will in this case be zero. In order to minimize the number of function evaluations needed per generation on a serial computer and gain maximum progress over G generations, λ may be chosen such that

$$\sum_{g=1}^{G/\lambda} \varphi^{(g)}$$

is maximal or equivalently [Sch95, p. 127]

$$\frac{\partial}{\partial \lambda}\left(\frac{\varphi}{\lambda}\right)\Big|_{\lambda=\lambda_{opt}} \stackrel{!}{=} 0 \tag{2.29}$$

For the two strategies then; $\lambda_{opt} = 1$ $(1+\lambda)$, and $\lambda_{opt} = 2.4142$ (2 or 3) $(1, \lambda)$. Similarly, optimal values have been computed using the normal distribution for various function models. This analysis has shown that for $\lambda \geq 5$ the maximal rate of progress is practically independent of whether or not the parent survives. It follows that for a (μ, λ) strategy the ratio λ/μ should not be less than 6 [Sch95, p. 145], and from practical experience, it is recommended that μ/λ be approximately 1/7 [Sch95, p. 148].

Finally, the expected running time is of central interest. For a continuous search space, it is possible to determine the number of generations for constant stationary normalized progress rate $\varphi^\star = \varphi\, n/s$ and a relative distance change to the optimum s_o/s [Bey96],

$$G = \frac{n}{\varphi^\star} \ln(s_o/s) \tag{2.30}$$

where s_o is the initial distance to the local optimum and s is the desired distance. For the case when the progress rate is bounded by c from below, i.e. $\varphi^\star \geq c > 0$, then one may bound the total number of generations needed by

$$G < \frac{n}{c} \ln(s_o/s) \tag{2.31}$$

Although these results are for the hypersphere model [Bey96], it is claimed that similar results might hold for the general case[2] but proof is unavailable. Showing that $c > 0$ is a difficult theoretical problem, especially when one considers the *full* implementation of an evolutionary algorithm.

[2]Communication with Hans-Georg Beyer.

It is fair to say that progress rate theory has been limited to single peaked fitness landscapes, or local minima. This guarantees that individuals ranked by fitness values is equivalent to ranking them in accord with their distance to the optimum. This does, however, not hold for a multi-peaked fitness landscape. This case is of greater interest and will therefore be given special attention is chapter 5.

2.2.4 Global Convergence

Another important theoretical requirement of optimization is *global convergence*. This is just as important as fast convergence but always in direct opposition to approaching an optimum quickly and accurately. A globally optimal solution $\mathbf{x}^*$ is one where

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall\, \mathbf{x} \in \mathcal{S} \cap \mathcal{F}. \tag{2.32}$$

An evolutionary algorithm will locate the global optimum, with probability one, by means of the reachability property [Bäc96, Rud97a],

$$\mathrm{P}\Big\{ \lim_{g\to\infty} \mathbf{x}^{(g)} = \mathbf{x}^* \Big\} = 1 \tag{2.33}$$

Reachability implies that any point in the search space can be generated from any other arbitrary point by either mutation or recombination. The global convergence property is valid for algorithms that maintain the best found individuals for the entire evolutionary run or use elitist selection. The theorems of global convergence follow from the realization that points generated for a sufficiently long time eventually generate the global optimum. Global convergence theorems of this type don't help too much for real world applications where good solutions are desired in reasonable times. This is discussed further and illustrated empirically in section 5.4.

2.3 Connectionist Models

Connectionist models, often called parallel distributed models or neural networks, consist of nothing more than *nodes* and *connections*. Nodes are abstractions of biological neurons. It is generally understood that biological neural functions are stored in the neurons and the connections between them. The

nodes are simple processing units. Processing takes place via the propagation of activation among the units; via real-valued weighted connections. The 'knowledge' that governs processing consists of the values of the connection weights and learning occurs through the changes of the connection weights. The evolutionary algorithm described in the previous section may be used to optimize these weights and, therefore, 'evolve knowledge'. The details presented in this section are sufficient for the implementation of connectionist models for knowledge representation in problem solving.

The origin of the modern view of a network of artificial neurons may be traced back to the work of McCulloch and Pitts [MP43]. They demonstrated

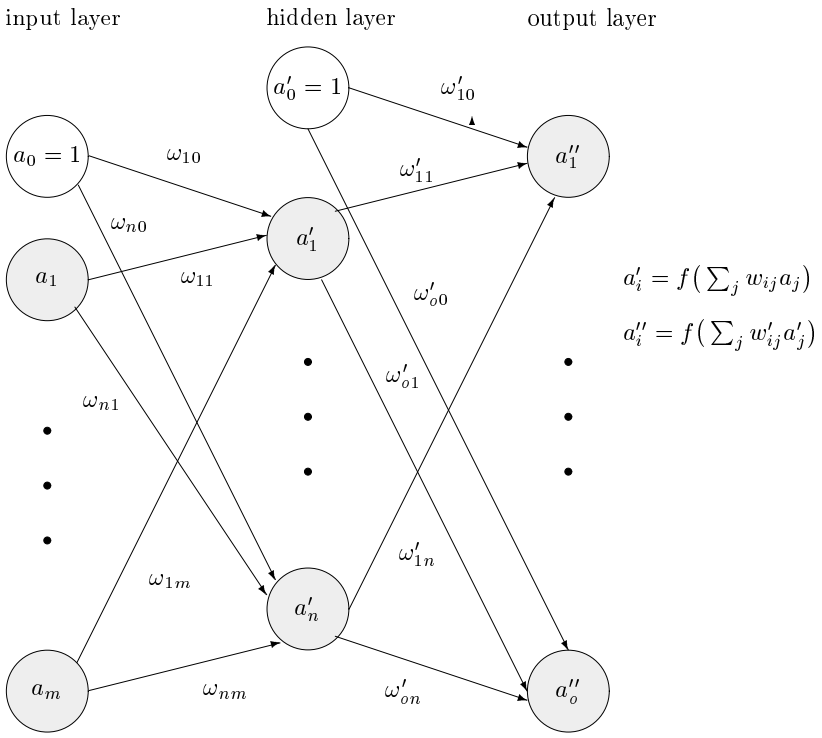


Figure 2.3: A multilayered feed-forward neural network. The node activations are labelled by the variable a and the connection weights by w. An input activation layer is fed through the network to the output layer as shown. The number of hidden layers may be any number.

that a neural networks could, in principle, compute any arithmetic or logical function. Later it was shown that a single hidden layer of neurons is sufficient for a network to be a universal function approximator [Fun89, HSW89, Cyb89]. A number of neurons, acting in parallel, form what is known as a *layer*. The network inputs then form the *input layer* and the layer whose output is the network's output is called an *output layer*. All other layers are *hidden layers*. Figure 2.3 shows a sample network, a three layered feed-forward neural network where the nodes are the circles and the connections are the vector lines drawn between them. The *architecture* of network refers to the particular way the network is assembled. The inputs to a node i originate from other nodes or external sources that have a numerical activation value, denoted here by a_j. The sending nodes activity travels along a connection line to the receiving node. The connection lines have varying connection strengths that are represented by the numerical value w_{ij}. A connection weight multiplies the signal travelling through it. The receiving node will then process the signal by first summing up the net inputs, that is $\sum_j w_{ij}a_j$. This net activation may be positive (excitatory) or negative (inhibitory) and is decided by a threshold that is determined by the bias node. The bias nodes are those units in figure 2.3 whose value is one. The activations at any given unit are then usually 'squashed' by a nonlinear activation function resembling a sigmoid. The most common functions are the log sigmoid transfer function

$$a_i' = \frac{1}{1 + \exp(-\sum_j w_{ij}a_j)} \tag{2.34}$$

and the hyperbolic tangent sigmoid transfer function

$$a_i' = \frac{2}{1 + \exp(-2\sum_j w_{ij}a_j)} - 1. \tag{2.35}$$

2.4 Summary

In this chapter, background and literature for problem solving and numerical optimization by evolutionary algorithms was presented. Details of the implementation of a particular version of the evolution strategy, applied in this work, was described in detail. It was shown how the convergence rate for a $(1 \overset{+}{,} \lambda)$ evolution strategy may be computed in terms of the expected useful distance covered towards a known optimum. Optimal offspring numbers for serial computers were discussed and a lower bound on the number

of generations needed to reach the vicinity of the optimum. Conditions for convergence to a global optimum for an arbitrary problem was given. Finally, a brief overview of connectionist models was presented. The following chapter is devoted to setting up problems and their candidate solutions for effective and efficient evolutionary search.

Chapter 3

Problem-Solution Abstraction

The aim of this chapter is to increase the understanding of how a situation should be described on a computational device for evolutionary problem solving. The description must embody the principles of evolution by natural selection, which are [Lew70]:

1. *Phenotypic variation* – there are processes that produce variations among individuals.

2. *Differential fitness* – different phenotypes have different rates of survival and replication in different environments.

3. *Fitness is heritable* – there is a correlation between parents and offspring in the contribution of each to future generations.

In order to apply the principles, the stage must be set. What is needed is a description of the organism and its environment, which, in turn, represents in some way the situation to be optimized. It must then be determined whether the organism and the environment are to be treated in isolation or as a whole. This will be the topic of discussion in the following section. Only the fitness need be inherited. Differential fitness is due to different phenotypes. Different phenotypes in turn are due to the processes, variation operators, which produce them. Understanding this relationship is the key to designing efficient and effective evolutionary problem solvers. With the aid of an example, these issues will be addressed in the second section presented. A summary with main conclusions are given in the final section.

3.1 Environment and Organism

The notion of a problem–solution model for natural systems is awkward. In problem solving, there is the notion that a problem preexists its solution. This attitude is reflected in a world model where an outside force, the preexisting environment, sets up the 'problem' an organism must solve and the inside forces for variation that generate the organisms' 'solution' to the 'problem' [Lew00]. For example, an aquatic environment poses the problem of swimming and the fish is a solution.

On a digital computer the problem and its solution must be described by data structures or *symbols*, which the machine is capable of interpreting. The interpreter is a program that is determined by a system of *rules* that govern transitions from one state to the next. An analogy between symbols and genes is commonly drawn. The interpretation of the genome is presupposed and invariant. This is what is required in setting up the situation. In this model, only the genome is randomly varied, hence, the gene is often referred to as the unit of heredity. Differences in the genome encode for differences in the phenotype. In general, there exists a function

$$f_o : \mathcal{C} \to \mathcal{S} \tag{3.1}$$

mapping points in the genotype or representation space $\mathcal{C}$ to the phenotype space $\mathcal{S}$. Together f_o and $\mathcal{C}$ form a *representation* of the attributes of the organism. In short, representations are caricatures of selected attributes of the problem-solution. Essentially, it is the designer of the algorithm who sets up the problem and the plausible solutions to it. The contents of the symbolic representation are the contents of the designer's thought. The rules manipulating and transforming the symbols imitate the designer's cognitive process, the process by which knowledge is acquired. The designer is, however, free of any commitment as to how this knowledge is represented.

Let a function

$$f : \mathcal{S} \to \mathcal{F} \tag{3.2}$$

map points in the phenotype space to a *fitness* space $\mathcal{F}$. This is the *fitness function*. In natural systems it is expected that a small change in the phenotype will result in a small change in reproductive fitness. Neighbourhoods in phenotype space map into neighbourhoods in fitness space [Lew85, p. 79].

The phenotype and fitness value taken together form a continuous *fitness landscape*. As a consequence, correlation analysis of fitness landscapes have been proposed as a means of characterizing difficulty for evolutionary search methods [Man97, VFM00]. This involves estimating the landscape 'ruggedness'. The efficiency of the evolutionary search is then reflected in the ability to anticipate how differences in the genotype are translated to differences in the phenotype. With this understanding, it is possible to design a variation operator that transforms a symbolic representation to a neighbourhood in the fitness space. However, this in itself is a meta-problem – the *coding relationship problem*.

Different interpretation of symbols is equivalent to solving another problem. A local optimum under one interpretation may not be one under another. Escaping local optimum may therefore be possible under a changing representation. The fitness landscape changes and there exists an interpretation for which the original problem becomes trivial [LV90]. Searching for an effective interpretation of the symbols implies learning something about the problem. It may then also be possible to use this domain knowledge to solve another problem instance of the same class more efficiently in the future. The space of all possible interpretations is, however, larger than the original search space. When the domain knowledge space is represented again with symbols, *meta-rules* will be needed to interpret them otherwise the coding relationship problem will persist. Numerous attempts have been made to dynamically adjust symbolic representations in order to escape local optima [BWW00]. However, as long as these approaches consist of pure syntax operations carried out on symbols, no improvement can be expected in general. It is quite possible that the genetic code is some form of symbolic knowledge, but it can only be understood on the basis of pre-existing non-symbolic forms. Symbolic representations are theoretical primitives and there are no conceptual categories available for specifying the situation before the symbols come into being [Ste95]. Unless the rules for transforming and manipulating the symbols, based on their meaning, are clearly specified by the designer, it is impossible to know how to search in a symbolic space.

The frequently cited alternative to a symbolic representation is the connectionist representation. The connectionist representation is *propositional* because the environment proposes the appropriate action. The symbolic representation requires a symbol for each plausible action. That is; the symbol

currently in use in the symbolic structure dictates the action taken. This kind of representation is *compositional*, which is characteristic of symbolic approaches. A propositional representation is more suitable for an alternative world view for living systems. This is the view that there is no predetermined problem but it is rather the activity of living systems that determines both the problems and solutions simultaneously. There is no environment without an organism and there is no organism without an environment [Lew83]. The organism both constructs and selects its environment. The organism determines the problem by establishing what parts of the environment are relevant. This view is expressed in figure 3.1. There is be no link from environment to organism in the conventional world view. The information flow would only be one way, from genome to environment. The same figure shows how the genome is taken out of the organism-environment loop. The genome is that part of the problem solving architecture which is randomly varied, independent of the organism and its milieu. When identifying a problem, all information potentially relevant to the problem at hand must be abstracted. Information are data structures plus their meaning. Information that are 'easy' to define and compute are known as *primitives*. The environment primitives are the simple sensations perceived by the organism. Instances of a primitive in problem solving may for example be the information regarding the contents of every square on a chess board, whose move it is, and so on. Activities of the organism are then reflected in legal chess moves. The *unknown* is the appropriate activity

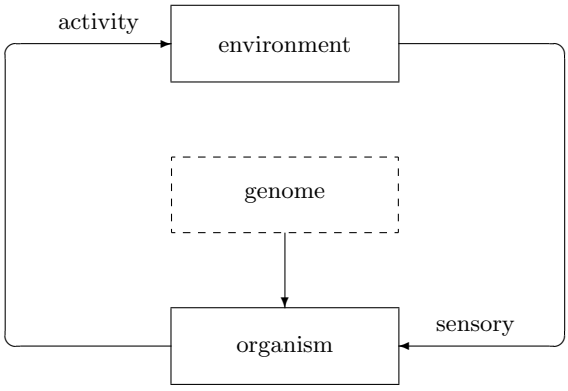


Figure 3.1: Interpretation of the environment-organism and genome system.

for a given set of primitives. This interpretation is an ongoing process.

There are two approaches to modelling the alternative world view. The first is a model which maps a sensory state directly to action. The second requires a *scoring function* or *approximate cost-to-go function*, which can determine the appropriate activity for a given sensory or problem solving state. The plausible activity (actions) and environment primitives must be pre-specified. The world model depicted in figure 3.1 is also found in dynamic programming [Ber95] and reinforcement learning [SB98]. These are so called actor-critic models [BT96, p. 36]. Figure 3.1 should in this case be reinterpreted as follows: the genome determines of the optimal cost-to-go function, the generation of new cost-to-go functions is the result of random variations of the genome, the actor (controller) is the organism, the system corresponds to the environment, and the critic is the result of selection.

Scoring functions are based upon features that describe the problem solving states, an evaluation function may be very sensitive to the set of features chosen. In neuro-dynamic programming [BT96], the function is approximated by an artificial neural network. When lookahead is used, only a single output node is needed to estimate the worth of an action given the resulting state. The evaluation of states are used to determine whether one state is preferable to another. If no lookahead is used, then the neural network must have as many output nodes as there are plausible actions. In practice, this is not commonly done, since this requires a more complex scoring function. It may, however, be advantageous if the computational costs are high or lookahead is not possible. Note, that the objective of the task, the end goal, is not explicitly programmed.

From the organism-environment perspective, it is possible to reinterpret the evolution strategy [Sch95, p. 106]. Consider the sphere model

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

where x_i, $i = 1, \ldots, n$ are environment primitives. They may represent some nutrients absorbed by the organism and their values are the amount. The organism transforms these nutrients to products some of which are harmful to it. The concentration of these products, now part of the environment, is given by the value $f(\mathbf{x})$. In order for the organism to maximize its survival rate, it must posses behaviour which will minimize $f(\mathbf{x})$. The object of evolution

is then simply the organism's environment. However, it is the activity of the *population* that determines the environment. For the evolution strategy the 'self-adaptive parameter' σ, the mean step size, determines this activity. With the inclusion of random changes, its variation is given by

$$\sigma_i^{(g+1)} = \sigma_i^{(g)} Z^{(g)} \tag{3.4}$$

where Z is a random number. The organism is the object of these internal forces of variation, which operate independent of its functional needs or its relation to the outer world. That is what is meant by mutations being 'random'. The main properties of the random change is that there is no deterministic drift in behaviour without selection and that small changes in behaviour occur more frequently than large ones [Sch95, page 143]. In this particular case, these requirements are satisfied when Z is log-normal distributed. If the mutation of σ is dependent on x, then the mutation is called a directed mutation. The replicated organism selects or constructs its environment according to the following behaviour,

$$x_i^{(g+1)} = x_i^{(g)} + \sigma_i^{(g+1)} N_i(0,1), \quad i = 1, \ldots, n.$$

In this interpretation σ is the organism primitive and Z its internal variation. The environment primitives are x and $f(x)$, and $N(0,1)$ the source of developmental noise. Evolution is not a property of these primitives, but of the whole ensemble.

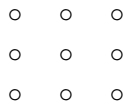
The ability to select and construct ones environment comes as a direct consequence for interacting with ones world. The ability to have knowledge about ones world necessitates an interaction with it. The organism is the medium for this interaction [Lew83] and the 'knowledge' is formed in the organization of the organism-environment system. The sharing of the environment with other organisms is essentially a medium for communication about the world. Problems, interpreted as such, may quite simply be solved by a team of organisms, each performing their behaviour independently. There is then no necessity in having a central directing agent in problem solving [Bro91, Hut95]. This is yet another possible problem solving architecture, however, not explored any further here.

Both symbolic and connectionist descriptions presuppose the existence of some representation before they can discover other useful representations. This makes the initial architecture extremely important. If not chosen properly,

it places additional burdens on problem solving. The following section will illustrate by example the use of symbolic and connectionist representations.

3.2 Representing the World

In this section, the nine-dot problem [NS72, Hay81] is considered to illustrate the concepts developed in the previous section. The problem is commonly found in the psychology literature (see for example [LD85]). The problem is such that it is difficult for the designer to formulate a problem-solution description and consequently, the change needed to improve a solution. This is the main reason for choosing this particular example. The problem is that a subject is presented with a square array of nine dots;



and is directed to draw four straight lines without raising the pen from the paper so, that each of the dots is touched by at least one of the lines. If unfamiliar with the problem, the reader may want to try to solve it before proceeding. The appropriate questions for identifying the problem are [Pol57]: *what is the unknown?* four sequentially drawn lines, *what are the data?* the nine dots given in position, and *what is the condition?* the four lines should pass all nine dots at least once.

Draw any four lines without raising the pencil. All possible lines drawn correspond to the *problem space* [New90, Hay81]. The sketching of four lines *represents* an idea of the solution. The *knowledge* required to draw these four lines abstracts from this *representation*. The difficulty is to determine how this knowledge is to be represented on a computational device.

3.2.1 Symbolic

The classical view is to treat a representation as a syntaxically-manipulative symbol. It is also imposed by the fact that a representation must be *implemented* in a computer by some data structure. Previously, the reader may have drawn four lines in an attempt to solve the problem. These lines must now be abstracted to symbols and what will be modelled is the abstract computation

achieved by the manipulation of the symbols. This is the replacement for the reader's cognitive processes and activity that are now absent from the system [Hut95, p. 363].

Consider the general idea of a solution to the nine-dot problem as a drawing of four lines. Perhaps the most commonly used symbolic representation of a drawing in a computer is a plot-file. For example the HP-GL plotting commands needed to draw four lines sequentially are

$$[\ \texttt{PU}\ x_1\texttt{,}y_1\texttt{;}\ \texttt{PD}\ x_2\texttt{,}y_2\texttt{;\$}\ \texttt{PD}\ x_3\texttt{,}y_3\texttt{;\$}\ \texttt{PD}\ x_4\texttt{,}y_4\texttt{;\$}\ \texttt{PD}\ x_5\texttt{,}y_5\texttt{;\$}\ ]. \tag{3.7}$$

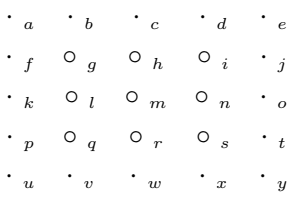
The string is interpreted by reading it from left to right. The symbol `PU` represents the pen at the specified point, this is also known as the *initial state*. Symbol `PD` represents a line from the current point to the specified point, these instruction produce *intermediate states*. The actual solution is obtained when the entire string has been completely interpreted and will be referred to as the *final state*. There are a number of special symbols like `,;$` which assist the plotter in interpreting these instructions, for example `$` is a terminator character. Other possible symbols may represent a circle, ellipse, arc, pen number, pen speed, pen pressure, etc. The representation is transparent and easy to apply. The plot-file is a symbolic representation of potentially *any drawing* with plotter-pens. However, the plotting commands used in (3.7) do not represent just any drawing, but a drawing which 'naturally' satisfies conditions of the problem. These are the conditions that only four lines may be drawn without lifting the pen. The symbols, which are *unknown*, are the five arbitrary coordinates:

$$[\ (x_1, y_1)\ (x_2, y_2)\ (x_3, y_3)\ (x_4, y_4)\ (x_5, y_5)\ ] \tag{3.8}$$

The coordinates represent numbers. Usually the ten digit symbols "0", "1", ..., "9", are used to represent the numbers 0 through to 9. All other numbers may be represented by concatenating this small set of symbols.

When choosing a representation it is common to hypothesize what the solution may look like. These speculations may be interpreted as hypothetical conditions which could be satisfied by the representation. *How are the unknowns linked to the data?* [Pol57] How are the lines linked to the nine dots? The common assumption is that the nine-dot problem can be solved as a dot-to-dot puzzle. The lines are simply drawn by connecting the dots. For representation (3.8), this would imply that the coordinates should only

correspond to the coordinates of any of the nine dots. The domain of solutions under this hypothesis does, however, not contain the correct solution. If the coordinates cover the real number space then most of the search will be devoted to touching any dot at all. Another hypothesis is then that the solution lies on an extended grid rather than just the nine dots. In this case, the nine dots may be placed on a grid as shown:



Lines may now be drawn simply by connecting dots. It is assumed that the correct solution lies on the grid and luckily for this hypothesis it does. This illustrates how important the expressive adequacy of the representation is. It must span that portion of the world which contains the correct solution, but at the same time it should be as small as possible. By labeling the grid with letters from the alphabet, four sequentially drawn lines may then be represented for example by the symbolic string

$$[\ g\ i\ s\ q\ g\ ]. \tag{3.10}$$

The string is interpreted in a similar fashion to the previous string representations.

The representations presented 'naturally' satisfy most of the conditions of the problem and hypothesis of what the solution may look like. If the representation is complete then the remaining conditions will be satisfied by at least one of the alternatives specified by it. The search for alternatives will clearly depend on the representation chosen.

Operations performed on symbols without a regard to their meaning may be viewed as the great strength of the symbolic approach. If an organism is computational, its operations may apply to other computational systems as well. By imitating nature's computational ways one may hope to discover some universal 'tricks' to general problem solving. One point crossing over and point mutations are two examples. The manipulation of symbols, without regard to their meaning and selection, may then in principle solve any problem. In order to justify such an approach, the *schema theorem* was proposed [Hol75, Hol00].

Attempts to explain the behaviour of such syntactic systems by interpretations of the schema theorem have, however, been unsuccessful. The notion of implicit parallelism, a misinterpretation of the schema theorem, has been used to argue that symbols of low-cardinality are ideal [Gol89], i.e., all binary. In this case, for the nine-dot problem, assuming a 4×4 grid, a solution may be represented as

$$[\ 0000\ 1111\ 1010\ 1001\ 0000\ ]. \tag{3.11}$$

For the reasons given, binary representations are commonly found in the theoretical genetic algorithm literature. However, no efficient non-deterministic approach for arbitrary pseudoboolean optimization problems exists [Bäc96, p. 56]. Gray codes have been shown to perform better empirically than natural binary encoding on discretized real-valued fitness landscapes [WMRD96]. This is because neighbourhoods in the real-valued space are preserved by Gray codes. In this case, the structure of the problem provides useful topological

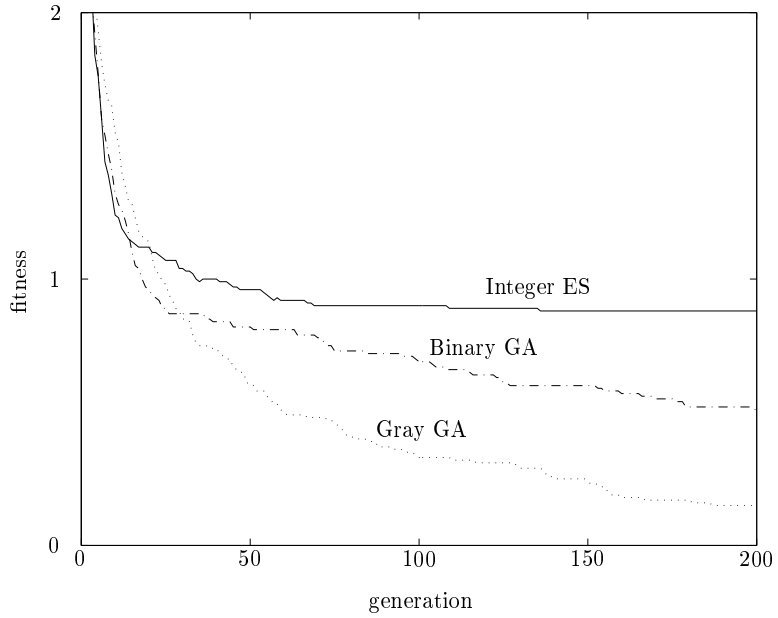


Figure 3.2: Comparing integer, binary and Gray encoding for the nine dot problem.

information for the algorithm.

To illustrate the effect the representation has on evolutionary problem solving, an empirical comparison of three different symbolic representations for the nine dot problem is performed. A GA using natural binary and Gray coding of (3.8) are compared. This algorithm uses one point crossover and bit-wise mutations with probability $1/\ell$, where the string length $\ell = 20$. The population size is 50, using binary tournament selection, and an elitist strategy (the best found individual is preserved in the population). The average fitness of the best individual from 100 independent runs is depicted in figure 3.2, where the fitness value is simply the number of dots which remain untouched. From these runs the Gray GA performs better than the binary GA on average. If we assume this is because neighbourhoods in the integer-valued space is preserved, then perhaps operating directly on the integer space is best. There exists an ES algorithm for integer programming which does precisely this [Rud94]. The algorithm uses a mutation distribution constructed from the principles of maximum entropy. For a further analysis of integer programming using ES see [RSJ00]. A $(10, 50)$ selection and a lower bound on the global mean step size of one is used. The result is worse than both the GA runs on average. For all experiments, attempts were made to tune the algorithm's parameters, but the overall results remained the same. Is the reason for the Gray GA's success because the integer 1 is the neighbour of 4? This example illustrates the coding relationship problem discussed in the previous section. It is not known how to reason with the symbols and therefore designing the appropriate variation operator is a hopeless task, which can only be solved by trial and error. It may be said that that representation is not inferentially adequate.

3.2.2 Connectionist

Evolutionary algorithms have been criticized for their lack of interaction with the environment. "Evolutionary methods ignore much of the useful structure of the reinforcement learning problem: they do not use the fact that the policy they are searching for is a function from states to actions" [SB98]. This is, however, not so much a criticism of the evolutionary approach but rather of the conventional world view chosen. Ultimately, an evolutionary algorithm need only satisfy the three principles which embody evolution by natural selection. These principles are independent of the particular world view chosen.

The connectionist models are propositional and hence a suitable representation for when the environment states propose appropriate action. The environment states, connectionist network architecture, and all plausible activity must be pre-specified. For the nine dot problem the activities are all the possible lines which may be drawn from the current point. If a 4×4 grid is assumed, as in the previous section, then the plausible activities are 16 at any given state. A scoring function, approximated by a numerical artificial neural network, will determine which activity is to be performed. The drawing of the four lines starts at a randomly chosen point on the grid. That is, the initial state is random. Now all 16 plausible lines drawn from this point must be examined by the scoring function. This is done by a one step lookahead procedure. A line is drawn and the resulting state is used as input to a linear network. The input nodes (sensory) correspond to the nine dots and their state (environment). When crossed the state is 1, and when not it is 0. If the pen is currently located on one of the nine dots then its state is set to -1. The network has one output node and no bias node. The connection weights are denoted by

$$\mathbf{w} = [\ w_g, w_l, w_q, w_h, w_m, w_r, w_i, w_n, w_s\ ] \in \mathcal{R}^9, \tag{3.12}$$

corresponding to the numbering in the figure given on page 34, with its first row and column removed. The scoring function is then defined as

$$f_{go}(\mathbf{x}) = \sum_{i=1}^{9} w_i x_i \tag{3.13}$$

where x are the states described. The activity resulting in a state with the highest output node activity, $f_{go}(\mathbf{x})$, is the chosen activity. The process is repeated until all four lines have been sequentially drawn.

The data structure to be varied and selected are the nine connection weights in (3.12). Unlike the symbolic data structures from the previous section, these are numerical. More importantly, it is known that *a small change in the weights will result in a small change in behaviour*. The coding relationship problem, encountered by the symbolic representation in the previous section, is therefore nonexistent here. The neural network is evolved using the evolution strategy where the variation of the connection strength is

$$w_i^{(g+1)} = w_i^{(g)} + \sigma^{(g+1)} N(0,1), \quad i = 1, \ldots, 9 \tag{3.14}$$

and the 'mean step size' is modified, as described previously by (3.4). The selection strategy used is $(10, 50)$ and the number of generations is 50. The result is compared with that of the symbolic representation from the previous section and shown in figure 3.3. The result is the average of 100 independent runs. The results illustrate how the connectionist representation has trivialized the nine-dot problem.

The ability to gain knowledge about the world necessitates an interaction with it. The solutions become 'intelligent' as a consequence. The evolutionary search no longer results in a single best solution but the best behaviour for a given state. For the nine dot problem, a random initial state was chosen, as a consequence the network needed to find the optimal policy for all initial states. Therefore, the solution found can solve the nine dot problem starting

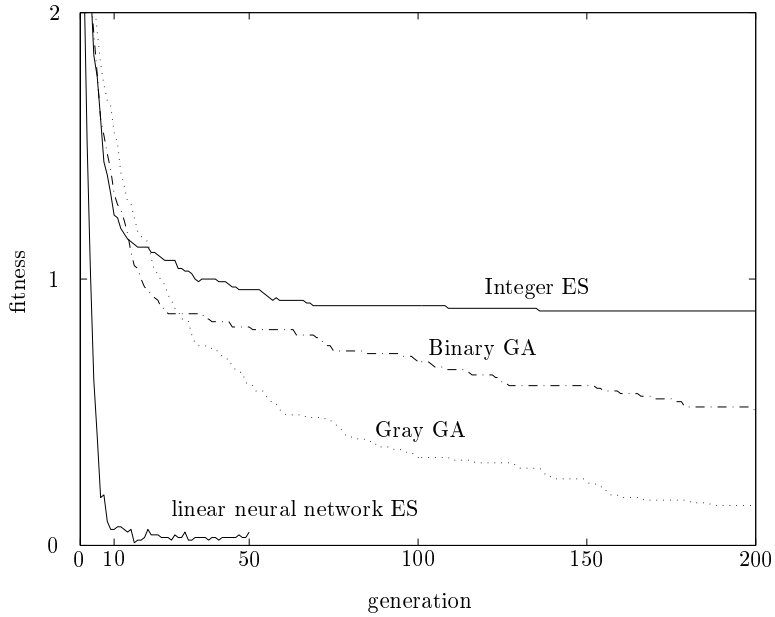


Figure 3.3: Linear neural network compared with symbolic representations for the nine dot problem.

from any of the dots g, l, h, s or y shown on page 34. An example solution is

$$\mathbf{w} = [\ -.67, -.03, 1.28, .71, .54, .89, 1.91, .61, .81\ ].$$

The connectionist representation used has eliminated the need for re-optimization, which would be the case for the symbolic representation for any change in initial conditions. The representation supports generalization and specialization. A small change in information does not require the modification of the representation. This type of representation is called acquisitionally adequate.

3.3 Summary and Conclusion

In this chapter it has been argued that the process of problem-solution abstraction is essentially that of knowledge abstraction. It follows that in choosing a suitable representation for evolutionary algorithms, similar issues to knowledge representation techniques in cognitive science must be addressed. They are [SWC+95, page 153]:

1. *Ease of encoding* – a concise and transparent encoding. Is simple knowledge encoded simply?

2. *Expressive adequacy* – the portion of the world spanned by the representational primitives contains the solution sought.

3. *Acquisitional adequacy* – the modification of information. Is the representation modular? How is old knowledge related to the new?

4. *Inferential adequacy* – the possibility to make logical deductions, or some sort of induction, when faced with uncertainty or missing knowledge.

5. *The scheme and its maintenance* – is everything encoded in advance or can the interpreter fill in the gaps? Can a new primitive be created and used?

Representations are chosen so that they can be reasoned with. The reasoning should reflect the designer's anticipation of how variations in represented situations correspond to fitness change. The properties of the variation operator are as follows:

1. The variation must guarantee an average neutrality in the absence of selective pressure. The variation is random.

2. Small variations correspond to small changes in fitness. This implies that a distance metric space must be pre-specified, and that the fitness landscape is smooth. The variation operator works in the metric space, and should be capable of varying the distances travelled in this space. This introduces the notion of a 'mean step size'.

3. At least one variation should on average be better than the best parent in the population. This is the attempt to maintain positive progress towards the goal state. The likelihood of this is increased when the landscape is smooth and smaller steps are taken more frequently than larger ones. The rate of progress may be maximized by adapting the mean step size.

An intelligent problem solver does not limit itself to direct search, but will attempt to gain domain knowledge during search. This is the knowledge that guides the problem search in an attempt to maximize progress. This knowledge also requires a representation. However, meta-knowledge is needed to search this space. When this meta-knowledge is not available, connectionist representations are the better alternative. They are numerical and it is possible to design variation operators where small steps correspond to small changes in 'knowledge'. Therefore, a more detailed discussion on connectionist representation is presented in the following chapter.

Chapter 4

Connectionist Models

The state-of-the-art evolutionary problem solver will use learning to avoid the need for the designer to provide knowledge for search. In the previous chapter it was claimed that connectionist models are well suited for this task. This chapter expounds the idea of using connectionist models as a problem solving architecture. The architectures presented may be classified into two distinct models of cognition. In the first model 'intelligent' search is partly the result of deliberation over possible courses of action based on some learned knowledge about the world. This is a *deliberative* model of cognition for which the connectionist model presented, in the previous chapter, serves as a typical example. In the second model, there is no deliberation but rather a direct mapping from perception (sensory) to action. This is a *reactive* model of cognition, which avoids the intermediate steps of representation and reasoning. In both models domain knowledge is adapted during search as a result of the random variation of connection weights and selection. There is an additional price paid in computation time for acquiring domain knowledge. This loss in efficiency will hopefully be compensated by the efficiency gained from applying the learned knowledge for search. This is in principle equivalent to the self-adaption of local search step sizes in evolution strategies in order to maximize progress. In some instances, the domain knowledge may be reused such that further problem instances of the same class may be solved in a shorter time. Furthermore, these architectures will (in some cases) eliminate the need for continued search, if problem conditions are modified slightly. Adaptive solutions of this kind are desirable for critical applications where sudden changes in the environment must be met with a compromise solution immediately.

An example for both, a deliberative and a reactive connectionist model for problem solving, is presented. These models serve to illustrate two plausible connectionist architectures for problem solving. It will not be claimed that the architectures designed are optimal in any sense. They serve only to illustrate the novelty of using connectionist models for evolutionary problem solving. The claims made will be supported by experimental studies presented previously in [RJ99a, RJ99b, RJ00].

4.1 Deliberative

Consider a production system, a program structure consisting of a simple set of *production rules*, of the form: *if condition – then action.* The condition part of a production rule consists of a series of condition elements, which describe the conditions that must be true for the rule to be valid. The action part of the rule describes the actions that are taken should the rule fire. Production systems are convenient to program and have been shown to be computationally complete [Pos43]. When building a production system, it must be known what conditions are pertinent and what the appropriate actions are for those conditions. The deliberative connectionist model is equivalent to a production system where these conditions are evolved and, therefore, must not be supplied by the designer. Production rules may be described by feed-forward connectionist models where data attributes are assigned input nodes, target concepts are assigned output units, and intermediate concepts or hypotheses are hidden units.

When using a symbolic production system all rules must be pre-specified. In problem solving there may be conflicting opinions on what rules should be used. Hence many different procedures may be suggested for the same premise. The production system then contains rules with equivalent conditions but different action parts. Purely legal actions are those that have no other condition than that of being valid. A genetic algorithm which resolves rule conflicts by evolving rule patterns was first implemented in Holland's classifier systems [BGH89]. A symbolic string is formed by concatenating production rules. The rules are then fired consecutively by reading the string from one end to the other. In this model a gene represents a set of activity or the behaviour of an organism. Each locus is occupied by alternative forms of a gene, which are known as alleles of one another. Each allele corresponds to

a unique production rule. The allele will at any given locus determine the action to be taken. If the activity suggested is illegal, the interpretation must be relaxed and rule conflicts are restored. Generally, production systems solve resolution conflicts by ranking the rules according to some priority. This type of predetermined priority may be a source of bias for the search. The bias could be avoided if a valid rule is chosen in a random manner. Figure 4.1 shows an example of a genetic string of preferred production rules used by a production system. The third gene from the left suggests a rule that cannot be fired because its actions are illegal. This is represented by a dashed box in the figure. The production system will select either the default rule "A" or some other random valid rule "?" instead of "c". The consequences of selecting the different production rule may have varied effects on the validity of other suggested rules in the string as shown by the dashed boxes in the figure. This is the brittle nature of a symbolic production system.

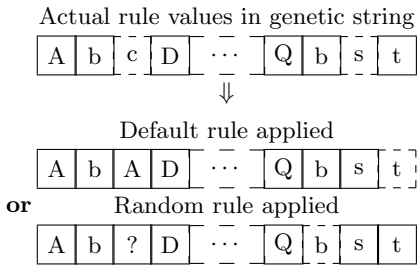


Figure 4.1: An example genetic string of rules is pictured, the third rule from the left requires a interpretation since rule "c" is illegal – denoted by a dashed box, either default rule "A" or a random valid rule "?" is fired instead.

Hinton and Nolan [HN87] denoted adaptive alleles as learnable "?"s. This notation is adopted here also as shown in figure 4.1. The rule fired is randomly chosen when a "?" is encountered. An allele of this type may also be added to the genetic string to force 'symbolic learning' between generations. The number of times the string is evaluated denotes the number of learning steps taken. The same string will produce varying solutions and the fitness value of the best solution found is returned. The learning process influences, therefore, only selection. Baldwin called this *organic selection* [Bal96]. Individuals possessing the ability (plasticity) to increase their fitness through

learned behaviour have a greater chance of being selected. This is the first step of the so called Baldwin effect, the second is *genetic assimilation* of the learned behaviour. There are two necessary conditions for genetic assimilation to occur: high correlation between genotype and phenotype spaces, and high relative costs for learning [Mar96]. The increase in an individual's fitness may vary if learning contains a stochastic element. Consequently, the cost of learning is observed in the unreliable increase in fitness, which puts pressure on a more stable behaviour. As a result the number of "?" in the population will tend to dwindle over generations. Similarly, if random rules are applied to illegals rather than some default hierarchy and the same learning strategy is applied, there will be pressure on the individuals to become 'more legal' during evolution. Initially, however, it is expected that symbolic learning of this kind is favoured, but later selected against [Mar96].

In the symbolic production system all described rules are predefined. They reflect the designer's idea of the most plausible rules for constructing solutions to the problem. It is then effectively the symbolic string that determines the activities performed by the organism. There is no feedback from the environment to the organism as pictured in figure 3.1 on page 29. When a connectionist model is used, complete rules are not supplied by the designer. The designer need only describe the organism's environment and activities. It is then the environment, which proposes the activity of the organism. For example, when a cell encounters an increased concentration gradient of molecules of lactose, certain genes will be activated that allow the organism to feed upon this food source. The connectionist model will learn the appropriate activity for any given environmental state. Here the gene coordination is modelled by simple feed-forward neural networks. An analogy is drawn between a neural network and a network of interactions among information macromolecules responsible for gene coordination [Zuc97]. As a result evolutionary problem solving becomes an adaptive search. The approach produces *plastic* solutions, which are solutions capable of immediate self-organization in the event of an environmental change. If an allele is deleted, other genes will alter their expression pattern so that, a solution with (near) equivalent fitness is obtained. This is accomplished through the local gene interaction networks that coordinate gene expression. Genes regulate each other's activity directly or through their products via these networks. Like living cells, the solutions will differ because they have dissimilar patterns of gene activity, not because they have

different genes [Kau91].

The gene coordination network's task is to determine, which gene is to be expressed as a function of the environmental state of the genome. As genes are expressed, their products change the environment. Through the environment or directly, genes are capable of regulating the activity of other genes and so the regulation of transcription is a result of transcription itself. There is no predetermined environmental problem for which there is a solution; the genome *constructs* the environment and hence determines both the solution and problem simultaneously [Lew82]. The construction and reconstruction of their environment is, however, constrained by what they already are. The genome alters its environment based on patterns of the world, which are presented to the gene coordination network. The network consists of nodes and connections. The nodes are simple processing units whose activation is the weighted sum of their input from the environment and from other nodes. Knowledge is stored in the connections among the processing units and learning takes place through the adjustment of their connection strengths.

Each state and corresponding response could be considered in isolation by the network, if an absolute value judgement is given. The response strength, gene activity, would then be an intervening variable, reinforced by some function of the individual's fitness. In essence, the network would be attempting to predict the resulting individual's fitness, based on the current environmental state and actions taken. Formulating reinforcement in isolation is, however, not a simple procedure. An evolutionary approach, such as that proposed in [CF99] and the one presented in the previous chapter (see section 3.2.2), produce absolute network output, but only make sense for the ranking of alternatives. It is believed that choice procedures may provide a better measure of the effects of reinforcement. The measures of 'absolute values' are only a result of choice dynamics [Wil94], which is the approach taken here. Gene expression is the result of choices made from a set of competitive compromises.

In figure 4.2, a section of the genome model, depicted as a genetic string, is illustrated. Two different loci (l and m) are shown. An individual is imagined to contain multiple alleles. In general, however, living organisms have only two alleles, although a greater variety exists within the population. In the model, multiple alleles will compete for the right to be expressed, but only two of them at a time. The competition is resolved by the gene coordination network. As for any connectionist model, assumptions must be made about the number of

nodes, arrangement of connectivity and their interaction with the environment. Since the network is a competitive or choice network, the input will be the difference between two competing environmental states associated with the alleles. An array of environment state differences at locus l is denoted by $\Delta E_l^{i,j} = E_l^i - E_l^j$, where allele i is competing with allele j. The environmental differences are connected to a hidden layer of nodes, which are connected to two output nodes as shown in the figure. The activations of the two output nodes – $\mathcal{O}_{\text{lhs}}$ and $\mathcal{O}_{\text{rhs}}$ – are real numbers between 0 and 1. The node with the higher activity is chosen. For each competition performed, two separate inquiries are made as to whether an allele should be chosen over the currently successful one. The results must be consistent and if the challenge of allele j is to be successful over allele i then: $\mathcal{O}_{\text{lhs}}^{i,j} < \mathcal{O}_{\text{rhs}}^{i,j}$ and $\mathcal{O}_{\text{lhs}}^{j,i} \geq \mathcal{O}_{\text{rhs}}^{j,i}$ must be satisfied, where $\Delta E^{i,j} = -\Delta E^{j,i}$. If the above inequality does not hold, allele i remains successful. With no useful information available from the environment, the network may respond in a contradictory manner and the successful gene will hold its position independently of changes in the environment. To achieve this, the model must remember, which allele was expressed previously at that locus. Loci which are sensitive to the environment are known as *plasticity loci*, and those insensitive to the environment *mean loci*. A genome containing only plasticity loci has been labelled the *pleiotropy* model by Scheiner [Sch98]. The pleiotropy model is a special case of the *epistasis* model, which contains also mean loci.

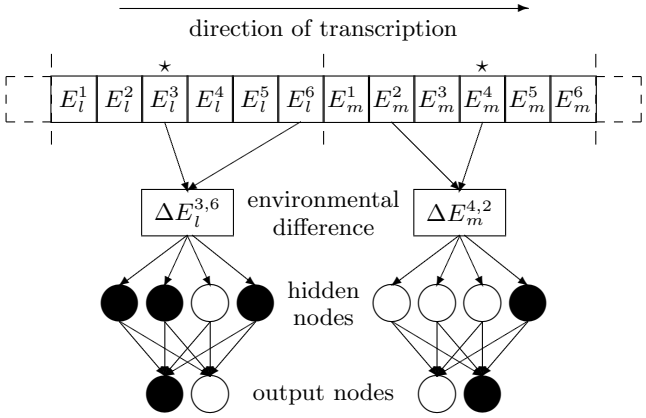


Figure 4.2: Genome model. Competing alleles at loci l and m where ($\star$) marks the currently successful allele.

Two types of perturbations are possible at a given locus. The first is a *minimal perturbation*, where a previously expressed allele is forgotten and therefore will not hold through to the next generation. If the locus is a plasticity locus, it is likely that the allele will regain its position. If, however, the locus is a mean locus, the allele will loose its position. The second is a *structural perturbation*, where the gene coordination network itself is modified. This may be a modification of the network architecture or of the connection strengths. Viewed in this manner, a structural perturbation constitutes learning. The names for these perturbations are taken from [Kau91]. Additionally, it is possible to extend this model so that a previously expressed allele, and/or other alleles, may be deleted (removed from a locus) and others added.

The entire genome is represented by a string containing ℓ loci as shown in figure 4.2. The string is transcribed systematically from left to right, processing one locus at a time. Within each locus there exist m alleles competing to be transcribed. Random competitions are held, where alleles compete with the currently successful one for its right to be expressed. The competition continues until a maximum number of competitions is reached or some time has elapsed. Each locus is occupied by a data structure containing a neural network for gene regulation and a list of competing alleles. The data structure may also hold information about which allele was previously expressed, training history for the network, etc. In the model presented the previously expressed allele will be remembered and in the next generation will be the default successful allele. If, however, this allele happens to be illegal in the following generation, a random legal allele is selected as the default, which then other alleles must compete with.

There exist a number of possible methods for evolving the connection strengths of the gene coordination network. A simple and effective way would be to apply an evolution strategy as done in the previous chapter. A perhaps more elaborate method would be to use incremental learning procedures such as the methods of temporal difference [Sut88]. Indeed, the gene coordination network may be thought of as having the ability to predict the future based on the organisms environment and actions. Supervised learning, using backpropagation, is another possible learning strategy. Training data for learning may be sampled from the current population. During transcription, the environmental states (associated with the expressed alleles) are recorded in the loci data structure. This may be used as input training data. Once the genome

has been expressed completely, its total fitness is known. The output training pattern is then formed using this fitness. From a population of λ unique individuals a training set of the size $\lambda(\lambda - 1)$ can be sampled. Should there be any useful information in this data, the network may learn it and hopefully generalize this knowledge. New individuals may be formed using standard recombination operators. Networks may be exchanged between two or more individuals using one, two, or multiple crossover sites. Mutation will play an important role in maintaining a diverse environment. Minimal perturbations will attempt to knock out successful alleles. It is expected that minimal perturbation will have less influence on plasticity loci. Structural perturbation will reset the connection strengths for the gene coordination networks and will permanently damage loci. It is also possible to view the training of a network as a more complex structural perturbation. The following section will illustrate the methods discussed on some well known benchmark problems for job scheduling.

Experimental Study

In this section, the evolutionary problem solving architectures, described in the previous section, are tested on two frequently studied job-shop scheduling problems. They serve to illustrate further the difference between the symbolic and connectionist representation. The complete studies are presented in two separate papers devoted to *genetic production systems* [RJ99b], and *gene coordination networks* [RJ99a] respectively. In the symbolic model, the algorithm designer supplies the domain knowledge by hand. For the deliberative connectionist model, domain knowledge is acquired during search using the gene coordination network described in the previous section.

In the job-shop problem, a set of jobs must be scheduled on a set of machines. Each job consists of a number of operations, which are processed on the machines in a predetermined order. The goal is to assign jobs on machines such that the overall production time, the makespan, is minimal. Schedules are formed by systematically assigning one job after the other at its earliest convenience. In the experiments an allele will determine the activity of the organism. The activity is jobs assignment.

Symbolic

The classic 10 job 10 machine job-shop scheduling problem posed by Fisher and Thompson is studied [FT63]. There exist numerous rules for scheduling. A survey of one hundred such rules was given by Panwalker and Iskander [PI77]. For the simulations performed in this section, three priority scheduling rules are used. They are, assign job; (A) operation with earliest completion time, (B) first in, and (C) operation with shortest processing time. Since the machine order is fixed for each job, the legal moves are limited to selecting any of the uncompleted 10 jobs for scheduling: $a, b, ..., i, j$. Adaptive or learnable alleles "?" are added to give the individuals the plasticity to increase their fitness through learned behaviour. These alleles may be regarded as adaptive rules. Two approaches to learning are examined: Lamarckian learning and the Baldwin effect. The number of learning steps is one hundred. The Baldwin effect uses the "?" alleles to target its learning. The Lamarckian learning is a simple hill-climber where a random allele is mutated and kept if the modification is an improvement. This is also repeated one hundred times. The Lamarckian learning modifies the genotype whereas the Baldwin effect is a result of the selection process.

Since there are in all 100 operations to be scheduled, the genetic string of rules will be of the equivalent length ℓ. The production system is simply built by reading the symbolic string of production rules from left to right. The scheduling rules will at all times be valid and the legal moves only when the job it represents has not been completed. Two approaches are taken to decipher illegals. The first fires the default heuristic A. The second approach is to interpret illegals as learnable alleles "?" and apply one hundred learning steps. The second approach will be referred to as random decipher with Baldwin effect. The four different evolutionary approaches for the scheduling problem are then: 1.) without learning, 2.) Baldwin effect random decipher of default, 3.) Lamarckian learning, and 4.) Baldwin effect with "?" alleles added. All methods use by default rule A for illegals with the exception of approach 2.) which randomly deciphers illegals.

The evolutionary algorithm used to evolve the rule patterns is a simple genetic algorithm. Individuals survive to the next generation by competing in a tournament. Two individual are sampled randomly out of the population and the better of the two is passed on to the next generation. The process

Scheduling rule	A	B	C	Default	None
Without learning	22.8	11.5	4.9	21.1 (A)	39.6
	(4.2)	(3.6)	(2.3)	(4.1)	(4.4)
Baldwin effect	30.3	13.7	6.1	10.7 (?)	39.2
random decipher	(4.2)	(3.5)	(2.3)	(2.4)	(4.3)
Lamarckian	19.7	9.9	1.0	11.6 (A)	57.9
	(3.5)	(3.0)	(1.0)	(3.0)	(4.2)
Baldwin effect	20.8	12.8	4.6	19.9 (A)	42.0
with "?"	(3.7)	(3.6)	(2.4)	(3.9)	(4.2)

Table 4.1: The mean and standard deviation of the percentage of time a rule is used to build the best schedule found.

is repeated until there are as many individuals in the next generation as in the previous. Sampling is done without replacement. All individuals in the new generation undergo recombination where alleles are randomly exchanged between any two individuals. This is known as uniform crossover [Sys89]. An average of 40% of the alleles on a string will be crossed in this manner. Following recombination mutation is applied to the string with a probability of $1/\ell$. Mutation modifies alleles by giving them other randomly chosen values. The population size used in the runs is one hundred and the best individual will always survive to the next generation (elitist).

One hundred independent runs are performed. The average percentage of scheduling rules used in the final populations is given in table 4.1. Of the scheduling rules used rule A is most frequently applied followed by B and then C. The number of times that legal moves are preferred to the priority scheduling rules are, however, high. This would tend to indicate that the scheduling

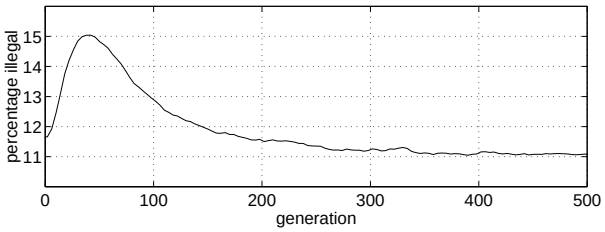


Figure 4.3: Average number of illegal alleles in the population as a function of generation for the random decipher method.

rules applied are inefficient. The Lamarckian runs, which effectively 'legalize' the solutions during learning, still contain 11% illegal rules on average. This also applies for the random decipher method and tends to indicate that the 1% mutation rate, mutating a single rule in the string, has the effect of making an average of 11 rules illegal. Figure 4.3 shows how the mean number of illegals develop over generations. The characteristic 'hump' is observed where learning (the illegals in this case) is favoured for initially and later selected against.

In table 4.2 the statistics for the best solutions found for the runs is presented. The difference in the performance of evolution without learning, as to the random decipher method, indicates that the default reasoning approach for interpreting illegals is on average not the best method. Learning requires, however, more computation time and is proportional to the number of learning steps taken. The evolution that was stopped at one thousand generations and the improvement after 200 generations was minimal. The mutation operator can cause a disastrous change in an individual's fitness value. The local minima found by Lamarckian learning produce, therefore, too diverse rule patterns. The Lamarckian runs are extremely noisy and, hence, unable to converge.

Makespan	Best	Mode	Mean	Worst
Without learning	939	982	974	997
Baldwin effect random decipher	940	961	967	985
Lamarckian	977	998	994	1014
Baldwin effect with "?"	958	963	966	985

Table 4.2: Makespan for the 100 independent runs taken, the optimal solution is 930.

Discussion on the symbolic approach

In this study, a genetic production system for optimization and search was presented. The domain specific knowledge is introduced to the algorithm without having to modify the search process. This was done by letting the genetic algorithm evolve rule patterns, which represent the domain knowledge.

The variation of rules is performed without any regard to their meaning. Although the computational evidence from a single test problem is too small to serve a definite conclusion, it does validate some discussion. The results of the experiments indicate that evolution favours some rules more than others. Different stages of search favour different rules; this may be observed from the adaptive alleles examined over the generations. The frequency of purely legal moves preferred may be an indicator of the effectiveness of the heuristic rules suggested by the designer of the algorithm.

The Baldwin effect was the superior one of the two learning approaches studied. The Lamarckian learning approach was too disruptive for the rules being processed by the genetic algorithm. Alternatively, it may be expected that Lamarckian learning forces premature convergence. Learning, using the adaptive alleles, converged too rapidly and the best solution found is similar to the mode found using the random decipher approach. The algorithm without learning did not perform badly considering it found the best solution and has the least computation time. The algorithm that assimilated legal solutions via Baldwin effect and random deciphering of illegals is on average the best. This indicates that improvement may be expected for a genetic algorithm without learning if the illegals are deciphered in a more intelligent manner.

Connectionist

In the symbolic study it was found that designing general rules for problem solving may be difficult. Its success is highly dependent on the designer's insight into the problem being tackled. Different stages of the search also require different rules. Furthermore, the variation of a single rule may have drastic consequences on the validity of other rules used in the production system. It will now be demonstrated how most of these difficulties may be overcome by using the gene coordination network discussed in the previous section. The evolutionary algorithm used in the experimental study may be summarized as follows [RJ99a]:

1. Initialize population. Connection strengths for the gene coordination network is randomly set $\in [-1, 1]$ at each locus.

2. Loop through the following steps until the termination criterion is met:

   (a) Transcribe loci by performing m random competitions at each locus

with the successful allele. Record which allele was transcribed and its environmental state. Compute individual's fitness.

(b) Store elite individual.

(c) Select individuals from the current population using tournament selection to form the new population for the next generation.

(d) Train gene coordination networks in the new population at loci which have not been trained before with a probability P_l. The P_l parameter will essentially dictate the initial rate of learning. Training samples are taken from the old population. When a network has been trained a training flag T for that locus is set to *false*.

(e) If the elite has been lost inject a copy into the new population (elitist).

(f) Shuffle new population and perform a two point crossover in order to exchange loci between selected individuals. The probability of crossover is P_c.

(g) Perform a minimal perturbation with probability P_m by exchanging the currently successful allele at a locus by another randomly chosen allele. Perform a structural perturbation with probability P_s by randomly resetting the connection strengths for a network at a locus. In both cases set the training flag T to *true*.

The test problems taken from [MT63] are of sizes 6×6 and 10×10. The optimal makespans are known and are 55 and 930 respectively. As a schedule is being constructed, a number of features of the solution become available. These are the environment states which may be associated with a job (allele). Alleles corresponding to jobs that have been completed are illegal and will not compete for expression. For the simulation performed three environment states are used: The time a job is available, the time it may be expected to finish and the total time still needed to complete the entire task (work remaining). These environment states are used as input data for the gene coordination network, which has one hidden layer with 6 nodes. For the training of the network the output pattern used is $f_i \leq f_j$ for the left output node and $f_i \geq f_j$ for the right output node, where f is the fitness value corresponding to environment states i and j. Note that these are Boolean operators and that the problem is one of minimization. A sample size, twice the size of the population, is extracted as discussed in the previous section. Samples for which $\Delta E = 0$ are ignored.

The training algorithm used is back-propagation. The log-sigmoid transfer function returns the activations of the nodes squashed between 0 and 1. The mean square error is used as the performance function. A network is trained for 100 epochs and, if it survives, it may be trained further in some future generations. A population size of 30 is used for the 6×6 problem and 50 for the 10×10 problem. The probability for crossover is $P_c = 1$, for learning $P_l = 0.2$, and for minimal perturbations $P_m = 1/\ell$. The probability of structural perturbation for the smaller problem was zero. For the larger problem, it was found to be useful to add a very slight chance (0.01%) of a structural perturbation and an increase in minimal perturbations. Ten independent runs were taken for each problem. For the 6×6 problem, the optimal solution was found within 40 generations. The larger problem was stopped after 200 generations and the results varied from a makespan of 960 to 990.

Results for a typical solution found for the two problems will be presented. To test the plasticity of the solutions found, all loci are systematically perturbed by deleting the successful allele and putting another in its place. This can be done in $m - 1$ different ways at each locus. The result of this is that

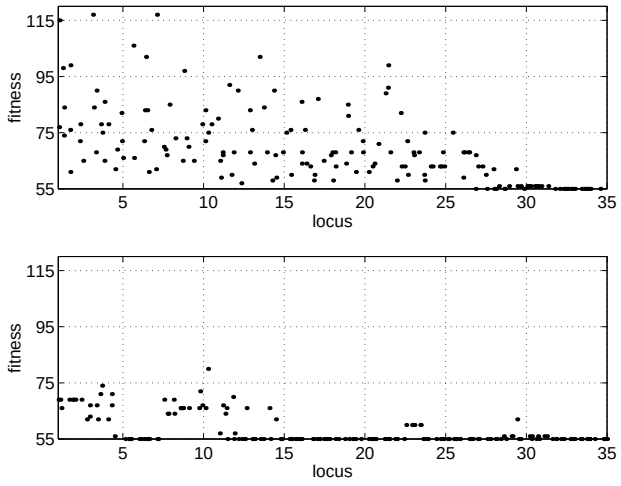


Figure 4.4: Perturbation results. The top figure shows the fitness at a locus which has had its successful allele deleted and all loci to the right of it have been structurally perturbed. The bottom figure shows the fitness at a locus which has only had its successful allele deleted at that point.

on average 50% of the loci are immune to the perturbation for the 6×6 problem. Either other loci performed its function or another phenotype was produced that gave the same fitness value. Figure 4.5 shows six different solutions resulting from these perturbations. The bottleneck remains on the same machine but some job orders have changed. The means by which a solution is constructed as well as the problem itself are redundant. The bottom plot in figure 4.4 shows which loci are most immune to the perturbation by deletion. Regions that are in the start of the string are more susceptible to damage. This is reasonable since they must essentially predict much further into the future. To eliminate the possibility that this is a result of some other factors, such as the constraining of the problem-solution space, all networks to the right of the damaged locus were structurally perturbed. The result of this is shown in the top plot in figure 4.4 and illustrates how the fate of the last m loci is determined independent of the network when $M - m$ loci have been expressed.

Figure 4.6 shows the choice landscape for the 6×6 problem where the difference in the work remaining has been set to zero. The horizontal axis

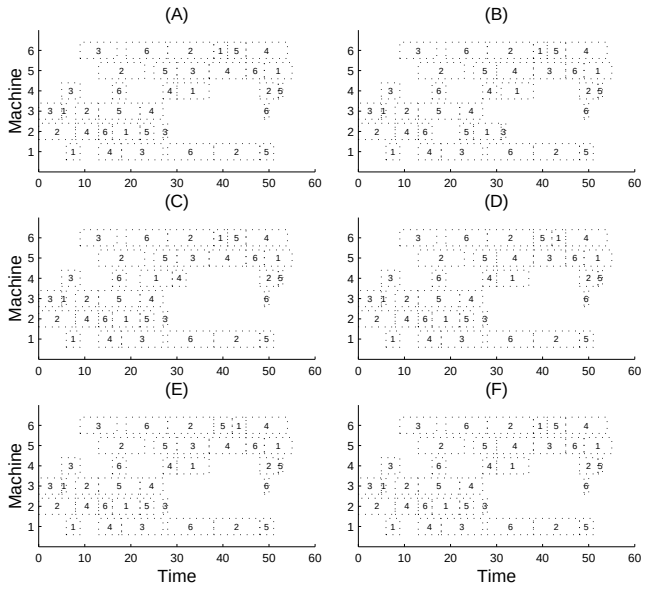


Figure 4.5: Gantt charts. Six different solutions obtained due to perturbation by deletion for the 6×6 job-shop problem's elite solution.

displays the difference in time of completion and the vertical axis when a job is available. On both axes the scale is from -50 to 50. The landscape is the difference between the two output nodes, $\mathcal{O}_{\text{lhs}} - \mathcal{O}_{\text{rhs}}$. The darker regions are positive values whereas the lighter (white) are negative. The network, for example at locus 24, will prefer a job (allele) with a sooner completion time and later availability, but at locus 34, early completion time is preferred regardless of availability. Some of the nets are contradictory which will make their corresponding loci a mean loci. Examples of these are the first and the last locus. It is understandable that the last locus could be a mean locus since its fate is always decided. The first loci has one environment state always equal to zero. When we examine the choice landscapes also with respect to the work remaining, we find that most loci are of the plastic type.

The same perturbations by deletion were performed on a 10×10 solution. The result was that on average 60% of the loci were immune to deletion. The results are depicted in figure 4.7. When an arbitrary gene is deleted, how many genes alter their expression pattern? The single perturbation brings about a cascade of change in the patterns of gene activation. If the mutations

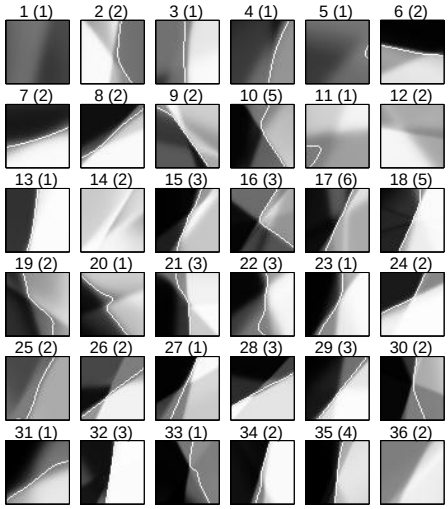


Figure 4.6: Network landscapes. The choice landscapes for the gene coordination networks per locus. The locus number is given above each map with the number of time the network has been trained during its evolution over generations in parenthesis.

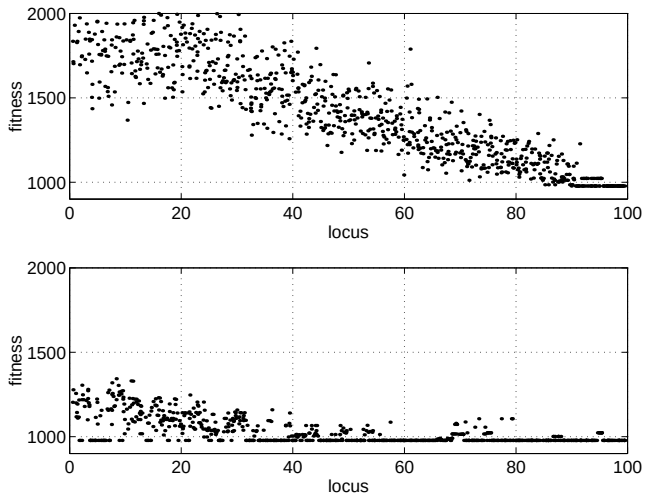


Figure 4.7: Perturbation results. The top figure shows the fitness at a locus which has had its successful allele deleted and all loci to the right of it have been structurally perturbed. The bottom figure shows the fitness at a locus which has only had its successful allele deleted at that point.

are neutral the resulting solution – the phenotype – remains the same or the phenotype has changed but its fitness remains unchanged. In figure 4.8 the number of genes which alter their expression is plotted against the locus where the deletion occurred. Only the cases where the equivalent elite fitness was obtained is shown. Commonly it suffices that just one additional gene changes its expression to compensate for the damage. Changes for up to 19% of the string are also observed!

Discussion on the connectionist approach

An application of a general epistasis model for adaptive search and optimization was presented. The development of the model is like creating a language whose rules have not been formalized and where there is no priori definition of 'purpose' or 'sense'. As the gene coordination networks evolve, their meaning develops alongside or with it. Gene expression is controlled by these networks where two alleles are matched up at a time. This is a contingent process.

The proposed method performed well on two job-shop benchmark problems and the solutions found were highly plastic. For plasticity to be selected

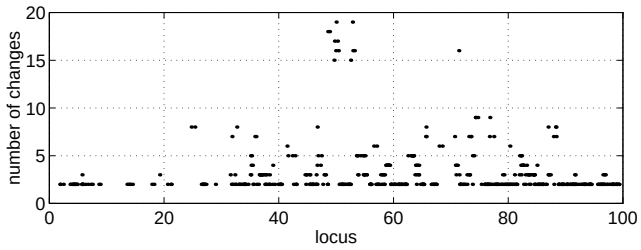


Figure 4.8: Expression changes. The figure shows how many genes have changed their expression as a function of locus where the successful allele was deleted.

for, the environment must be predictable. The redundant nature of both the problem and method by which solutions are built, has been a factor in producing highly fit plastic loci. Formulating gene expression as an 'intelligent' process introduces new possibilities for the role of learning in difficult search problems. Problem domain knowledge is introduced to the evolutionary algorithm without much involvement from its designer.

4.2 Reactive

The deliberative connectionist model, similar to the one presented in the previous section, is suitable when there is a small number of activities to select from. In this section an example of a reactive connectionist model is described for which the model's output is the activity and, is continuous. The activity of the connectionist model is that of a learning rule for the optimization of neural networks. Connectionist models are universal function approximators and, therefore, the space searched already contains a simple gradient descent search procedure like back-propagation. The reactive connectionist model will rediscover, and perhaps improve, known mathematical methods. In essence, it is proposed that a learning rule be part of the neural network structure and, therefore, globally distributed. Although the method is a non-local learning rule, it does not require an 'error' signal to be propagated directly to the neuron in question. The 'error' signal will be channeled through the learning rule, i.e. the network. The viewpoint expressed challenges the idea that 'learning rules' must be local for biological plausibility. Indeed, how can a simple local

rule give rise to anything cognitively significant like "learning"? If it does, it must be the result of some higher-order organization of the system as a whole.

The 'biological implausibility' of learning algorithms is commonly debated in the literature [DG99], for example: "it is unlikely that animal networks employ any of the mechanisms used in training back-propagation networks (gradient descent, stochastic descent, genetic algorithms, etc.) because they all use information from across the entire network to decide how to adjust the value of an individual weight" [Bax92]. It is for this reason that Hebb-like [Heb49] local learning rules are regarded as biologically plausible. However, there do exist many return pathways in the brain although, they may not carry all the information needed for back-propagation learning [Cha90]. Another move towards biological relevance [MAJ91] are reinforcement learning methods [WGM73, BA85]. With these methods, learning may be achieved by a single reinforcer. A part from these arguments, there is the issue of the implementation of the 'learning rule' itself.

Previous attempts for evolving learning rules have relied on predefined functions [Cha90, Bax92, BBC95]. For example, a simple quadratic function (where the delta rule could be found as a special case) with evolvable parameter was used by Chalmers [Cha90]. Chalmers recommends genetic programming for the evolution of even more complicated learning rules [Cha90]. It will show that sophisticated learning rules may be found using the neural networks themselves. It is well known that a neural network is a capable function approximator and for this reason quite able to both approximate and implement any learning rule. The learning rules will be discovered by an evolutionary search method. The implications of such a system may be the following:

- faster trainers for given problem classes
- a new approach to learning
- different types of feedback (reinforcers)
- a greater variety of learning rules

The current work looks at the single layered neural network only. Because the decision boundary is linear, the single-layer network can only be used to recognize patterns that are linearly separable. Experimental studies for some linearly separable problems are presented in the following section. The working principles developed are fundamental to more complicated neural network

architectures. For non-linear problems, architectures similar to that of fixed on-line learning [YCC99] may be used. The concept developed is similar to fixed weight on-line learning, but the weights are not fixed, they are evolved.

A learning rule is a procedure for modifying the weights and biases of a network, this procedure is also referred to as the training algorithm. The weight updating rule at time (t) is given by:

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij} \tag{4.1}$$

where w_{ij} is the weight of the connection from node j to node i. The commonly used learning rule for the change in the weights Δw_{ij} is

$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} \tag{4.2}$$

where E_p is the error for pattern p, or by the delta rule [RHW86]:

$$\Delta w_{ij} = \eta \delta_{ip} o_j. \tag{4.3}$$

The change in the weights is a function of the *learning rate* η which is typically less than 1 (e.g. 0.1 or 0.3). A small learning rate is chosen because the final weights sought should work for all patterns and changing the weights per pattern must be done cautiously. The error term δ_{ip} is the product of the actual error output, which is the difference between the target for node i, denoted by t_{ip}, and the actual output of that node o_{ip}, multiplied by the derivative of the node's activation function. The error for the node is also related to the signal from the sending node, which is o_j. For a pure linear activation function, the output layer is modified by,

$$\Delta w_{ij} = \eta (t_{ip} - o_{ip}) o_j \tag{4.4}$$

and for the commonly used sigmoid activation function by

$$\Delta w_{ij} = \eta o_{ip} (1 - o_{ip}) (t_{ip} - o_{ip}) o_j. \tag{4.5}$$

In other words the learning rule is a function of the input and output node activations and the target value:

$$\Delta w_{ij} = \eta \ f(o_j, o_{ip}, t_{ip}). \tag{4.6}$$

This function may be approximated by a neural network and hence the proposal to use another part of the network to model the learning rule. Computing the error term for hidden nodes is usually done by letting them inherit the errors of all nodes that they activate. Therefore a more general learning rule would take the form:

$$\Delta w_{ij} = \eta \ f(o_j, o_{ip}, \delta_{ip}) \tag{4.7}$$

where now δ_{ip} reflects in some way the 'error' assigned to w_{ij} by the principles of credit and blame [RHW86],[WGM73], or reinforcement [BA85].

The neural network approximating equations (4.6) or (4.7) would have an output value of $\Delta w'_{ij}$, which may be either positive of negative. For this reason, the output node's activation function must allow for both signs. By using the hyperbolic tangent sigmoid transfer function so the learning rule becomes:

$$\Delta w_{ij} = \eta \Delta w'_{ij} = \eta \left(\frac{2}{1 + \exp(-2 \sum_k \omega_k a_k)} - 1 \right) \tag{4.8}$$

where ω_k are the weights of the learning rule and a_k are the signals from the hidden nodes' activation. Other activation functions may be equally accept-

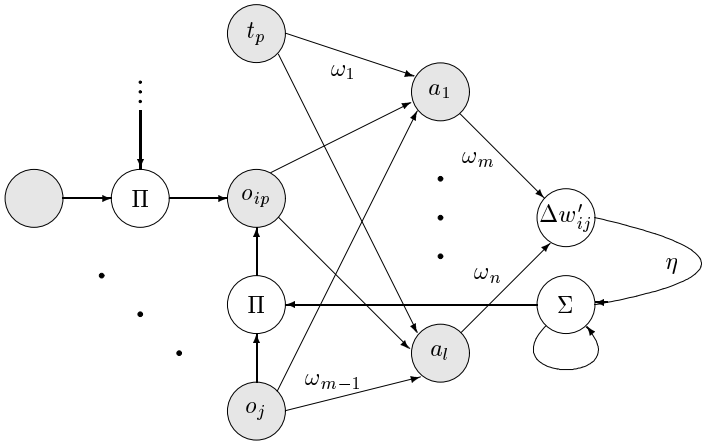


Figure 4.9: The neural network structure (bias is not shown). A Π (multiplier) node and accumulator node have been added to implement the weight update. Both t_p and o_j are sensors or the result of sensory input. The effector is o_{ip}, proposes the appropriate action.

able but (4.8) will be used here. The inputs for the neural network, the learning rule, are the function arguments in equation (4.6) or (4.7). In Chalmers' [Cha90] quadratic function, the weights themselves (w_{ij}) are part of the learning rule, however, there is no reason for including them here. The complete neural network structure is given in figure 4.9 for a learning rule with one hidden layer training a single layer network.

Experimental Study

The evolutionary optimization algorithm presented is based on the evolution strategy, ES(μ, λ), [Sch95]. It follows the typical cycle for the evolution of learning rules [Yao99] and may be formulated as follows:

0. Initialize a population of λ individuals randomly, set the generation counter to $g = 0$ and the desired maximum number of generations G. Each individual is a vector $(\omega_1, \ldots, \omega_n, \sigma_1, \ldots, \sigma_n)$ where $\omega = (\omega_1, \ldots, \omega_n)$ are the neural network weights for the learning rule, and $\sigma = (\sigma_1, \ldots, \sigma_n)$ are the 'mean step' sizes adapted during the search. The initial weights are randomly distributed between -1 and 1 and the initial mean step sizes are $2/\sqrt{n}$.

1. Use each learning rule $\omega_i \; \forall \; i \in \{1, \ldots, \lambda\}$ to train a neural network on a given set of training patterns. The initial weights $\mathbf{w}$ of the network to be trained are randomly distributed between -1 and 1. A number of different training patterns are used. The fitness of the training algorithm, ω_i, is the averaged 'mean root square error' of the last epoch.

2. Select the best μ individuals to generate on average λ/μ offspring for the next generation.

3. The 'mean step sizes' are updated according to the log-normal update rule: $i = 1, \ldots, \mu$, $h = 1, \ldots, \lambda$, and $j = 1, \ldots, n$,

$$\sigma_{h,j}^{(g+1)} = \sigma_{i,j}^{(g)} \exp(\tau' N(0,1) + \tau N_j(0,1))). \tag{4.9}$$

4. The neural network weights are varied using the normal distribution:

$$\omega_{h,j}^{(g+1)} = \omega_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1) \tag{4.10}$$

5. Increment the generation counter $g = g + 1$ and go to step 1 unless the maximum number of generations (G) has been reached.

The algorithm presented is a simple evolution strategy which does not use any recombination. Some of the factors which we expect will influence the search for learning rules are:

- whether the training samples can be learned by the chosen network architecture
- the number of training samples used (the set must cover the problem space 'sufficiently')
- the number of epochs used in training
- the complexity of the learning rules (the number of hidden nodes)
- repeated training on the same task in order for the results to be statistically significant.

Some of these issues will be examined in the following experimental study.

Linearly Separable Tasks

Linear prediction is a task for which single-layer networks are commonly used. Here experiments with Chalmers' [Cha90] eight linearly separable tasks given in table 4.3 are performed. The tasks have only one output unit since a single layer network with more than one output unit is equivalent to a number of disjointed networks with a single output unit [Cha90]. Task 3 is, however, not linearly separable and therefore will not be used in the experiments. The effect of learning rule complexity and training time is examined in this study.

The input and outputs are zeros and ones and the activation function for the output node of this single layer network is the log-sigmoid activation function:

$$o_{ip} = \frac{1}{1 + \exp(-\sum_j w_{ij} o_j)} \tag{4.11}$$

Both the learning rule and the network being trained use a bias which is not shown in figure 4.9.

Table 4.3: Chalmers' eight linearly separable tasks [Cha90].

o_1	o_2	o_3	o_4	o_5	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
1	1	1	1	1	1	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	0	1	0	1
0	1	1	1	0	0	0	1	0	1	0	1	0
1	1	0	0	0	0	1	1	1	0	1	1	1
1	0	1	0	1	1	0	0	0	0	0	1	1
0	1	1	0	0	0	0	1	0	1	1	0	1
0	1	1	1	1	1	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0	1	0	1	1	1
1	0	0	1	0	0	0	1	1	1	0	1	1
1	0	1	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	0	0	1

The evolution strategy, ES$(30, 200)$, was run for 400 generation on the following experimental setups:

	10	20	30	hidden nodes
10	A	D	G	
30	B	E	H	
60	C	F	I	
epochs				

where $A, \ldots, I$ denote the experiment number. Each of the experiments was conducted five times with a learning rate of $\eta = 1/2$.

In all of the experiments similar response surfaces were evolved. The response surface for experiment F is compared with that of the delta rule given by equation (4.5). This comparison is shown in figure 4.10 for a target value of 1 and in figure 4.11 for a target value of 0. The figures also illustrate the actual change in the network behaviour by multiplying the change in weight by the input signal o_j.

The striking difference between the evolved networks and the delta rule is that the delta rule does not change its weights when the output node is zero or one, regardless of the error signal. For the evolved network, when the target value is $t_{ip} = 1$ (see figure 4.10), the change in weights is minimal when

Table 4.4: The last epoch error for the best rules found for each experiment.

F	I	G	B	E	H	A	C	D
0.0059	0.0070	0.0086	0.0120	0.0127	0.0137	0.0138	0.0227	0.0240

$o_{ip} \approx 1$, but when $o_{ip} \approx 0$ the change is maximal, but again little for $o_j \approx 0$. Similar behaviour is observed for when the target value is 0 (see figure 4.11), that is, maximal change when $o_{ip} \approx 1$, but again minimal for $o_{ip}, o_j \approx 0$. When multiplying the input signal with the weight change, as shown in both figures 4.10 and 4.11, it is noticed that the major difference is that *the delta rule does not modify weights when the error is at the extremum, however, the evolved network does.*

The best evolved learning rules for each of the experiments conducted was used to train a network on Chalmers' tasks again but now for 200 epochs. The last epoch error for these learning rules, sorted in order of best to worst performance, are given in table 4.4 above. None of the training rules diverged but fluctuated slightly about a fixed error level. From these experiments is

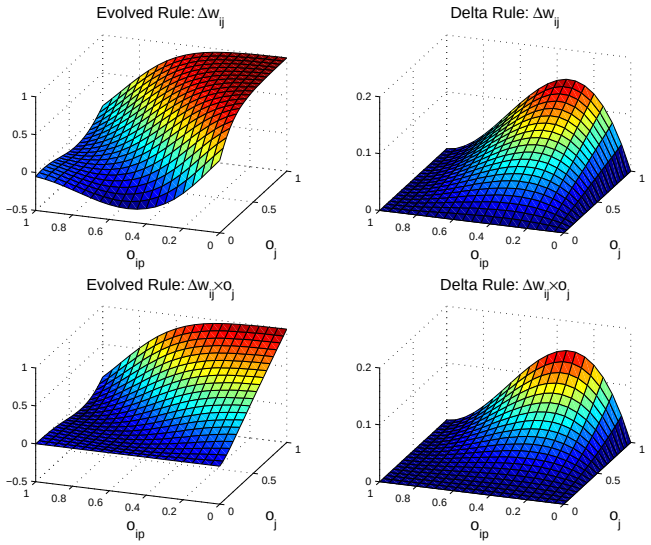


Figure 4.10: The target value is $t_{ip} = 1$ and the learning rate is one. The learning rule has 20 hidden nodes and a training time of 60 epochs during the evolution.

seems that the more complicated learning rules, with 20 and 30 hidden layers, performed better. Also, a longer training time resulted in a learning rule with lower errors.

Figure 4.12 shows the evolution of error as a function of epoch for the best learning rule from experiments A and F. For comparison these tasks were trained using the delta rule with a η value five times higher than that used for the evolved rule (see figures 4.10 and 4.11). Indeed, a very large η value is needed for the delta rule to achieve the same learning rate as the evolved rules, however, in this case the weights do oscillate [MMR97, page 81].

The performance of the evolved rules on other unseen tasks were also good, even for Chalmers' third task (which is not linearly separable) all but one pattern was solved. A similar result is obtained using the delta rule on this task. As a further example consider the following two cases: the logical `OR` function and the logical `AND` function. The training results for these cases are depicted in figure 4.13. It is clear from this demonstration that the evolved learning rules are biased.

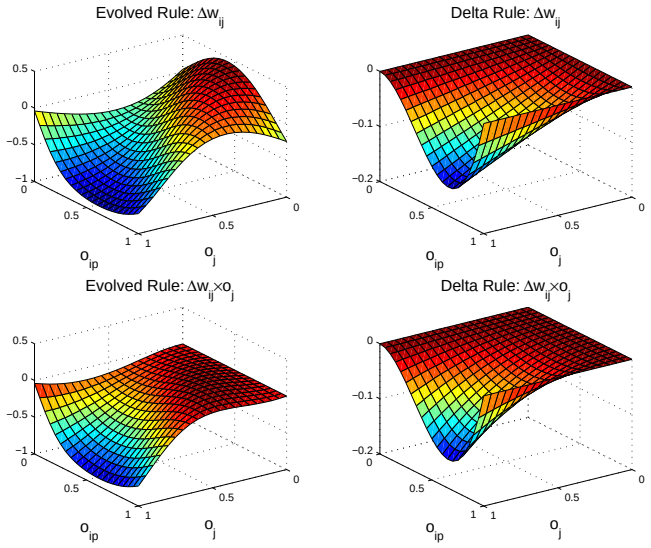


Figure 4.11: The target value is $t_{ip} = 0$ and the learning rate is one. The learning rule has 20 hidden nodes and a training time of 60 epochs during the evolution.

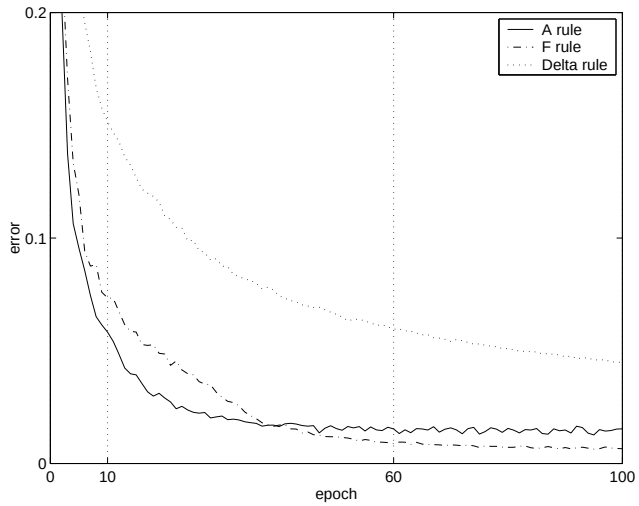


Figure 4.12: The learning process for the evolved rules from experiment A and F and using the delta rule on Chalmers' tasks.

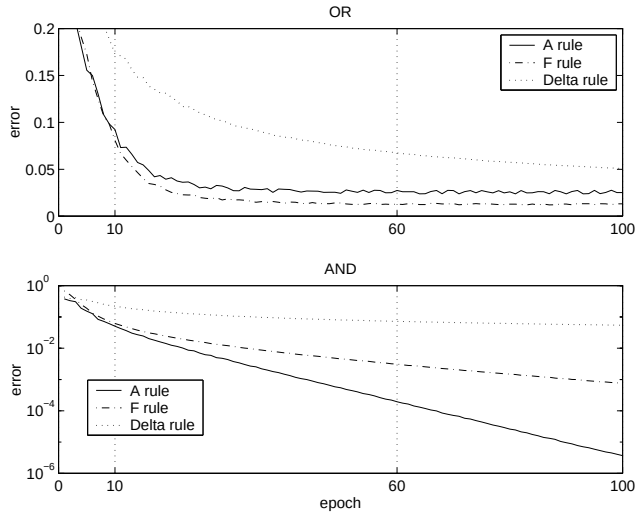


Figure 4.13: The learning process for the evolved rules from experiment A and F and using the delta rule on the logical `OR` (top) and `AND` (bottom) functions.

Discussion

The experimental results may be summarized as follows: 1) a neural network is capable of representing a learning rule, 2) the evolved rules are fast neural

network trainers, 3) the training algorithms do not diverge when a longer period of learning is used, 4) the accuracy achieved is limited, and 5) the learning rule is biased. The results indicate also that the shorter the training time given to the evolved learning rule, the less accurate the results but faster the learning. Conversely, the longer the training time the more accurate the results, but slower the learning. This seems reasonable since even an algorithm like back-propagation requires a certain number of epochs to reach a quality solution. All of the learning rules evolved have a limited accuracy and appear to be noisy. However, a network approximating equation (4.5) would also make for a 'noisy' learning rule.

The learning rule is biased. It is 'tuned' to solve a given problem class fast. Even using the back-propagation algorithm, the tuning of learning and momentum rates is often required when new problems are solved. Note that a recurrent neural network structure could be used to model a learning rule with momentum. It is believed that the approach taken here opens up many possibilities for evolving fast problem specific optimizers.

4.3 Summary

In this chapter *deliberative* and *reactive* evolutionary problem solvers using connectionist models were presented. The first model requires some sort of deliberation process before action is taken. The second is reactive in the sense that the action taken is a direct mapping from environment state. Their possible application has also been presented. Having proposed various architectures for problem-solution abstraction, the question of search must now be addressed. The following chapter is devoted to understanding the conditions needed for efficient end effective evolutionary search.

Chapter 5

Evolutionary Search

The previous two chapters are devoted to describing situations, or problem-solution abstraction. This chapter is committed to the operators that transform described situations to a desired one or a goal. This is the fundamental heuristic method in problem solving identified by Newell and Simon, known as *means-ends analysis* [NS72] (see also [Hay81, p. 33]). In means-ends analysis, the differences between current situation or state $\mathbf{x}$ and the goal state $\mathbf{x}^*$ are listed and the operators appropriate for reducing it found. In evolutionary problem solving, this operator is probabilistic and denoted by variation operator ν,

$$\mathbf{x}^{(g+1)} = \nu(\mathbf{x}^{(g)}) \quad \mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{C}^n. \tag{5.1}$$

One way to list the difference between two states is by defining a distance metric $d_{\mathcal{C}}(\mathbf{x}^{(g)}, \mathbf{x}^{(g+1)}) \geq 0$. In metric space $(\mathcal{C}^n, d_{\mathcal{C}})$ the sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(g)}$,... of points in $\mathcal{C}^n$ *converges in mean* to $\mathbf{x}^* \in \mathcal{C}^n$, if

$$\lim_{g \to \infty} \mathrm{E}\Big[d_{\mathcal{C}}(\mathbf{x}^{(g)}, \mathbf{x}^*)\Big] = 0.$$

For real number spaces, the distance metric commonly used is the Euclidean norm: $d_{\mathcal{C}}(\mathbf{x}^{(g)}, \mathbf{x}^{(g+1)}) = \|\mathbf{x}^{(g)} - \mathbf{x}^{(g+1)}\|$. In order to estimate the operator's effectiveness in reducing $d_{\mathcal{C}}(\mathbf{x}^{(g+1)}, \mathbf{x}^*)$, an *expected rate of progress* measure φ may be defined. In terms of the average distance to the optimum, the expected rate of progress, using (μ, λ) selection, could be computed as [Sch77]:

$$\varphi_x = \mathrm{E}\left[\frac{1}{\mu}\sum_{i=1}^{\mu} d_{\mathcal{C}}(\mathbf{x}_i^{(g)}, \mathbf{x}^*) - \frac{1}{\mu}\sum_{i=1}^{\mu} d_{\mathcal{C}}(\mathbf{x}_{i:\lambda}^{(g+1)}, \mathbf{x}^*) \middle| \mathbf{x}_1^{(g)}, \ldots, \mathbf{x}_\mu^{(g)}\right] \tag{5.2}$$

where $\mathbf{x}_{i:\lambda}$ is the ith *order statistic* ($i = 1, \ldots, \lambda$) determined by the fitness,

$$f(\mathbf{x}_{1:\lambda}) \leq f(\mathbf{x}_{2:\lambda}) \leq \ldots \leq f(\mathbf{x}_{\lambda:\lambda}). \tag{5.3}$$

The progress rate may also be defined in terms of closeness to the optimal fitness function value $f(\mathbf{x}^*)$ [Bey95b, page 387],

$$\begin{aligned}\varphi_f =& \mathrm{E}\left[\frac{1}{\mu}\sum_{i=1}^{\mu}\left(f\left(\mathbf{x}_i^{(g)}\right) - f\left(\mathbf{x}^*\right)\right) - \frac{1}{\mu}\sum_{i=1}^{\mu}\left(f\left(\mathbf{x}_{i:\lambda}^{(g+1)}\right) - f\left(\mathbf{x}^*\right)\right)\middle|\mathbf{x}_1^{(g)}, \ldots, \mathbf{x}_\mu^{(g)}\right] \\ =& \mathrm{E}\left[\frac{1}{\mu}\sum_{i=1}^{\mu} f\left(\mathbf{x}_i^{(g)}\right) - \frac{1}{\mu}\sum_{i=1}^{\mu} f\left(\mathbf{x}_{i:\lambda}^{(g+1)}\right)\middle|\mathbf{x}_1^{(g)}, \ldots, \mathbf{x}_\mu^{(g)}\right].\end{aligned} \tag{5.4}$$

For a constant stationary normalized progress rate $\varphi^\star \geq c > 0$, a linear convergence order is expected in general (see equation (2.31) and discussion on page 21). Computing φ analytically is a difficult theoretical problem, especially when considering the full implementation of an evolutionary algorithm. Furthermore, the analysis is only valid for the fitness function investigated. To extrapolate the results to other problem classes it must be assumed that they have similar properties. For example, it is often supposed that the local structure of the fitness landscape can be approximated by a spherical or corridor model [Bey96].

An operator should reflect the designer's anticipation of where improved situations are most likely to be located for any arbitrary situation. In evolution strategies, there is a preference for small phenotypic change over larger ones, "as common in nature" [BSK96]. This suggests that the fitness landscape is smooth and that evolutionary search is in some sense a hill climbing procedure. The designer may therefore want to choose a distance metric where a small variation is reflected in a small change in fitness. The next step is then to design a variation operator that works in this distance metric space. The operator is probabilistic and has no preference for the direction traversed in this space. The operator must, however, be generalized to jump arbitrary distances. This requires the introduction of a strategy parameter controlling the mean step size taken. This strategy parameter is then adapted during search in order to maintain maximal progress. In this way, an evolution strategy can be formalized for any problem, not just for real valued functions. The designer's domain knowledge is reflected in the selection of a distance metric space, which now promises to form at least a semi-smooth fitness landscape.

At first glance, it may seem that the variation operator designed is a *local search* operator. A vector $\mathbf{x}^*$ is said to be a *local minimum* of f over the set $\mathcal{C}^n$ if it is no worse than its feasible neighbour. Formally, if there exists a $\varepsilon > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall\, \mathbf{x} \in \mathcal{C}^n \text{ with } d_{\mathcal{C}}(\mathbf{x}, \mathbf{x}^*) < \varepsilon.$$

The operator is therefore only "local" to the effect of the mutation strength, or the mean step size taken. A local minimum is also a *global minimum* when it is no worse than all other vectors that is, $f(\mathbf{x}^*) \leq f(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{C}^n$. When the search space is bounded, and mutation strength is large, one might call this *global search*. A function is called unimodal if it has exactly one local minimum, otherwise it is called multimodal. When the fitness landscape is unimodal then a local search corresponds to a global search.

The distance metric and, hence, the fitness landscape has now been defined. The landscape promises to be smooth but may be multimodal. An operator has been designed capable of modifying individuals by some average distance in this landscape. There is now one factor which needs to be investigated; selection and results in a new landscape concept, called the *expected fitness landscape*, and is introduced in the following section. The properties of this landscape will determine how fast individuals will be located in the global region and whether global convergence in linear time may be expected in general. Step size control and directed search are then discussed in relation to the expected fitness landscape. The final section of this work investigates the use of large step sizes for faster search.

5.1 Expected Fitness Landscape

Essentially the progress of the single best individual is of central interest. Therefore, the alternative progress rate definition for evolutionary algorithms may be used [Rud97a, page 207]:

$$\varphi'_f = \mathrm{E}\Big[\min\{f(\mathbf{x}_i^{(g)}) : i = 1, \ldots, \mu\} - f(\mathbf{x}_{1:\lambda}^{(g+1)}) \Big| \mathbf{x}_1^{(g)}, \ldots, \mathbf{x}_\mu^{(g)} \Big] \tag{5.5}$$

and similarly

$$\varphi'_x = \mathrm{E}\Big[\min\{d_{\mathcal{C}}(\mathbf{x}_i^{(g)}, \mathbf{x}^*) : i = 1, \ldots, \mu\} - d_{\mathcal{C}}(\mathbf{x}_{1:\lambda}^{(g+1)}, \mathbf{x}^*) \Big| \mathbf{x}_1^{(g)}, \ldots, \mathbf{x}_\mu^{(g)} \Big] \tag{5.6}$$

The progress rates computed from fitness values, as the ones given by (5.5) and (5.4), indicate the progress towards a local minimum only. Progress towards the global minimum in a multimodal landscape can only be computed in terms of the distance metric and when the global minimum is known. When the support of the variation distribution covers the feasible region of the optimization problem, global convergence may be proven [Bäc96, p. 130]. These results, however, do not of have much practical interest as they require unlimited time for the algorithm to converge. Most of the theoretical analysis of convergence rates for evolutionary algorithms are based on unimodal fitness functions. In this case, there is only one local minimum and the progress in terms of fitness corresponds to progress in terms of the distance metric. Fast global convergence for multimodal functions are nevertheless more interesting. As long as $\varphi_x^\star \geq c > 0$, the evolutionary algorithm works. For this reason an attempt will be made to investigate when this may be the case for multimodal landscapes.

The approach taken is to establish the condition for which the progress in fitness corresponds to progress towards the global optimum defined in terms of the distance metric. That is, all parents, $\mathbf{x} \in \mathcal{C}^n$, satisfy the following condition

$$\mathrm{E}\Big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}\Big] < f(\mathbf{x}) \Leftrightarrow \mathrm{E}\Big[d_{\mathcal{C}}(\mathbf{x}_{1:\lambda/\mu}, \mathbf{x}^*)\big|\mathbf{x}\Big] < d_{\mathcal{C}}(\mathbf{x}, \mathbf{x}^*). \tag{5.7}$$

When (5.7) holds then positive progress is maintained and the evolutionary algorithm will converge in mean to the global optimum. Now consider the case when

$$\mathrm{E}\Big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}_j\Big] < \mathrm{E}\Big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}_k\Big] \Leftrightarrow d_{\mathcal{C}}(\mathbf{x}_j, \mathbf{x}^*) < d_{\mathcal{C}}(\mathbf{x}_k, \mathbf{x}^*) \tag{5.8}$$

where $j, k \in \{1, \ldots, \mu\}, j \neq k$ denote any two parents. This implies that the *expected fitness landscape*, $\mathrm{E}[f\big(\mathbf{x}_{1:\lambda/\mu}\big)|\mathbf{x}]$, is unimodal and monotone decreasing. Furthermore let the progress rate in terms of the fitness values be positive, i.e.

$$\mathrm{E}\Big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}_j\Big] < f(\mathbf{x}_j) \text{ and } \mathrm{E}\Big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}_k\Big] < f(\mathbf{x}_k). \tag{5.9}$$

When (5.8) and (5.9) are satisfied then (5.7) holds. This can be verified by substituting $\mathbf{x}_k = \mathbf{x}$ and $\mathbf{x}_j = \mathrm{E}[\mathbf{x}_{1:\lambda/\mu}|\mathbf{x}]$ in (5.8) and using condition (5.9). The left hand side of (5.8) will then read: $\mathrm{E}[f(x_{1:\lambda/\mu})|(x_{1:\lambda/\mu}|\mathbf{x})] < \mathrm{E}[f(x_{1:\lambda/\mu})|\mathbf{x}]$ which is equivalent to $\mathrm{E}\big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}_j\big] < f(\mathbf{x}_j)$ and therefore holds by (5.9).

The right hand side of (5.8) will simply read: $\mathrm{E}\big[d_{\mathcal{C}}(\mathbf{x}_{1:\lambda/\mu}, \mathbf{x}^*)\big|\mathbf{x}\big] < d_{\mathcal{C}}(\mathbf{x}, \mathbf{x}^*)$. Also, since $\mathbf{x} = \mathbf{x}_k$ then from (5.9) $\mathrm{E}\big[f\big(\mathbf{x}_{1:\lambda/\mu}\big)\big|\mathbf{x}\big] < f(\mathbf{x})$. The offspring produced by a parent closer to the optimum will on average be preferred to an offspring whose parent is further away from the optimum. The combined effect of variation and selection is to smooth the fitness landscape. If the expected fitness landscape is unimodal and its minimum is the same as the global minimum, the evolutionary algorithm will converge in mean to the global optimum when the progress rate in terms of fitness is positive. This concept will now be illustrated by a number of numerical examples and its consequences discussed.

Having chosen a meaningful distance metric for the task, the expected fitness landscape will now depend on the chosen variation operator working in this metric space. A mutation operator is random in the sense that the variation resulting from it is independent from the organism and its milieu. The operator should guarantee an average neutrality of the evolution process

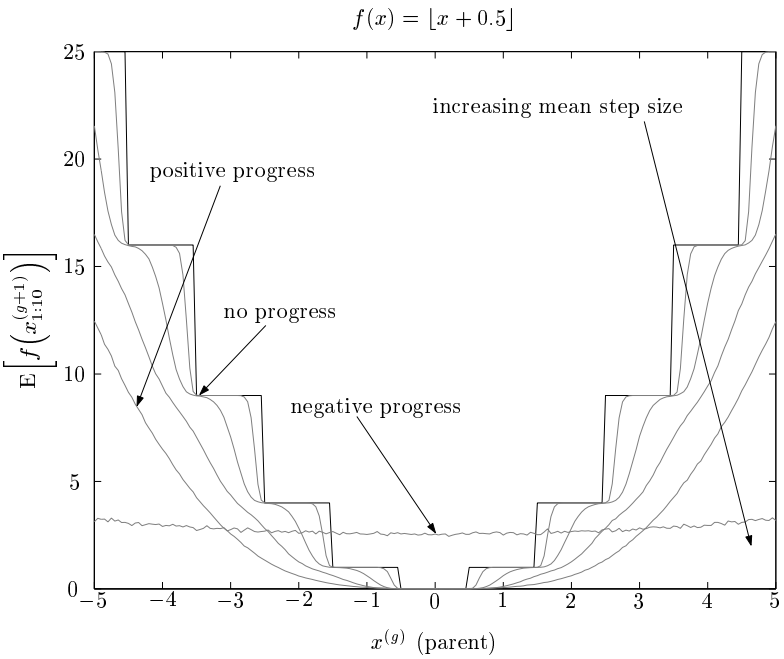


Figure 5.1: Expected fitness landscapes for the step function using the mean step sizes of $\sigma = 0, 0.1, 0.3, 0.6, 1$, and 10

in the absence of selective pressure. There should be no deterministic drift as a result of this mutation without selection. It is natural selection alone which is responsible for the search bias. That is, the expectation of the variation of $\mathbf{x}$ should be $\mathbf{x}$, i.e. $\mathrm{E}(\nu(\mathbf{x})) = \mathbf{x}$. The central limit theorem of statistics says that data which are influenced by many small and unrelated random effects are approximately normally distributed. Assuming this to be the common case for natural systems the variation operator may be written as,

$$x^{(g+1)} = x^{(g)} + \sigma N(0,1), \tag{5.10}$$

where $N(0,1)$ is a Gaussian distributed random variable with an expectation of zero and variance one. The variation operator is altered by modifying the mean step size σ. Consider the step function shown in figure 5.1, where the horizontal axis takes on the parent values $x^{(g)}$ and the vertical axis the expected fitness on the best offspring out of λ/μ, i.e. $\mathrm{E}\big[f(x^{(g+1)}_{1:\lambda/\mu})\big]$. The expectation is computed numerically by performing $10,000$ 'one generation experiments' for $\lambda/\mu = 10$. The mean step sizes taken are $\sigma = 0, 0.1, 0.3, 0.6, 1$, and 10. For $\sigma = 0$ the function topology corresponds to the original step function. In the case where $\sigma > 0$, the fitness landscape is transformed to an expected fitness landscape. A positive progress rate follows from all curves below the original function. Negative progress is obtained when the expected fitness curve lies above the original function curve. No progress is expected where the curves touch. See figure 5.1 for examples of negative, zero, and positive progress for the various mean step sizes used. For an algorithm using variable step sizes the function topology with the lowest expected fitness value $\mathrm{E}\big[f(x^{(g+1)}_{1:\lambda/\mu})\big]$ is selected. In other words, not only is the best x selected, but also the corresponding σ and, hence, progress is maximized. A fixed mean step size of 0.6 or 1, for the step function, results in a unimodal expected fitness function and positive progress. In this case the algorithm will converge to the global optimum on average.

Finding the global optimum is more challenging when the expected fitness landscape is multi-modal. This is illustrated using Rastrigin's function shown in figure 5.2. The step sizes applied in the previous example are used again here. In this case positive progress is assured most of the time for steps sizes 0.6 and 1. However, as the parents near the global optimum the progress becomes negative. This region is shown enlarged in figure 5.2. Notice that when a parent is located near the global minimum, a lower expected fitness value may be maintained by decreasing the mean step size. When the step

size is reduced, the expected fitness landscape becomes multimodal and there is a danger of being trapped in a local minimum. If no parent is located in the global region, getting there will now on average be impossible. Only by enforcing a larger step size, or perhaps when using a probabilistic selection method, could an individual escape the local minimum. The only reliable method, however, is to shift the unimodal expected fitness landscape down such that it lies at least below all local minima. This can only be achieved by increasing λ before a decision is made to decrease the mean step size and turn the unimodal expected fitness landscape to a multimodal one. Elitist selection is also a means of extending the number of offspring created by a parent, but this is done over a number of generations.

Assuming the global region is reached, there exists still the possibility of being pushed out of it. This is the case for Shaffer's function shown in figure 5.3. The figure shows the expected fitness landscapes for the mean

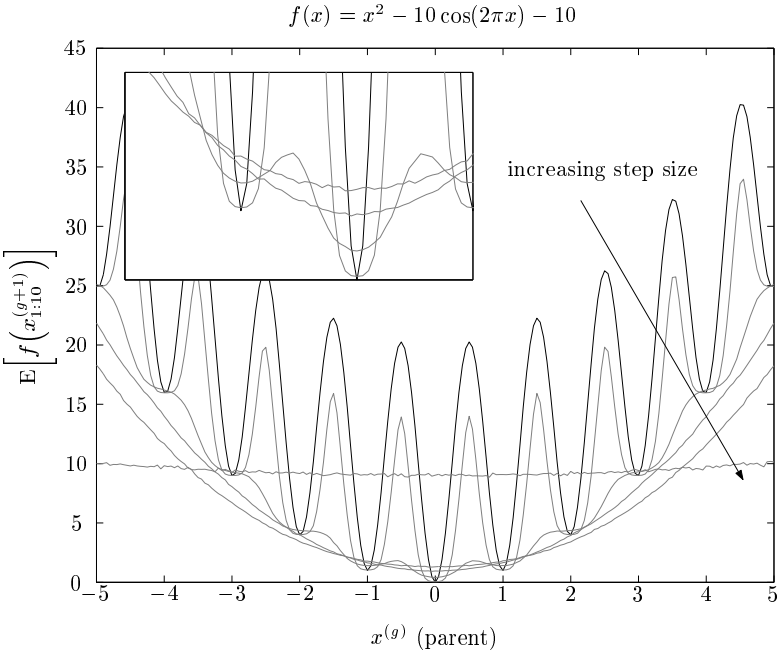


Figure 5.2: Expected fitness landscapes for Rastrigin's function for the mean step sizes of $\sigma = 0, 0.1, 0.3, 0.6, 1, 10$

step sizes 1 and 5. For a step size of 5, the expected fitness landscape is unimodal, however, as the step size decreases a peak emerges at the global optimum making the neighbouring local minima more attractive locations. The population is effectively pushed out of the global region and will on average be located in a local region. Staying in the global region would require a sharp decrease in the mean step size. To illustrate this effect, the function is optimized by a $(15, 100)$ ES using the lognormal self-adaptation of the mean step size both with and without global intermediate recombination and an initial step size of 5. The result of these experimental studies are given in table 5.1, and are labelled (i) and (ii) respectively. It is anticipated from figure 5.3 that only a fixed step size of 5, which results in a unimodal expected fitness landscape, will on average locate the global optimum. However, the number of function evaluations required are greater, since $\lambda/\mu = 10$ will result in negative progress near the global optimum. Two experiments are performed

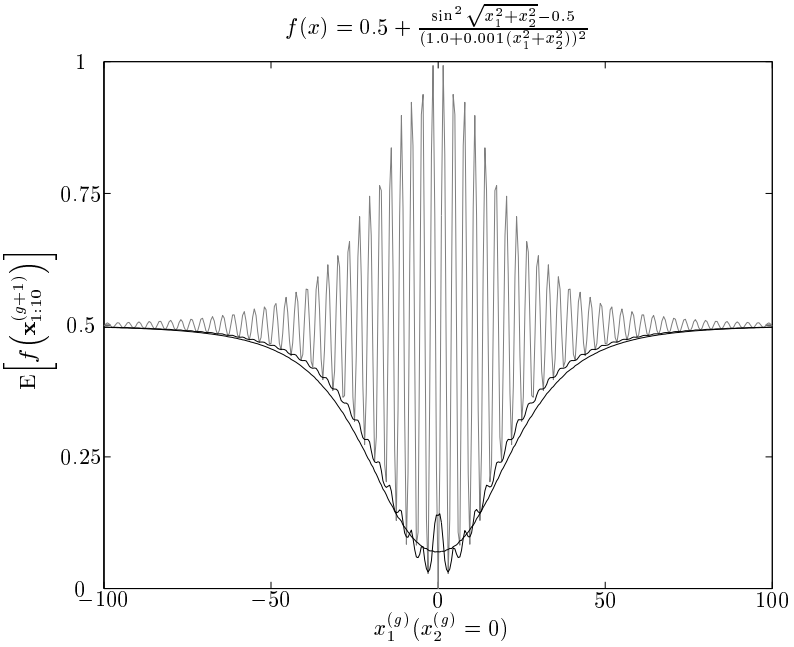


Figure 5.3: Expected fitness landscapes for Shaffer's function with the mean step sizes: $\sigma = 0, 1, 5$

with a fixed step size. The first uses the same population size as the first two experiments, in this case negative progress is expected near some local minima. The second experiment uses a larger population size, $\lambda = 400$, and maintains positive progress at all local minima. The total number of function evaluations used for each experiment is fixed at 100×1000 and is used as the termination criterion. The results are summarized also in table 5.1, and labelled by (iii) and (iv) respectively. As expected, only a fixed step size of 5 reliably locates the global optimum. This is guaranteed for experiment (iv) using the larger population size. For experiment (iii), which on average requires as many function evaluation to locate the global optimum, this is not as obvious. It may perhaps be explained by the fact that the individuals are located near the global region on average and that the mutation distribution covers the global optimum point.

Table 5.1: Optimization of Shaffer's function using the ES$(15, 100)$ with an initial step size of 5 using i) lognormal update, ii) global intermediate recombination and lognormal update, iii) no update and iv) ES$(15, 400)$ no update.

study	best	median	mean	st. dev.	worst	G_m	global hit
(i)	3.0E−3	1.7E−2	3.9E−2	3.9E−2	1.8E−1	7	1/30
(ii)	3.0E−4	9.7E−3	8.0E−3	3.2E−3	1.0E−2	100	9/30
(iii)	6.1E−5	1.3E−3	2.0E−3	1.8E−3	7.4E−3	413	30/30
(iv)	7.1E−5	7.5E−4	1.1E−3	8.8E−4	3.2E−3	119	30/30

5.2 Step Size Control

The expected fitness landscapes, plotted for the various examples in the previous section, assumed that a fixed mean step size is used to produce all offspring. The actual evolution strategy introduces, however, a mutative step-size control that introduces random fluctuations [OGH95]. Intermediate recombination in large populations may lead to a significant reduction in these fluctuations. It has been suggested that this effect may perhaps be explained by the genetic repair hypothesis [Bey97]. This is possibly reasonable for strategy parameters working on the same *local expected fitness landscape*. For another 'derandomizing' step size control and further discussion on this problem see [OGH95]. One of the main differences between the current implementation of steps size control in evolution strategies and evolutionary programming is the use of

recombination. Evolution strategy prefers the use of global intermediate recombination of the mean step sizes [Sch95, p. 148]. In this section the role of recombination of the mean step sizes is examined empirically. This is done by optimizing the 23 benchmark functions investigated in [YLL99], using the evolution strategy described in section 2.2.1, with and without recombination. A complete description of the benchmark functions is given in appendix A.2.

The number of function evaluations used in the experiments are the same as in [YLL99]. For a population size of $\lambda = 100$, the number of generations used for the 23 problems are: 100 for test functions f_{u14}, f_{u16}, f_{u17}, f_{u18}, f_{u19}, f_{u21}, f_{u22}, f_{u23}, 1500 for f_{u1}, f_{u6}, f_{u10}, f_{u12}, f_{u13}, 5000 for f_{u3}, f_{u4}, f_{u9}, 2000 for f_{u2}, f_{u11}, and 20000, 3000, 9000, 4000, 200 for functions f_{u5}, f_{u7}, f_{u8}, f_{u15}, f_{u20} respectively. The experimental results for the test problem studied using no recombination are presented in table 5.2. When comparing these results with those using recombination, shown in table 5.3, it is clear that for these test cases recombination does on average better. However, for test problems f_{u3} and f_{u9} a slightly better result is observed using no recombination. The

Table 5.2: Traditional ES using **no** recombination of mean step sizes.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{u1}	0.0	2.1E−05	13.4	52.6	166.1	372.3	1499
f_{u2}	0.0	0.09	3.34	4.87	7.85	15.02	1999
f_{u3}	0.0	431.19	3266.68	3894.93	4965.84	10073.8	4999
f_{u4}	0.0	0.36	1.38	1.72	1.98	4.25	4999
f_{u5}	0.0	18.10	163.93	9213.54	44293.0	95941.2	18106
f_{u6}	0.0	73.0	773.0	1175.87	2209.04	4764	1146
f_{u7}	0.0	0.063	0.269	0.471	0.815	1.909	2408
f_{u8}	−12569.5	−9667.65	−8857.22	−8760.21	1215.7	−7476.32	6975
f_{u9}	0.0	56.77	78.98	78.72	24.39	106.50	3579
f_{u10}	0.0	5.025	9.757	9.719	3.852	13.889	1499
f_{u11}	0.0	0.039	1.422	1.830	3.761	10.848	1999
f_{u12}	0.0	0.522	4.154	5.256	6.502	14.874	1499
f_{u13}	0.0	0.318	6.544	33.977	179.878	528.18	1499
f_{u14}	1.0	1.003	2.677	3.153	3.399	7.874	6
f_{u15}	0.0003075	0.0003075	0.0003087	0.0011395	0.005840	0.017466	3998
f_{u16}	−1.0316285	−1.03163	−1.03163	−1.03163	8.1E−16	−1.03163	50
f_{u17}	0.397887	0.397887	0.397887	0.397887	0.0E+00	0.397887	52
f_{u18}	3.0	3.0	3.0	3.0149	0.146	3.4465	49
f_{u19}	−3.86278	−3.86278	−3.86278	−3.86278	4.0E−16	−3.86278	73
f_{u20}	−3.322	−3.322	−3.322	−3.283	0.10	−3.203	186
f_{u21}	−10.153	−10.153	−10.153	−7.347	6.3	−2.630	98
f_{u22}	−10.403	−10.403	−8.235	−6.922	6.5	−2.326	98
f_{u23}	−10.536	−10.536	−5.374	−6.711	6.2	−2.427	99

result for these two test cases are nevertheless poor for both versions. Note that a lower bound on the steps sizes is *not* implemented.

One way of increasing the reliability of the algorithm may be to combine the strategies by using a mixed implementation of the global intermediate recombination. This may be performed as follows,

$$\hat{\sigma}_{h,j}^{(g)} = \begin{cases} (\sigma_{i,j}^{(g)} + \sigma_{k_j,j}^{(g)})/2, & k_j \in \{1,\ldots,\mu\}, \quad \text{if } U_j > 1/2 \\ \sigma_{i,j}^{(g)} & \text{otherwise,} \end{cases} \tag{5.11}$$

where k_j is an index generated at random and anew for each j and U is a random number uniformly distributed over $(0, 1]$. The experimental results using this mixed strategy is given in table 5.4. The result is a surprising improvement for functions f_{u3} and f_{u9} over previous strategies. On average the results are better or similar to the best result using the other two strategies.

It is clear that different problems, and different stages of search, may require different step size controls as observed here and in the previous section. There exist yet another means of achieving this goal adaptively. Assumptions

Table 5.3: Traditional ES using intermediate recombination of mean step sizes.

`fcn`	optimal	best	median	mean	st. dev.	worst	G_m
f_{u1}	0.0	2.6E−14	7.1E−13	4.1E−11	2.2E−10	5.9E−10	1498
f_{u2}	0.0	5.2E−10	6.2E−09	1.9E−08	5.8E−08	1.5E−07	1998
f_{u3}	0.0	0.25	10230.6	8795.25	9662.21	17861.3	3492.5
f_{u4}	0.0	1.1E−06	1.1E−05	1.5E−4	8.8E−04	2.3E−03	4993
f_{u5}	0.0	8.7E−07	3.6E−05	0.313	1.852	3.987	19999
f_{u6}	0.0	0.0	0.0	0.0	0.0	0.0	532
f_{u7}	0.0	0.063	0.114	0.112	0.045	0.177	1810
f_{u8}	−12569.5	−11997	−11464	−11460	754.9	−10319	2688
f_{u9}	0.0	194.05	225.86	225.08	18.99	245.31	2782
f_{u10}	0.0	6.1E−08	1.7E−06	0.0547	0.5379	1.6413	1497
f_{u11}	0.0	0.0	1.7E−15	0.0077	0.024	0.0492	1992
f_{u12}	0.0	2.8E−14	5.9E−11	0.0138	0.064	0.10367	1498
f_{u13}	0.0	8.6E−13	9.3E−10	0.0066	0.036	0.1099	1496
f_{u14}	1.0	0.998	1.047	1.602	1.55	4.007	25
f_{u15}	0.0003075	0.0003075	0.0003075	0.0003075	2.9E−19	3.1E−04	2427
f_{u16}	−1.0316285	−1.03163	−1.03163	−1.03163	8.1E−16	−1.03163	67
f_{u17}	0.397887	0.397887	0.397887	0.397887	0.0E+00	0.397887	68
f_{u18}	3.0	3.0	3.0	3.0	0.0E+00	3.0	68
f_{u19}	−3.86278	−3.86278	−3.86278	−3.86278	5.8E−15	−3.86278	92
f_{u20}	−3.322	−3.322	−3.322	−3.301	0.081	−3.203	182
f_{u21}	−10.153	−10.153	−10.153	−7.5871	5.29	−2.6304	99
f_{u22}	−10.403	−10.403	−10.403	−10.051	2.40	−5.1288	99
f_{u23}	−10.536	−10.536	−10.536	−10.273	2.58	−2.6434	98

made about the environment may be used to control search direction and step sizes. These are so called directed search methods and will be discussed briefly in the following section.

Table 5.4: Traditional ES using mixed intermediate recombination of mean step sizes.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{u1}	0.0	3.4E−20	1.2E−17	6.3E−15	5.3E−14	1.6E−13	1499
f_{u2}	0.0	1.1E−14	1.2E−11	1.4E−05	1.4E−04	4.3E−04	1999
f_{u3}	0.0	5.6E−13	9.0E−10	4.1E−09	1.8E−08	5.4E−08	4999
f_{u4}	0.0	8.4E−04	9.6E−03	2.4E−02	4.8E−02	1.2E−01	4999
f_{u5}	0.0	1.5E−09	4.1E−06	1.3E+00	3.4E+00	4.0E+00	19999
f_{u6}	0.0	0.0E+00	0.0E+00	1.2E+00	1.2E+01	36.0E+00	709
f_{u7}	0.0	4.9E−02	6.2E−02	6.4E−02	2.1E−02	9.8E−02	1774
f_{u8}	−12569.5	−11108.7	−10447.4	−10368.4	788.2	−9310.8	1337
f_{u9}	0.0	18.9	42.8	44.2	18.5	676.6	2169
f_{u10}	0.0	2.2E−11	1.6E+00	1.7E+00	2.2E+00	4.8E+00	1423
f_{u11}	0.0	0.0E+00	2.0E−02	3.5E−02	8.7E−02	2.2E−01	1744
f_{u12}	0.0	2.9E−20	1.0E−01	2.8E−01	1.0E+00	2.9E+00	1499
f_{u13}	0.0	3.7E−20	1.1E−02	2.3E−01	1.3E+00	3.7E+00	1499
f_{u14}	1.0	0.998	1.992	2.276	2.60	6.903	11
f_{u15}	0.0003075	3.1E−04	3.1E−04	1.3E−03	5.0E−03	1.2E−02	3588
f_{u16}	−1.0316285	−1.03163	−1.03163	−1.03163	8.1E−16	−1.03163	53
f_{u17}	0.397887	0.397887	0.397887	0.397887	0.0E+00	0.397887	55
f_{u18}	3.0	3.0	3.0	3.0	0.0E+00	3.0	55
f_{u19}	−3.86278	−3.86278	−3.86278	−3.86278	4.1E−15	−3.86278	72
f_{u20}	−3.322	−3.32237	−3.32237	−3.3075	7.0E−02	−3.20316	146
f_{u21}	−10.153	−10.1532	−5.38515	−6.79	5.7	−2.63047	98.5
f_{u22}	−10.403	−10.403	−10.403	−8.6612	5.4	−2.7519	98
f_{u23}	−10.536	−10.536	−10.536	−9.1502	5.1	−2.5497	99

5.3 Directed Search

A directed variation is dependent on the organism-environment system. The mutation itself is under an environmental feedback. Consider a direct search technique like the method of Nelder-Mead [NM65], the algorithm maintains a simplex of approximations to an optimal point. The algorithm proceeds by

changing the worst individual, $((\lambda-1)+\lambda)$ selection[1], to

$$\mathbf{x}^{(g+1)} = \frac{(1+\chi)}{\lambda-1}\sum_{i=1}^{\lambda-1}\mathbf{x}_{i:\lambda}^{(g)} - \chi\mathbf{x}_{\lambda:\lambda}^{(g)} \tag{5.12}$$

where $\lambda = n+1$ and typically $\chi \in \{1, 2, 1/2, -1/2\}$ depending on whether there is a reflection, expansion, outside or inside contraction step taken in the Nelder-Mead iteration [Kel99]. Similar strategies using different vertices are employed in differential evolution [SP97], generalized intermediate recombination [Bäc96, p. 74], and other direct search methods [DT91]. In the case of evolutionary gradient search [Sal98]

$$\mathbf{x}^{(g+1)} = \mathbf{x}^{(g)} - \sigma^{(g+1)}\frac{\widetilde{\nabla f}}{\|\widetilde{\nabla f}\|} \tag{5.13}$$

where the estimated gradient is computed as a weighted sum of λ trial points $\mathbf{t}_i$,

$$\widetilde{\nabla f} = \sum_{i=1}^{\lambda}\big(f(\mathbf{t}_i) - f(\mathbf{x}^{(g)})\big)\big(\mathbf{t}_i - \mathbf{x}^{(g)}\big), \text{ and}$$

$$\mathbf{t}_i = \mathbf{x}^{(g)} + N_i\big(0, \tfrac{\sigma^2}{n}^{(g)}\big)$$

The weights are determined by the fitness function.

Using local landscape information may be misleading. It is expected that the simplex and gradient approximations will find the global optimum of some well behaved functions only. The methods do not utilize the smooth expected fitness landscape. If they would then the estimated gradient in (5.13) would use the parent position and best offspring fitness. Similarly for the directed search strategy, the simplex would be estimated from the parents but the ranking according to their best offspring. The strategies (5.13) and (5.12) are effectively single parent strategies and therefore cannot utilize information from the smoothed fitness landscape resulting from variation and selection.

5.4 Hopeful Monsters

The variation needed to satisfy the reachability property is called a *hopeful monster* mutation. According to the reachability property, discussed in sec-

[1] When no improvement is obtained all but the best individual are modified in a shrink step.

tion 2.2.4, the variation guarantees the location of the global optimum with probability one, although the waiting time may be long.

It has been expressed that large mutations or jumps may keep the evolutionary algorithm from being caught in local minima [YL96, Kap96]. For this reason, a Cauchy mutation has been proposed as the alternative to the Gaussian one. However, for a decreasing step size even the Cauchy mutation will get stuck in a local minimum [Rud97b]. Only a Cauchy mutation with a fixed step size, or lower bound on the step size, can secure coverage [Rud97b]. The mutation described in this section is based on a uniform distribution over the search space and is independent of the parent's position. The hopeful monster mutation is defined as

$$x_{h,j}^{(g+1)} = \begin{cases} \underline{x}_j + U(\overline{x}_j - \underline{x}_j), & \text{if } U_h < \mu/\lambda \text{ and } U_j < p_m \\ x_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1) & \text{otherwise.} \end{cases} \tag{5.14}$$

where U is a random number uniformly distributed over $[0,1]$. On average

Table 5.5: Hopeful monster mutation and mixed recombination of mean step sizes.

func.	optimal	best	median	mean	st. dev.	worst	G_m
f_{u1}	0.0	3.2E−16	2.3E−14	6.2E−12	4.5E−11	1.4E−10	1499
f_{u2}	0.0	3.2E−12	7.7E−10	2.7E−06	2.5E−05	7.6E−05	1998
f_{u3}	0.0	2.6E−10	3.5E−09	5.5E−08	3.1E−07	8.6E−07	4999
f_{u4}	0.0	1.4E−04	1.3E−02	2.3E−02	6.0E−02	1.7E−01	4999
f_{u5}	0.0	5.6E−09	5.4E−05	7.3E−02	4.3E−01	1.2E+00	19999
f_{u6}	0.0	0.0E+0.0	0.0E+0.0	0.0E+0.0	0.0E+0.0	0.0E+0.0	686
f_{u7}	0.0	4.3E−02	6.6E−02	6.7E−02	2.8E−02	1.0E−01	1891
f_{u8}	−12569.5	−12569.5	−12569.5	−12569.5	3.3E−12	−12569.5	1810
f_{u9}	0.0	0.0E0.0	0.0E0.0	1.4E−15	1.3E−14	3.9E−14	3272
f_{u10}	0.0	8.0E−09	1.3E−05	1.5E−01	6.1E−01	1.2E+00	1498
f_{u11}	0.0	0.0E+00	2.1E−02	3.4E−02	6.5E−02	1.2E−01	1877
f_{u12}	0.0	1.2E−17	4.4E−13	3.5E−03	3.4E−02	1.0E−01	1499
f_{u13}	0.0	1.1E−15	1.3E−10	4.2E−03	9.6E−03	1.1E−02	1499
f_{u14}	1.0	0.998004	0.998004	0.998011	5.1E−05	0.998146	93
f_{u15}	0.0003075	0.0003075	0.0003075	0.0003075	2.1E−18	0.0003075	3942
f_{u16}	−1.0316285	−1.0316	−1.0316	−1.0316	1.2E−15	−1.0316	77
f_{u17}	0.397887	0.397887	0.397887	0.397887	0.0E+00	0.397887	57
f_{u18}	3.0	3.0E+0.0	3.0E+0.0	3.0E+0.0	9.6E−15	3.0E+0.0	83
f_{u19}	−3.86278	−3.86278	−3.86278	−3.86278	4.1E−15	−3.86278	80
f_{u20}	−3.322	−3.322	−3.322	−3.303	8.1E−02	−3.203	163
f_{u21}	−10.153	−10.153	−10.153	−7.746	5.5	−2.683	99
f_{u22}	−10.403	−10.403	−10.403	−9.290	4.3	−2.752	98
f_{u23}	−10.536	−10.536	−10.536	−7.364	6.3	−2.427	98

Table 5.6: Result for 30 independent runs using Gaussian mutation only.

function	optimal	best	mean	st. dev.	worst	G_m	global hit
f_{u8}	−12569.5	−11108.7	−10368.4	788.2	−9310.8	1337	0/30
f_{u9}	0.0	18.9	44.2	18.5	676.6	2169	0/30
f_{u14}	1.0	0.998	2.276	2.60	6.903	11	15/30
f'_{u14}	1.0	0.998	2.073	0.99	4.002	8	14/30

only one offspring will undergo this mutation. The mutation is aligned to the coordinate system when $p_m = 1/n$, and $p_m = 1$ is required to cover uniformly the entire search space. The univariate Cauchy mutation [Rud97b], as well as small mutation rates [Sal96], bias the search toward a grid that is aligned with the coordinate system. This is also true for a Gaussian mutation when there is no correlation between multiple mean step sizes.

In order to evaluate the effectiveness of the hopeful monster mutation, experiments are carried out on the 23 benchmark function described in [YLL99]. The complete set of results are given in table 5.5. The hopeful monster experiments using $p_m = 1/n$ showed a remarkable improvement for the generalized Schwefel's problem 2.26 (f_{u8}), generalized Rastrigin's (f_{u9}), and Shekel's foxholes function (f_{u14}). These results, summarized in table 5.7, are better than those obtained using the univariate Cauchy mutation [YLL99]. The results using $p_m = 0$ for the same problems are summarized in table 5.6.

The reason for this great success of the hopeful monster mutation on the generalized Schwefel and Rastrigin functions is simply due to the independence of the parameters, which the $p_m = 1/n$ mutation exploits [Sal96]. The dependence on the rotation of the coordinate system can be further demonstrated by optimizing the rotated function. Shekel's foxholes function, shown in figure 5.4, may be rotated by $\pi/4$ by letting $\mathbf{x}' = [(x_1 - x_2), (x_1 + x_2)]/2$. The rotated function is then denoted by $f'_{u14}(\mathbf{x}) = f_{u14}(\mathbf{x}')$. The number of global hits using the hopeful monster mutation is now equivalent to that of

Table 5.7: Result for 30 independent runs using the hopeful monster mutation along coordinate axis and Gaussian mutation.

function	optimal	best	mean	st. dev.	worst	G_m	global hit
f_{u8}	−12569.5	−12569.5	−12569.5	3.3E−12	−12569.5	1810	30/30
f_{u9}	0.0	0.0E0.0	1.4E−15	1.3E−14	3.9E−14	3272	30/30
f_{u14}	1.0	0.998	0.998	5.1E−05	0.998	93	30/30
f'_{u14}	1.0	0.998	1.837	0.96	4.950	49	14/30

Table 5.8: Result for 30 independent runs using the hopeful monster mutation uniformly distributed over the search space and Gaussian mutation.

function	optimal	best	mean	st. dev.	worst	G_m	global hit
f_{u8}	−12569.5	−11286.4	−10457.2	470.0	−9529.5	2006	0/30
f_{u9}	0.0	29.8	49.7	11.9	85.6	2116	0/30
f_{u14}	1.0	0.998	1.841	1.189	5.916	15	19/30
f'_{u14}	1.0	0.998	1.459	0.688	3.968	71	21/30

using none. This may be verified by comparing the last rows in tables 5.6 and 5.7.

Finally, in order to implement the hopeful monster mutation properly, i.e. independent of the coordinate system, then $p_m = 1$. This result is given in table 5.8. In comparison with the results in table 5.6, it seems that no significant improvement may be expected using hopeful monster mutations. They do, however, guarantee global convergence in infinite time, but the experiments were conducted in finite time.

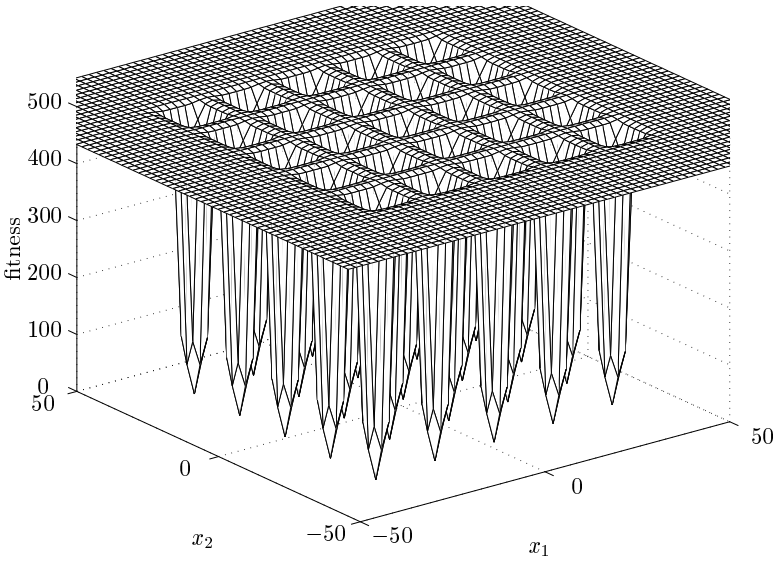


Figure 5.4: Shekel's foxholes function

5.5 Summary

This chapter has given a new interpretation of when one may expect evolutionary search to perform effectively and efficiently. This is done by visualizing an expected fitness landscape. The topology of this landscape aids in the interpretation of when and how evolutionary search works. When this landscape is unimodal and its optimum is also the global optimum, the population will be located in the global region. Furthermore, if positive progress in fitness is maintained, the evolutionary algorithm will converge to the global optimum in mean. Different step size controls are needed for different problems. A small change in the mean step size is not necessarily the optimal strategy for all problems, Shaffer's function is an example of this. Directed search techniques do not take advantage of the structure of the fitness landscape smoothed out by selection. They are effectively single parent strategies. Finally, although larger step sizes may increase the probability of escaping local minima for some problems, they may not contribute to faster evolutionary search for other problems. A more reliable means of escaping local minima is to preserve a larger step size and increase the number of offspring. One means of achieving this adaptively would be to generate the pre-specified number of offspring and when no improvement over the parent is found continue generating individuals until an improvement is obtained.

Up until now it has been assumed that all situation descriptions naturally satisfy any constraint. However, this is not possible to implement in every case. The following chapter addresses this issue in detail.

Chapter 6

Constraint Handling

When a representation and variation operators are unable to 'naturally' satisfy the constraints of a problem, then the only approach available is to eliminate infeasible solution by selection. This requires transforming the fitness function so that it includes a measure of the constraint violation. This is known as the penalty function method discussed in section 2.2.2 and is written as

$$\psi(\mathbf{x}) = f(\mathbf{x}) + r_g \ \phi(\mathrm{g}_j(\mathbf{x}); j = 1, \ldots, m) \tag{6.1}$$

where $\phi \geq 0$ is a real valued function which imposes a 'penalty' controlled by a sequence of *penalty coefficients* $\{r_g\}_0^G$. The penalty function ϕ used in this study is the quadratic loss function (2.18), but any other penalty function is equally valid. Different penalty functions characterize different problems. It is unlikely that a generic penalty function exists which is optimal for all problems. The introduction of penalties may transform the original smooth objective function into a rugged multimodal landscape. As discussed in the previous chapter, the search may then become easily trapped in local minima. For this reason, it is necessary to develop a penalty function method which attempts to preserve the topology of the objective function and yet enables the search to locate feasible solutions. A new way of interpreting penalty function methods for evolutionary search is discussed in the following section. The result is a new ranking method for constraint handling, which is presented in the second section. This is then followed by a section devoted to experimental studies illustrating the effectiveness of the proposed approach.

6.1 Penalty Method

For a given penalty coefficient $r_g > 0$ let the ranking of λ individuals be

$$\psi(\mathbf{x}_1) \leq \psi(\mathbf{x}_2) \leq \ldots \leq \psi(\mathbf{x}_\lambda) \tag{6.2}$$

where ψ is the transformation function given by equation (6.1). Now examine the adjacent pair i and $i+1$ in the ranked order:

$$f_i + r_g\phi_i \leq f_{i+1} + r_g\phi_{i+1}, \qquad i \in \{1, \ldots, \lambda - 1\}, \tag{6.3}$$

where the notation $f_i = f(\mathbf{x}_i)$ and $\phi_i = \phi(\mathrm{g}_j(\mathbf{x}_i), j = 1, \ldots, m))$ are used for convenience. Introduce a parameter, $\check{r}_i$, which will be referred to as the *critical penalty coefficient* for the adjacent pair i and $i+1$,

$$\check{r}_i = (f_{i+1} - f_i)/(\phi_i - \phi_{i+1}), \qquad \text{for } \phi_i \neq \phi_{i+1}. \tag{6.4}$$

For the given choice of $r_g \geq 0$, there are three different cases which may give rise to the inequality (6.3):

1. $f_i \leq f_{i+1}$ and $\phi_i \geq \phi_{i+1}$: the comparison is said to be *dominated by the objective function* and $0 < r_g \leq \check{r}_i$ because the objective function f plays the dominant role in determining the inequality. When individuals are feasible $\phi_i = \phi_{i+1} = 0$ and $\check{r}_i \to \infty$.

2. $f_i \geq f_{i+1}$ and $\phi_i < \phi_{i+1}$: the comparison is said to be *dominated by the penalty function* and $0 < \check{r}_i < r_g$ because the penalty function ϕ plays the dominant role in determining the inequality.

3. $f_i < f_{i+1}$ and $\phi_i < \phi_{i+1}$: the comparison is said to be *nondominated* and $\check{r}_i < 0$. Neither the objective nor the penalty function can determine the inequality by itself.

When comparing nondominant and feasible individuals, the value of r_g has no impact on the inequality (6.3). In other words, it does not change the order of ranking of the two individuals. However, the value of r_g is critical in the first two cases as $\check{r}_i$ is the flipping point that will determine whether the comparison is objective or penalty function dominated. For example, if r_g is increased to a value greater than $\check{r}_i$ in the first case, individual $i+1$ would change from a fitter individual into a less-fit one. For the entire population,

the chosen value of r_g used for comparisons will determine the fraction of individuals dominated by the objective and penalty functions.

Not all possible r_g values can influence the ranking of individuals. They have to be within a certain range, i.e. $\underline{r}_g < r_g < \overline{r}_g$, to influence the ranking, where the lower bound $\underline{r}_g$ is the minimum critical penalty coefficient computed from adjacent individuals ranked only according to the objective function and the upper bound $\overline{r}_g$ is the maximum critical penalty coefficient computed from adjacent individuals ranked only according to the penalty function. In general, there are three different categories of r_g values:

1. $r_g < \underline{r}_g$: *All* comparisons are based only on the fitness function. r_g is too small to influence the ranking of individuals, call this *under-penalization.*

2. $r_g > \overline{r}_g$: *All* comparisons are based only on the penalty function. r_g is so large that the impact of the objective function can be ignored, call this *over-penalization.*

3. $\underline{r}_g < r_g < \overline{r}_g$: *All* comparisons are based on a combination of objective and penalty functions.

All penalty methods can be classified into one of the above three categories. Some methods may fall into different categories during different stages in search. It is important to understand the difference among these three categories because they indicate which function (combination of functions) is driving the search process and how search progresses. For example, most dynamic methods start with a low r_g value (i.e., $r_g < \underline{r}_g$) in order to find a good region that may contain both feasible and infeasible individuals. Towards the end of search, a high r_g value (i.e., $r_g > \overline{r}_g$) is often used in order to locate a good feasible individual. Such a dynamic method would work well for problems for which the unconstrained global optimum is close to its constrained global optimum. It is unlikely to work well for problems for which the constrained global optimum is far away from its unconstrained one because, the initial low r_g value would drive the search towards the unconstrained global optimum and thus further away from the constrained one.

The traditional constraint-handling technique used in evolution strategies falls roughly into the category of over-penalization since all infeasible individuals are regarded worse than feasible ones [Sch95, HS96, Deb99, JV99]. In fact, canonical evolution strategies allow only feasible individuals in the initial

population. To perform constrained optimization, an ES may be used to find a feasible initial population by minimizing the penalty function ([Sch95], page 115). Once a feasible population is found, the ES algorithm will then minimize the objective function and reject all infeasible solutions generated.

It has been widely recognized that neither under- nor over-penalization is a good constraint-handling technique and there should be a balance between preserving feasible individuals and rejecting infeasible ones [GC97]. In other words, ranking should be dominated by a combination of objective and penalty functions and so the penalty coefficient r_g should be within the bounds: $\underline{r}_g < r_g < \overline{r}_g$. It is worth noting that the two bounds are not fixed. They are problem dependent and may change from generation to generation as they are also determined by the current population.

A simple way to measure the balance of dominance of objective and penalty functions is to count how many comparisons of adjacent pairs are dominated by the objective and penalty function respectively. Such a number of comparisons can be computed for any given r_g by counting the number of critical penalty coefficients given by (6.4) which are greater than r_g. If there is a predetermined preference for the number of adjacent comparisons that should be dominated by the penalty function then a corresponding penalty coefficient could be found.

It is clear from the analysis in this section that all a penalty method tries to do is to obtain the right balance between objective and penalty functions so that the search moves towards the optimum in the feasible space, not just towards the optimum in the combined feasible and infeasible space. One way to achieve such balancing effectively and efficiently is to adjust such balance directly and explicitly. This is what stochastic ranking, described in the next section, does.

6.2 Stochastic Ranking

Since the optimal r_g is hard to determine, a different approach is used here to balance the dominance of the objective and penalty function. A probability P_f of using only the objective function for comparisons in ranking in the infeasible regions of the search space is introduced. That is, given any pair of two adjacent individuals, the probability of comparing them (in order to determine which one is fitter) according to the objective function is 1 if both

individuals are feasible, otherwise it is P_f. This appears to be similar to the use of a probability by Surry and Radcliffe [SR97] in deciding the outcome of competitions between two individuals in tournament selection. The technique presented is, however, quite different because it uses rank-based selection and does not require any extra computational cost for self-adapting P_f. More importantly, the motivation of stochastic ranking comes from the need for balancing objective and penalty functions directly and explicitly in optimization. Surry and Radcliffe's method [SR97] does not attempt to balance the dominance of penalty and objective functions in a population.

Ranking is achieved by a bubble-sort-like procedure[1] in this work. The procedure provides a convenient way of balancing the dominance in a ranked set. In the bubble-sort-like procedure, λ individuals are ranked by comparing adjacent individuals in at least λ sweeps[2]. The procedure is halted when no change in the rank ordering occurs within a complete sweep. Figure 6.1 shows the stochastic bubble sort procedure used to rank individuals in a population.

The probability of an adjacent individual winning a comparison, i.e., holding the higher rank, in the ranking procedure is

$$P_w = P_{fw}P_f + P_{\phi w}(1 - P_f) \tag{6.5}$$

given that at least one individual is infeasible. P_{fw} is the probability of the individual winning according to the objective function and $P_{\phi w}$ is the probability of the individual winning according to the penalty function. In the case where adjacent individuals are both feasible $P_w = P_{fw}$, the probability of winning k more comparisons than losses is examined. The total number of wins must be $k' = (N + k)/2$ where N is the total number of comparisons made. The probability of winning k' comparisons out of N is given by the binomial distribution[3]

$$P_w(y = k') = \binom{N}{k'} P_w^{k'}(1 - P_w)^{N-k'}. \tag{6.6}$$

The probability of winning *at least* k' comparisons is

$$P'_w(y \geq k') = 1 - \sum_{j=0}^{k'-1} \binom{N}{j} P_w^j(1 - P_w)^{N-j}. \tag{6.7}$$

[1] It can be regarded as the stochastic version of the classic bubble sort.

[2] It would be exactly λ sweeps if the comparisons were not made stochastic.

[3] The standard deviation of the binomial distribution is $\sqrt{NP_w(1 - P_w)}$.

```
1   I_j = j ∀ j ∈ {1, ..., λ}
2   for i = 1 to N do
3       for j = 1 to λ − 1 do
4           sample u ∈ U(0, 1)
5           if (φ(I_j) = φ(I_{j+1}) = 0) or (u < P_f) then
6             if (f(I_j) > f(I_{j+1})) then
7                swap(I_j, I_{j+1})
8             fi
9           else
10            if (φ(I_j) > φ(I_{j+1})) then
11               swap(I_j, I_{j+1})
12            fi
13          fi
14      od
15      if no swap done break fi
    od
```

Figure 6.1: Stochastic ranking using a bubble-sort-like procedure where $U(0,1)$ is a uniform random number generator and N is the number of sweeps going through the whole population. When $P_f = 0$ the ranking is an *over-penalization* and for $P_f = 1$ the ranking is an *under-penalization*. The initial ranking is always generated at random.

Equations (6.6) and (6.7) show that the greater the number of comparisons (N) the less influence the initial ranking will have. It is worth noting that the probability P_w usually varies for different individuals in different stages of ranking (sorting). Now consider a case where P_w is constant during the entire ranking procedure, which is the case when $f_i < f_j$, $\phi_i > \phi_j$; $j \neq i, j = 1, \ldots, \lambda$. Then $P_{fw} = 1$ and $P_{\phi w} = 0$. If $P_f = \frac{1}{2}$ is chosen then $P_w = \frac{1}{2}$. There will be an equal chance for a comparison to be made based on the objective or penalty function. Only a feasible individual is desired as the final solution, therefore P_f should be less than $\frac{1}{2}$ so that there is a bias against infeasible solutions. The strength of the bias can be adjusted easily by adjusting only P_f. When parameter N, the number of sweeps, approaches ∞ then the ranking will be determined by the bias P_f. That is if $P_f > \frac{1}{2}$ the ranking is based on the objective function, and when $P_f < \frac{1}{2}$ the ranking is the over-penalty ranking. Hence, an increase in the number of ranking sweeps is effectively equivalent to changing parameter P_f, i.e., making it smaller or larger. Thus

$N = \lambda$ can be fixed and P_f adjusted to achieve the best performance. These points are illustrated by optimizing a set of benchmark functions presented in appendix A.1 using different P_f values. Table 6.1 presents the average results for 30 independent runs of the evolutionary strategy presented in section 2.2.1 using stochastic ranking. The numbers in the table indicate the percentage of feasible individuals in the final population. The details about the experiment are given in the following section. It is quite clear from the table that as $P_f > \frac{1}{2}$ finding feasible solutions becomes very difficult unless the unconstrained optimum happens to be the same as the constrained optimum, as is the case for problem `g12`.

Table 6.1: Average percentage feasible individuals in the final population as a function of P_f ($N = \lambda$) and test function presented in appendix A.1.

fcn\P_f	0.525	0.500	0.475	0.450	0.000
g01	0	0	11	83	82
g02	4	30	50	57	58
g03	0	0	0	2	22
g04	0	17	78	78	80
g05	0	0	78	82	81
g06	0	0	0	53	81
g07	0	1	71	80	79
g08	0	100	100	100	100
g09	0	16	83	83	86
g10	0	0	0	74	73
g11	0	0	85	90	70
g12	100	100	100	100	5
g13	0	0	73	79	44

6.3 Experimental Study

The evolutionary optimization algorithm applied in this study is the evolution strategy described in section 2.2.1. One reason for choosing evolution strategy is that it does not introduce any specialized constraint-handling variation operators. It will be shown that specialized and complex variation operators for constrained optimization problems are unnecessary although they may be quite useful for particular types of problems (see for example [MNM96]). A simple extension to the evolution strategy, i.e., the use of the stochastic ranking scheme proposed in the previous section, can achieve significantly better results than other more complicated techniques [RY00].

Thirteen benchmark functions are studied. The first 12 are taken from [KM99] and the 13th from [Mic95]. The details, including the original sources, of these functions are listed in appendix A.1. Problems f_{c2}, f_{c3}, f_{c8}, and f_{c12} are maximization problems. They may be transformed to minimization problems using $-f(\mathbf{x})$. For each of the benchmark problems 30 independent runs are performed using a $(30, 200)$-ES and the stochastic ranking procedure described in the previous section. All runs are terminated after $G = 1750$ generations except for f_{c12}, which was run for 175 generations. Problem f_{c12} is the harder version studied in [KM99] where the feasible region of the search space consists of 9^3 disjointed spheres with a radius of 0.25.

Table 6.2 summarizes the experimental results obtained using $P_f = 0.45$. The median number of generations for finding the best solution in each run is indicated by G_m in the table. The table also shows the known 'optimal' solution for each problem and statistics for the 30 independent runs. These include the best objective value found, median, mean, standard deviation and worst found. The statistics are based on feasible solutions only. All equality constraints have been converted into inequality constraints, $|\mathrm{h}(\mathbf{x})| - \delta \leq 0$, using the degree of violation $\delta = 0.0001$. As a result of this approximation, some results might be better than the optimum. However, the tolerated violation is more stringent than others [Mic95] where $\delta = 0.001$ was used.

In order to evaluate the impact of P_f on the results generated by the algorithm, the same set of experiments are run many times using $P_f = 0, 0.025, \ldots,$

Table 6.2: Experimental results on thirteen benchmark functions using ES with stochastic ranking ($P_f = 0.45$). 30 independent runs were carried out.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{c1}	−15.000	−15.000	−15.000	−15.000	0.0E+00	−15.000	741
f_{c2}	−0.803619	−0.803515	−0.785800	−0.781975	2.0E−02	−0.726288	1086
f_{c3}	−1.000	−1.000	−1.000	−1.000	1.9E−04	−1.000	1146
f_{c4}	−30665.539	−30665.539	−30665.539	−30665.539	2.0E−05	−30665.539	441
f_{c5}	5126.498	5126.497	5127.372	5128.881	3.5E+00	5142.472	258
f_{c6}	−6961.814	−6961.814	−6961.814	−6875.940	1.6E+02	−6350.262	590
f_{c7}	24.306	24.307	24.357	24.374	6.6E−02	24.642	715
f_{c8}	−0.095825	−0.095825	−0.095825	−0.095825	2.6E−17	−0.095825	381
f_{c9}	680.630	680.630	680.641	680.656	3.4E−02	680.763	557
f_{c10}	7049.331	7054.316	7372.613	7559.192	5.3E+02	8835.655	642
f_{c11}	0.750	0.750	0.750	0.750	8.0E−05	0.750	57
f_{c12}	−1.000000	−1.000000	−1.000000	−1.000000	0.0E+00	−1.000000	82
f_{c13}	0.053950	0.053957	0.057006	0.067543	3.1E−02	0.216915	349

Table 6.3: Experimental results on thirteen benchmark functions using ES with stochastic ranking ($P_f = 0$). 30 independent runs were carried out.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{c1}	−15.000	−15.000	−15.000	−15.000	0.0E+00	−15.000	697
f_{c2}	−0.803619	−0.803578	−0.785253	−0.783049	1.5E−02	−0.750656	1259
f_{c3}	−1.000	−0.327	−0.090	−0.105	7.2E−02	−0.014	61
f_{c4}	−30665.539	−30665.539	−30665.538	−30664.710	3.8E+00	−30644.897	632
f_{c5}	5126.498	5126.945	5225.100	5348.683	2.7E+02	6050.566	213
f_{c6}	−6961.814	−6961.814	−6961.814	−6961.814	1.9E−12	−6961.814	946
f_{c7}	24.306	24.322	24.367	24.382	5.9E−02	24.598	546
f_{c8}	−0.095825	−0.095825	−0.095825	−0.095825	2.7E−17	−0.095825	647
f_{c9}	680.630	680.632	680.657	680.671	3.8E−02	680.772	414
f_{c10}	7049.331	7117.416	7336.280	7457.597	3.4E+02	8464.816	530
f_{c11}	0.750	0.750	0.953	0.937	5.4E−02	0.973	1750
f_{c12}	−1.000000	−0.999972	−0.999758	−0.999766	1.4E−04	−0.999488	90
f_{c13}	0.053950	0.919042	0.997912	0.993372	1.5E−02	0.998316	1750

0.525. As expected, neither small nor large (i.e., > 0.5) P_f gave very good results. The best results are obtained when $0.4 < P_f < 0.5$. This indicates that a minor bias toward the dominance of the penalty function encourages the evolution of feasible solutions while still maintaining infeasible regions as potential 'bridges' to move among feasible regions in the whole search space. This is illustrated by plotting the expected fitness landscape for the highly disjointed test function f_{c12}. The result is shown in figure 6.2 for a mean step size of 0.2 and the two values of $P_f = 0$ and $P_f = 0.45$. The dotted line in the figure represents the unconstrained function. The second and third variables are held at the constant value 5. For an over-penalization ($P_f = 0$) there exist regions of negative progress. Positive progress may be regained in these regions by increasing the value of P_f towards 0.5. The rate of progress for this particular problem will be at a maximum when optimizing the unconstrained problem ($P_f = 1$). By maintaining positive progress towards the optimum in the infeasible regions the evolutionary algorithm is able to traverse these areas.

Tables 6.3 and 6.4 give two sets of experimental results when $P_f = 0$ and $P_f = 0.475$ respectively. $P_f = 0$ is an extreme case where all infeasible individuals were ranked lower than feasible individuals. Among feasible solutions, the ranking was based solely on the objective function. Among infeasible solutions, the ranking is based only on the penalty function. This extreme case is somewhat similar to [Deb99], but not the same because it does not use

the worst fitness value of feasible solutions. Although the algorithm did not perform as well as when $P_f = 0.45$ for problems f_{c3}, f_{c4}, f_{c5}, f_{c11}, f_{c12}, and f_{c13}, it performed roughly the same as when $P_f = 0.45$ for other problems. When $P_f = 0.475$, the penalty against infeasible solution is weakened. The algorithm could only find a feasible solution 6 times out of 30 runs for problem f_{c10} although it found a feasible solution 100% times for all other problems. In general, the algorithm improved its performance and found best solutions when P_f was changed from 0.45 to 0.475, except for problems f_{c1}, f_{c3}, and f_{c6}. The improvement is especially noticeable for functions f_{c13} and f_{c4}.

It is important to emphasize that the performance of any evolutionary algorithm for constrained optimization is determined by the constraint-handling technique used as well as the evolutionary search algorithm (including parameters). Throughout the study, modification of the ES have been kept to the

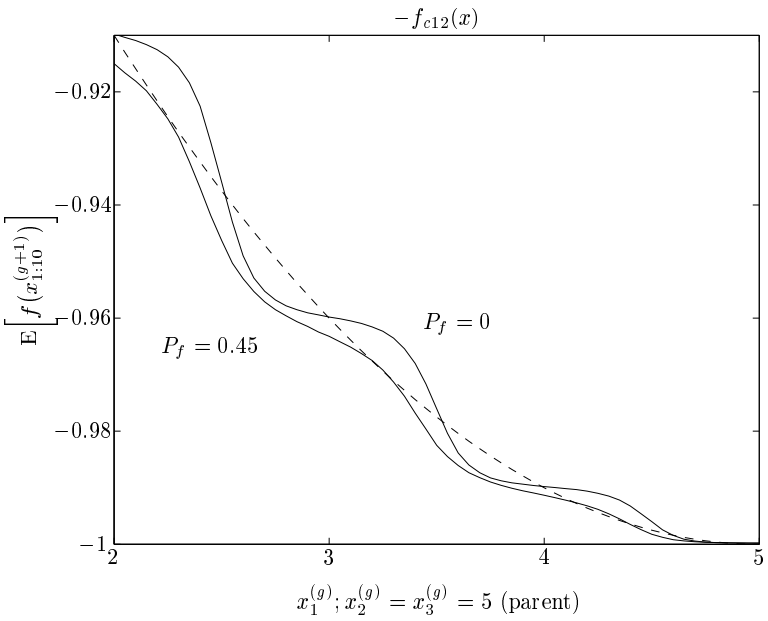


Figure 6.2: Expected fitness landscapes for Michalewicz and Koziel's function (f_{c12}) for the mean step sizes of $\sigma = 0.2$ and $P_f = 0, 0.45$. The dotted line represents the unconstrained function value.

Table 6.4: Experimental results on thirteen benchmark functions using ES with stochastic ranking ($P_f = 0.475$). 30 independent runs were carried out. $^{\star)}$ Based on the 6 feasible solutions found out of the 30 runs.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{c1}	−15.000	−15.000	−5.736	−6.793	3.5E+00	−2.356	951
f_{c2}	−0.803619	−0.802760	−0.792272	−0.786084	1.8E−02	−0.731226	1301
f_{c3}	−1.000	−0.998	−0.995	−0.995	2.1E−03	−0.990	935
f_{c4}	−30665.539	−30665.539	−30665.539	−30665.539	1.1E−11	−30665.539	349
f_{c5}	5126.498	5126.518	5127.276	5128.538	3.1E+00	5141.085	448
f_{c6}	−6961.814	−6871.345	−6603.846	−6572.309	2.1E+02	−6058.588	14
f_{c7}	24.306	24.307	24.317	24.328	2.6E−02	24.392	1568
f_{c8}	−0.095825	−0.095825	−0.095825	−0.095825	2.7E−17	−0.095825	463
f_{c9}	680.630	680.630	680.634	680.640	1.4E−02	680.676	1449
$f_{c10}^{\star)}$	7049.331	7202.108	7343.603	7384.116	1.8E+02	7688.864	1321
f_{c11}	0.750	0.750	0.750	0.750	8.7E−06	0.750	118
f_{c12}	−1.000000	−1.000000	−1.000000	−1.000000	0.0E+00	−1.000000	69
f_{c13}	0.053950	0.053945	0.054000	0.054179	5.0E−04	0.056224	573

minimum, i.e. changing only the selection scheme. The parameters were also set according to previous recommendations in published books and articles. This, however, does not imply that the search algorithm plays an unimportant role in constrained optimization. To illustrate that the combined effect of a constraint handling technique and a search algorithm can make a big difference, the experiment on function f_{c10} is repeated using $\varphi^* = 1/4$ (instead of 1) for the computation of the learning rates τ and τ'. These results are given in table 6.5. A significant improvement was achieved in comparison with the results in tables 6.2 and 6.3.

Table 6.5: Experimental result on function f_{c10} using ES with stochastic ranking and $\varphi^* = 1/4$.

P_f	optimal	best	median	mean	st. dev.	worst	G_m
0.45	7049.331	7049.852	7054.111	7056.163	5.7E+00	7068.633	1733
0.00	7049.331	7049.955	7062.673	7074.044	3.1E+01	7196.647	1745

An interesting question that arises naturally here is whether or not the ES was fully responsible for the good results obtained, e.g., those in table 6.2, in other words, whether stochastic ranking contributed anything to the good results. To answer this question, an additional set of experiments was carried out using exactly the same ES but with a different constraint handling technique — the dynamic penalty method of [JH94]. The results are summarized

Table 6.6: Experimental results using the dynamic penalty method of [JH94] with $r_g = g/2$. The subscript in the function name indicates the number of feasible solutions found if it was between 1 and 29 inclusive. "–" means no feasible solutions were found.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{c1}	−15.000	−14.990	−14.970	−14.968	1.3E−02	−14.943	122
f_{c2}	−0.803619	−0.803597	−0.783042	−0.777241	2.3E−02	−0.710725	1007
f_{c3}	−1.000	–	–	–	–	–	–
f_{c4}	−30665.539	−30648.7	−30007.6	−30021.8	1.7E+02	−29804.6	5
f_{c5}	5126.498	–	–	–	–	–	–
f_{c6}	−6961.814	−6897.969	−6534.206	−6502.478	2.3E+02	−5962.775	12
f_{c7}	24.306	24.347	24.417	24.479	1.6E−01	24.934	109
f_{c8}	−0.095825	−0.095825	−0.095825	−0.095354	1.5E−03	−0.087752	278
f_{c9}	680.630	680.632	680.638	680.648	2.7E−02	680.761	109
f_{c10}	7049.331	–	–	–	–	–	–
$f_{c11(16)}$	0.750	0.750	0.750	0.758	2.5E−02	0.850	14
f_{c12}	−1.000000	−1.000000	−1.000000	−1.000000	0.0E+00	−1.000000	65
$f_{c13(25)}$	0.053950	0.281242	0.448114	0.474460	1.1E−01	0.901263	1750

in tables 6.6 and 6.7.

Comparing tables 6.2 and 6.6, it is clear that stochastic ranking performed better than the dynamic penalty method with $r_g = g/2$ [JH94] according to all four criteria (*best, median, mean*, and *worst*) for all benchmark functions except for f_{c2}, f_{c9}, and f_{c12}. The two methods performed the same on problem f_{c12}. The dynamic penalty method found a better *best* than stochastic ranking for problem f_{c2}, but performed worse than stochastic ranking according to *median, mean*, and *worst*. On the other hand, stochastic ranking found a better *best* (i.e., the optimum) for problem f_{c9}, but performed worse than the dynamic penalty method according to *median, mean*, and *worst*.

The results in table 6.7 with $r_g = (g/2)^2$ improved those in table 6.6 for most, but not all, problems. Feasible solutions can now be found for problem f_{c3}. The results for several problems are now better than those in table 6.6. However, none of these improvements has changed the general picture. The results from stochastic ranking are still better than those from the dynamic penalty method [JH94] with $r_g = (g/2)^2$. In fact, the dynamic penalty method with $r_g = (g/2)^2$ is only better than stochastic ranking for problem f_{c2}, but has lost its advantage for problem f_{c9}. This is not very surprising because the dynamic penalty method relies on a predefined sequence of r_g while stochastic ranking is an adaptive method without any predefined sequence.

Table 6.7: Experimental results using the dynamic penalty method of [JH94] with $r_g = (g/2)^2$. "–" means no feasible solutions were found.

fcn	optimal	best	median	mean	st. dev.	worst	G_m
f_{c1}	−15.000	−15.000	−15.000	−15.000	7.9E−05	−15.000	217
f_{c2}	−0.803619	−0.803587	−0.785907	−0.784868	1.5E−02	−0.75162	1235
f_{c3}	−1.000	−0.583	−0.045	−0.103	1.4E−01	−0.001	996
f_{c4}	−30665.539	−30365.488	−30060.607	−30072.458	1.2E+02	−29871.44	4
f_{c5}	5126.498	–	–	–	–	–	–
f_{c6}	−6961.814	−6911.247	−6547.354	−6540.012	2.6E+02	−5868.028	13
f_{c7}	24.306	24.309	24.375	24.421	2.2E−01	25.534	180
f_{c8}	−0.095825	−0.095825	−0.095825	−0.095825	2.8E−17	−0.095825	421
f_{c9}	680.630	680.632	680.648	680.659	3.2E−02	680.775	1739
f_{c10}	7049.331	–	–	–	–	–	–
f_{c11}	0.750	0.750	0.750	0.750	9.1E−06	0.750	61
f_{c12}	−1.000000	−1.000000	−0.999818	−0.999838	1.3E−04	−0.99957	68
f_{c13}	0.053950	0.514152	0.996674	0.965397	9.4E−02	0.99816	1750

Predefined sequences are unable to adjust r_g according to different problems and different search stages for a problem. While a predefined sequence may work well for one problem it may not for a different problem. This is what happened to the dynamic penalty method. On the other hand, an adaptive method like stochastic ranking can adjust the balance between objective and penalty functions automatically for different problems and during different stages of evolutionary search. However, there is no free lunch in optimization [WM97]. The price paid by an adaptive method is slightly longer search time for the algorithm to adapt. This can be observed from the last column in tables 6.2 and 6.6.

6.4 Summary

This chapter proposed a new constraint handling technique — stochastic ranking. The technique does not introduce any specialized variation operators. It does not require a priori knowledge about a problem since it does not use any penalty coefficient r_g in a penalty function. Stochastic ranking is motivated from the analysis of penalty methods from the point of view of dominance. The balance between the objective and penalty functions is achieved through a ranking procedure based on the stochastic bubble sort algorithm. The introduction of a single probability of P_f produces a convenient means of specifying an agreeable bias towards the objective function in ranking in-

dividuals. Experimental results have been presented and are an improvement on the performance of previous evolutionary optimization methods [RY00]. The experimental results suggest that a value of $0.4 < P_f < 0.5$ would be appropriate for many constrained optimization problems.

Chapter 7

Conclusion

The difference between human problem solving and problem solving by natural systems is, that for the latter case problems are not predefined, but rather they co-evolve with their solution. Essentially, evolutionary problem solving requires setting up a situation in order to get the simulation off the ground. The form of a solution must be pre-specified. The problem-solution must be abstracted and so the search space is defined. On a computational device, this information appears in the form of a data structure, similar to what base sequences of nucleic acids seem to provide in natural systems. With information comes meaning, instructions to build solutions, which make only sense in relation to the problem (environment). The search for solutions requires variations of the data structure. For intelligent search, these variations effectively reason with the structure. The variation is an inference that determines the location of the most viable solutions. It is often thought that random variations operate blindly on data structures without any regard to their meaning. This gives the impression of mutations doing more harm than good, but can only be presumed when the meaning of the data is unknown. For 'intelligent' search the meaning of the data and its reasoning by variation must be co-adapted for every problem-solution. It is the designer of the algorithm who supplies this knowledge and when sufficient, search may not be necessary at all. The more 'intelligent' problem solver is nevertheless capable of automatic learning, requiring fewer instructions from its designer, but this also requires search. For 'intelligent' general problem solving, an architecture must be chosen such that the search is meaningful, regardless of what the problem may be. The symbolic structures do not have this property, they can only be understood on the

basis of pre-existing non-symbolic forms. Connectionist models do, however, provide meaningful search, small changes in the connection weights provide in general small changes in behaviour. The meaning of the connectionist model adapts to its variation.

7.1 Discussion of Contributions

The primary objective of this work has been to enhance the understanding of how evolutionary problem solving works. This has been done by treating the method like any other problem solving strategy using a description of a situation, operators for changing the situation, and an evaluation of the situation.

1. **Problem-solution description**

An attempt has been made to gain a wider understanding of how problems and their candidate solutions may be described for 'intelligent' problem solving. The problem corresponds to the environment and the solution to the organism. This is illustrated in figure 3.1. The most common approach in evolutionary computation is to isolate the problem from its solution. By doing so all knowledge about the problem must be reflected in the solution representation and variation operators. How this may be done is not always trivial, but when done properly search is probably unnecessary. This difficulty is inherent for symbolic representations as illustrated by example in sections 3.2.1 and 4.1. 'Intelligent' search requires automatic learning about the problem. This can be done more effectively and efficiently when the solution and problem interact as shown in figure 3.1. The search for the knowledge of how to solve a problem requires itself meta-knowledge. A meaningful search for this knowledge was illustrated using connectionist models in section 3.2.2. These connectionist models were extended in chapter 4 and two novel architectures for evolutionary problem solving were presented. The first architecture, a deliberative connectionist model, produces *plastic* solutions. It not only solves the problem set but also variations of it. This implies that re-optimization is not necessary when any conditions are changed. This would, however, be the case for the symbolic model. The novelty of the approach was demonstrated for scheduling problems. The second architecture, a reactive connectionist model,

learned how to optimize neural network weights. Any new linearly separable task is, therefore, solved without the need for any further evolutionary search.

2. **Evolutionary search**

It may be that the "building block thesis" [Hol00] holds, but understanding *the algorithm's behaviour* remains difficult when the 'blocks' are manipulated without any regard to their meaning. When symbolic representations are used, it must first be understood how the system functions *without* a code and then it may be imagined what would be necessary so that some aspects of this functioning could be 'encoded'. Once the 'coding relationship' is in place, it will be known how the 'blocks' should be varied. It is believed that the architecture of natural systems is such that random variations correspond to intelligent inferences. Meaningful variations are created by specifying meaningful distance metrics for the problem-solution studied. Small changes correspond to small changes in fitness. The variation operates in the metric space. The data structures and corresponding fitness values form a fitness landscape. A number of probabilistic inferences are made in order to locate fitter solutions. Only one of these need be successful on average in order to maintain search progress. The result of generating a number of alternative solutions, followed by selection, is effectively to smooth the fitness landscape. This introduces a new landscape concept, called the *expected fitness landscape*. Neighbourhoods are determined by the parents and the fitness by the expected fitness of their best offspring. This landscape is shaped by the distance metric chosen, the operators working in this space, the number of offspring produced, and their fitness value. In order for the evolutionary search to work, this landscape must be unimodal, or at least locally so, and the local optimum of the expected fitness landscape must be equivalent to that of the goal. Furthermore, if global convergence in mean is to be obtained, positive progress in terms of fitness must also be maintained on average. In some cases, this may be achieved by adaptively increasing the number of offspring generated.

3. **Constraint handling**

When constraints are not 'naturally' satisfied by the problem-solution description or the variation operator, it must be treated by selection. This requires an additional measure for constraint violations. Individuals are selected based

on their original fitness value and the degree of constraint violation. Preference must be given to solutions that satisfy all constraints. In order to understand how different constraint handling techniques perform, evolutionary search must first be understood. Clearly the evolutionary search must also work for the unconstrained problem. This understanding was enhanced by the description of expected fitness landscapes. This concept can now be applied to interpret the influence of a constraint handling technique on evolutionary search. By introducing a measure for constraint violation a smooth landscape may be transformed to a rugged one. The more emphasis that is put on the constraints, the more rugged it becomes. This was illustrated by example in figure 6.2 by plotting the expected fitness landscape for when the infeasible individuals are *over-penalized*. When a balance is struck between the unconstrained fitness and the measure of constraint violations, as illustrated in the same figure using stochastic ranking, then the degree of ruggedness may be controlled. Individuals are able to traverse infeasible regions, but only when they locate superior feasible regions. For this a balance must be maintained between venturing across infeasible regions and exploring the current feasible region. This is what the new constraint handling technique, stochastic ranking, does effectively.

7.2 Directions for Future Work

A problem solving architecture must include in some form representations which can be interpreted by the human designer, otherwise the solutions will have no meaning. Similarly, designers must be capable of describing their goals and conditions. In other words, some representation will always be presupposed, this is what is required in setting up a situation. An 'intelligent' problem solver will be capable of adapting any further representations and variations thereof for efficient and effective search. This, in turn, requires a greater understanding of the architectures of cognition and the nature of representation. With this insight, it may be possible to design *general 'intelligent' problem solvers* in the future. Currently, connectionist models look as if they are the more suitable universal representation for evolutionary search.

Appendix A

Test Function Suite

A.1 Constrained

Floundas & Pardalos

Minimize [FP87]:

$$f_{c1}(\mathbf{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \tag{1.1}$$

subject to:

$$\begin{aligned}
\mathrm{g}_1(\mathbf{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
\mathrm{g}_2(\mathbf{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
\mathrm{g}_3(\mathbf{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
\mathrm{g}_4(\mathbf{x}) &= -8x_1 + x_{10} \leq 0 \\
\mathrm{g}_5(\mathbf{x}) &= -8x_2 + x_{11} \leq 0 \\
\mathrm{g}_6(\mathbf{x}) &= -8x_3 + x_{12} \leq 0 \\
\mathrm{g}_7(\mathbf{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
\mathrm{g}_8(\mathbf{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
\mathrm{g}_9(\mathbf{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned} \tag{1.2}$$

where the bounds are $0 \leq x_i \leq 1$ $(i = 1, \ldots, 9)$, $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The global minimum is at $\mathbf{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ where six constraints are active ($\mathrm{g}_1, \mathrm{g}_2, \mathrm{g}_3, \mathrm{g}_7, \mathrm{g}_8$ and g_9) and $f_{c1}(\mathbf{x}^*) = -15$.

Keane

Maximize [KM99]:

$$f_{c2}(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right| \tag{1.3}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned} \tag{1.4}$$

where $n = 20$ and $0 \leq x_i \leq 10$ $(i = 1, \ldots, n)$. The global maximum is unknown, the best we found is $f_{c2}(\mathbf{x}^*) = 0.803619$ (which, to the best of our knowledge, is better than any reported value), constraint g_1 is close to being active ($\mathrm{g}_1 = -10^{-8}$).

Michalewicz et. al.

Maximize [MNM96]:

$$f_{c3}(\mathbf{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i \tag{1.5}$$

$$\mathrm{h}_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0 \tag{1.6}$$

where $n = 10$ and $0 \leq x_i \leq 1$ $(i = 1, \ldots, n)$. The global maximum is at $x_i^* = 1/\sqrt{n}$ $(i = 1, \ldots, n)$ where $f_{c3}(\mathbf{x}^*) = 1$.

Himmelblau, problem 11

Minimize [Him72]:

$$f_{c4}(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{1.7}$$

subject to:

$$\begin{aligned}
\mathrm{g}_1(\mathbf{x}) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0 \\
\mathrm{g}_2(\mathbf{x}) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 \\
\mathrm{g}_3(\mathbf{x}) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0 \\
\mathrm{g}_4(\mathbf{x}) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0 \\
\mathrm{g}_5(\mathbf{x}) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0 \\
\mathrm{g}_6(\mathbf{x}) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0
\end{aligned} \tag{1.8}$$

where $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ and $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$. The optimum solution is $\mathbf{x}^*$ =(78, 33, 29.995256025682, 45, 36.775812905788) where $f_{c4}(\mathbf{x}^*) = -30665.539$. Two constraints are active (g_1 and g_6).

Hock & Schittkowski, problem 74

Minimize [HS81]:

$$f_{c5}(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3 \tag{1.9}$$

subject to:

$$\begin{aligned}
\mathrm{g}_1(\mathbf{x}) &= -x_4 + x_3 - 0.55 \leq 0 \\
\mathrm{g}_2(\mathbf{x}) &= -x_3 + x_4 - 0.55 \leq 0 \\
\mathrm{h}_3(\mathbf{x}) &= 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \\
\mathrm{h}_4(\mathbf{x}) &= 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \\
\mathrm{h}_5(\mathbf{x}) &= 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0
\end{aligned} \tag{1.10}$$

where $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. The best known solution [KM99] $\mathbf{x}^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ where $f_{c5}(\mathbf{x}^*) = 5126.4981$.

Flounds & Pardalos

Minimize [FP87]:

$$f_{c6}(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3 \tag{1.11}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= -(x_1-5)^2-(x_2-5)^2+100 \le 0 \\ \mathrm{g}_2(\mathbf{x}) &= (x_1-6)^2+(x_2-5)^2-82.81 \le 0 \end{aligned} \tag{1.12}$$

where $13 \le x_1 \le 100$ and $0 \le x_2 \le 100$. The optimum solution is $\mathbf{x}^* = (14.095, 0.84296)$ where $f_{c6}(\mathbf{x}^*) = -6961.81388$. Both constraints are active.

Hock & Schittkowski, problem 113

Minimize [HS81]:

$$\begin{aligned} f_{c7}(\mathbf{x}) = \; & x_1^2+x_2^2+x_1x_2-14x_1-16x_2+(x_3-10)^2+4(x_4-5)^2+(x_5-3)^2+ \\ & 2(x_6-1)^2+5x_7^2+7(x_8-11)^2+2(x_9-10)^2+(x_{10}-7)^2+45 \end{aligned} \tag{1.13}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= -105+4x_1+5x_2-3x_7+9x_8 \le 0 \\ \mathrm{g}_2(\mathbf{x}) &= 10x_1-8x_2-17x_7+2x_8 \le 0 \\ \mathrm{g}_3(\mathbf{x}) &= -8x_1+2x_2+5x_9-2x_{10}-12 \le 0 \\ \mathrm{g}_4(\mathbf{x}) &= 3(x_1-2)^2+4(x_2-3)^2+2x_3^2-7x_4-120 \le 0 \\ \mathrm{g}_5(\mathbf{x}) &= 5x_1^2+8x_2+(x_3-6)^2-2x_4-40 \le 0 \\ \mathrm{g}_6(\mathbf{x}) &= x_1^2+2(x_2-2)^2-2x_1x_2+14x_5-6x_6 \le 0 \\ \mathrm{g}_7(\mathbf{x}) &= 0.5(x_1-8)^2+2(x_2-4)^2+3x_5^2-x_6-30 \le 0 \\ \mathrm{g}_8(\mathbf{x}) &= -3x_1+6x_2+12(x_9-8)^2-7x_{10} \le 0 \end{aligned} \tag{1.14}$$

where $-10 \le x_i \le 10$ $(i = 1, \ldots, 10)$. The optimum solution is $\mathbf{x}^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$ where $f_{c7}(\mathbf{x}^*) = 24.3062091$. Six constraints are active (g_1, g_2, g_3, g_4, g_5 and g_6).

Schoenauer & Xanthakis

Maximize [KM99]:

$$f_{c8}(\mathbf{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)} \tag{1.15}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= x_1^2 - x_2 + 1 \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{aligned} \tag{1.16}$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is located at $\mathbf{x}^* = (1.2279713, 4.2453733)$ where $f_{c8}(\mathbf{x}^*) = 0.095825$. The solution lies within the feasible region.

Hock & Schittkowski, problem 100

Minimize [HS81]:

$$\begin{aligned} f_{c9}(\mathbf{x}) = \quad & (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + \\ & 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \end{aligned} \tag{1.17}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\ \mathrm{g}_3(\mathbf{x}) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\ \mathrm{g}_4(\mathbf{x}) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \end{aligned} \tag{1.18}$$

where $-10 \leq x_i \leq 10$ for $(i = 1, \ldots, 7)$. The optimum solution is $\mathbf{x}^* = (2.330499,\ 1.951372,\ -0.4775414,\ 4.365726,\ -0.6244870,\ 1.038131,\ 1.594227)$ where $f_{c9}(\mathbf{x}^*) = 680.6300573$. Two constraints are active (g_1 and g_4).

Hock & Schittkowski, heat exchanger design

Minimize [HS81]:

$$f_{c10}(\mathbf{x}) = x_1 + x_2 + x_3 \tag{1.19}$$

subject to:

$$\begin{aligned} \mathrm{g}_1(\mathbf{x}) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\ \mathrm{g}_2(\mathbf{x}) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\ \mathrm{g}_3(\mathbf{x}) &= -1 + 0.01(x_8 - x_5) \leq 0 \\ \mathrm{g}_4(\mathbf{x}) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\ \mathrm{g}_5(\mathbf{x}) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\ \mathrm{g}_6(\mathbf{x}) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0 \end{aligned} \tag{1.20}$$

where $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$ $(i = 2, 3)$ and $10 \leq x_i \leq 1000$ $(i = 4, \ldots, 8)$. The optimum solution is $\mathbf{x}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$ where $f_{c10}(\mathbf{x}^*) = 7049.3307$. Three constraints are active (g_1, g_2 and g_3).

Maa & Shanblatt

Minimize [KM99]:

$$f_{c11}(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \tag{1.21}$$

subject to:

$$\mathrm{h}(\mathbf{x}) = x_2 - x_1^2 = 0 \tag{1.22}$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The optimum solution is $\mathbf{x}^* = (\pm 1/\sqrt{2}, 1/2)$ where $f_{c11}(\mathbf{x}^*) = 0.75$.

Michalewicz & Koziel

Maximize [KM99]:

$$f_{c12}(\mathbf{x}) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100 \tag{1.23}$$

subject to:

$$\mathrm{g}(\mathbf{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0 \tag{1.24}$$

where $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds. The optimum is located at $\mathbf{x}^* = (5, 5, 5)$ where $f_{c12}(\mathbf{x}^*) = 1$. The solution lies within the feasible region.

Hock & Schittkowski

Minimize [HS81]:

$$f_{c13}(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5} \tag{1.25}$$

subject to:

$$\begin{aligned} \mathrm{h}_1(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ \mathrm{h}_2(\mathbf{x}) = x_2 x_3 - 5 x_4 x_5 = 0 \\ \mathrm{h}_3(\mathbf{x}) = x_1^3 + x_2^3 + 1 = 0 \end{aligned} \tag{1.26}$$

where $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. The optimum solution is $\mathbf{x}^* = (-1.717143,\ 1.595709,\ 1.827247,\ -0.7636413,\ -0.763645)$ where $f_{c13}(\mathbf{x}^*) = 0.0539498$.

A.2 Unconstrained

Sphere Model

$$f_{u1}(x) = \sum_{i=1}^{30} x_i^2$$

$$-100 \leq x_i \leq 100, \quad \min(f_{u1}) = f_{u1}(0, \ldots, 0) = 0$$

Schwefel's Problem 2.22

$$f_{u2}(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

$$-10 \leq x_i \leq 10, \quad \min(f_{u2}) = f_{u2}(0, \ldots, 0) = 0$$

Schwefel's Problem 1.2

$$f_{u3}(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$-100 \leq x_i \leq 100, \quad \min(f_{u3}) = f_{u3}(0, \ldots, 0) = 0$$

Schwefel's Problem 2.21

$$f_{u4}(x) = \max_i \{|x_i|, 1 \leq i \leq 30\}$$

$$-100 \leq x_i \leq 100, \quad \min(f_{u4}) = f_{u4}(0, \ldots, 0) = 0$$

Generalised Rosenbrock's Function

$$f_{u5}(x) = \sum_{i=1}^{29} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

$$-30 \leq x_i \leq 30, \quad \min(f_{u5}) = f_{u5}(1, \ldots, 1) = 0$$

Step Function

$$f_{u6}(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$$

$$-100 \leq x_i \leq 100, \quad \min(f_{u6}) = f_{u6}(0, \ldots, 0) = 0$$

Quartic Function with Noise

$$f_{u7}(x) = \sum_{i=1}^{30} i x_i^4 + random[0, 1)$$

$$-1.28 \leq x_i \leq 1.28, \quad \min(f_{u7}) = f_{u7}(0, \ldots, 0) = 0$$

Generalised Schwefel's Problem 2.26

$$f_{u8}(x) = -\sum_{i=1}^{30} \left(x_i \sin\left(\sqrt{|x_i|} \right) \right)$$

$$-500 \leq x_i \leq 500, \quad \min(f_{u8}) = f_{u8}(420.9687, \ldots, 420.9687) = -12569.5$$

Generalised Rastrigin's Function

$$f_{u9}(x) = \sum_{i=1}^{30}[x_i^2 - 10\cos(2\pi x_i) + 10)]$$

$$-5.12 \leq x_i \leq 5.12, \quad \min(f_{u9}) = f_{u9}(0,\ldots,0) = 0$$

Ackley's Function

$$f_{u10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30}x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i\right) + 20 + e$$

$$-32 \leq x_i \leq 32, \quad \min(f_{u10}) = f_{u10}(0,\ldots,0) = 0$$

Generalised Griewank Function

$$f_{u11}(x) = \frac{1}{4000}\sum_{i=1}^{30}x_i^2 - \prod_{i=1}^{30}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \leq x_i \leq 600, \quad \min(f_{u11}) = f_{u11}(0,\ldots,0) = 0$$

Generalised Penalised Functions

$$\begin{aligned} f_{u12}(x) &= \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2\right\} \\ &\quad + \sum_{i=1}^{30}u(x_i,10,100,4) \end{aligned}$$

$$-50 \leq x_i \leq 50, \quad \min(f_{u12}) = f_{u12}(1,\ldots,1) = 0$$

$$\begin{aligned} f_{u13}(x) &= 0.1\Bigg\{\sin^2(\pi 3x_1 + \sum_{i=1}^{29}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + \\ &\quad (x_n-1)[1+\sin^2(2\pi x_{30})]\Bigg\} + \sum_{i=1}^{30}u(x_i,5,100,4) \end{aligned}$$

$$-50 \leq x_i \leq 50, \quad \min(f_{u13}) = f_{u13}(1,\ldots,1) = 0$$

where

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

Shekel's Foxholes Function

$$f_{u14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1}$$

$$-65.536 \le x_i \le 65.536, \quad \min(f_{u14}) = f_{u14}(-32, -32) \approx 1$$

where

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

Kowalik's Function

$$f_{u15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$$

$$-5 \le x_i \le 5, \quad \min(f_{u15}) \approx f_{u15}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$$

Six-hump Camel-Back Function

$$f_{u16} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$$

$$-5 \le x_i \le 5$$

$$x_{min} = (0.08983, -0.7126), \ (-0.08983, 0.7126)$$

$$\min(f_{u16}) = -1.0316285$$

Table A.1: Kowalik's Function f_{u15}

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

Branin Function

$$f_{u17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

$$-5 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 15$$

$$x_{min} = (-3.142, 12.275),\ (3.142, 2.275),\ (9.425, 2.425)$$

$$\min(f_{u17}) = 0.398$$

Goldstein-Price Function

$$\begin{aligned} f_{u18}(x) \quad = \quad & [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ & \times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned}$$

$$-2 \leq x_i \leq 2, \quad \min(f_{u18}) = f_{u18}(0, -1) = 3$$

Hartman's Family

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right]$$

with $n = 3, 6$ for $f_{u19}(x)$ and $f_{u20}(x)$, respectively, $0 \leq x_j \leq 1$. The coefficients are defined by Tables A.2 and A.3, respectively.

Table A.2: Hartman Function f_{u19}

i	$a_{ij}, j = 1, 2, 3$			c_i	$p_{ij}, j = 1, 2, 3$		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

Table A.3: Hartman Function f_{u20}

i	$a_{ij}, j = 1, \cdots, 6$						c_i	$p_{ij}, j = 1, \cdots, 6$					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

For $f_{u19}(x)$ the global minimum is equal to -3.86 and it is reached at the point $(0.114, 0.556, 0.852)$. For $f_{u20}(x)$ the global minimum is -3.32 at the point $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$.

Shekel's Family

$$f(x) = -\sum_{i=1}^{m}[(x - a_i)(x - a_i)^T + c_i]^{-1}$$

with $m = 5, 7$ and 10 for $f_{u21}(x)$, $f_{u22}(x)$ and $f_{u23}(x)$, respectively, $0 \leq x_j \leq 10$.

These functions have 5, 7 and 10 local minima for $f_{u21}(x)$, $f_{u22}(x)$, and $f_{u23}(x)$, respectively. $x_{local\_opt} \approx a_i$, $f(x_{local\_opt}) \approx 1/c_i$ for $1 \leq i \leq m$. The coefficients are defined by Table A.4.

Table A.4: Shekel Functions $f_{u21}, f_{u22}, f_{u23}$

i	$a_{ij}, j = 1, \cdots, 4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

Bibliography

[BA85] A.G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Trans. Syst. Man Cybern.*, 4:360–375, 1985.

[Bäc96] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[Bal96] J. M. Baldwin. A new factor in evolution. *American Naturalist*, 30:441–451, 1896.

[Bax92] J. Baxter. The evolution of learning algorithms for artificial neural networks. *Complex Systems*, 1992.

[BBC95] S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for ANNs. *Neural Processing Letters*, 2(4):1–5, 1995.

[Ber95] D. P. Bertsekas. *Dynamic Programming and Optimial Control, Vols. I and II.* Athena Scientific, Belmont, Massachusetts, 1995.

[Bey95a] H.-G. Beyer. Toward a Theory of Evolution Strategies: On the Benefit of Sex – the $(\mu/\mu, \lambda)$-Theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[Bey95b] H.-G. Beyer. Toward a Theory of Evolution Strategies: The (μ, λ)-Theory. *Evolutionary Computation*, 2(4):381–407, 1995.

[Bey96] H.-G. Beyer. Toward a Theory of Evolution Strategies: Self-Adaptation. *Evolutionary Computation*, 3(3):311–347, 1996.

[Bey97] H.-G. Beyer. An Alternative Explanation for the Manner in which Genetic Algorithms Operate. *BioSystems*, 41:1–15, 1997.

[BGH89] L. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1-3):235–282, 1989.

[Bro91] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[BS84] J. D. Bransford and B. S. Stein. *The ideal problem solver: A guide for improving thinking, learning, and creativity.* Freeman, New York, 1984.

[BSK96] T. Bäck, M. Schütz, and S. Khuri. Evolution strategies: An alternative evolution computation method. In J. M. Alliot, E. Lutton, E. Ronald, M. Schoenhauer, and D. Snyers, editors, *Artificial Evolution*, pages 3–20. Springer, Berlin, 1996.

[BT96] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmount, Massachusetts, 1996.

[BWW00] L. Barbulescu, J.-P. Watson, and D. Whitley. Dynamic representations and escaping local optima: Improving genetic algorithms and local search. In *The Seventeenth National Conference on Artificial Intelligence*. AAAI, 2000.

[CF99] K. Chellapilla and D. B. Fogel. Evolving neural networks to play checkers without expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, November 1999.

[Cha90] D. Chalmers. The evolution of learning: An experimentation in genetic connectionism. In *Connectionists models: Proceedings of the 1990 summer school*, pages 81–90. San Mateo, CA: Morgan Kaufmann Publishers, 1990.

[Cyb89] G. Cybenko. Approximation by superposition of a sigmoid function. *Mathematics of Control, Signal and Systems*, 2:303–314, 1989.

[Deb99] K. Deb. An efficient constrained handling method for genetic algorithms. In *Computer Methods in Applied Mechanics and Engineering*, page in press, 1999.

[DG99] I. E. Dror and D. P. Gallogly. Computational analysis in cognitive neuroscience: In defence of biological implausibility. *Psychonomic Bulletin & Review*, 6(2):173–182, 1999.

[DT91] J. E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, November 1991.

[EHM99] Á. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[FM68] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. Wiley, New-York, 1968.

[Fog95] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[FP87] C. Floundas and P. Pardalos. *A Collection of Test Problems for Constrained Global Optimization*, volume 455 of *Lecture Notes in Computar Science*. Springer-Verlag, Berlin, Germany, 1987.

[FT63] H. Fisher and G. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*. Prentice Hall, 1963.

[Fun89] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[GC97] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. Wiley, New-York, 1997.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[Hay81] J. R. Hayes. *The Complete Problem Solver*. The Franklin Institute Press, USA, 1981.

[Heb49] D. O. Hebb. *The Organization of Behaviour*. Wiley, New York, 1949.

[Him72] D. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, New-York, 1972.

[HN87] G. Hinton and S. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

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

[Hol00] J. H. Holland. Building blocks, cohort genetic algorithms and hyperplane-defined functions. *Evolutionary Computation*, 8(4):373–391, 2000.

[HS81] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin, Germany, 1981.

[HS96] F. Hoffmeister and J. Sprave. Problem independent handling of constraints by use of metric penalty functions. In L. J. Fogel, P. J. Angeline, and Th. Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 289–294, Cambridge MA, 1996. The MIT Press.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward network are universal approximators. *Neural Networks*, 2:359–366, 1989.

[Hub87] V. Hubka. *Principles of Engineering Design*. Edition Heurista, Zürich, Swiss, 1987.

[Hut95] E. Hutchins. *Cognition in the Wild*. The MIT Press, Cambridge, Massachusetts, 1995.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proceedings of the 6th Int. Conference on Genetic Algorithms*. Kaufman, 1995.

[JH94] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems

with GAs. In *Proc. IEEE International Conference on Evolutionary Computation*, pages 579–584. IEEE Press, 1994.

[JV99] F. Jiménez and J. L. Verdegay. Evolutionary techniques for constrained optimization problems. In *Proc. of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Germany, Berlin, 1999. Springer-Verlag.

[Kap96] C. Kappler. Are evolutionary algorithms improved by large mutations? In *Parallel Problem Solving From Nature – PPSN IV*, Berlin, 1996. Springer.

[Kau91] S. A. Kauffman. Antichaos and adaptation. *Scientific American*, 265(2):64–70, 1991.

[Kau93] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, New York, 1993.

[Kel99] C. T. Kelley. Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. *SIAM Journal of Optimization*, 10:43–55, 1999.

[KM99] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[KP98] S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase in the dynamic penalty terms. In *Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 211–220, Berlin, Germany, 1998. Springer.

[LD85] C.-T. Lung and R. L. Dominowski. Effects of strategy instructions and practice on nine-dot problem solving. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 11(4):804–811, 1985.

[Lew70] R. C. Lewontin. The units of selection. *Annual Review of Ecology and Systematics*, 1:1–18, 1970.

[Lew82] R. C. Lewontin. Organism and environment. In H.C. Plotkin, editor, *Learning, Development and Culture*, page 162. John Wiley & Sons Ltd, New York, 1982.

[Lew83] R. C. Lewontin. The organism as the subject and object of evolution. *Scientia*, 118:63–82, 1983.

[Lew85] R. C. Lewontin. *The Dialectical Biologist*, chapter Adaptation, pages 65–84. Harward University Press, New York, 1985.

[Lew00] R. C. Lewontin. *The Triple Helix: Gene, Organism, and Environment*. Harvard University Press, Cambridge, Massachusetts, 2000.

[LV90] G. Liepins and M. Vose. Representation issues in genetic algorithms. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:101–115, 1990.

[LYL+98] K.-H. Liang, X. Yao, Y. Liu, C. Newton, and D. Hoffman. An experimental investigation of self-adaptation in evolutionary programming. In *et al.* Porto, editor, *Proc. of the 7th Annual Conference on Evolutionary Programming*, pages 291–300, Berlin, Germany, 1998. Springer-Verlag.

[MA94] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In L. J. Fogel and A.V. Sebald, editors, *Proc. of the 2nd Annual Conference on Evolutionary Programming*, pages 98–108, River Edge, NJ, 1994. World Scientific Publishing.

[MAJ91] P. Mazzoni, R. A. Andersen, and M. I. Jordan. A more biological plausible learning rule for neural networks. *Proc. Natl. Acad. Sci. USA*, 88:4433–4437, May 1991.

[Man97] B. Manderick. Fitness landscapes: Correlation analysis. In T. Bäck, D.B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages B2.7:10–14. IOP Publishing Ltd and Oxford University Press, 1997.

[Mar96] G. Marley. Landscapes, learning costs, and genetic assimilation. *Evolutionary Computation*, 4(3):213–234, 1996.

[MF00] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin Heidelberg, 2000.

[Mic95] Z. Michalewicz. Genetic algorithms, numerical optimization and constraints. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 151–158, San Mateo, CA, July 15-19 1995. Morgan Kaufman.

[MMR97] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. The MIT Press, Cambridge, Massachusetts, 1997.

[MNM96] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In L.J. Fogel, P.J. Angeline, and T. Bäck, editors, *Proc. of the 5th Annual Conference on Evolutionary Programming*, pages 305–312. MIT Press, Cambridge, MA, 1996.

[MP43] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MT63] J. F. Muth and G. L. Thompson. *Industrial Scheduling*. Prentice Hall, New Jersey, 1963.

[New90] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[NS72] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[OGH95] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1995.

[PI77] S. S. Panwalker and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[Pol57] G. Polya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, Princeton, New Jersey, second edition, 1957.

[Pos43] E. L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.

[Rec94] I. Rechenberg. *Evolutionstrategie '94*. Frommann-Holzboog, Stuttgart, 1994.

[Ree97] C. R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–247, 1997.

[RHW86] D. E. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1. Foundations*, pages 318–362. Cambridge, MA: MIT Press, 1986.

[RJ99a] T. P. Runarsson and M. Th. Jonsson. Evolution of gene coordination networks. In B. McKay, X. Yao, C.S. Newton, J.-H. Kim, and T. Furuhashi, editors, *Lecture Notes in Artificial Intelligence 1585*, pages 430–437. Springer, 1999.

[RJ99b] T. P. Runarsson and M. Th. Jonsson. Genetic production systems for intelligent problem solving. *Special Issue: Computational Intelligence in Intelligent Manufacturing*, 10(3):181–186, April 1999.

[RJ00] T. P. Runarsson and M. Th. Jonsson. Evolution and design of distributed learning rules. In *The first IEEE symposium on Combinations of Evolutionary Computation and Neural Networks*. IEEE, May 2000.

[RSJ00] T. P. Runarsson, R. Sarker, and M. Th. Jonsson. Constrained nonlinear integer programming, self-adaptation and evolution strategies. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):164–171, July 2000.

[Rud94] G. Rudolph. An evolutionary algorithm for integer programming. In *Parallel Problem Solving From Nature*, volume 3, pages 139–148, 1994.

[Rud97a] G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač, Hamburg, 1997.

[Rud97b] G. Rudolph. Local convergence rates of simple evolutionary algorithms with cauchy mutations. *IEEE Transactions on Evolutionary Computation*, 1(4):249–258, 1997.

[RY00] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[Sal96] R. Salomon. Some comments on evolutionary algorithm theory. *Evolutionary Computation Journal*, 4(4):405–415, 1996.

[Sal98] R. Salomon. Evolutionary algorithms and gradient search: Similarities and differences. *IEEE Transactions on Evolutionary Computation*, 2(2):45–55, 1998.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[SC97] A. E. Smith and D. W. Coit. Penalty functions. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook on Evolutionary Computation*, pages C5.2:1–6. Oxford University Press, 1997.

[Sch77] H.-P. Schwefel. *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basle, Switzerland, 1977.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New-York, 1995.

[Sch98] S. M. Scheiner. The genetics of phenotype plasticity. vii. evolution in a spatially-structured environment. *Journal of Evolutionary Biology*, 11(3):303–320, 1998.

[SP97] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.

[SR97] P. D. Surry and N. J. Radcliffe. The COMOGA method: Constrained optimisation by multiobjective genetic algorithms. *Control and Cybernetics*, 26(3), 1997.

[SS89] W. Siedlecki and J. Sklansky. Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In *International Conference on Genetic Algorithms*, pages 141–149, 1989.

[Ste95] J. Stewart. Cognition = life: Implications for higher-level cognition. *Behavioural Processes*, 35:311–326, December 1995.

[Sut88] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(9–44), 1988.

[SWC+95] N. A. Stillings, S. E. Weisler, C. H. Chase, M. H. Feinstein, J. L. Garfield, and E. L. Rissland. *Cognitive Science: An Introduction.* MIT Press, Cambridge, Massachusetts, second edition, 1995.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[VFM00] V. K. Vassilev, T. C. Fogarty, and J. F. Miller. Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60, 2000.

[WGM73] B. Widrow, N. K. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Trans. Syst. Man Cybern.*, 4:455–465, 1973.

[Wil94] B. A. Williams. Reinforcement and choice. In N.J. Mackintosh, editor, *Animal Learning and Cognition*, pages 81–108. Academic Press Inc., London, 1994.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[WMRD96] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence Journal*, 85, 1996.

[Wri31] S. Wright. Evolution in mendelian populations. *Genetics*, 16:97, 1931.

[Yao99] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[YCC99] A. S. Younger, P. R. Conwell, and N. E. Cotter. Fixed-weight online learning. *IEEE Transactions on Neural Networks*, 10(2):272–283, March 1999.

[YL96] X. Yao and Y. Liu. Fast evolutionary programming. In *Proceeding of the Fifth Annual Conference on Evolutionary Programming*, pages 451–460, Cambridge (MA), 1996. MIT Press.

[YLL99] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, July 1999.

[Zuc97] E. Zuckerkandl. Neutral and nonneutral mutations: the creative mix-evolution of complexity in gene interaction systems. *Journal of Molecular Evolution*, 44(Suppl 1):S2–S8, 1997.

List of Tables

List of Figures

Mathematical Symbols

Symbol	Short Description
n	search space dimension
$\mathbf{x}$	vector of objective variables x
$\mathbf{x}^*$	global minimum point
λ	number of offspring individuals
μ	number of parent individuals
σ	mean step size
τ_o	learning rate, $\varphi^*/\sqrt{n}$
τ	individual step size learning rate, $\varphi^*/\sqrt{2\sqrt{n}}$
τ'	global step size learning rate, $\varphi^*/\sqrt{2n}$
φ_x	rate of progress in terms of distance metric
φ_f	rate of progress in terms of fitness
φ^*	expected rate of convergence
ω	connectionist model's connection strength
ψ	fitness function
ϕ	penalty function
r_g	penalty coefficient
ℓ	length of string
$E[\cdot]$	expectation value
$P\{\cdot\}$	probability of an event
P_f	probability of using objective function in stochastic ranking
$N(0,1)$	normally distributed random variable with 0 mean and variance 1
$U,\ u$	random number uniformly distributed over [0 1]
$G,\ G_m$	total and median number of generations
(g)	generation counter

Glossary

Allele alternative forms of a gene.

Building block a short above-average schemata.

Building block hypothesis states that a genetic algorithm using crossover works well when short, low-order, highly fit schemas recombine to form even more highly fit, higher-order schemas.

Chromosome a chain of genes.

Gene a unit of heredity.

Genome the collection of genes held by the organism.

Locus position on a chromosome occupied by a gene.

Phenotype the manifested attributes of an organism.

Schema a *schema* describes a subset of strings that have similarities at certain positions. Schemata are defined as strings of length ℓ in the extended representation space $I^\star = I \times \star$, where $\star$ denotes a 'wildcard' matching any alphabet in I.

Schema theorem a theorem devised by Holland [Hol75] to give a lower bound for the expected number of instances of a *schema* that are represented in the next generation in a canonical genetic algorithm using one-point crossover, mutation, and proportional selection.