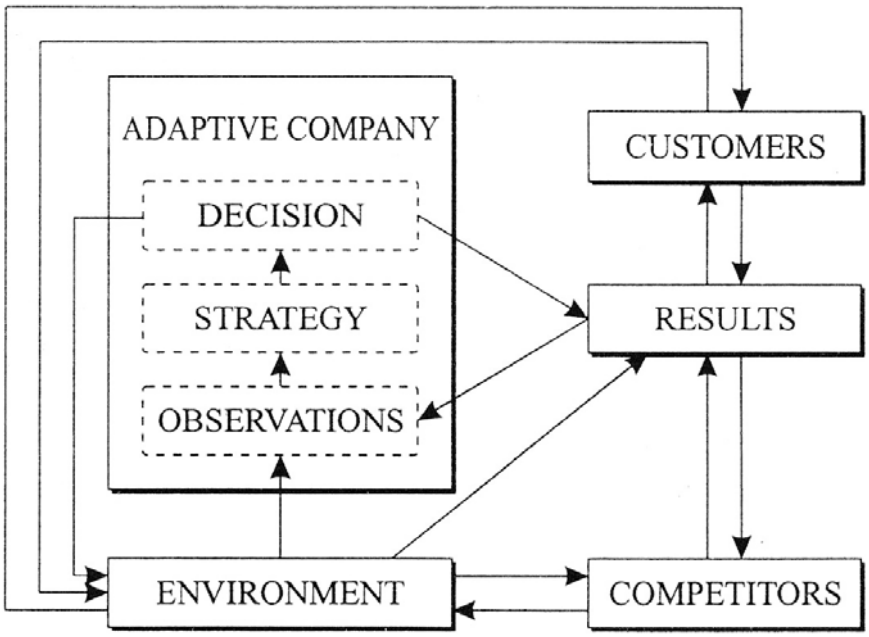
FORTSCHRITTSBERICHT
SIMULATION FBS 10



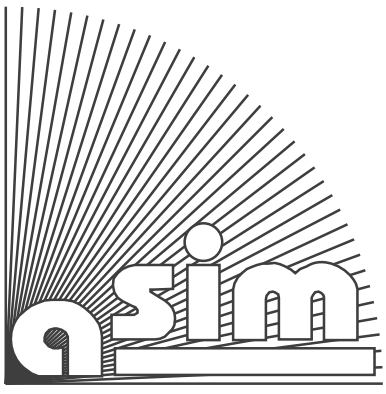
Thomas Fent

Applications of Learning Classifier Systems for Simulating Learning Organizations

ARGESIM

ISBN Ebook 978-3-903347-10-6
ISBN Print 978-3-901608-60-5

DOI: 10.11128/fbs.10


argesim


ARGESIM

Fortschrittsberichte Simulation

FBS Band 10

Herausgegeben von **ASIM**
Arbeitsgemeinschaft **Sim**ulation, Fachausschuß 4.5 der GI

Thomas Fent

Applications of Learning Classifier Systems for Simulating Learning Organizations

ARGESIM / ASIM – Verlag, Wien, 2001
ISBN Print 978-3-901608-60-5

Ebook Reprint 2020
ISBN Ebook 978-3-903347-10-6
DOI: 10.11128/fbs.10

ASIM Fortschrittsberichte Simulation

Herausgegeben von **ASIM**, Arbeitsgemeinschaft Simulation, Fachausschuß 4.5 der **GI** und der **ARGESIM**

Betreuer der Reihe:

Prof. Dr. G. Kampe (ASil\1)
Fachhochschule Esslingen
Flandernstraße 101, D-73732 Esslingen
Tel: +49-711-397-3741, Fax: -3763
Email: **kampe@ti. fht-esslingen. de**

Prof. Dr.-Ing. D.P.F. Möller (ASIM)
Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Str. 30, D-22527 Hamburg
Tel: +49-40-42883-2438
Email: **dietmar.moeller@informatik.uni-hamburg.de**

Prof. Dr. F. Breitenecker (ARGESIM / ASIM) Abt.
Simulationstechnik, Technische Universität Wien Wiedner
Hauptstraße 8 - 10, A - 1040 Wien
Tel: +43-1-58801-10115
Email: **Felix.Breitenecker@tuwien.ac.at**

FBS Band 10

Titel: Applications of Learning Classifier Systems for Simulating Learning Organizations

Autor: Dr. Thomas Fent
Österreichische Akademie der Wissenschaften
Institut für Demographie
Hintere Zollamtsstraße 2b
1033 Wien, AUSTRIA
Email: **Thomas.Fent@oeaw.ac .at**

Begutachter des Bandes:

Prof. Dr. F. Breitenecker TU Wien; Prof. Dr. D.P.F. Möller, Univ. Hamburg

ARGESIM / ASIM – Verlag, Wien, 2001
ISBN Print 978-3-901608-60-5

Ebook Reprint 2020
ISBN Ebook 978-3-903347-10-6

DOI: 10.11128/fbs.10

DISSERTATION

Applications of Learning Classifier Systems for Simulating Learning Organizations

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

o.Univ.Prof. Dr. G. FEICHTINGER
E1192
Institut für Ökonometrie, Operations Research und Systemtheorie

eingereicht an der Technischen Universität Wien
Fakultät für Technische Naturwissenschaften und Informatik

von
Thomas Fent
89255245

Wien, am 25. April 2001 .....

Kurzfassung

In der mathematischen Modellierung ökonomischer Phänomene erfreuen sich analytische Modelle, die das Vorhandensein eines stabilen Gleichgewichtszustandes postulieren großer Beliebtheit. Dabei wird meist unterstellt, dass die den Systemzustand definierenden Größen sich allmählich dem Gleichgewicht annähern, und anschließend in dieser Ruhelage verharren. Weiters wird oft davon ausgegangen, dass sich die handelnden Wirtschaftssubjekte stets vollkommen rational verhalten — d.h. dass sie stets die Entscheidungen treffen, die ihren Nutzen maximieren.

Zur Modellierung von Wirtschaftssystemen, die einem ständigen und sich fortlaufend beschleunigenden Wandel unterworfen sind, sowie zur Beschreibung von Individuen, die nicht nur mit Intelligenz, sondern auch mit Emotionen ausgestattet sind, sind solche Ansätze nur sehr eingeschränkt geeignet. Deshalb macht die Theorie der *komplexen adaptiven Systeme* von Computersimulationen Gebrauch, deren erfolgreiche Implementierung nicht notwendigerweise mit der Existenz von Gleichgewichten bzw. von vollkommen rational handelnden Agenten zusammenhängt.

In der vorliegenden Arbeit werden zunächst die Notwendigkeit und die Zweckmäßigkeit des Einsatzes von *komplexen adaptiven Systemen* zur Beschreibung aktueller wirtschaftlicher Vorgänge erörtert. Anschließend werden Methoden, die zur Implementierung solcher Systeme geeignet sind erläutert. Insbesondere wird dabei ausführlich auf Classifier Systeme und genetische Algorithmen eingegangen. Zur Illustration der Einsatzmöglichkeiten bzw. der Grenzen des Einsatzes dieser Verfahren dienen einige daran anschließende Beispiele. Auf diesen Ausführungen aufbauend werden zwei umfangreiche betriebswirtschaftliche Modelle formuliert.

Das erste Modell behandelt das Problem der Kommunikation innerhalb eines Betriebes, der ein neues Produkt entwickelt. In großen Unternehmen stellt die Gestaltung eines effektiven und effizienten Informationsflusses eine große Herausforderung dar. Weiters treten in verteilten Entscheidungsprozessen mitunter einander widersprechende Ziele auf. In den Prozeß der Gestaltung eines neuen Produktes sind von der Marktforschung bis zur Einlastungsplanung und Instandhaltung unzählige Mitarbeiter mit unterschiedlichen Sichtweisen und oftmals stark voneinander abweichendem Fachvokabular involviert (siehe Abb. 4.1 auf S. 63).

Zur Lösung dieses Problems schlugen (Hauser and Clausing, 1988) ein Kommunikationsschema vor, das sie „*House of Quality*“ nannten. In der vorliegenden Arbeit wird eine darauf aufbauende Simulation, die in MATLAB implementiert wurde, vorgestellt. Die Entscheidungsträger (die adaptiven Agenten) sind als Classifier Systeme modelliert, und versuchen unter Verwendung von genetischen Algorithmen eine sinnvolle Strategie zu lernen. Zur Bewertung der so erzielten Ergebnisse ziehen wir die mittels vollständiger Enumeration erhaltenen optimalen Strategien zum Vergleich heran. Es zeigt sich, daß die genetischen Algorithmen tatsächlich in der Lage sind, brauchba-

re Entscheidungsregeln zu generieren. Die so erzielten Regeln zeigen auf, wie sich die verantwortlichen Individuen in unterschiedlichen Situation verhalten sollten.

Im zweiten Modell wird ein heterogener Markt von substituierbaren Gütern untersucht. Die Aufgabe der lernenden Agenten besteht darin, ihre Produkte so zu plazieren, dass sie von möglichst vielen Kunden gekauft werden. Auf diesem Markt treten insgesamt vier Klassen von untereinander konkurrierenden Agenten auf. Zwei dieser Gruppen beinhalten lernende Agenten. Die erste Gruppe beobachtet die Positionen (= Bedürfnisse) der Kunden direkt. Die zweite Gruppe beobachtet die Positionen der Anbieter, und die von ihnen erziehlten Gewinne. Aufgrund dieser Beobachtungen werden die Entscheidungen für die nächste Verkaufsperiode getroffen. Weiters gibt es eine Gruppe von Anbietern, die ihre Produkte gemäß eines *„random walk“* im Markt positioniert, sowie einen Anbieter, der sich immer auf die Position des in der Vorperiode erfolgreichsten Anbieters setzt.

Zum Vergleich dieser Strategien werden drei unterschiedliche Verhaltensmuster auf Seiten der Kunden betrachtet: i) statisch, ii) zyklisch zwischen zwei Zuständen wechselnd und iii) random walk. Um gezielte Aussagen über den Zusammenhang zwischen dem Kundenverhalten und dem Erfolg der Strategie eines Anbieters zu ermöglichen, weisen innerhalb einer Simulation stets alle Kunden das gleiche Verhaltensmuster auf. Es zeigte sich, dass in den Fällen i) und iii) die Strategie des Imitierens optimal ist — unter der Voraussetzung, dass nur ein einziger Anbieter diese Strategie verfolgt. Verhalten sich die Kunden hingegen gemäß ii), so sind die lernenden Agenten, die die Kunden direkt beobachten, am erfolgreichsten (siehe Abb. 5.5 auf S. 92).

Abstract

In the field of mathematical modelling of economic phenomena analytical models, assuming the existence of a stable equilibrium, are very popular. It is often expected, that the quantities defining the state of the system gradually approach to the equilibrium, and remain there in the following. Moreover, it is often assumed, that the interacting individuals behave perfectly rational — i.e they always take those decisions, that maximize their utility.

Such approaches, however, have only limited suitability for modelling economic systems subject to a permanent and succesively accelerated change. Moreover, they are only partially capable to describe individuals that do not only possess intelligence but also emotions. Therefore, the theory of *complex adaptive systems* applies computer simulations whose implementations does not necessaryly depend on the existence of equilibria or perfectly ratinal agents.

At the beginning of this thesis the necessity and the practicality of the employment of *complex adaptive systems* for describing recent economic happenings is discussed. Then methods that are qualified to implement such systems — in particular classifier systems and genetic algorithms — are being explained. In the following some examples are provided to illustrate possibilities and also restrictions of the usage of such procedures. Based on this elaborations two comprehensive economic models are formulated.

The first model is about the problem of communication within a firm developing a new product. In big enterprises it is often a big challenge to establish an effective and efficient flow of information. Moreover, in distributed decision making processes conflicting objectives may occur. Many different groups of employees are cooperating in the process of designing a new product. The tasks reqired to successfully introduce a new product involve employees from market research, engineering, scheduling, maintenance, and many others (see fig. 4.1 on p. 63). These people may possess different viewpoints and different technical languages.

To solve this problem (Hauser and Clausing, 1988) suggested a communication scheme called *"House of Quality"*. This thesis introduces a simulation based on the *"House of Quality"*, which was implemented in MATLAB. The decision makers are implemented as classifier systems, and apply genetic algorithms to learn a meaningful solution. To evaluate the rules generated by this adaptive learning process, the obtained results are compared with the results gained by full enumeration. It turns out that the genetic algorithms indeed create pretty good decision rules. These rules illustrate how the responsible individuals should react to the encountered situations.

The second model examines a heterogenous market of goods that can be substituted for each other. The task of the learning agents is to place there products in the market such that the number of customers who buy their products is maximized. Four classes of different agents occur in this market. Two of these groups contain learning agents. The first group observes the

positions (= the needs) of the customers directly. The second group observes the positions of the suppliers and their profits. The decisions for the next planning period are based on these observations. Additionally, there is a group of suppliers placing their products according to a *"random walk"*, and another supplier who always takes over the position of the most successful seller of the previous period.

To compare these strategies three different behaviour patterns of the demand side are taken into consideration: i) static, ii) cyclic, and iii) random walk. To allow accurate conclusions about the relation between customer behaviour and the success of a certain selling strategy, all the customers exhibit the same behaviour pattern within one particular simulation. It turned out, that in the cases i) and iii) the strategy of imitating the most successful seller is optimal — under the assumption, that only one supplier follows that strategy. If the customers behave according to ii), on the other hand, the agents observing the customers directly are the winners (see fig. 5.5 on p. 92).

Software tools

The simulations described in this thesis and most of the graphics were done with the help of Matlab (5.2, 5.3, and 6.0). The typesetting and some of the graphics were done with LaTeX2e.

Acknowledgements

Many people have contributed to the present work. First of all, I would like to thank my parents who enabled me to succeed in my study. I would also like to thank Prof. Feichtinger and Prof. Breitenecker who supervised me in analysing the models. I am greatly indepted to (in alphabetical order) Christian Almeder, Maria Dworak, Josef L. Haunschmied, Markus Klug, and Siegfried Wassertheurer who helped me solving the problems I sometimes had with computers. Finally, I would like to thank Prof. Herbert Dawid, Prof. Georg Dorffner, Prof. Andreas Geyer-Schulz, Prof. Richard F. Hartl, Prof. Werner Jammernegg, Prof. Daniel Polani, Michael Schedl, Prof. Alfred Taudes, and Prof. R. Vetschera for their many helpful hints.

This piece of research was supported by the Austrian Science Foundation (FWF) under grant SFB # 10 ('Adaptive Information Systems and Modelling in Economics and Management Science').

Thank you very much

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many methods used in today's management science are based on micro-economic theory. Therefore, neoclassic approaches assuming that there exists a stable equilibrium play a major role. Such approaches typically assume

1. perfect information about the analysed problem and its structure,

2. diminishing returns, and

3. only perfectly rational individuals.

However, in reality there are many situations in which these assumptions are not fulfilled. Individuals lack complete information and different participants interpret the same information in a different way. Moreover, individuals do not always behave rational, both due to lacking abilities and due to lacking motivations. This results in an increasing popularity of models taking into account the effects of so-called *bounded rationality*. The second assumption fails in case of network externalities.

Some of these models are very elegant from a mathematical point of view. They may be appropriate for describing an agricultural or manufacturing economy, but they are not at all suitable for a market determined by innovation, change, and uncertainty. A typical neoclassical model is based on simple assumptions about the individuals' behaviour. The benefits obtained by such models are the mathematical proofs confirming the results. The disadvantage, however, is the restriction of the behaviour of the interacting agents to the simple assumptions. If the assumptions exclude important aspects of the real world, then the answers delivered by the model are irrelevant. In order to (at least partially) overcome those shortcomings we will make use of computer simulations based on assumptions that are too complex to be included into a neoclassical analytical model. This gives us the chance to consider aspects of learning, adaption, uncertainty and bounded rationality.

The remainder of this chapter is organized as follows. In section 1.1 some thoughts about the shortcomings of neoclassical equilibrium-based economic models are elaborated. In section 1.2 complex adaptive systems (CASs) are

introduced as a tool to model dynamic markets determined by learning individuals in a more realistic way. Section 1.3 describes the second industrial revolution which started in the nineties of the twentieth century. In section 1.4 some thoughts about the problems accompanying the increasing bulk of information are formulated. Finally, in section 1.5 some arguments emphasizing the increasing importance of knowledge management in modern enterprises are collected.

In chapter 2 classifier systems (CSs) and genetic algorithms (GAs) are discussed in detail as two possible tools to simulate CASs. Some examples to illustrate the capabilities and limitations of these tools are given in chapter 3. In chapter 4 we build and analyse a model to simulate the process of developing new products in a big company consisting of departments with partially conflicting interests. In chapter 5 we formulate a model of product placement in a market with several competing vendors supplying highly replaceable products. The needs of the consumers cannot be observed directly but only the sales of all the suppliers including the own sales. Therefore, the process of adaption has to be capable of detecting indirect connections. Finally, in chapter 6 we summarize the results obtained by our simulations, draw some conclusions, and outline a couple of promising possible extensions of the present work.

1.1 Shortcomings of equilibrium-based economic models

The world in which we live is not static, nor does it converge to a stable-state equilibrium at all. If it were, it would be almost impossible for a new entrepreneur to succeed in a market segment that is already covered by big suppliers with decades of experience. Innovation and growth cannot be explained as internal effects of an equilibrium-based model but just as a result of random exogeneous shocks (see Beinhocker, 1997).

Bloomberg News for instance showed that it is possible for a newcomer to outperform established competitors within a few years. What they did, was reinventing existing services and providing additional features which were experienced as additional values by the customers. Such changes which take place in many lines of todays business usually do not occur within a mathematical framework based on equilibrium theory. Neither can such a framework describe the appearance of deregulation, profound technological change, industry convergence, globalisation, and increasing returns. The latter phenomena for instance occur in internet business or telecommunication services which ensures that even the second of the above assumptions (p. 1) must be challenged.

To choose a certain strategy of any kind, the situation has to be analysed first. Based on some observations one might choose an appropriate strategy and, finally, a decision (an action) is derived as an outcome of the chosen

strategy. Managers must take into account such conflicting objectives as their companies competitive situation, the needs of the customers, capital markets, the regulatory environment, new technology, the structure of its industry, and the strengths and weaknesses of itself as well as of its competitors. Having a dominant strategy, however, is not enough. Another important aspect is the ability to implement it. Hence, in the simulation in chapter 4 a weighted sum of implementation costs and opportunity costs will be considered.

The long-run success of a company highly depends on the alignment between the abilities and the requirements for executing the strategies. Aspesi and Vardhan (1999) distinguish two major kinds of strategies: transformational and operational. Transformational strategies are applied in an environment determined by significant uncertainty. Such strategies often attempt to change the game in the industry, and address substantial customer, channel, or competitive challenges. When the level of uncertainty is low, and the company just tries to play the same old game better than the competition, then the strategy is called operational. There is no global rule determining or distinguishing which of the two kinds of strategies is better in general. Neither is it possible to say which of the two is the easier way. It always depends on the particular situation and the internal abilities.

In many business fields changes of paradigms take place indicating the transition from the old–world of business to the e-world of business. The success of some of the so–called dot–coms is often due to their understanding that although offering recent technology is important, business model innovation can be the most important property determining global market shares. Such business model innovations often include changes of paradigms that do not only require transformations at the level of business processes and process workflows, but also fundamental alterations of the general business model and the information flows between organizations and industries. Many traditional companies that try to enhance their distribution channels by entering into e-business encounter serious challenges in integrating their physical and virtual value– and supply–chains.

Two examples are given in (Aspesi and Vardhan, 1999). The first addresses the situation of gas and electric power industries. In the United States — similarly to Europe — originally protected markets have become open to competition. The customers have the choice among several suppliers and, on the other hand, the suppliers are no more obliged to serve the customers with a fully vertically integrated chain from production to supply. In California these deregulations resulted in substantial instabilities in the supply of electricity. Nevertheless, this chapter addresses the challenges and opportunities of a deregulation from the viewpoint of a supplier. Two types of reactions are possible: transformational and operational.

Implementing an operational strategy could mean, for instance, to invest in technological research to increase the efficiency of power plants. Moreover, many processes required for running and maintaining a power plant can be improved to reduce costs and increase productivity. This could strengthen

the competitive position and help to survive in the opened market. Choosing a transformational strategy, on the other hand, could mean to improve the position in the retail business. A provider originally limited to its local market can think about selling energy to new groups of customers and, thus, take profit from the deregulation of the formerly regulated market.

The second example deals with the changes in the US pharmaceutics industry, undertaken in the 1990s. Originally, patent protection excluded price competition, and health insurances avoided the development of customers' price sensitivity. Physicians, consuming the drug companies sales provisions, took the decisions, and the consumers had no incentive to call that in question. The appearance of managed health care suddenly jeopardized these habits. Doctors were encouraged to use low-cost alternatives instead of the well-known but costly brands. As a consequence, profits and share prices of drug companies decreased remarkably.

Similarily to the engergy business, again, an operational strategy may be based on improving product research and sales. Reducing the time from conception to clinical tests can help to defend market shares against alternative manufacturers. Hiring and educating a strong sales force will also strengthen the competitiveness. A possible transformational strategy, on the other hand, is to integrate vertically by purchasing a pharmacy benefits manager. In reality, this was achieved by installing two committees for governing a pharmaceutical company. The first one was responsible for operations and the second for strategic decisions. One major task of the strategic division was to evaluate the business portfolio and divest the majority of the noncore business units. This, in turn, generated cash which was invested in the core business.

According to Aspesi and Vardhan (1999) all of the four approaches mentioned in the previous paragraphs were successful in a real-world scenario. Therefore, even when looking at one particular industry it is impossible to determine, whether it is better to follow an operational or a transformational approach. An organization can gain great power after recognizing, what it does well and focusing on it. A strategy cannot be recommended, nor can it be rejected without looking at the skills of the company that has to implement it. Both, operational and transformational strategies can result in an outstanding performance in the same industry and at the same time.

Whenever an individual takes a decision based on information collected one time step ago, the decision can only be optimal with respect to the environment as it was one time step before. If, in the sequel, the individual sticks to the strategy that was good with respect to the formerly observed situation, the difference between the assumptions and reality becomes bigger and bigger.

Thus, real world individuals usually do not stick to one strategy for a long time. If they do, they will not survive within a changing environment. Therefore, in the following we will elaborate on models that are capable of describing environmental changes and learning individuals being confronted

with limited information.

1.2 Complex adaptive systems

A dynamic market makes it difficult for the participants to maintain their competitiveness and survive in the long-run. Companies with a lot of experience in their main business units might feel motivated to rely on their strategies that have proven to be successful in the past. To survive dramatic shifts of customers' tastes or sudden technological changes, a company must be as good or even better at evolving as the environment. If the company is not capable of performing the required level of evolution, then the market will do it by the entry and exit of firms. Therefore, companies that want to remain successful within a dynamic environment have to keep on observing, analysing, learning, and adopting continously. This process is illustrated in figure 1.1.

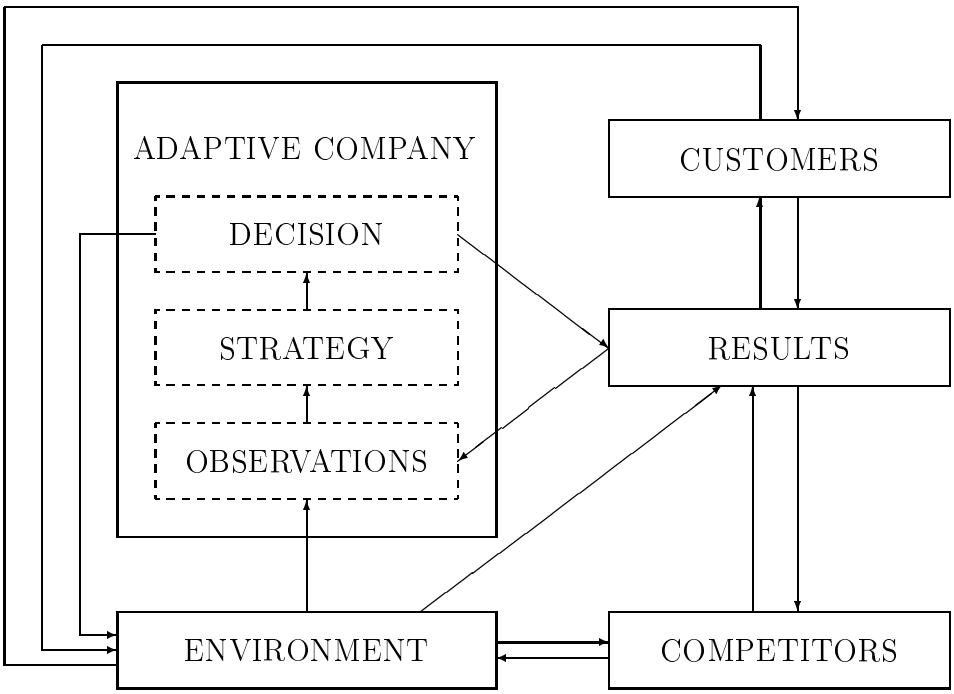


Figure 1.1: The process of adaption

Moreover, they must be prepared to face a big variety of situations and adopt their behaviour even when the system is in a state that has never been experienced before. Companies' competitive survival and long lasting sustenance highly depend on their ability to continuously redefine and adapt organizational goals, purposes, and the organization's "way of doing things". Therefore, it has been suggested that the role of senior management needs to change from command and control to sense and response. The recent dynamics of economic environments force market participants to put less

emphasis on playing by predefined rules and more on understanding and adapting as the rules of the game — as well as the game itself — keep changing. Hence, a big bundle of optional strategies might be a valuable tool in a modern economy. A market built up by many individuals of that kind can be seen as a complex adaptive system (CAS). According to (Beinhocker, 1997) the main characteristics of CASs are:

open and dynamic: Only a closed system without external (exogeneous) influences can tend to a stable-state equilibrium and persist there for a lengthy time interval. In a CAS the individuals are always aware of unpredictable changes and quickly adapt their behaviour whenever such shifts are encountered. In figure 1.1 external shocks (external from the viewpoint of the considered individual) can originate from the customers and from the competitors as well. The customers might change their taste (e.g. yesterdays trendy items might run out of fashion today) and their buying habits (people who went to the retailer on a regular basis might get used to order products via internet). The competitors, on the other hand, can perhaps launch a completely new technology and, therefore, gain a sudden competitive advantage. The actions of the illustrated adaptive company, on the other hand, are considered as exogeneous shocks by the other agents.

interacting agents: Decisions taken by one agent have an impact on the environment of all the agents and, in turn, they all have to adjust their strategy to the new situation. Thus, when one agent, due to evolution, changes her/his main strategy, this does not only effect the environment but also the evolution of the other agents. This causes the inherent dynamics of a CAS which makes it so unlikely to arrive to a steady-state. Therefore, in figure 1.1 there is an arrow pointing from the decision block to the environment block. This indicates that the agents decision does not only influence the outcome in the near future (direct effects) but can also cause (or at least initiate) significant and long lasting changes of the whole system (indirect effects). A characteristic example are the huge changes of stock prices. In a typical classical model they can only be explained by external perturbations, while in fact they are just a result of the investors' manners of trade or expectations, respectively.

emergence and self-organisation: There is no central planner deciding what and when things happen, but there are many intelligent individuals taking their own decisions. Only the whole bundle of the participating individuals' actions and reactions can determine the behaviour of the system. As a result it is very difficult to predict the dynamics of a CAS.

Practical examples of CASs are cities (not located in a dictatorial country), ecosystems, the internet, economies, and firms with a fractal structure (see excursion 1).

Excursion 1 (Fractal firm)

The reorganization of production processes by introducing independently operating but connected working units can be seen as a central task of management in the post Taylor age. A mathematical view – in particular a fractal geometrical view – can help to explain these new production structures.

The organization of working processes introduced by Taylor is based on dividing big and complex processes in many small and simple steps. This enables unskilled workers to learn their part of the big puzzle and do the same task many times a day. As a consequence, efficiency increases and mass production of standardized goods becomes possible. Taylorism is characterized by multi-layer hierarchies, low level of education and responsibility, growing bureaucracy, low flexibility, and fixed working times. In such a structure, research and development has to be completely separated from production. Moreover, high levels of inventory emerge due to the many buffers required between the steps of the production.

Improved skills of contemporary employees, general movements towards emancipation, higher complexity of products, and the increased demand for customized products force companies to shift to an organization that supports the creativity of the workers. Thus, the working process becomes more customer oriented. Now, technological changes are no more introduced by a big bang in the whole company at once, but in small steps at the group level. Each working group operates like a company itself, taking the responsibility from customers demands until delivering the finished products. Optimization and improvement is no more a task of top level management, but of the groups themselves.

This kind of organization led to the keyword "fractal firm", because a small part of the company has the full functionality of the whole company (see figure 1.2). This is what such a fractal firm has in common with the objects known from fractal geometry.

The business is seen as a learning organism and must organize itself to survive in the market competition. Consequently, each co-worker is service oriented rather than looking only at his/her own task. In contrast to Taylorism, higher levels of order are built up from the lower ones. This is only possible, because the level of education increased rapidly since Henry Ford installed the first conveyor belts.

Another reason, why microeconomics cannot always describe an economy, is the shortcoming of human reasoning. In many situations humans are just overtaxed, when they have to analyse all the information they have, transform it to knowledge, think of all the connections that might exist, and finally, take the right decision taking into consideration several conflicting objectives. Therefore, most individuals use their experience. They compare the current situation with things that happened in the past, and guess which of their rules created previously fits best to the new problem. This even qualifies such models to explain, why two individuals facing the same pattern of recent

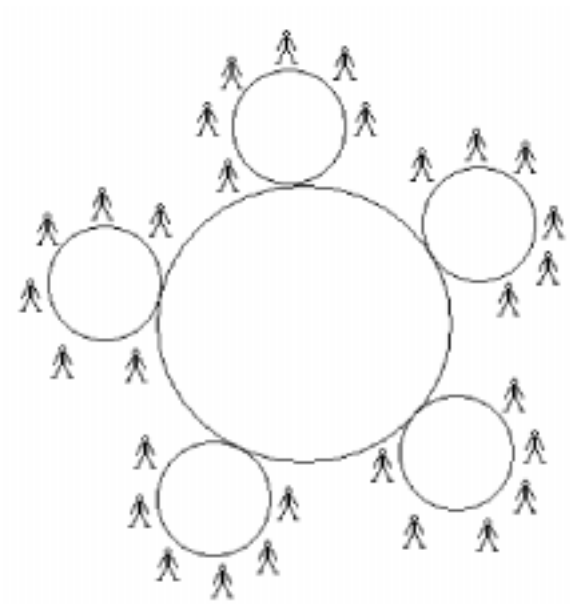


Figure 1.2: The structure of a fractal firm

information behave differently due to their different experience. In a classical microeconomic model such effects are very difficult to give reason for.

Human beings often use induction instead of deduction, and that's exactly, what agents in CASs are doing. Both, the real-world individuals and the agents in CASs, try to recognize patterns and develop and apply inductive rules of thumb. This even works in case of incomplete or changing information. Most of the time not even all the available information is actually taken into consideration. Real-world individuals have different reasons to change their behaviour, such as mistakes, curiosity, or external perturbations. Typical artificial agents use variation, elimination, and imitation to update their rule-base.

Two examples of methods which can be used to simulate such an adaptive company are *classifier systems* (CS) and *genetic algorithms* (GA), which will be described in detail in chapter 2. CSs have the ability to pursue several paths simultaneously, which is a prerequisite to prosper in a CAS. On the other hand, GAs enable our artificial agents to discard unsuccessful rules, recombine the successful ones, and make some random changes to create rules that might be helpful in completely new situations. Brenner (1996) claimed that evolutionary algorithms cannot describe the change of an individual's behaviour due to dissatisfaction about the achieved results. Nevertheless, in this study it will be shown that a synthesis of CSs and GAs can do that.

1.3 The second industrial revolution

While the so-called *lean-management* philosophy, which was very popular until the early nineties, was highly focused on industrial enterprises, we are now in the era of a new industrial revolution. Around 1900 most people worked in agriculture and only a small share of an economy's workforce was allocated to the information sector. Nowadays these relations have changed dramatically. Only a small share of the population is working in agriculture while employment in the information sector has boomed amazingly. Moreover, even countries that gained their wealth with petrol have started to invest in information technology to be prepared for the demands of the future. The first industrial revolution provided considerable wealth to the vast majority. The result of the current industrial revolution, on the other hand, is the expanded access to accurate and comprehensive information and education.

Today, capital is no more a limited resource — at least it is no more the most important one. *In the new world of e-business, the scarce resource is not information, but human attention* (see Malhotra, 2000, p.11). Knowledge is the primary resource for individuals and for the economy in general. Land, labour and capital — the economist's traditional factors of production — do not disappear, but they become secondary. There are several examples of small Internet companies whose level of stock–value at least temporarily exceeds or exceeded the stock–value of huge traditional companies with an evident physical value much higher than the upcoming dot–coms. Obviuosly such strange effects depend on the availability of risk capital. In the third world or developed regions exposed to underdeveloped stock and capital markets this is very unlikely to happen.

Such startups are often founded by university dropouts. Their competitive advantages are enthusiasm and creativity. On the other hand, when a few employees bearing the key information responsible for a company's success leave that company it may lead to a sudden crash of the stock–value. Banks do a lot of advertisement to offer loans to the customers. Never before wealth was accessible to so many people — partially due to inheritance. Both observations indicate that there is more than enough capital available.

While capital, on the one hand, is indivisible (i.e. the capital of one individual cannot be shared among several persons without reducing the individuals amount) knowledge, on the other hand, is divisible. When the employes exchange their knowledge it usually does not mean that the knowledge of the best informed individual is smaller after the exchange. Nevertheless, in some situations the value of knowledge might decrease when it is shared with others. The exploration of the traditional factors of production is limited since any increase in land, labour, and capital leads to diminishing returns on the production outcomes. Information assets, on the other hand, seem to obey a completely different law of economic returns. Investment in an additional unit of information or knowledge created and used results in a higher return (in contradiction to the second assumption on page 1). This is often attributed to

externalities. The more individuals become members of the network and use its services, the greater will be the value added to the network. Knowledge is the only productive factor that cannot be transfered directly. It can only be perceived indirectly, it has the inert ability to be multiplied, and finally, utilization does not reduce the supply of knowledge.

Many countries move from an industrial society to an information society. Some typical paradigms characterizing the difference between the logic of an industrial society and the logic of an information society are listed in table 1.1.

	industrial society
–	the firm is a machine (Input → output)
–	making money is the highest target
–	leadership is mostly accomplished by central planning, delegation, motivation, and control
–	control is based on power and money
–	strategy is the exclusive task of top management
–	the value of a firm is determined by its book value
–	employees are basically paid for the time they spend for the company
–	time is the limited resource
	information society
–	the firm is a human(e) community
–	firms follow a commonly shared vision (economic performance is a consequence)
–	leadership is accomplished by common visions and definition of boundaries
–	control is based on knowledge
–	strategy is everybody's task
–	the value of a firm is determined by the market
–	employees are basically payed for their performance
–	attention is the limited resource

Table 1.1: Industrial society vs. information society

1.4 The homo informaticus

Information from everywhere in the world provided by newspapers, television, or internet penetrate our habitat day by day. The new media did not only create a new infrastructure facilitating the spread of news and facts but also initiated the occurence of bulks of news. The evil about this flood is the impossibility of escaping from it. Recent information has become a vitally necessary constituent of our everyday life, no matter whether it is just a short message, explosive news, or a commodity. Classical (vertical) means

of control like power, money or the law have been replaced. An intact flow of information as a new (horizontal) control medium determines wealth and competitiveness in the modern information society. Decentralisation and pluralism may help to avoid the emergence of totalitarian monopolies. Political hierarchies become obsolete in the internet. Some important aspects of the information superhighway with respect to socio-political consequences are concentration of power, information priority, selection processes undertaken by the users, and self regulation of markets.

As a result of progressive globalisation and networking of the mass media it is becoming more difficult for dictators to refuse their citizens access to international and independent information channels. Reversely, the responsibility of the companys for their employees is increasing due to the faster expiration of valuable knowledge.

Information may be transformed to knowledge and power. It takes over the role of a raw material and a productive factor at the same time. From an economic point of view, information has to be seen as commodity but from a sociological viewpoint, on the other hand, it is seen as the feeding of knowledge. Thus, in the latter case information is a process serving the removal of ignorance. In the context of business additional information transformed into knowledge leadership means a surplus value.

Additional information can also lead to confusion. The information society is not represented as a homogenously well-informed society. The bulk of information subject to rapid expiration creates confusion and becomes the dilemma of the global citizen. The fast spread of Internet technology does not only enlarge the individual's chances, but also result in irritation and desorientation. The post-industrial age is characterized by a new type of human being. The change from the industrial society to the information society replaces the "homo oeconomicus" by the "homo informaticus". Optimal exploitation of information becomes a key issue.

The ability of differentiation defines new social splitting lines. Orientation and purposeful utilization of the huge stream of available information requires an intact selection mechanism. The information chaos surfeited with redundant facts may degenerate the global village into a global jungle. Those who cannot participate in the global data exchange and those who cannot differentiate are in danger of social isolation. Commercialization is a solution to this problem and a part of this problem at the same time.

On one hand increased commercialisation will improve the services and methods of information brokers offering individually customized information services to those who can afford it. This already indicates that only the upper class will be able to take profit from advanced information technologies. The social loosers will be forced to search on the data garbage dump for facts that have been disposed by the winners — similar to the cycle of physical goods which go first to the rich and then to the poor.

On the other hand a data supply that has to pass a tailored but commercial check only supports information with a commercial value. Organizations

offering non-profit information may be the loosers of that kind of developments. A big danger occurs when some groups of information brokers form a monopoly. This would give them an amazing amount of power since control of information will provide more influence in the near future than control of physical goods or political power. To guarantee a free flow of information an information structure allowing the customers to perceive the difference between infotainment and infotisement (information and advertisement) is required.

1.5 Knowledge management

It is a well-known phenomenon, that pupils are more successful in gaining knowledge when they are taught in groups than when they have to learn on their own. They motivate each other and, moreover, the observation that colleagues are able to understand the contents of the lectures encourages to succeed as well. Thus, in some situations, groups can learn more than the single members of the group could do. Moreover, big organizations even have the potential to learn more than the groups inside the organizations could do. Therefore, keywords like knowledge management (KM), organizational learning, information economy, and so on are very common in modern management literature. Why is it possible to understand these keywords as indicators of a new management philosophy. Didn't organizations always learn?

What makes a company successful or unsuccessful is no more a matter of capital but a matter of knowledge and knowledge management. The important difference is the ability to utilize any kind of knowledge, ranging from scientific or technical knowledge to social, economic, and management knowledge. Because of the accelerated expiration of knowledge successful application of knowlege has become more desireable rather than just accumulation. The focus has moved from storing facts to storing information about processes. A learning organization must exhibit the capacity to design its own future.

When a firm differs significantly from its competitors, when it is able to create a product or a service that generates value inside the market, then it is only due to its knowledge. Information is a raw material and a productive factor at the same time. *Information is any difference that will make a difference at a later event* (Bateson and Bateson, 1972). Examples of companies succeeding in using their employees' capabilites to create customer value in an efficient way are for instance Gilette, SAP, and Li Kashing. Their and other companies strategies are described in detail in (Baghai et al., 1999).

Peter Drucker's (see Drucker (1970)) definition of the knowledge worker points out that her loyalty is primarily to herselfe and to her career development, rather than to her current host organisation. Encouraging and maintaining loyalty inside ones workforce is therefore a striking challenge for knowledge-based companies. Stock options as a part of the employment

package are widely used as loyalty-bonds. Another method are profit sharing systems that generate a payoff consumable with a reasonable time lag. Both are preferably used in knowledge-intensive branches like for instance the software industry. Corporate sponsorships for further qualifications and learning, sabbatical opportunities, and even corporate universities, are on the rise.

What is knowledge management? It can be seen as some combination of technology supporting a strategy for sharing and using both the brainpower resident within an organization's employees and internal and external information found in "information containers" (primarily documents). Malhotra (2000, p.11) gives the following definition:

> *Knowledge management caters to the critical issues of organizational adaption, survival, and competence in face of increasingly discontinuous environmental change. Essentially, it embodies organizational processes that seek synergistic combination of data and information–processing capacity of information technologies, and the creative and innovative capacity of human beings.*

The goal of knowledge management is to simultaneously manage data, information and explicit knowledge while leveraging the information hidden in the people's heads (tacit knowledge) through a combination of technology and management practices. Before discussing knowledge management, the question must be addressed, how knowledge differs from information.

In short, there are three levels of "information" — *data, information, and knowledge* — with a great deal of overlap between them. Nevertheless, within an environment determined by an overwhelming flood of information it is becoming extremely important for the success of nearly any organization to understand the strategic distinction between knowledge and information. One way to think of the difference is that knowledge is a finished product and information and data are raw materials. I.e. knowledge may also be interpreted as the *potential for action.* It can be distinguished from information in terms of its direct connection with performance. Further, there are two categories of knowledge: tacit and explicit. Tacit knowledge is what is stored somewhere in our heads. It's a combination of context, information, experience, intuition, talent, and so on (see table 1.2). Tacit knowledge often can only be shared with others by means of demonstration. For instance, one may think about attempting to learn skiing by reading a book about it.

That makes tacit knowledge extremely hard to transfer. Accessing tacit knowledge is the goal of knowledge management. Explicit knowledge is an attempt to place our tacit knowledge in some sort of container, most commonly a document. Knowledge in this sense, as a body of information (chemistry, biology, etc.) is currently managed. In many firms most of the valuable knowledge is implicit. Every task in the business process that is performed by routine without further analysis is part of a company's tacit knowledge.

Yet, the implementation of knowledge management tools and processes is still at the beginning. Many enterprises' knowledge management is based on

implicit knowledge	**explicit knowledge**
bound to a person	not bound to a person
complex	compressed
integrated	detailed
experience	facts
language-independent	language-dependent
practice	theory

Table 1.2: Implicit vs. explicit knowledge

models that are well-suited to satisfy the demands of the industrial epoch. Often it is expected that storing all kind of data in a central repository would somehow ensure that everyone equiped with access to that data–warehouse is capable and willing to explore the information stored therein. In reality, despite the availability of comprehensive reports and data–bases, many executives base their decisions on interactions with others who they consider to be knowledgeable about the issues in question. Furthermore, the assumption of singular meaning of information, though desireable for seeking efficiencies, excludes creative abrasion and conflict that is neccessary for comprehensive business reengineering.

Instead of benefiting these organizations, such antiquated models weaken their information strategies. Even companies and organizations aligned on professional knowledge–based services like consultants, newspapers, publishers of specialized journals, libraries, and so on hardly differ from the companies in the seventeenth century concerning knowledge management. The arrangement of lists, card index boxes and data–bases is designed to avoid access rather than encouraging it.

According to a comprehensive questionary (Fraunhofer Institut für Arbeitswirtschaft und Organistion, Deutsche Bank AG, 1999), 66% of the companies do not have the feeling to have a comprehensive overview about the knowledge available inside the company, 80% assume that a medium or even high utility gets lost because of not exploited knowledge, and even 80% feel a demand for an improvement of knowledge management. Conversely, about 80% of the initiatives to facilitate knowledge management are cancelled unsuccessfully within less than 24 months.

Implementing an enterprise knowledge management system is such a lengthy, expensive — and contentious — process that initiatives often run out of time, money or political support before they can contribute real value. Meanwhile, if knowledge workers believe that the chores of contributing to the knowledge management program benefit only their bosses, they may decide the best way to take advantage of the value of their individual knowledge is to change jobs or go independent. It seems obvious, but it is not often said that knowledge management works best when knowledge workers take the initiative and responsibility for what they know, don't know and need to know.

Doing so not only makes the individual more valuable to the corporation, it also enhances the value of intellectual capital for the corporation.

In attempting to apply their collective expertise for competitive advantage, corporations often overlook the fundamental truth that knowledge begins and ends as personal. Nearly all the knowledge is stored in the peoples heads. Without human understanding, personal context and immediate utility, all we have is data. This is also consistent with the observations of Churchman (1971):

> *knowledge resides in the user and not in the collection of information ... it is how the user reacts to a collection of information that matters.*

Moreover, Nonaka and Takeuchi (1995) state that only human beings can take the central role in knowledge creation. Despite the fact that automatically generated data may not be the right medium to store human interpretation for potential action, knowledge always depends on the user's subjective context of action based on that information.

During the last decades of the twentieth century important changes concerning data–processing took place: modern information technology now offers many opportunities to create competitive advantage; increased demand and supply in the field of IT–outsourcing, considering information as a utility, comparable to electric power or telphone services; and the e–everything phenomenon allowing internet and e–commerce emerge as key factors of the business– and IT–strategy. Moreover, a transition from an era of competitive advantage based on information to one based on knowledge creation is taking place. The former was characterized by modest and predictable changes that most information systems were capable to process. Regarding to (Malhotra, 2000) three phases in the evolution of the information–processing paradigm can be distinguished (see also figure 1.3):

Automation increased efficiency of operations

Rationalization of procedures streamlining of procedures and eliminating obvious bottlenecks that are revealed by automation for enhanced efficiency of operations

Reengineering radical redesign of business processes that depends on information technology–intensive radical redesign of workflows and work processes

The development of information technology during those phases was facilitated by a rather predictable view of products and services. Despite the risk and returns due to the increased investments in IT there were only little efforts concerning business model innovations.

Successful knowledge management does not just mean to

- provide laptop computers,

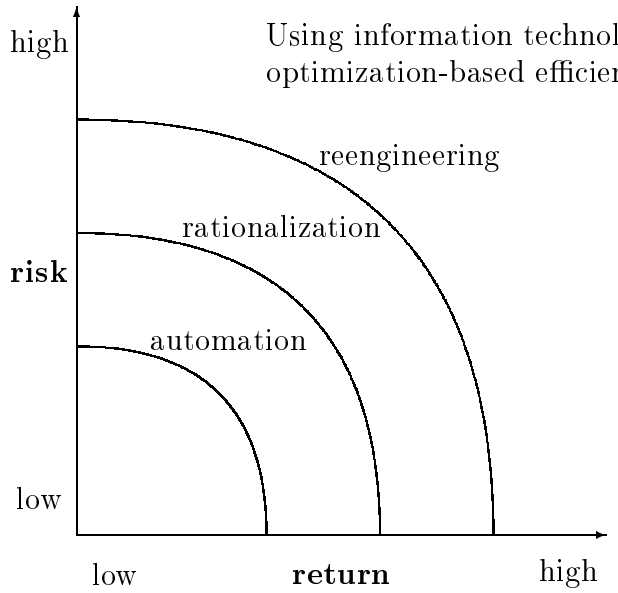


Figure 1.3: Risk and return in the "old world of business"

- send the employees a few days longer to advanced training seminars,
- allocate a little bit more money to the R&D department,
- rename the IT-department to KM-department, and finally,
- use any opportunity to criticize the lack of adaptability within the companys workforce.

But it does mean to

- develop a common understanding of the business,
- determine a knowledge strategy and define knowledge targets,
- communicate the knowledge strategy such that each coworker can recognize her contribution to reach the strategy,
- make knowledge assessable and remunerate the staff according to performance, and
- provide resources and the infrastructure to encourage organizational learning.

The continued challenge for modern knowledge management–systems is to ensure the adaptability and flexibility of information interfaces and information flows — both internally and externally — required for coping with dynamically changing business and competitive environments. The simulations discussed in chapters 4 and 5 elaborate on the obstacles arising when a complex organizational structure is required to act within a dynamically

changing environment. Moreover, they deal with the question of how to use incomplete information within an organization and also address the problem of transfering knowledge within an organization. In particular the incentive scheme to encourage staff to submit information to other departments and also consider information received from the others is given attention. Similar models can be found in (Aoki, 1996), (Carley and Svoboda, 1996), (Carley and Lee, 1998), (Kollman et al., 2000), (Marengo, 1992), (Page and Ryall, 1998), and (Paul et al., 1997).

Chapter 2

Genetics-based machine learning

So far, we have elaborated why methods that are capable to simulate complex adaptive systems are useful in many situations to analyse economic phenomena. In this chapter some of these methods will be described in detail. In section 2.1 we will introduce rule-based decision mechanisms called classifier systems. The following section 2.2 deals with genetic algorithms that, for instance, can be used to update such rule-bases. Concluding, in section 2.3 some simple examples are worked out to give a first view about the possibilites given by the described algorithms.

2.1 Classifier systems

To model a connection between input and output signals consisting of vectors of integer (or binary) entries we use classifier systems (CS). CS were first introduced by (Holland, 1976) as a tool for pattern recognition. They can be seen as a vehicle to use GAs in studies of machine learning (Holland, 1995).

2.1.1 The rule base

The main part of a CS is the rule base consisting of the condition part and the action part (see Fig. 2.1). The conditions within one particular row plus the action in the same line represent a rule, which can also be called a classifier. The conditions may contain integer (binary) entries plus the so called **don't care** symbols #. Thus, in the most simple implementation only the three symbols 0, 1, and # are permitted. On the other hand, the entries of the message list and the action part are restricted to integer (binary) values. The message list represents the information the individual receives, already encoded in a way appropriate for further computation. First, the information in the message list has to be compared with the conditions. Whenever there exists a message that is equal to a condition, except those bits where a # occurs, then the condition is considered to be fulfilled. A whole rule is fulfilled,

when all its conditions are fulfilled. Thus in the example in figure 2.1 the first, the third, and the fourth rule are fulfilled.

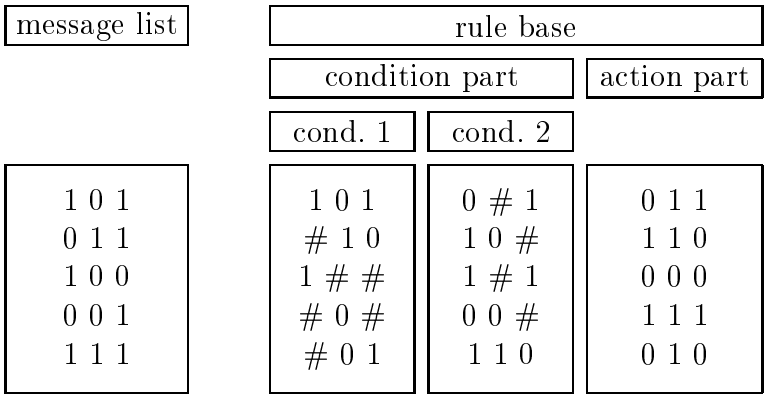


Figure 2.1: The message list and rule base of a classifier system

2.1.2 Choosing an action

Now all the fulfilled rules become candidates to post an action. If, like in the previous example, more than one rule is fulfilled, one of them has to be selected randomly. Usually the strength of the rule, which depends on the success of this rule in the past, and the specifity are used to weigth the rules. The specifity is a measure of the frequency of the don't care symbols. A very general rule, i.e. a rule containing many #, certainly has a higher chance to get selected, because it will be fulfilled more often. To compensate this, it is neccesary to favour the more specific rules. Moreover, we can assume that a more specific rule might yield a better solution to a particular situation, another reason to favour those rules containing only a few #.

The chosen action may be posted directly to the environment, or it may be used as an internal message, and be brought back to the message list. Thus, it will be considered as an input in the next time step. In figure 2.2 the core part of the classifier system, which has already been shown in detail in figure 2.1, is placed into a dashed box. The arrow pointing from the right edge of the rule base to the message list illustrates the stream of internal messages. To distinguish between internal messages and output signals, one bit of the action part has to be reserved to determine the type of the signal.

Now, how can the classifier system get connected to the environment? In general, it is assumed that there exists an input interface, which translates the information available in the environment into signals that can be interpreted by the classifier system. Thus, the signals generated by the input interface must be vectors with a fixed length, containing only integer (binary) entries. The input vector may represent the state in a chess game, or it may contain the temperature, atmospheric pressure, and the atmospheric humidity. These

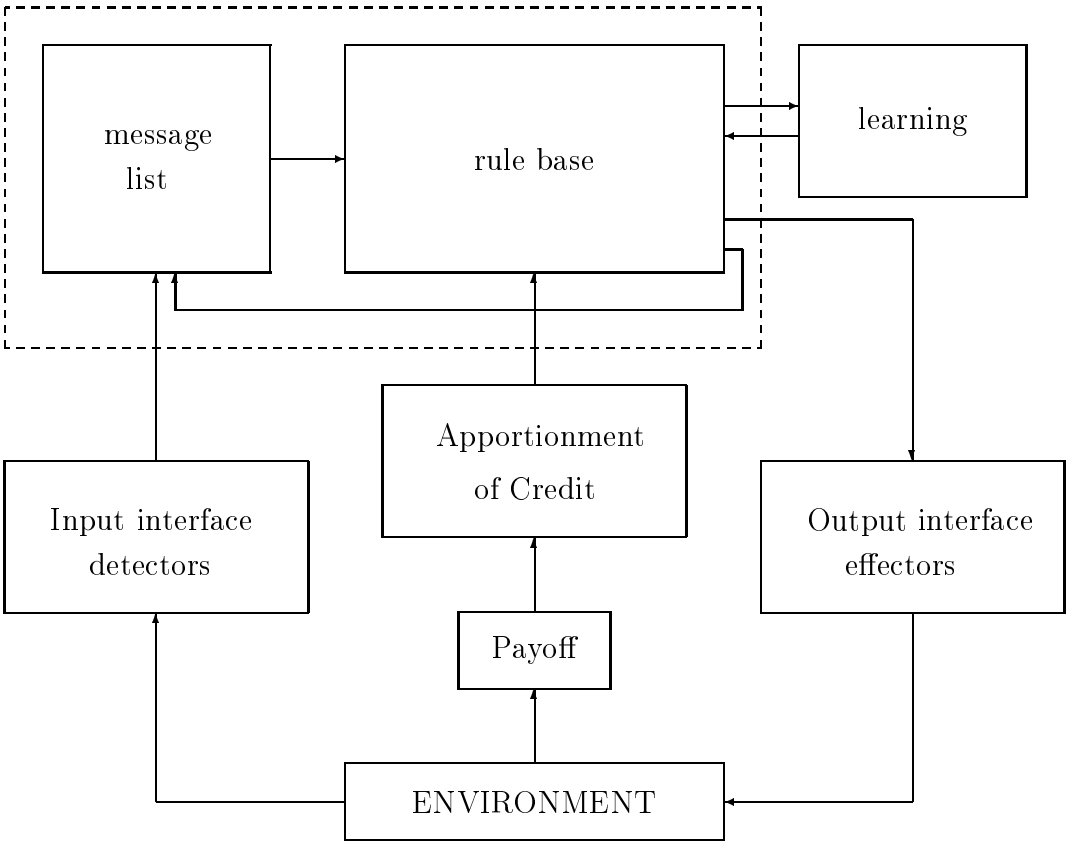


Figure 2.2: A classifier system

signals are collected in the message list, and proceeded as described in the above paragraphs.

In case the action is posted to the environment, it has to be translated by the output interface. For instance, if a CS is used to play chess, the output interface translates each possible vector into a particular move. When it is used to forecast the weather, it may contain the data being of interest for the meteorologist. Whatever kind the chosen actions are, it influences the environment, and may yield a good or bad situation for the individual represented by the CS, resulting in a certain payoff.

2.1.3 Coupled rules

So far we have only discussed how one rule, activated by one or more input messages, can post a message to the environment and, therefore, cause a good or bad result for the CS as a whole. However, CSs can also produce organized sequential activities. This can only be achieved by means of internal messages. In figure 2.3 we see a simple example of coupled rules containing only one condition part. At time t a rule is activated by an input message. We assume that its action part is tagged as an internal message. Thus, it enters the message board of the next time step, and, in turn, may fulfill another rule's condition. Thus, the effect of the input message that entered the system at time t occurs at time $t + 1$, which leads to a time lag between input and output. If more rules are coupled like that, also bigger time lags may appear.

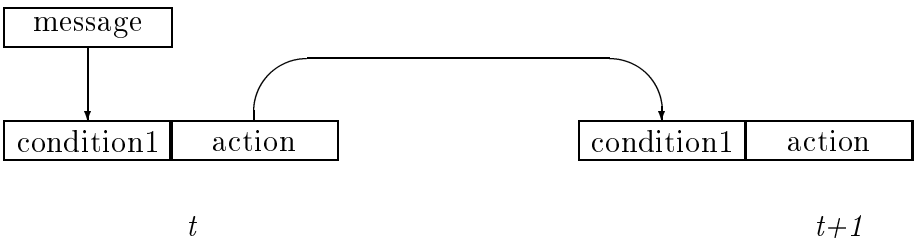


Figure 2.3: Coupled Rules in case of one condition part

In case of two or more condition parts the emergence of rule coupling is a little bit more complex. In figure 2.4 a message matching condition1 enters the system at time t. Assuming there is another message, which fulfills condition2, the corresponding classifier may be selected to post. In the sequel (time $t + 1$) there might be another classifier with condition2 matching that message. Activating that classifier requires another message fulfilling conditon1. Nevertheless, whenever an action labeled as internal message matches any condition of another rule these two rules are called coupled.

Summarizing we can say that rule coupling occurs, when a rule active at time t may result in activating another rule at time $t + 1$.

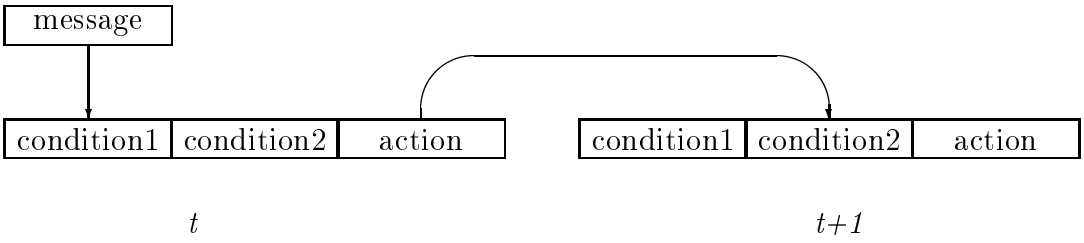


Figure 2.4: Coupled Rules in case of two condition parts

1. The rule and message (performance) system is set up for stimulus response boolean function learning.
2. The bidding structure is ideal.
3. Default hierarchy formation is perfect.
4. Context sensitivity is negligibly small.

Table 2.1: Restrictions for an idealized classifier system

Although such complicated connections within a CS violate the restrictions defining an idealized CS (Goldberg et al., 1992, p.1-2) (see table 2.1), they are widely used to implement complex types of behaviour such as time lags between emergence of a signal and the appropriate reactions, or combining information of the past with recent inputs to completely understand the encountered situation.

2.1.4 Apportionment of credit

Somehow the system must recognize and recompence the rules responsible for its success. The inductive mechanism must execute three different but connected tasks. It has to

- evaluate the systems rules as tools for accomplishing a certain goal, improve them when necessary and favour the better ones in application;
- create useful new rules that are capable to detect and apply regularities in experience; and
- provide associations between and clustering among classifiers to generate more complex knowledge structures leading to efficient modelling of the environment.

Hence, we need a mechanism to evaluate all the situations that can occur in the environment. This may be a problem sometimes. If again we think about a CS trained to play chess there are three possible outcomes, a win, a loss, and a draw. Although it seems to make sense to rate them with 1, -1, and 0 – or to use any equivalent scale – there arises the problem, that only at the end of the game the result is known. If the value of the result of the game gets assigned to all the rules activated during the game, it may happen that extremely good rules get assigned a bad value and vice versa.

If we think about checkers as another example, it is easy to agree that a triple jump is something very valuable, hence, the according rule should get assigned a very high strength. However, how can we find out which classifiers made it possible to enter a state that allows such a triple jump? Moreover, even when there is a possibility to evaluate the state at every time step, in a CS with internal messages there is still the problem of assigning meaningful values to those rules that caused internal messages.

Several techniques to deal with that problem have been developed. Basically, we can distinguish between local and nonlocal techniques for apportioning credit. If the rules are strengthened directly, without storing intermediary results, then this is called a local technique, otherwise it is nonlocal.

2.1.5 The bucket brigade algorithm

In this section we will describe a very popular local method called the bucket brigade algorithm (see Goldberg (1989, p. 255 ff) or Holland et al. (1997, chap. 10)). This procedure can be seen as an information economy. The main components of this economy are the auction and the clearinghouse. The classifiers take over the role of the trading agents who can buy and sell the right to trade information. The rules place bids in exchange for the opportunity to post their actions. The bid is a function of the rule's strength, the specifity, and the support it gaines from the messages that match it. With this bid a rule can participate in an auction and, as a result, the message of the winning rule will be posted. Rather than always choosing the message with the highest bid, it makes sense to use a random process with the bids being the weights.

Whenever a rule is chosen to post its action part, the message carries support equal to the bid that was previously placed. Consequently, this support can be consumed by another classifier in the next time step. Therefore it influences the auction taking place in the following cycle. As a result, the support of the messages has immediate consequences for the further development of the system. Moreover, the bids are also used to update the rules' strengths on the basis of current bids. The influence of those adaptions of the strengths lasts much longer than the change in support.

In a sequence of coupled rules each rule can be interpreted as a middleman in a complex supply chain. The suppliers are those rules providing messages that match a condition of the considered rule. The customers, on the other

hand, are those rules whose conditions match with the action of the rule taken into consideration.

When a rule wins an auction and, therefore gets the right to post, it pays its bid to its suppliers and receives payments from its consumers in exchange for providing support. Thus, the strength of the rule reflects the ability to create a profit. A rule that receives higher payments from its customers than it payed to its vendor makes a profit, which results in an increased strength. The more profitable the consumers are, the more likely it is for a rule to make a profit itself. This ensures to detect successful chains of coupled rules rather than only rewarding the last rule for the good result at the end of a complex series of decisions.

The following equation (2.1) shows the whole process of updating the strength,

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t)\,. \tag{2.1}$$

The strength of the i-th rule at time t is donated by $S_i(t)$. Each matching rule offers a bid $B_i = c_{bid} \cdot S_i$, and an auction-mechanism with random noise is applied to select a rule. Afterwards, the wining rule has to pay $P_i(t) = B_i(t)$. All the rules – also those, that do not match – have to pay a tax $T_i = c_{tax} \cdot S_i$. This ensures that the strength of useless rules decreases, and they can get discarded by the learning algoritm, for instance a genetic algorithm. Finally, the reward possibly gained in the previous cycles adds to the strength. Thus, rules that led to a successful outcome get an encouragement.

The ultimate rules gain profit directly from the environment. At this moment the payoff is added to all the rules that are active. When a chain of rules leads to a bad result, then the reward will be low. In the sequel this rule will not be able to pay a competitive bid anymore. As a result, the suppliers of this rule have to sell their actions for a lower price, which reduces their strengths as well. After several time steps the chain of coupled rules will not be able to win the bidding processes anymore and alternative paths get explored. The bucket brigade algorithm conserves strong links and repairs weak links.

In chapter 4 we will introduce a method similar to the bucket brigade algorithm to solve the problem of apportionment of credit in case of a series of classifier systems.

2.1.6 Learning better rules

A classifier system creates new classifiers (rules) by running a genetic algorithm (see section 2.2) — or any other suitable learning algorithm (see also sections 2.3.3 and 2.3.4) — on the present population of classifiers. In figure 2.2 the box in the right upper corner is connected with the rule base by two opposite arrows. This indicates that the learning process takes the current population of classifiers and their recent strengths as an input to create the new population which will be used in the next time step. To avoid an extremely volatile behaviour of the system, the incoming messages have to be

processed through the classifier system several times before the learning algorithm may be invoked. In chapters 4 and 5 the variable r_p is used to denote the number of repetitions.

2.2 Genetic algorithms

Genetic algorithms (GA) basically have been created to optimize technical systems. Later on, it turned out that they might also be of interest in modeling human behaviour. This is due to the strong analogy between the genetic operators and human learning by trying, experiencing, and imitating, which is outlined in more detailed in the subsequent paragraphs. This makes them a powerful tool to simulate social interaction. Todays applications of GAs cover such fields as optimization, automatic programming, machine learning, economics, operations research, ecology, population genetics, studies of evolution and learning, and social systems.

A GA works on a population of strings with a fixed length. These strings are called genomes and, similar to genomes in nature, they carry all the information required to identify an individual. In genetic algorithms a genome usually represents a possible solution within a given field of feasible solutions of a particular problem. When GAs are used to optimzie a CS, then each genome represents a rule. These strings may contain only binary digits (i.e. only 0 and 1), integer numbers, real values, symbols from any appropriate alphabet, or even such complex entries like tree representations. In case of a CS the genomes can only take binary or integer numbers. Although originally GAs were developed for operating on binary strings only, the main principle remains the same when the operators are modified to be able to handle more sophisticated alphabets. Michalewicz (1994) has shown, that real valued GAs are more efficient than binary GAs in terms of consumption of CPU time. Nevertheless, the whole population of strings must be of the same type.

The set of all possible genomes is called the search space. Applying a GA only makes sense, when the search space is too large to fully enumerate it, and when there is no structure that allows a directed search or even an analytical optimization procedure. Depending on the kind of data to be used, the genetic operators differ sligthly.

The solution process works as follows. First an initial population of genomes is generated by a random processor, or manually, if some predetermined knowledge should be inserted. Each string gets assigned a particular fitness value by some meaningful evaluation process. Then the GA tries to improve the average fitness in the population by an iterative procedure. The old population and its fitness values are the basis for creating a new generation of strings.

The main operations of GAs are

- selection,
- replication,

- recombination, and
- mutation.

Certainly, it does make a big difference for the individual if the evolution of the system as a whole is caused by natural selection (i.e. elimination of unsuccessful individuals) or by learning. In contrast to GAs, learning processes are characterised by

- variation,
- satisfaction, and
- imitation.

A nice distinction between learning and evolution is given for instance by (Brenner, 1998). The main difference lies in the fact that a learning process changes the involved individuals, while an evolutionary process changes the distribution of different types of individuals within a population, without any modifications of the individuals. However, here the GAs are used to force evolution and improvement within the individuals' rule-bases rather than within the population of agents itself. The share of those rules that performed well in the past increases, while shares of rules that led to an outcome below the average decreases. Thus, only rules can be eliminated, but not the individual as a whole. Therefore, the improvement within the population of agents takes place due to learning effects rather than selection among individuals.

In the following we will have a closer look to the main steps of genetic algorithms.

2.2.1 Selection

The selection operator chooses strings out of the population that are allowed to place their offsprings into the next generation. The selection process is highly influenced by the fitness values. A meaningful selection operator must secure that genomes with a high fitness are more likely to be chosen. Otherwise previously gathered knowledge may get lost. Individuals may also be selected more than once, and all individuals – also those which performed very badly in the previous steps – have a chance to be selected. Thus, the selection is based on a probabilistic procedure. Common schemes of selection are roulette wheel selection, scaling techniques, tournament, elitist models, and ranking methods.

Many popular selection methods assign a probability P_i to each genome, depending on its fitness. A sequence of random numbers x_n is generated and compared with the cumulative probability $C_i = \sum_{j=1}^{i} P_j$. For each random number x_n that lies within the interval $(C_{i-1}, C_i]$ the individual i is chosen. Several methods (roulette wheel, linear ranking, and geometric ranking) make use of this idea. They only differ in the mechanism of assigning probabilities.

Roulette wheel selection

Roulette wheel selection, developed by Holland (1995), defines the probability according to

$$P_i = \frac{F_i}{\sum_{j=1}^{J} F_j} ,$$

with F_i (F_j) denoting indivial i's (j's) fitness and J being the size of the population. This formula only produces meaningful results, when the objective is to maximize the (average or maximum) fitness within the population and all fitness values assigned take only nonnegative values. In an attempt to allow for minimization and negative fitness values as well, extensions such as windowing and scaling have been proposed. Nevertheless, the basic principle is still the same.

Remark: It follows immediately that the above formula guarantees that the probabilities sum up to one.

Ranking methods

A ranking method can always be applied when the evaluation process maps the genomes to a partially ordered set. First, all the solutions are sorted – i.e. there must be a preference scheme among the genomes. The probability P_i then depends on the rank of solution i within the population. Thus, it is not crucial, how well a genome solves a given problem, but how well it does in comparison to the other genomes. Joines and Houck (1994) suggest a scheme called normalized geometric ranking that assigns probabilities

$$P_i = q'(1-q)^{r_i - 1} , \tag{2.2}$$

where

q	=	parameter determining the probability of selecting the best individual,
r_i	=	rank of the individual i, 1 being the best,
q'	=	$\frac{q}{1-(1-q)^J}$, and
J	=	population size, like in the previous subsection.

For big populations q approximately equals the probability of selecting the best individual, because of

$$\lim_{J\to\infty} q'(1-q)^{r_i-1} = \lim_{J\to\infty} \frac{q}{1-(1-q)^J}(1-q)^0 = q$$

Besides the bigger range of possible applications ranking selection also provides an advantage compared to roulette wheel selection in terms of computing time. Since the fitness levels within the population may change from generation to generation, the selection probabilities taken into consideration by roulette wheel selection have to be computed separately for each generation. When ranking methods are applied, the selection probabilities only

depend on the parameter q, and on the size of the population. Therefore, they only need to be calculated at the beginning of the iteration process and may be used again for all later generations.

An important difference with respect to roulette wheel selection is that the difference of the probability of choosing the fittest, second–fittest, and so on remains the same, without regarding the difference between their fitnesses. Thus, for ranking selection the selection pressure remains constant, which does not hold true for fitness proportional selection. To illustrate the effects of geometric ranking, in table 2.2 numerical values of P_i are listed in case of a population of 5 individuals and $q = 0.2, 0.3, 0.4, \text{and}\, 0.5$.

	q			
rank	0.2	0.3	0.4	0.5
1	0.2975	0.3606	0.4337	0.5161
2	0.2380	0.2524	0.2602	0.2581
3	0.1904	0.1767	0.1561	0.1290
4	0.1523	0.1237	0.0937	0.0645
5	0.1218	0.0866	0.0562	0.0323

Table 2.2: Selection probabilities in case of geometric ranking

Remark: The above formula also guarantees that the probabilities sum up to one, since $\sum_{i=1}^{J} P_i = \sum_{i=1}^{J} q'(1-q)^{r_i - 1} = \sum_{i=1}^{J} \frac{q}{1-(1-q)^J}(1-q)^{r_i-1}$ $= \frac{q}{1-(1-q)^J} \cdot \frac{1-(1-q)^J}{1-(1-q)} = 1.$

Tournament selection

Tournament selection also requires to map all genomes into a partially ordered set. A fixed number of individuals is chosen without regarding their fitness, nor their rank. The best out of these is selected. Repeating this procedure until the desired population size is achieved concludes the selection process.

2.2.2 Crossover

After being selected the genomes are placed into a mating pool. Yet, selection alone does not enable the GA to explore new regions in the search space. Therefore, genetic operators are required that use the previously gained information to create new genomes. The first, and most important operator is crossover. Before crossover is applied, all the strings selected in the mating pool are grouped to pairs. Then, with some crossover probability p_c a new pair of strings is created by exchanging parts of an existing pair of strings. The newly generated strings are called offsprings, since they contain features of both parents. With a probability of $1 - p_c$ no crossover takes place and

both individuals are members of the new generation without any changes so far. This procedure is applied to each pair independently.

Simple crossover

The most simple crossover operator just cuts the individuals at a randomly chosen position and exchanges the strings' tails, as illustrated in figure 2.5.

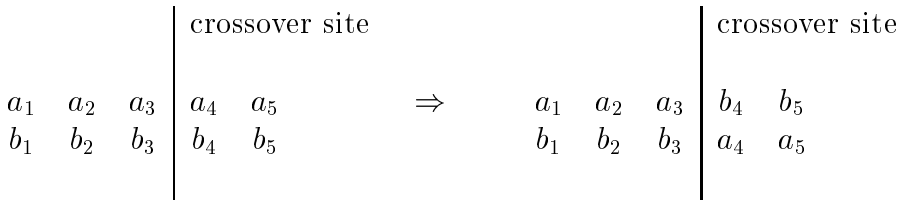


Figure 2.5: Simple crossover

This operator can be applied to binary, integer and real valued strings, since there are no requirements that must be fulfilled by the entries a_i and b_i.

Crossover masks

A more complicated method, that also can be used for any kind of data, exchanges multiple parts of the parents defined by a randomly generated crossover mask. The crossover mask is a binary string of the same length as the considered genomes. One offspring is created by placing the gene a_i at each position where the crossover mask contains a 1, and b_i otherwise. Another offspring is done just the other way round, as illustrated in figure 2.6.

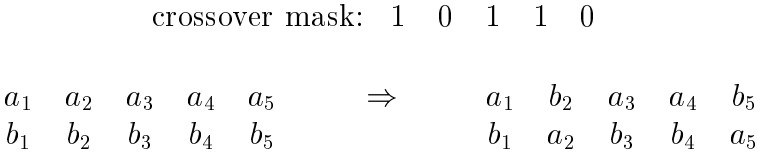


Figure 2.6: Crossover with a mask

Compared with simple crossover applying a crossover mask may enable the GA to explore more distant regions in the search space, because genes are exchanged more extensively. On the other hand, it is also more likely to loose good solutions, because sucessful combinations of genes may be disrupted. The interested reader will find a comprehensive analysis concerning the survival of schemes in (Goldberg, 1989, p. 28-33). It depends on the kind of problem and the chosen encoding of the search space which of the two methods is more efficient.

Despite the fact that simple crossover and crossover with masks can be applied to all kind of genetic encodings, they may not always succeed in finding the right genomes. Since they both just exchange genes between two individuals but never create new entries, they can only explore those regions within the search space that can be reached by those entries that already exist in the randomly created initial population. For binary strings and integer strings with a rather small difference between lower and upper bound this will not be a problem, since all the possible entries may already occur in the initial population. However, when the genomes contain real numbers, then the set of possible values is infinite. Therefore, it is very unlikely that a finite initial population already contains those values required for generating an individual that leads to an acceptable result. Two possible tools to overcome this shortcoming are big populations and crossover operators. The first tool consumes a lot of memory and CPU time, the latter is based on chance. In the following sections two crossover operators will be introduced that provide the ability to create new genes based on information that is already available within the existing population. This results in a directed search within the set of real numbers. In chapter 3 it will be shown, that those operators may produce better results in a learning domain built up by real numbers.

Arithmetic crossover

In contrast to the crossover methods examined so far, arithmetic crossover can only be applied to a population of real-valued strings. Yet, a random number α within the interval $[0, 1]$ is used to compute a linear combination of the genomes in consideration. Thus, each gene (position) of the new genomes is influenced by both parents. This process is illustrated in equation (2.3).

$$\begin{aligned} a_i^{t+1} &= \alpha \cdot a_i^t + (1-\alpha) \cdot b_i^t \\ b_i^{t+1} &= \alpha \cdot b_i^t + (1-\alpha) \cdot a_i^t \end{aligned} \tag{2.3}$$

Remark: If there are some restrictions that must be fulfilled by the components – i.e. a lower and an upper bound – then they are fulfilled, if the parents fulfill them.

Heuristic crossover

Heuristic crossover is another method that only makes sense when the search space is encoded by real-valued strings. While arithmetic crossover can be seen as some kind of linear combination that searches for intermediate solutions of the parents, heuristic crossover can be interpreted as an extrapolation of the existing solutions. To determine in which direction the extrapolation should take place, the fitness values of the existing strings are utilized. It is assumed that the fitness might become better when the neighbourhood of the more successful one of the two individuals is explored. Again a random number $\alpha \in [0, 1]$ is generated. Now (w.l.o.g.) let us assume that (a_i) is the

better performing genome. Then the new individuals become

$$\begin{aligned} a_i^{t+1} &= a_i^t + \alpha \cdot (a_i^t - b_i^t)\,, \\ b_i^{t+1} &= a_i^t\,. \end{aligned} \tag{2.4}$$

However, after computing the new individuals according to equation (2.4) it is checked whether the new individual (a_i^{t+1}) is feasible, i.e. if all its components a_i lie between some lower and upper bounds $\underline{a}_i$ and $\overline{a}_i$, limiting the region of useful solutions within the search space. If this restrictions are not fulfilled, then another random number is computed and the extrapolation is repeated. Nevertheless, it must be ensured that this process terminates within finite time. Thus, a maximum number of trials is defined in advance. If no feasible solution is found after the maximum number of trials, then both genomes remain unchanged and the offsprings in the next generation are identical to their parents.

2.2.3 Mutation

The operators described so far enable the GA to explore new regions of the search space while previously collected knowledge is taken into consideration. Although it sounds meaningful to create new generations by selecting genomes regarding their fitness and combining them, this alone would not make GAs such a powerful tool for optimizing complex functions. When only selection and crossover are implemented, the GA is tempted to converge very quickly. Hence, it is possible that those regions of the search space containing the best solutions in terms of fitness are left out. To avoid such an overhasty convergence, a mutation operator slightly changes the offsprings gained by crossover. Thus, there is a chance to jump into regions that have not been visited so far. To avoid that a whole population of successful individuals gets destroyed, mutation should only take place with a very small probability. If the mutation probability p_m is fixed, then values around 0.001 have turned out to be best. More sophisticated GAs start with a higher mutation probability, to avoid a quick convergence at the beginning, and reduce it later on, helping the GA to arrive at a steady state before termination. This method will be shown in more detail in section 3.3.

Binary mutation

The simplest mutation operator is called binary mutation, and – as the name already indicates – it can only be applied to binary genomes. If mutation takes place, a 0 becomes a 1 and vice versa. Each bit is fliped with a probability of p_m. Like in any other mutation technique, all the random processes used to determine whether a change takes place are independent.

Uniform mutation

For integer or real valued strings the method described in the section before does not work well, since there are more than two values that can occur in each gene. In case of mutation, a random variable is inserted at the chosen position. As already mentioned in the section about heuristic crossover, we need a lower bound $\underline{a}_i$ and an upper bound $\overline{a}_i$. For integer values the discrete uniform distribution over the set $\{\underline{a}_i, \underline{a}_i + 1, \ldots, \overline{a}_i - 1, \overline{a}_i\}$ is used. In case of real-valued genomes the random variable follows a uniform distribution over the interval $[\underline{a}_i, \overline{a}_i]$.

Boundary mutation

Boundary mutation operates very similar to uniform mutation. The difference is that only the lower and upper boundaries ($\underline{a}_i$ and $\overline{a}_i$) are candidates for being inserted. Usually both are chosen with equal probability. Consequently, the chance, that a certain gene may be changed to its lower (upper) bound is $p_m/2$.

Non-uniform mutation

Another procedure similar to uniform mutation is non-uniform mutation. The only difference is the random distribution used to determine the new elements. Houck et al. (1995) suggest the following method. Two random variables $\alpha_1, \alpha_2 \sim U_{[0,1]}$ are generated to derive the new gene

$$a_i^{t+1} = \begin{cases} a_i + (\overline{a}_i - a_i) f(j) & \text{if} \quad \alpha_1 < 1/2\,, \\ a_i - (a_i - \underline{a}_i) f(j) & \text{if} \quad \alpha_1 \geq 1/2\,, \end{cases} \tag{2.5}$$

where

$$\begin{array}{rcl} f(j) & = & (\alpha_2(1 - \frac{j}{G}))^b, \\ j & = & \text{the index of the current generation,} \\ G & = & \text{the maximum number of generations (explained} \\ & & \text{in detail in a later section), and} \\ b & = & \text{a shape parameter.} \end{array}$$

The function $f(j)$ is strictly decreasing in its argument j. Therefore, the changes caused by mutation are bigger at the beginning and become less dramatic when the generation counter approaches close to termination.

Multi-non-uniform mutation

A very severe kind of mutation is the multi-non-uniform mutation. This method applies the operator described in the section before to all the entries in the parent string. Thus, the string experiences a complete change, possibly interfering all the schemes that it represents.

Since $f(j)$ in equation 2.5 takes non-integer values, (multi-)non-uniform mutation is only meaningful when the genome population is real-valued.

2.2.4 Termination

Although GAs have been applied successfully in many fields, there is no guarantee that they will always converge towards an acceptable solution. Therefore, it is recommended to define a fixed maximum number of generations to ensure that the GA terminates. Additionally, it is possible to implement further termination criteria. Such criteria for instance could be the attainment of an acceptable fitness level. The fitness under consideration can be the maximum fitness, if only one good solution is required, or the average fitness, if a whole bundle of good genomes is needed. The latter is the case when GAs are used to optmize CSs, where a list of rules leading to satisfying results in many situations is desired. Another meaningful approach is to stop the iteration procedure, when the fitness level has stopped increasing for several periods. However, such a feature should always be implemented with caution, since in very complex search spaces it may happen that the GA seems to converge, due to an arrival at a local extrema, while within some more iterations it might find an even better region, due to a successful shift caused by the mutation operator. In cases where only one solution is desired, the algorithm may also be terminated, when the deviations among the whole population fall below a specified level.

2.3 Opportunities and limitations of genetics-based machine learning

According to Korzybski (1950) animals and plants can be seen as *space-binders*, while humans can be better characterized as *time-binders*. This is due to humans capabilities of learning from experience and passing what was learned to succeeding generations. Experience can be transfered into later periods. Humans can collect knowledge from the past and inform future generations about their own knowledge. In this chapter the possibilities and the limitations of describing those abilities by algorithms and implementing them in artificial learning systems will be elaborated.

2.3.1 Gaining knowledge

At the beginning newborn children have almost no knowledge, except their innate instincts. Motivated by their curiosity they observe their environment and try to influence it with their own actions. The reactions of the environment are also observed. Many of these observations (if not all) are stored in the brain, and gradually some connections and interdependencies emerge. The child learns to predict the results of different actions under given circumstances. If a completely new situation arises, the learning individual presumably tries to compare it with a similar situation already encountered before. If that doesn't work either, the child will try any action — this action can also mean to do nothing. Based on this single observation a completely

new rule will be formed. As time elapses, these rules may change, due to forgetting, trying new actions, or when a prediction turns out to be wrong.

Machine learning

Algorithms applied in the field of machine learning (ML) are inspired by human learning. Therefore, procedures similar to those described in the previous paragraph can be found. Machine learning is the study of computer algorithms that improve automatically through experience. For instance, a rule-base is initialized with random numbers and a learning algorithm is used to update it. The rule-base may also be seeded with some rules to insert some existing knowledge. This existing knowledge can be the result of another search algorithm, or it can stem from the wisdom of human experts. If all the rules are given right from the beginning, this structure is called an expert system. Typical ML-systems do not apply given knowledge, but only learn through experience and make predictions about the future.

The main challenge is to describe all relevant states of the environment, all possible actions, and all possible results in a way that can be interpreted by a computer.

> *If we are ever to make a machine that will speak, understand or translate human languages, solve mathematical problems with imagination, practice a profession or direct an organization, either we must reduce this activities to a science so exact that we can tell a machine precisely how to go about doing them or we must develop a machine that can do things without being told precisely how ... In short, although it might learn to perform a task without being told precisely how to perform it, it would still have to be told precisely how to learn.*
>
> Friedberg (1958)

The process of ML starts with the identification of the learning domain and ends with testing and using the results of the learning. A learning domain is any problem or a set of facts, where it is possible to identify the "features" of the domain that are to be measured, and a result or several results that should be predicted. Certainly, this can only be done when there exists a connection between the features and the results. An ML-system goes through the learning set (a subset of the learning domain), and tries to learn from those examples. The validation set is another subset of the same learning domain. The inputs of the validation set are used to test, whether the ML system indeed has learned a meaningful connection between features and results, or it has just stored the learning set. The ability to apply the learned connections to new data sets is called generalization. The difference between several well-known ML-systems is mostly due to the applied learning algorithms.

2.3.2 Problem representations

The representation of an ML-system contains the definition, how possible solutions of the problem may look like, what kind of inputs are accepted, how the inputs are proceeded, and how the outputs are generated. In Banzhaf et al. (1998) five kinds of representation typical for ML systems are mentioned. In the following sections it will be shown that it is possible to transform all of them into an equivalent classifier system.

Boolean representations

The following example for illustrating the different representation concepts is given in Banzhaf et al. (1998). Let us assume a scientist wants to determine whether a particular character featured in a Dick Tracy cartoon is a "bad guy" or a "good guy". The researcher examines several episodes of Dick Tracy cartoons and recognizes the following features to be useful:

- shifty eyes
- scarred face
- skull tattoo
- slouches while walking
- hooked noose
- wears two-way wrist radio

All these features can be encoded by a boolean variable (*true* or *false*). There are two basic types of boolean systems:

Conjunctive boolean systems (CBS)

A CBS connects the features with a logical **AND**. A particular learning algorithm may have found the concepts given in table 2.3 to distinguish between "bad guys" and "good guys".

concept 1	shifty eyes **AND** scarred face **AND** skull tattoo
concept 2	hooked noose **AND** wears two-way wrist radio

Table 2.3: Learning concepts in a conjunctive boolean system

Yet, the concepts alone do not suffice. They must also be interpreted and classified. The classification again may be represented in different ways. Dick Tracy himself may choose the following method to use the two concepts. Concept 1 corresponds with his personal "crime watchers guide". Concept 2,

concept	value	bad guy?
1	*true*	*true*
2	*true*	*false*

Table 2.4: Classification of the concepts

on the other hand, is fulfilled by himself. Therefore, Dick Tracys classification of the concepts becomes as listed in table 2.4.

Each of the concepts only becomes valid, when all the features are fulfilled. Therefore, there might be many situations, where none of the two works. A CS implementation of those concepts is given in figure 2.7.

$$\begin{array}{cccccc|c} 1 & 1 & 1 & \# & \# & \# & 1 \\ \# & \# & \# & \# & 1 & 1 & 0 \end{array}$$

Figure 2.7: Classifier system implementation

To use this CS, the person under consideration has to be examined with respect to all the six features. The results are written into a row vector of length six. A 1 means that the according feature is *true*, a 0 means *false*. This vector becomes the input and is going to be compared with the condition part of the CS — the part on the left hand side. If the input corresponds with the output at all components, except those marked with a don't care symbol #, then the rule is fulfilled. If both or none of the two rules are matched, the CS does not help in classifying the person. If only one of the two rules is matched, then it can be used as a tool for characterization. In this example the output is only one-dimensional. A 1 indicates a "bad guy" and 0 a "good guy". In more complex problems the output may be an element of a vector-field of any finite dimension.

Disjunctive boolean systems (DBS)

In a DBS the features are connected with a logical **OR**. Whenever one of the simple concepts of a DBS is fulfilled, then the output also is defined to be *true*. Let's have a look at the concepts in table 2.5.

concept	feature	value
concept 1	shifty eyes	*true* or *false*
concept 2	scarred face	*true* or *false*
concept 3	skull tattoo	*true* or *false*

Table 2.5: Learning concepts in a disjunctive boolean system

Now, the CS implementation becomes as depicted in Fig. 2.8

$$\begin{array}{cccccc|c} 1 & \# & \# & \# & \# & \# & 1 \\ \# & 1 & \# & \# & \# & \# & 1 \\ \# & \# & 1 & \# & \# & \# & 1 \end{array}$$

Figure 2.8: Classifier system implementation

Combinations of CBS and DBS can describe more complex input-output relations which can be used for sophisticated decision support systems.

Treshold representations

Treshold representations are more powerful than boolean representations because of their higher degree of flexibility. A treshold unit only produces an output, when the input exceeds a certain level. For instance, a heating controlled by a thermostat is switched off, whenever the temperature in the room exceeds a certain upper bound. In an ML-system a treshold unit may be used to prepare an input before it is further proceeded by the system. In a multi-layer feedforward neural network, in each layer the data are proceeded by treshold units.

For example, it may be required that at least two out of the three features under consideration in concept 1 in the section about CBS (see table 2.3) are *true*. A rather tedious CS implementation of such a treshold unit is given by

$$\begin{array}{cccccc|c} \# & 1 & 1 & \# & \# & \# & 1 \\ 1 & \# & 1 & \# & \# & \# & 1 \\ \# & 1 & 1 & \# & \# & \# & 1 \end{array}$$

Figure 2.9: Classifier system implementation

With a customized input detector such a treshold unit can be implemented in a much more elegant way. Suppose that in a first step all the three features (shifty eyes, scarred face, skull tattoo) are checked. In the next step the fulfilled features are counted, and encoded as a binary string of length 2. Thus, the input vectors undergo the following transformation

$$\begin{array}{cccccc} 0 & 0 & 1 & \# & \# & \# \\ 0 & 1 & 0 & \# & \# & \# \\ 1 & 0 & 0 & \# & \# & \# \end{array} \Big\} \rightarrow 0\,1$$

$$\begin{array}{cccccc} 0 & 1 & 1 & \# & \# & \# \\ 1 & 0 & 1 & \# & \# & \# \\ 1 & 1 & 0 & \# & \# & \# \end{array} \Big\} \rightarrow 1\,0$$

$$\begin{array}{cccccc} 1 & 1 & 1 & \# & \# & \# \end{array} \Big\} \rightarrow 1\,1$$

Now the treshold unit simply becomes $1\,\#\,|\,1$. Certainly, the rule-base only is that small, because there are only three features under consideration. Nevertheless, also in more complex situations it is often possible to compress

the rule-base, if the input interface is designed to reduce the set of possible inputs. It may be worth to spend some extra efforts in a reduction of the search space in order to accelerate the learning process.

Case-based representations

Further common methods of machine learning store the data as representations of classes. Moreover, they may just store general descriptions of classes, derived by some averaging of the training data. The K-nearest neighbour method for instance interprets the data themselves as a part of the problem representation. A new input is assigned to the class containing most of the K nearest neighbours. Consequently, each new classified element immediately influences the problem representation and also the classification of further inputs.

Another possibility is to divide the learning domain into classes separated by some hyperplanes. If the data are discrete, then such a case-based representation can also be implemented as a classifier system.

Example: Let us assume the learning domain is given by the set $\{0,1,2,3,4,5,6,7\}^2$, and two classes are defined by

$$\begin{aligned} A: y &> \frac{9}{4} + \frac{x}{2} \\ B: y &< \frac{9}{4} + \frac{x}{2}\,. \end{aligned}$$

This situation is illustrated in figure 2.10.

If the coordinates of the elements of $\{0,1,2,3,4,5,6,7\}^2$ are described by vectors containing the binary representation of the components (e.g. $(3,4) = (0,1,1,1,0,0)$), then the above classification is accomplished by the system given in figure 2.11.

This example shows, that even very simple classifications require rather complicated rule-bases, when they are implemented as a CS. Therefore, a CS-implementation may not be the method of first choice, although it is a possible solution.

Tree representations

Many decision situations may be clearly illustrated by decision trees. Consequently, many ML-systems also use tree representations. Each node represents a feature. Each edge represents a value of the feature represented by the node above it. If we look again at the two concepts given in table 2.3, then each node means a feature like shifty eyes, scarred face, and so on. The edges below these nodes can take the values *true* or *false*. Thus, an equivalent tree representation is given by the two trees in figure 2.12.

Since a tree can only accept a finite set of different values, it is always possible to implement a classifier system that is equivalent to the tree representation. A CS equivalent to the tree in figure 2.12 was already given in figure 2.7.

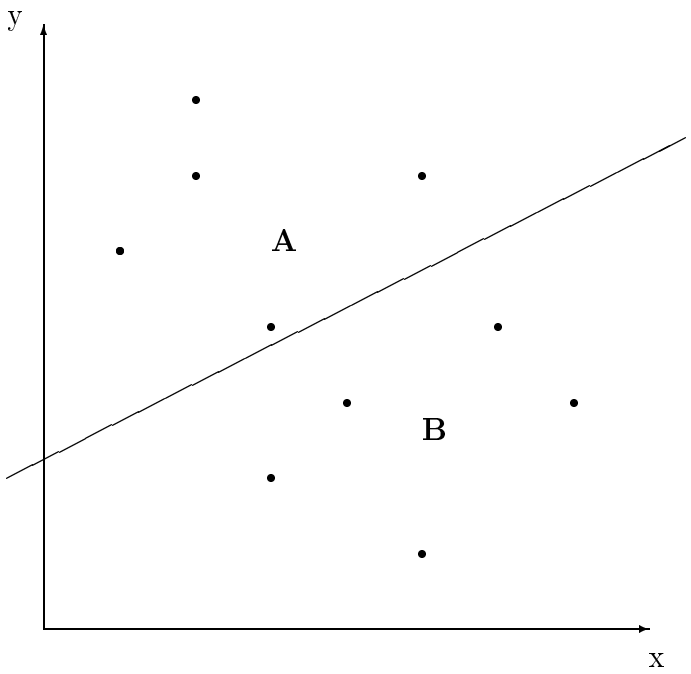


Figure 2.10: The sets A and B

0	0	#	#	1	1	A
0	1	#	1	#	#	A
1	0	#	1	#	1	A
#	#	#	1	1	#	A
#	#	#	0	0	#	B
#	#	#	0	1	0	B
#	1	#	0	1	1	B
1	#	#	1	0	0	B
1	1	#	1	0	1	B

Figure 2.11: Classifier system implementaion

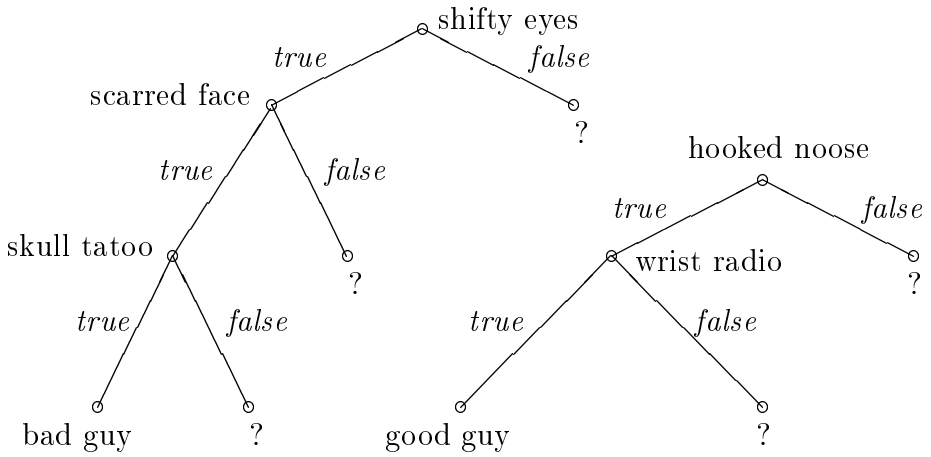


Figure 2.12: Tree representation

Genetic representations

The applications of genetic representations are manifold. Genetic algorithms (described in detail in section 2.2) often use binary strings of a fixed length. Each bit gets assigned a particular meaning. Since there are no limitations about this assignment, genetic representations offer a very high degree of freedom. As already illustrated, with some limitations all the concepts mentioned so far can be substituted by an equivalent genetic representation. Genetic algorithms do not use any information related to the meaning of the bits. Representations allowing more entries than just 0 and 1 — for instance integer numbers, real values, or alphabets of any symbols — often lead to an increased efficiency compared to pure binary encodings.

2.3.3 Search Strategies

Yet, the decision about the problem representation alone does not solve the problem. In the next step a suitable method to explore the solution space has to be chosen. The current literature (see for instance Goldberg (1989) or Banzhaf et al. (1998)) identifies three classes of search methods:

- calculus–based
- enumerative
- randomized

Calculus–based methods only work when the function that has to be optimized is known and sufficiently smooth (e.g. continuously differentiable). Since such functions are only a small subset of the whole range of possibilities

— and often it is only possible to evaluate single points without obtaining information about the universal connection between input and output — calculus–based methods often fail. Enumerative methods require evaluating every single point in the search space. The idea and also the implementation of such search strategies is rather straightforward. Dynamic programming is one of the most popular implementations of an enumerative scheme. However, in most cases evaluating the whole space is not possible due to the complexity of the problem and to time restrictions. Therefore, most ML-systems apply randomized methods to explore the solution space.

It is important to distinguish between random methods (e.g. blind search) and directed search methods that use some random choices. The search operators determine how and in which order the ML-system selects the possible solutions. It is evident that a proper ML-system chooses a path through the solution space that detects good solutions and omits bad ones. In general all well-known randomized search strategies belong to one of the following three categories.

Blind search

When applying blind search the elements of the solution space are chosen without considering any information about the structure of the problem or results of previous learning iterations. The algorithm continues until a termination criterion is met. Such termination criterions may for instance consider the number of evaluations or the quality of the solution. The termination criterion may be the same for all the three categories mentioned here. When the search space is sufficiently small, blind search may be the best alternative — in terms of CPU–time consumption. When problems show some level of complexity such simple search methods usually do no better than enumerative schemes.

Hill climbing

Hill climbing methods start at a single point in the search space, evaluate this single point and store the solution. Afterwards some movements are done. Only the best solution found so far is stored. If a new solution is better then the previously best solution, the new solution is stored and the old one is discarded. Otherwise, the new solution is discarded immediately. There is no record containing several of the past solutions.

Some examples of hill climbing algorithms are simulated annealing and many neural network training algorithms. At each time step only one solutions is being considered, and only one path through the solution space is investigated. For the sake of completenss it should be mentioned that also methods that are looking for a valley (minimum) on a fitness–landscape may belong to the group of hill–climbing methods, if the basic idea behind the search is the same.

Beam search

All the algorithms mentioned in the previous sections perform single point–to–point searches. Beam search methods always keep a record of a population of solutions. Thus, beam search can be seen as a compromise between exhaustive search (enumeration) and hill–climbing. An evaluation criterion is used to choose a certain number of individuals that are taken into the population. These individuals form the beam. All other solutions that have been evaluated are discarded. In the next time step this beam is used to find new solutions which again have to be evaluated. Thus, the search space is limited to those solutions that can be generated by applying some search operators to the individuals within the given population.

Compared to hill climbing the memory required to store the intermediate results increases but, on the other hand, the number of necessary function evaluations decreases. Some examples of beam search methods are genetic algorithms (see section 2.2), particle swarm optimization (see Eberhart and Kennedy (1995), Kennedy (1997), Kennedy and Eberhart (1995), Kennedy and Eberhart (1997), and Kennedy and Eberhart (1999)), ant colony optimization (see Dorigo et al., 1991), and other contemporary heuristic search methods. ML–systems have operators to determine the size, contents, and the ordering of the beam.

2.3.4 Learning methods

Among others, there are three main approaches that are of interest for genetics based machine learning.

Supervised learning

Every output produced by the learning agent is compared with a given desired output. The deviation or correspondence determines the fitness assigned to the objects that produced the output. This is for instance applied in a multilayer perceptron that has to find a nonlinear conncetion between multidimensional inputs and outputs. Examples of genetic algorithms performing supervised learning are given in sections 3.1, 3.3, and 3.4.

Unsupervised learning

When unsupervised learning is desired, the system does not get any information about the desired output. Rather it has to find it's own classification of inputs. Examples for unsupervised learning are Kohonen networks, or the K-nearest neighbour classification (see section 2.3.2, p.38).

Reinforcement learning

Reinforcement learning can be seen as a compromise between supervised and unsupervised learning. The system is not told directly what output would

have been desired. Instead there is an evaluation function providing information about the quality of the solution. Thus, the system obtains more information than in case of unsupervised learning, but the information is not very accurate. This is the most common approach in ML. Examples are given in section 3.2, and in chapters 4 and 5.

Chapter 3

Examples

In chapter 2 the concepts and methods of genetics-based machine learning were sketched briefly. In this chapter four examples of possible applications will be given. In sections 3.1 and 3.2 it will be shown that genetic algorithms may also find pretty good solutions when an exact analytic solution technique is available. Of course this is not for free. When the information utilized for obtaining the analytical solution is neglected and a heuristic search method is applied then this is done for the cost of performing significantly more computations. In section 3.1 the function under consideration is smooth and has only one local extremum, while in section 3.2 a function with several optima will be studied. Later on, in sections 3.3 and 3.4, GAs as a tool for training classifier systems will be examined. While the CS in section 3.3 performs a rather simple input-ouput relation, in section 3.4 a CS is required to store the inputs for one time unit, leading to certain difficulties concerning the apportionment of credit.

3.1 Ordinary least squares estimation

Let us assume that y is an affine function of the two variables x_1 and x_2 disturbed by some white noise ϵ, i.e. $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$. Given K samples of y_j, x_{j1}, and x_{j2} ($j \in \{1, 2, \ldots K\}$), the task of ordinary least squares estimation is to find estimators $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$, that minimize the sum of the squares of the errors

$$S\left(\vec{\beta}\right) := \sum_{j=1}^{K} \left[y_j - (\beta_0 + \beta_1 x_{j1} + \beta_2 x_{j2})\right]^2 . \tag{3.1}$$

If we define

$$\vec{\beta} := \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}, \; X := \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{K1} & x_{K2} \end{bmatrix}, \text{ and } \quad \vec{y} := \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix},$$

then $S\left(\vec{\beta}\right)$ can be rewritten as[1]

$$S\left(\vec{\beta}\right) := \left(y - X\vec{\beta}\right)' \left(y - X\vec{\beta}\right) .$$

The best linear unbiased estimator $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2)'$ is given by

$$\hat{\beta} = (X'X)^{-1}X'\vec{y}. \tag{3.2}$$

Proof: See (Caspary and Wichmann, 1994, p.198ff) or (Johnston and DiNardo, 1997, p.70ff).

If we assume that $\beta_0 = 1$, $\beta_1 = 0.3$, $\beta_2 = 0.5$, and $\epsilon \sim N(0, 0.01)$, then we might get the following samples:

x_1	x_2	ϵ	y
1.4	1.8	0.00000431918416	2.32000431918416
0.1	0	-0.03178594512477	0.99821405487523
0.1	0	0.10950037387875	1.13950037387875
0.9	0.6	-0.18739902576410	1.38260097423590
0.0	1.8	0.04281832730452	1.94281832730452
1.2	0.2	0.08956384712118	1.54956384712118
2.0	1.1	0.07309573384295	2.22309573384295
0.3	1.9	0.05778573463308	2.09778573463308
0.1	2.2	0.00403140316184	2.13403140316184
1.8	2.1	0.06770891875973	2.65770891875973

Table 3.1: Samples for the ordinary least squares estimation

According to equation (3.2), the best unbiased linear estimator becomes

$$\hat{\beta} = \begin{pmatrix} 1.00687068282030 \\ 0.30908175061436 \\ 0.50725393409751 \end{pmatrix} . \tag{3.3}$$

Applying these values we get the squared error (see equation 3.1) $S(\hat{\beta}) = 0.06525$, while when applying the actulally chosen parameter vector $\beta = (1, 0.3, 0.5)'$ the squared error becomes $S(\beta) = 0.07126$. So we see that it is impossible to find the originally chosen parameters by just observing the above sample. The sample data and their projections into the (hyper-)plane defined by the parameters given in equation (3.3) are shown in figure 3.1.

Now we will have a look at the capabilities of GAs to estimate the parameters. We start with a randomly chosen initial population of 10 individuals and compute the squared errors $S(\beta)$. The obtained results are listed in table 3.2.

To generate this population, a uniform random distribution over the interval $[0, 1.5]$ was used, and all the entries were rounded to two digits after

[1] The apostrophe $'$ indicates the transpose of a matrix or vector.

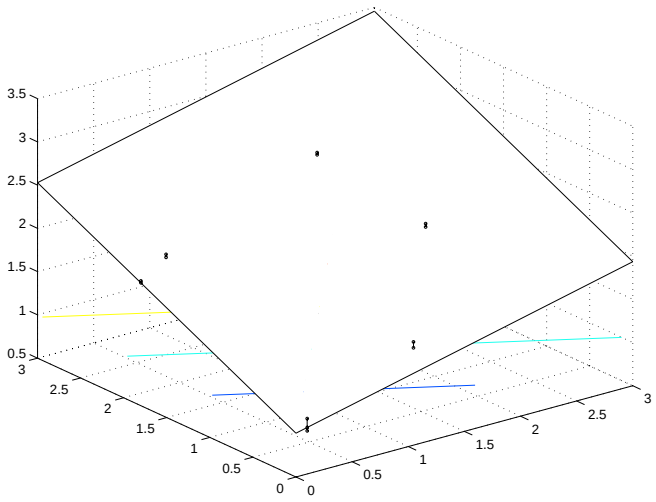


Figure 3.1: The sample data and their projections

index	β_0	β_1	β_2	$S(\beta)$	rank	P_i
1	0.99	1.45	1.59	63.8371	6	0.0519
2	1.80	0.62	1.91	87.4115	9	0.0178
3	1.64	1.68	1.05	68.7301	7	0.0363
4	1.29	1.14	1.76	74.7146	8	0.0254
5	1.64	0.74	0.35	7.3564	3	0.1513
6	1.32	1.41	1.96	107.5855	10	0.0125
7	0.68	1.09	0.54	4.4434	2	0.2161
8	0.58	0.89	0.50	1.8626	1	0.3087
9	0.68	1.39	1.75	58.9737	5	0.0741
10	1.07	1.24	1.47	49.9509	4	0.1059

Table 3.2: Initial population

the decimal point. This is obviously a very small population, but we want to keep the problem small such that it is possible to pursue the effects of the genetic operators. The average squared error within this initial population is $\bar{S}(\beta) = 52.4866$, which is about 800 times the squared error of the best solution $\hat{\beta}$. The best individual within this population reaches a squared error of 1.8626 (about 30 times $S(\hat{\beta})$), while the worst shows a value of 107.59 (about 1600 times $S(\hat{\beta})$). Starting with this population we try to find better parameter sets within the learning domain.

Selection

Obviously the squared errors $S(\beta)$ are used to rank the individuals. Since this value has to be minimized, roulette wheel selection cannot be applied straight forward. If there exists an upper bound S^{up} then a fitness value can be computed following $F_i = (S^{up} - S(\beta))/S^{up}$, which ensures, that F_i always stays within the interval $[0, 1]$. Since all the squared errors are positive, another possibility would be to use the inverse of the squared error — i.e. $F_i = 1/S(\beta)$. Both types of fitness values may be used to derive the selection probability P_i in the following. Nevertheless, a ranking method defined in equation (2.2), will be the method of choice. The ranks and also the selection probabilities for a parameter value of $q = 0.3$ are listed in table 3.2.

After applying the selection operator we might get a population containing three copies of the two best strings and two copies of the fourth and the fifth string. Therefore, the mean squared error has become 23.6767 and the maximum squared error has changed to 58.9737. Obviously the minimum error remains, since selection cannot find better individuals than those already represented in the original population. Nevertheless, the average fitness of the whole population has improved significantly.

Since the parameter q must be chosen appropriately with respect to the population size, in the following it will always be denoted as a fraction $q = Q/ps$, ps denoting the size of the population. Then we can apply the same parameter values for Q for different population sizes.

Crossover

In order to study the differences among several possible ways of crossover, simple, arithmetic, and heuristic crossver where implemented. Since the length of the strings under consideration is very small, it does not make a big difference whether simple crossover (at a single crossover point), or crossover with a mask is applied. Therefore, crossover masks will be omitted in this example. To get a good balance between preserving existing individuals and testing new ones, the crossover probability p_c has to be chosen carefully. To observe the effects of p_c, the values $0.25, 0.4, 0.5, 0.8$, and 1.0 were tried.

Mutation

Since the function $S(\beta)$ is smooth and has only one local minimum, mutation is not crucial for the success of the search. For the sake of completeness, mutation probabilities $p_m = 0$ and $p_m = 0.001$ have been chosen.

Results

In total 123 simulation runs — either 40 or 50 generations — were done with the following variations:

population size	10, 20, 40
q (ranking selection)	$1.1/ps$, $1.2/ps$, $1.3/ps$, $1.5/ps$, $3.0/ps$
crossover type	simple, arithmetic, heuristic
crossover probability	0.25, 0.4, 0.5, 0.8, 1.0
mutation probability	0, 0.001

Table 3.3 shows the top ten solutions found within this 123 simulations.

β_0	β_1	β_2	$S(\beta)$	pop. size	q	crossover	p_c	p_m
1.0097	0.3071	0.5064	0.0653	40	1.1/ps	heuristic	0.4	0
1.0198	0.3058	0.5065	0.0662	40	1.2/ps	heuristic	0.4	0.001
1.0014	0.3118	0.5157	0.0663	20	3.0/ps	arithmetic	0.8	0.001
0.9963	0.3443	0.5049	0.0739	20	3.0/ps	arithmetic	1.0	0
1.0294	0.2547	0.5193	0.0769	40	1.2/ps	heuristic	0.4	0
1.0255	0.2910	0.4821	0.0787	40	1.2/ps	heuristic	0.4	0
0.9991	0.3563	0.4834	0.0794	40	1.3/ps	heuristic	0.4	0
0.9740	0.3613	0.4986	0.0796	40	1.1/ps	heuristic	0.4	0
1.0297	0.2547	0.5188	0.0814	40	1.2/ps	heuristic	0.4	0
1.0400	0.2700	0.5400	0.0955	20	3.0/ps	simple	0.5	0

Table 3.3: Best results

The best solution fits the (hyper-)plane better into the sample than the parameters that were actually chosen for generating the sample. Thus, there is no doubt that the accuracy of this solution is sufficient. Nevertheless, it is indeed questionable whether it makes sense to find those parameters by GAs when there exists a simple procedure to find an exact solution. Therefore, this example — and also the follwing — should rather be seen as an illustration of the power of GAs, and not as a typical practical application.

It is not surprising at all, that bigger populations in general result in better outcomes. Unfortunately, this is achieved at the cost of more memory and more computing time as well. Another conclusion that may be drawn is that mutation does not affect the qualitiy of the solution in this example. As already mentioned this is due to the smoothness of the function and the uniqueness of the optimum, which is the only local optimum within the whole

learning domain. Probably the most important result concerns the crossover operator. It seems that heuristic crossover is the most appropriate one for the given problem. Heuristic crossover assumes that given two different individuals in the search space a move along the connecting line extrapolating the edge of the more successful of the two results in even better solutions. This assumption obviously is fulfilled in the case of least squares estimation. However, this is not a typical problem for applying genetic algorithms. Therefore, the superiority of heuristic crossover cannot be seen as a general property of this operator.

3.2 Functions with multiple maxima

The function

$$y = sin(x_1\pi) + sin(x_2\pi) - \frac{x_1(x_1 - 5) + x_2(x_2 - 5)}{100} \tag{3.4}$$

has nine local maxima within the region $0 \leq x_1, x_2 \leq 5$ (see figure 3.2). These maxima are listed in table 3.4.

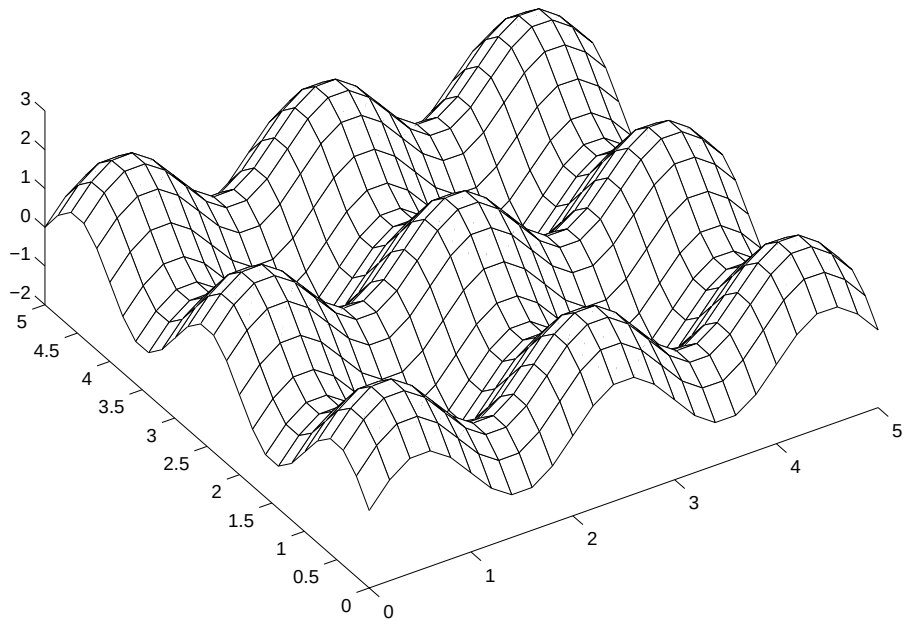


Figure 3.2: A function with 9 local maxima

The goal of this section is to test the GAs ability to find the global maximum even when there are several local maxima almost as good as the global one. Therefore, 3 initial random populations each containing 40 individuals

x_1	x_2	y
0.5	0.5	2.045
2.5	0.5	2.085
4.5	0.5	2.045
0.5	2.5	2.085
2.5	2.5	2.125
4.5	2.5	2.085
0.5	4.5	2.045
2.5	4.5	2.085
4.5	4.5	2.045

Table 3.4: Local extrema

were generated independently. Despite the fact, that this is a maximization problem (in contrast to the previous example), roulette wheel selection would still require a simple transformation to omit negative values. However, ranking selection can be implemented straight forward, and — as already mentioned — it reduces the amount of computations because the selection probabilities are only computed once at the beginning. Thus, again only ranking selection with four different parameters was examined.

Considering the results of the previous example, only arithmetic and heuristic crossover were applied. If heuristic crossover is done according to equation (2.4), then a random number $\alpha \in [0, 1]$ is used to define the stepsize. If the pair of strings taken to create an offspring have a big distance within the learning domain, this may lead to big jumps resulting in an individual far away from the profitable regions explored by its parents. To overcome this problem, the random numbers were generated from a uniform distribution over the interval $[0, \alpha_{max}]$ with $\alpha_{max} = 0.5, 0.8$, and 1.

Table 3.5 lists all the parameters that were used for running the GA.

parameter		values
size of population	ps	40
selection parameter	q	1.1/ps, 1.2/ps, 1.3/ps, 1.4/ps
type of crossover		arithmetic, heuristic
stepsize heuristic crossover	α_{max}	0.5, 0.8, 1.0
crossover probability	p_c	0.4, 0.6, 0.8, 1.0
mutation probability	p_m	0, 0.001, 0.002

Table 3.5: Simulation parameters

Each particular parameter set was applied three times for each of the three initial populations. In total this leads to 1728 independent simulations. Further, in each simulation 80 generations were created. The optimal solution $(x_1, x_2, y) = (2.5, 2.5, 2.125)$ was found 364 times (21.06 % of the simulations). Moreover, in 1515 cases (87.67 %) the best solution found was above $2.085+\epsilon$,

with ϵ being the smallest real number that can be seen to be greater than zero by the computer (actually ϵ was $2.2204e-016$). If a solution was found leading to a value $y > 2.085 + \epsilon$, this means there is at least one individual in the neighbourhood of the optimal solution, since the other local optimas' peaks lead to results of at most 2.085.

Within 80 generations the optimal solution was only found with heuristic crossover, but never with arithmetic crossover. With a stepsize $\alpha_{max} = 1$ and the selection parameter $q = 1.2/ps$ the optimum was found when $p_c = 1.0$ and $p_m = 0.001$ or $p_m = 0.002$. Under $\alpha_{max} = 1$ and $q = 1.3/ps$ or $q = 1.4/ps$ the optimum was always found. Further, when the stepsize α_{max} was chosen to be 0.5 or 0.8 then the optimum was found with all parameter settings listed in table 3.5. The best solutions found with arithmetic crossover are listed in table 3.6

x_1	x_2	y	q	p_c	p_m
2.5683	2.4763	2.1022	1.3/ps	0.8	0.002
2.5627	2.4343	2.0844	1.2/ps	0.8	0.001
2.5529	0.5040	2.0713	1.1/ps	0.4	0.002

Table 3.6: Best solutions found with arithmetic crossover

The iterations leading to the best two solutions in table 3.6 did not converge within 80 generations. Thus, with some more iterations perhaps also arithmetic crossover may have found the optimum in these two cases. The third best solution is in a neighbourhood of one of the wrong local maxima and, moreover the GA already converged — i.e. most of the individuals were concentrated at a small region around the local optimum. Thus, it is very unlikely that further generations would have improved the outcome significantly.

This motivates to guess that for the given problem heuristic crossover is the method of choice. The stepsize α_{max} should be small (i.e. significantly below 1.0), the selection parameter q should be between $1.3/ps$ and $1.4/ps$ (at least when the population size is 40), and the mutation probability should be significantly above 0.0.

Explanations:

- Arithmetic crossover fails because it is very unlikely that the global optimum is between the parents.
- Moving into the direction of the better individual helps to explore new and more successful regions. Heuristic crossover with a high crossover probability forces the GA to move towards better solutions (due to the smoothness of the examined function).
- The mutation operator prevents the GA from converging to a local optimum.

- Small stepsizes avoid movements too far away from the optimum.

Figure 3.3 shows four intermediate stages during the convergence of the GA towards the optimal solution. The chosen paramters are $\alpha_{max} = 0.8$, $q = 1.4/ps$, $p_c = 0.8$, and $p_m = 0.001$.

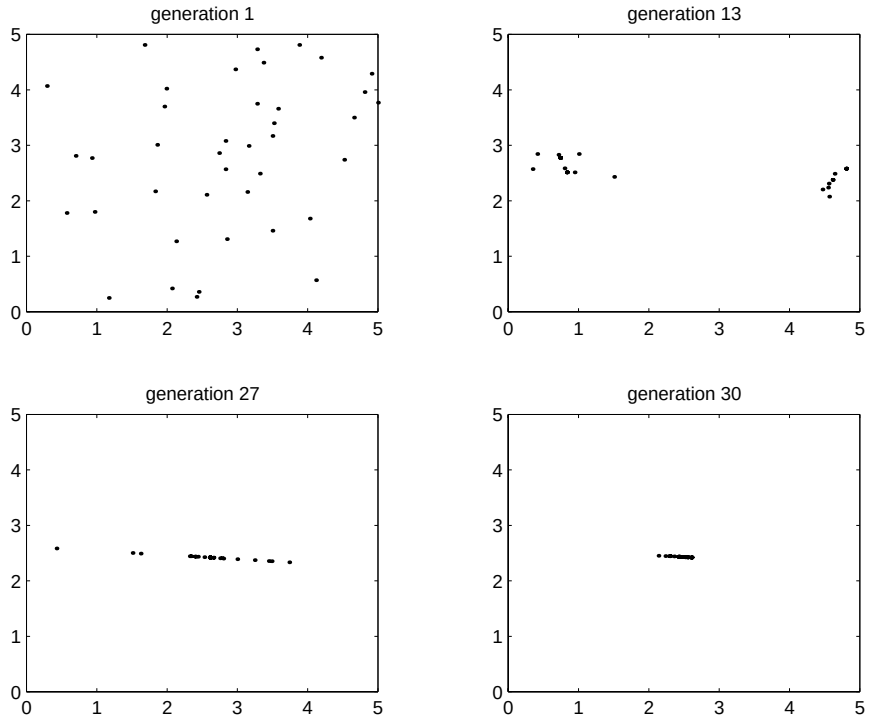


Figure 3.3: Convergence to the global optimum

The first graph shows the situation at the initial state. The individuals are distributed over the whole quadratic area $0 \leq x_1, x_2 \leq 5$. The second graph shows the situation of the 13th generation. The GA has already detected the two local optima $(0.5, 2.5)$ and $(4.5, 2.5)$. In the third graph (generation 27) many individuals have moved to the center of the quadratic area, due to a change caused by the mutation operator. Finally, the third graph shows generation 30, when all the individuals already moved into the neighbourhood of the global optimum at $(2.5, 2.5)$. The succeeding generations are needed to converge closer to the optimum.

Conclusion: The GA is able to find the best solution even when there are several local optima given a sufficiently big population spread over the whole learning domain, well adjusted genetic operators, and a reasonable mutation probability.

The following examples will illustrate the GAs capabilities to train classifier systems.

condition	action	
000	000	$(0 \to 0)$
001	010	$(1 \to 2)$
010	100	$(2 \to 4)$
011	110	$(3 \to 6)$
100	001	$(4 \to 1)$
101	011	$(5 \to 3)$
110	101	$(6 \to 5)$
111	111	$(7 \to 7)$

Figure 3.4: Rules performing the bit-shift operation

3.3 Training a simple classifier system

In this numerical example of a classifier system (CS) trained by a genetic algorithm (GA) the CS is required to perform the simple bit-shift operation $(a_1, a_2, a_3) \to (a_2, a_3, a_1)$ on the set $\{0, 1\}^3$. Since there are $2^3 = 8$ possible vectors, and each of them requires a unique response, the eight rules given in figure 3.4 are necessary for this bit-shift.

Because there are no opportunities for generalization, that means there is no possibility to set up one rule that covers several possible input scenarios, the don't care symbol # may be left out in this section. Thus, we could start with an initial population of classifiers containing only 0 and 1. Although there should not be any # symbols in the final CS which always produces the right outcome, it is nevertheless possible that inserting some # symbols into the initial population and also allowing them within intermediate iterations may help to train the CS.

Let us assume that the apportionment of credit assigns the number of correct digits contained in the chosen action to the rule that caused the output. This means for instance that the classifier 001 011 gets assigned a 2. Then the condition part of the classifier 11# 111 is fulfilled twice (inputs 110 and 111). Therefore, it has a higher chance to be chosen than classifiers without any #. Moreover, it gets assigned a 2 or a 3, which results in a very high fitness level. If the selection operator of the GA allows two reproductions in the mating pool, crossover or mutation may produce the two correct classifiers 110 101 and 111 111 during the following iterations. The simulation will show whether the learning domain $\{0, 1\}^6$ or $\{0, 1, \#\}^6$ is more successful.

Like in the previous examples, again the learning process is started with a random initial population. Here, each individual in the population is a string containing six components stemming from one of the alphabets given in the previous paragraph. Each rule gets assigned a one as initial fitness value. To evaluate these rules, each possible input is processed by the classifier system. When there are several rules matching an input, a roulette wheel selection mechanism is used to select a rule. The weights depend on the fitness, which

is equal at the beginning. Thus, if ps indicates the size of the population and f_i the fitness of the i-th rule, the selection probability of this rule becomes

$$p_i = \frac{f_j}{\sum_{i=1}^{ps} f_j} .$$

After selecting a rule its message is compared with the desired output (a_2, a_3, a_1) and the number of correct digits is stored in the variable cb (correct bits). Consequently, this value is used to update the fitness according to the formula

$$f_i(t) = \alpha \frac{cb}{disc} + (1 - \alpha) f_i(t - 1) . \tag{3.5}$$

The variables α and $disc$ are used to adjust the fluctuation of the fitness during the evaluation process. The fitness increases (decreases) when $cb/disc$ is greater (less) than $f_i(t-1)$ and it reaches an equilibrium when these expressions are equal. Thus, the parameter $disc$ can be used to determine which results lead to an increase of the fitness right from the beginning and which lead to a decrease. For example with an initial fitness of 1 and $1 < disc < 2$ the fitness of the rules with at least two correct bits in the action part increases and the fitness of rules with at most one correct digit decreases. Choosing $disc > 2$ leads to a decrease of the fitness even when two bits are correct. Thus, in the next cycle those rules that have never been tested are preferred compared to those that have been tested but only two bits are correct. Finally, the factor $alpha$ determines the speed of the change.

Before the genetic algorithm is invoked to update the rule base, the whole set of possible inputs is proceeded r_p times by the classifier system. In order to omit redundant numerical computations r_p should be chosen as small as possible. On the other hand, it is not desireable to produce results depending too much on random selections. Therefore, r_p has to be chosen sufficiently large such that the rules' fitness levels can converge towards their equilibrium level $f_i = \frac{cb}{disc}$. Since α influences the speed of the convergence significantly, it is important how many iterations are required in the worst case, when α is very small. In the numerical examples α-levels of 0.2, 0.4, and 0.6 were tried. Hence, $\alpha = 0.2$ becomes the worst-case scenario — with respect to the speed of the convergence towards the equilibrium. Table 3.7 shows how often a rule has to be chosen to get a fitness level of $1 + 0.95 \left(\frac{cb}{disc} - 1\right)$. That means the actual fitness has already performed 95% of the possible convergence. Close to the equilibrium the changes become extremly small such that a certain difference to the equlibrium must be tolerated.

Ignoring the zeros, which occur when the initial fitness equals the equilibrium, these results indicate that α is the only parameter influencing convergence velocity, while $disc$ and cb only affect the level of the equilibrium. This also holds true for other error tolerances. In a setup of ps rules not containing any don't care symbols and eight different input signals there are on average $ps/8$ rules handling the same input signal. Therefore, to avoid random

α	0.2				0.4				0.6			
$disc$	1	1.5	1.7	1.8	1	1.5	1.7	1.8	1	1.5	1.7	1.8
$cb = 0$	14	14	14	14	6	6	6	6	4	4	4	4
$cb = 1$	0	14	14	14	0	6	6	6	0	4	4	4
$cb = 2$	14	14	14	14	6	6	6	6	4	4	4	4
$cb = 3$	14	14	14	14	6	6	6	6	4	4	4	4

Table 3.7: Number of required iterations

results within the weights of the rules, r_p should be chosen approximately $ps/8$ times the value in table 3.7.

Now, there follow some remarks about the implemented genetic algorithm. Roulette wheel selection and ranking selection (with selection parameter $q = Q/ps$, see also section 3.1) are used. After selection simple crossover is applied with probability $p_c \cdot gfactor$. Finally, mutation is done with probability $p_m \cdot gfactor$. The expression $gfactor$ is used to accomodate the crossover and the mutation probability during the learning process. If G is the total number of generations and j the index of the current generation then $gfactor$ is computed according to

$$gfactor = \left(1 - \frac{j}{T}\right)^b$$

This feature enables the system to slow down the convergence at the beginning and avoid high fluctuations at the end of the learning process. A similar method is usually applied in simulated annealing. The exponent b is a shape parameter. Its effect is shown in figure (3.5).

In table 3.8 the considered numerical parameters and types of operators are listed.

Using several but not all possible combinations of these parameters and repeating each setting at least three times led to 8034 simulation runs. Within each generation the rate of correct bits in the outputs is computed — i.e. 1 is optimal. To avoid that sudden changes of the rule base due to crossover or mutation influence the comparison too much, the sum of these rates within the last ten generations is considered to compare the results. The best results obtained are given in table 3.9 — 10 being the optimal outcome. The population size is always 100 since the experiments with $ps = 40$ have not been successful. Moreover, inserting strings containing don't care symbols (#) did not deliver any acceptable outcomes. Thus, in this particular example generalization is useless since the task that has to be performed is so specific that handling several inputs with one rule is impossible. As a result, the learning domain under consideration is $\{0, 1\}^6$. Parameter settings that appear twice in the table indicate two different simulation runs with that parameter setting — i.e. these combinations of parameters are the most interesting ones since the good results do not depend too much on chance.

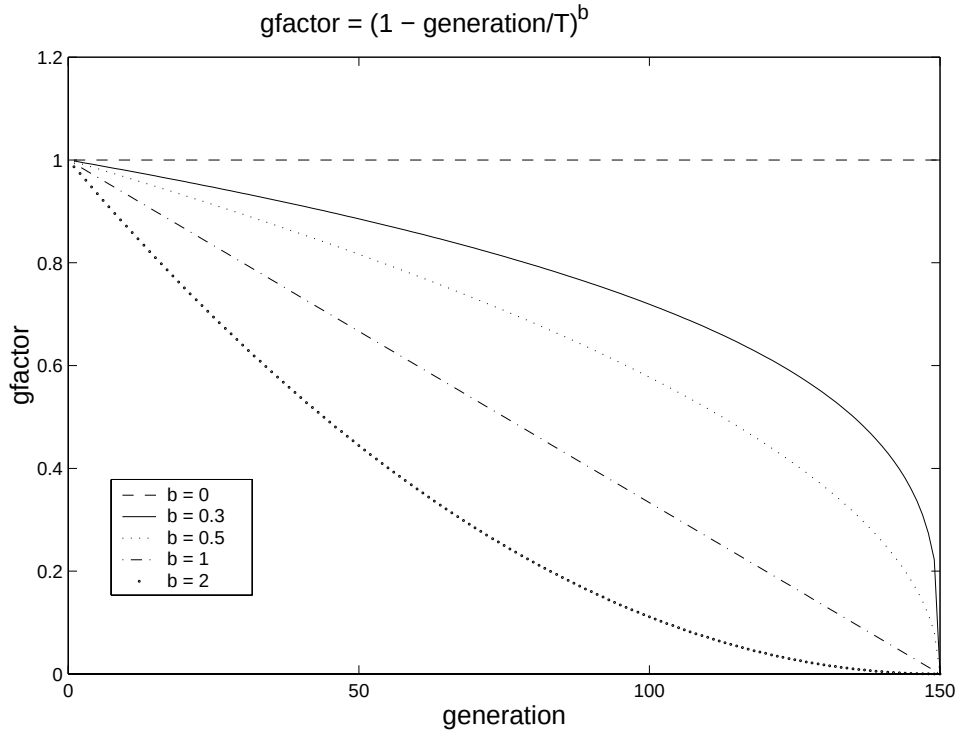


Figure 3.5: The generation factor

operator/parameter		values
don't care symbols	#	yes, no
selection operator		roulette wheel, ranking
size of population	ps	40, 100
number of generations	T	50, 100, 120, 150, 170
number of cycles	r_p	3, 5, 7, 12, 15, 20, 30, 35, 40, 80, 100, 120
fitness update factor	α	0.2, 0.4, 0.6
discount factor	$disc$	1, 1.5, 1.7, 1.8
selection parameter	Q	1.1, 1.2, 1.3
crossover probability	p_c	0.001, 0.002, 0.003, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.6
mutation probability	p_m	0, 0.001, 0.002, 0.003
shape parameter	b	0, 0.05, 0.3, 0.5, 0.7, 1, 1.5

Table 3.8: Simulation parameters

result	#	selection	T	r_p	α	$disc$	Q	p_c	p_m	b
10.000	no	ranking	150	100	0.2	1.7	1.3	0.1	0.003	1.5
10.000	no	ranking	150	100	0.2	1.7	1.3	0.2	0.002	1.5
10.000	no	ranking	150	100	0.2	1.8	1.3	0.1	0.001	1.5
10.000	no	ranking	150	100	0.2	1.8	1.3	0.2	0.003	1.5
10.000	no	ranking	150	100	0.4	1.7	1.3	0.1	0.002	1.5
10.000	no	ranking	150	100	0.4	1.8	1.3	0.2	0.001	1.5
10.000	no	ranking	150	100	0.6	1.8	1.3	0.1	0.003	1.5
10.000	no	ranking	150	120	0.2	1.8	1.3	0.2	0.001	1.5
10.000	no	ranking	150	120	0.4	1.7	1.3	0.1	0.001	1.5
10.000	no	ranking	150	120	0.4	1.8	1.2	0.1	0.001	1.5
10.000	no	ranking	150	120	0.4	1.8	1.2	0.1	0.001	1.5
10.000	no	ranking	150	120	0.4	1.8	1.3	0.2	0.001	1.5
10.000	no	ranking	150	120	0.6	1.7	1.2	0.1	0.002	1.0
10.000	no	ranking	150	120	0.6	1.7	1.2	0.1	0.002	1.5
10.000	no	ranking	150	120	0.6	1.7	1.2	0.1	0.003	1.5
10.000	no	ranking	150	120	0.6	1.7	1.3	0.2	0.002	1.5
10.000	no	ranking	150	120	0.6	1.8	1.2	0.2	0.002	1.5
10.000	no	ranking	150	120	0.6	1.8	1.3	0.2	0.001	1.5
9.9962	no	ranking	150	120	0.2	1.8	1.2	0.1	0.001	1.5
9.9958	no	ranking	150	100	0.4	1.7	1.2	0.2	0.002	1.0
9.9958	no	ranking	150	100	0.6	1.8	1.2	0.1	0.003	1.0

Table 3.9: Best results

From table 3.9 it follows that high values of r_p are extremely important to obtain good results. When r_p is too small, the CS is not capable to find exactly the desired rules to perform the bit–shift operation. Moreover, using a parameter value of $r_p = 120$ still improved the simulation results compared to those obtained with $r_p = 100$. The former appears in the list 12 times, while the latter appears 9 times. Presumably a further increase of r_p would do even better. However, this was not tested since the computing time becomes too long.

The shape parameter $b = 1.5$ clearly outperformed all other values that have been tested. Thus, the curve in figure 3.5 indicating the development of the generation factor should be convex. That means the reduction of the crossover and mutation probability is pretty high at the beginning and becomes more shallow at the end. This shows that only at the very beginning of the learning processes reasonable high mutation and crossover probabilities make sense. Later on, mutation and crossover tend to destroy the already found good solutions. The further improvement of the performance of the whole system is mostly due to the selection operator.

The factor α, responsible for the speed of the increase of the fitness may take any value. All the values that have been tested (0.2, 0.4, and 0.6) appear in the list with almost the same frequence. Those parameters that have not been discussed so far seem not to play a major role either.

The development of the rate of correct bits during the learning process in case of the parameters listed in the first row of table 3.9 is given in figure 3.6. It shows that beginning at generation 136 all the outputs are perfect. That means the CS trained by the genetic algorithm contains all and only those rules given in figure 3.4. Due to the decreasing crossover– and mutation–probability the correct CS was not destroyed within further iterations.

We may conclude that an artificial agent implemented with a CS and genetic operators is suitable to learn the required task. However, the successful parameter sets listed in table 3.9 require either $120000 (= 8 \cdot 150 \cdot 120)$ or even $144000 (= 8 \cdot 150 \cdot 120)$ function evaluations. Comparing that with $729 (= 3^6)$ function evaluations that would have been required for total enumeration, this method certainly is very unefficient. In the applications given in chapters 4 and 5 genetics based machine learning again is unefficient compared to full enumeration. However, in those models direct evaluation of the rules is not possible since the artificial agents lack information about the cost–functions (chapter 4) or about the consumers preferences (chapter 5). Therefore, only adaptive learning, for instance accomplished by CSs and GAs, can be taken into consideration.

3.4 Classifier systems with memory

The CS in the previous section was quite simple in the sense that it was completely memoryless. Its task was restricted to performing an input– output transformation. There are a lot of examples showing that this kind of CSs

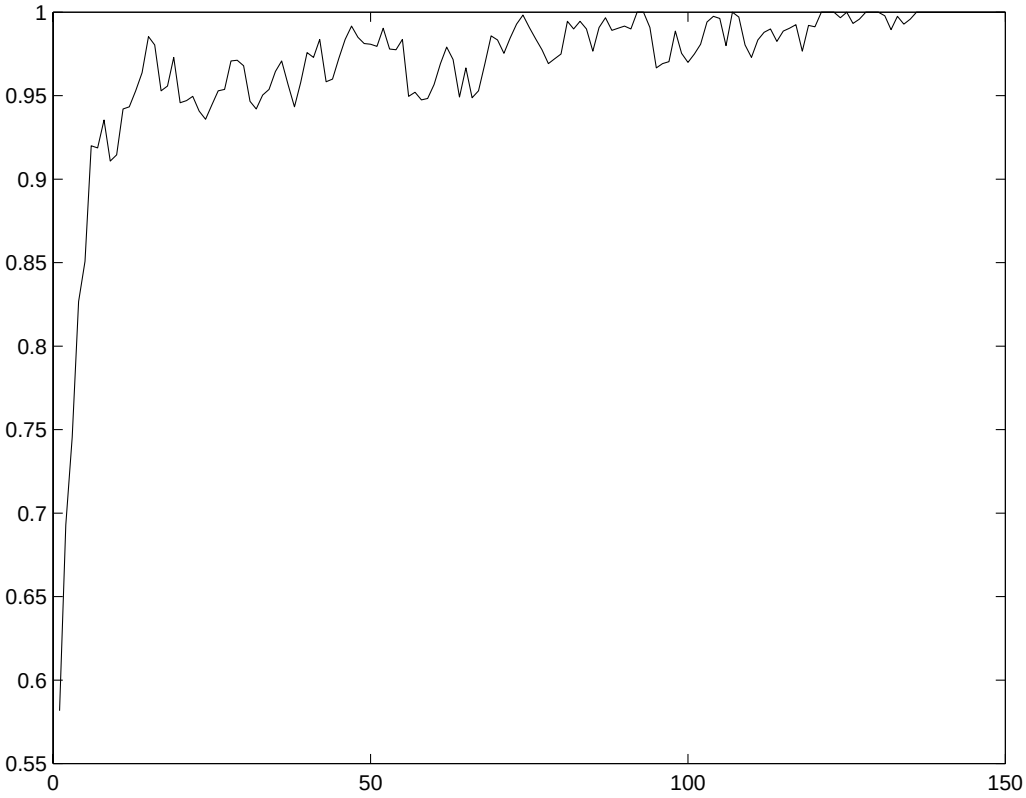


Figure 3.6: The learning process

indeed work without any troubles (see for instance Goldberg, 1989, p.218ff). In this section we will have a look at a more tricky problem. Here it will be examined whether a CS can learn to store an input $\vec{x} = (x_1, x_2, \ldots, x_n)$, and send it to the environment one time step later. This job is illustrated schematically in figure 3.7.

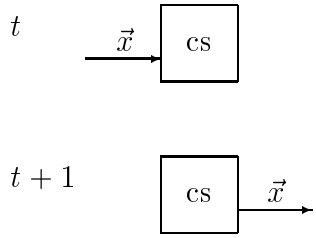


Figure 3.7: A classifier system with memory

In order to do that, it is necessary to implement internal messages. To determine, whether a string is an internal message, an additional bit is required. In the following any string with a zero at the last position will be considered to be a message coming from outside or an action to be posted to the environment. On the other hand, strings ending with a 1 are considered to be internal messages. To simplify the notation the abbreviations $(\vec{x}, 0) := (x_1, x_2, \ldots, x_n, 0)$ and $(\vec{x}, 1) := (x_1, x_2, \ldots, x_n, 1)$ will be used. The whole process of storing an input signal $\vec{x}$ for one time unit and then posting it is illustrated in figure 3.8.

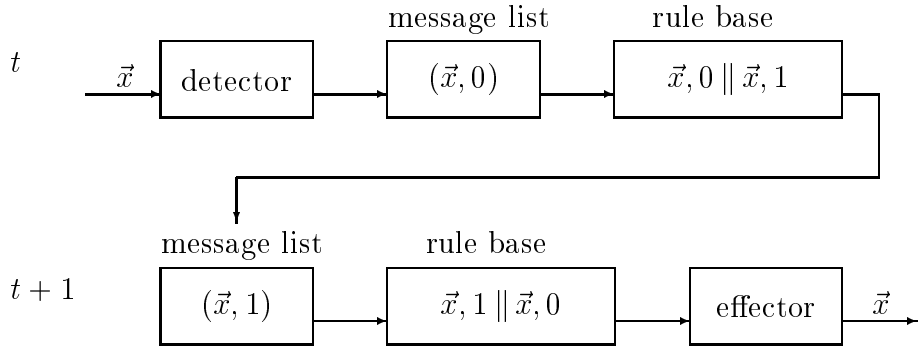


Figure 3.8: Storing a message for one time step

At time t the vector $\vec{x}$ enters the system via the detector which just adds a 0 at the end to enable the rule base to proceed with the input. The input $(\vec{x}, 0)$ is stored in the message list and compared with the condition parts of all rules. Given there is a rule $(\vec{x}, 0 \parallel \vec{x}, 1)$, and it is chosen among all fulfilled rules, an internal message $(\vec{x}, 1)$ is produced and written into the message list. Next, at time t this internal message might be chosen to activate the

rule $(\vec{x}, 1 \,\|\, \vec{x}, 0)$ which, in turn, sends the string $(\vec{x}, 0)$ to the effector. Finally, the effector removes the last bit and sends the signal $\vec{x}$ to the environment.

However, this job can only be performed in the described way, when the needed pair of coupled rules is available. Moreover, another pair of coupled rules, for instance $(\vec{x}, 0 \,\|\, \vec{y}, 1)$ and $(\vec{y}, 1 \,\|\, \vec{x}, 0)$ can do the same. Therefore, both pairs might get assigned a high fitness during the learning process (performed for instance by a genetic algorithm). Since both pairs do the right job, this does not disturb at all. What really causes big problems, is the fact that each single rule out of these two pairs may also be involved in a chain producing wrong results. A few examples of possible combinations are listed in table 3.10.

$\vec{x}, 0 \,\|\, \vec{x}, 1$	$\vec{x}, 1 \,\|\, \vec{y}, 0$
$\vec{x}, 0 \,\|\, \vec{y}, 1$	$\vec{y}, 1 \,\|\, \vec{y}, 0$
$\vec{y}, 0 \,\|\, \vec{x}, 1$	$\vec{x}, 1 \,\|\, \vec{x}, 0$
$\vec{y}, 0 \,\|\, \vec{y}, 1$	$\vec{y}, 1 \,\|\, \vec{x}, 0$

Table 3.10: Coupled rules leading to wrong outcomes

This results in assigning low fitness values to the involved rules when the above chains are activated during learning. Hence, it is unpredictable whether a rule required to perform the job of storing a message will be successful and survive several iterations. As a result, the considerations done in the previous example about convergence of a rule's fitness towards an equilibrium cannot be applied here. The aim of this setction is just to briefly sketch the problems arising when classifier systems are required to perform a time delay between input and output, i.e. when memory is required. Since the economic models in chapters 4 and 5 apply CSs without memory, this is not dicussed in more detail, and simulation results are omited.

Chapter 4

A model of new product development

Managing the information flow within a big organization is a challenging task. Moreover, in a distributed decision-making process conflicting objectives occur. In this chapter, artificial adaptive agents are used to simulate the process of designing a new product. None of the agents knows all the relevant details. Efficient communication within a network of decision makers is required to perform a complex task, which no one could do alone. The decision makers are implemented as classifier systems, and their learning process is simulated by genetic algorithms. To validate the outcomes we compared the results with the optimal solutions obtained by full enumeration. It turned out, that the genetic algorithm indeed was able to generate useful rules that describe how the decision makers involved in new product development should react to the requests they are required to fulfill.

4.1 Introduction

Designing new products is often a process that involves several departments of a company. Moreover, the people dealing with all aspects of launching a new product have varying educational backgrounds and, therefore, do not speak the same technical language. In some organizations it can indeed be a big challenge to manage the communication required to capture all aspects of product development. However, nowadays decreasing product life-cycles require an efficient communication in order to launch new products before the competitors do it. An efficient information-flow is an indispensable prerequisite for a high speed of innovation. In order to systematically produce innovative solutions a company has to perform four tasks:

- collect good ideas
- keep them alive
- find new applications

- test promising concepts

A collection of various case studies can be found in Hargadon and Sutton (2000). A well-known technique for interfunctional planning and communications is the *house of quality* introduced by (Hauser and Clausing, 1988).

"The foundation of the house of quality is the belief that products should be designed to reflect customers' desires and tastes" (Hauser and Clausing, 1988). A brief summary of the model can be found in (Chase and Aquilano, 1995). The main idea is a conceptual map that helps two departments to collect all the data that express the influence of the decisions of one group on the problem of the other group. The information flow works like it is shown in figure 4.1.

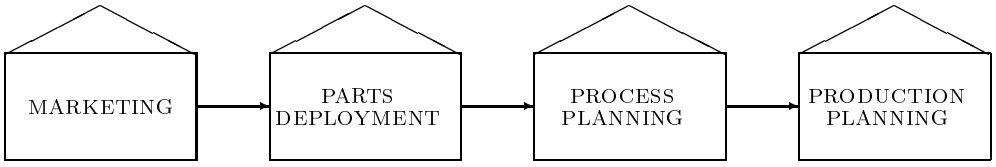


Figure 4.1: Flow of information

In the first step the purpose of the house of quality is translating customer attributes into engineering characteristics. Subsequently, the engineering characteristics are transformed into parts characteristics, which are then used to generate the key process operations, which, in the last step, are the basis of the production requirements. In all these steps data have to be communicated between different departments. The House of Quality approach helps them to talk together and is one of the most well-known instruments of quality function deployment. The main question addressed is: "How do we understand the quality that our customers expect and make it happen in a dynamic way?" Quality function deployment is a systematic way to make sure that customer requirements drive the design process. The first house is shown in detail in figure 4.2.

At the beginning of the process it is required to listen carefully to the customers to learn as much as possible about their requirements. These attributes together with their relative importances are then written into the left room of the house. Afterwards the room on the right-hand side is used to illustrate how customers experience the own product and the competitors products in terms of the attributes listed on the left-hand side. In the next step one has to think about the engineering characteristics (usually these are technical data that can be measured easily) that have some influence on the customer attributes. These technical features are written down in the room below the roof. Then the room in the center allows to mark the relations between engineering characteristics and customer attributes. There are different types of notations that can be used. Most of them have in common that they indicate whether there is a weak or a strong connection, and if the

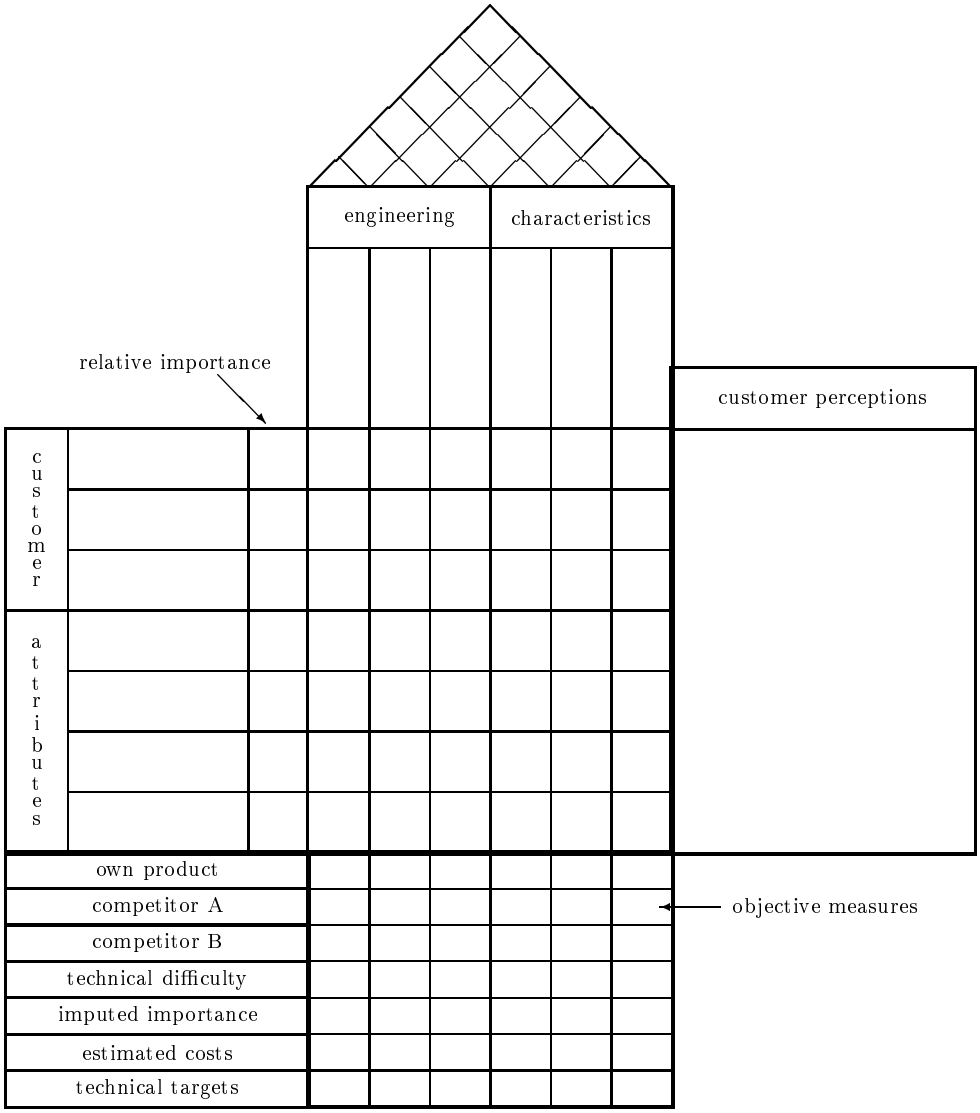


Figure 4.2: The house of quality

influence is positive or negative. When there is no relation, the according field in the matrix is left empty. However, there are not only correlations between engineering characteristics and customer attributes, but also among the engineering characteristics. These connections are marked at the roof of the house. Finally, the basis of the house is used to collect data about the measures of the engineering characteristics — again we compare the own product with the competitors products — the technical difficulty, the imputed importance, the estimated costs, and the technical targets.

Marengo (1992) studied an organization which has to forecast the customers' demands and to coordinate the production process at the same time. While the former is easier when the organization can rely on a diversified knowledge base, the latter depends on a common body of knowledge. Simulating this process with classifier systems showed, that in a static environment a centralized structure is sligthly advantageous compared to a decentralized one, while in a periodically changing environment a centralized structure completely fails. Another simulation model related to our work is given by DeCanio et al. (2000). They examine organizational structures that differ in their connectivity. The best performing information structures under several settings of cost parameters are identified in comprehensive numerical experiments.

In this chapter we will deal with the interaction within an organization and, therefore, the company itself is interpreted as a complex adaptive system. The groups involved in product development are the adaptive agents. Not all the influences they are confronted with are internal, and the customer's demand might change quickly. All the agents have to take their own decisions and, certainly there is interaction among them. Thus, all of the characteristics of a complex adaptive system described in section 1.2 arise.

The remainder is organized as follows. Section 4.2 explains the model and the algorithms we used. In section 4.3 we present and interpret the simulation results obtained with various parameter settings. Finally, in section 4.4 we provide a summary of some possible extensions.

4.2 The model

The well-known *house of quality* is a management tool that helps to illustrate the influence of abstract technical features on the customers' perception of the product. Although the practical value of such a concept is out of question, it is not obvious how such a communication scheme can be integrated into a numerical simulation of an artificial firm. For our attempt to simulate the behaviour of the individuals involved in the product design process we translate the basic ideas of the *house of quality* to a quantitative level. Hence, all the relevant information that is exchanged between the particular departments is supposed to be encoded in a string of fixed length containing only the digits $0, 1, 2, 3$, and 4.

4.2.1 The customer attributes

The customer attributes are the inputs of the first house. If we run our simulation with strings of length 3, there are 125 (5^3) possible input vectors. To initialize the customers' wishes we create all the 125 input strings, and select 20 of them for the validation set. The remaining strings are used to train the system while the validation data are stored for later comparisons between different simulation runs.

Obviously in case of a real product a vector of length 3 will never be suitable of carrying all the characteristics required to describe the product. However, since the algorithms we used do not take advantage of the structure of the functions that have to be optimized, increasing the size of the vector only increases computing time without any qualitative changes of the obtained results.

4.2.2 The decision making process

The messages created in the above section enter the first house — the marketing department, where the customer attributes are transformed to engineering characteristics. In our simulation this is done by a classifier system. The output — the engineering characteristics — is the input of the next house — the parts deployment. Yet, another classifier system uses this information to decide about the parts characteristics. In the same way the third house — process planning — transforms parts characteristics into key process operations, and, finally, these are transformed into production requirements by the department of production planning.

For an introductory description of classifier systems see (Holland, 1995, p. 13ff) or (Holland, 1976). Here, we use a very simple classifier system with only one condition part per rule and each message entering the classifier system alone. An application of more complex classifier systems containing more condition parts will be shown in chapter 5. A typical situation of our model is illustrated in figure 4.3, where '#' is the don't care symbol.

message	rules	actions
2 4 1	# 4 1	0 2 4
	3 4 #	2 3 1
	1 3 0	4 1 3
	2 4 1	1 0 2
	# 0 1	3 4 0

Figure 4.3: A typical decision situation

In this particular situation the first and the fourth rule are fulfilled by the incoming message. Thus, we choose one of them randomly. However, in most situations not all the fulfilled rules have the same chance to be chosen.

We use a weight which depends on the strength gained in the past and on the specivity of the rules. The specivity depends on the number of don't care symbols appearing in the considered rule. Thus, in this example the fourth rule is more strict than the first one.

Later, the selected rule posts its action part which, in turn, is the input of the following house, where a similar decision making process is applied again. Note, that each of the departments has it's own independent set of rules.

A description of more general classifier systems is given in (Geyer-Schulz, 1995), while (Holland, 1995) provides a survey of recent applications.

4.2.3 Creating new rules

If a message enters a house which does not fulfill any of the available rules, then a new rule has to be created. First, we compare the incoming message with each rule. At those indices, where the rule entry does not match with the message we compute the difference and derive the sum of all the differences for each rule. Finally, we choose one of those rules with the lowest sum of differences. The action part and the strength (see section 4.2.4) of this rule are inherited by the new rule. The incoming message itself is used as the new condition part.

4.2.4 Costs and fitnesses

Each decision taken by any department is assumed to cause two different types of costs. The first cost term represents those costs that originate from implementing the chosen decision while, on the other hand, there are some opportunity costs that arise from the difference between the actually chosen output and the theoretical optimal output with respect to the input.

Implementation costs

The costs of implementation depend on the numerical values of each particular component of the output vector plus a term that represents the correlation or synergy among the components. For instance, if the department chooses the row vector $\vec{x}$, then with certain cost parameters the implementation costs may become

$$c_1 = \vec{x}\ \vec{b} + \vec{x}\ \mathbf{B}\ \vec{x}' \ . \tag{4.1}$$

The first term $(\vec{x}\ \vec{b})$ in formula (4.1) represents the costs that stem from the row "estimated costs" in the foundation of the house (figure 4.2). The term $\vec{x}\,\mathbf{B}\,\vec{x}'$ stands for the technical correlation among the components which are noted in the roof of the house. Therefore, it makes sense to expect that in the main diagonal and below it there are only zeros. The other entries may be negative, if there are synergies that reduce the costs, as long as c_1 remains

nonnegative. A sufficient but not neccesary condition to fulfill $c_1 \geq 0$ can be given by the inequality

$$B_{ij} \geq -\frac{b_i}{4\ (n-1)} \quad \forall i, j. \tag{4.2}$$

Proof 1 *From equation (4.1) it follows that*

$$c_1 = \sum_{i=1}^{n} x_i \left(b_i + \sum_{j=1}^{n} B_{ij} x_j \right) .$$

All the entries along the main diagonal (i.e. all x_{ij} satisfying $i = j$) are equal to zero, and we assumed $x_i \in \{0, 1, 2, 3, 4\}$. Thus, the condition given in (4.2) guarantees that all the terms $x_i \left(b_i + \sum_{j=1}^{n} B_{ij} x_j\right)$ are nonnegative. □

If the designers decide to split the product into rather independent moduls there will be only a few entries in the matrix $\mathbf{B}$. A highly integrated design, on the other hand, leads to more nonzero entries in $\mathbf{B}$. For a comprehensive elaboration of modular product design and its impact on the organizational structure see Göppert and Steinbrecher (2000).

Opportunity costs

We assume that there might exist an affine transformation that assigns an optimal output to each input,

$$x^* = A\ x^{input} + a\,. \tag{4.3}$$

That means x^* is the best reply to the input x^{input} from a pure technical point of view — not regarding any considerations about the costs and difficulties arising when this technically optimal decision is being implemented. This input – output relation $(x^{input} \to x^*)$ is determined by the entries in the center of the house. In order to obtain a feasible solution we have to check if each component is in the intervall $[0, 4]$ and reduce or increase those components that are too big or too small. Finally, the distance between the chosen output and the optimal output determines the opportunity costs,

$$c_2 = \beta \|x^* - x\|\,. \tag{4.4}$$

The parameter β is fixed and it is used to adjust the influence of implementation and opportunity costs. Thus, we are able to control the relative importance of c_1 and c_2. Note, that in our model an overshoot is considered to be as bad as a shortfall. In most cases this might obviously make sense. However, if we think about excessive precision this statement is not so clear. In such a situation the costs might become too high and, therefore, it is feasible to assign high opportunity costs to such a solution.

Total costs and fitness

The total costs are the sum of implementation costs and opportunity costs,

$$c = c_1 + c_2 \ .$$

These costs are used to compute the fitness of the chosen rule. Higher costs lead to a lower fitness and vice versa.

Local optimization

In an organization consisting of locally optimizing agents only those costs arising from the particular department are considered for evaluating the fitness. For simplicity we want to achieve fitness values in the interval $[0, 1]$. Therefore, we first compute the highest total costs possible,

$$\bar{c} = \vec{x}\ \vec{b} + \vec{x}\ \hat{\mathbf{B}}\ \vec{x}' + \beta \cdot ||\vec{x}||,$$

with $\vec{x} = (4, 4, 4)'$, and $\hat{\mathbf{B}}$ containing the absolute values of the entries of $\mathbf{B}$. The parameter β determines the weight of the opportunity costs. Then we obtain the fitness

$$f = \frac{\bar{c} - c}{\bar{c}} \ . \tag{4.5}$$

Global optimization

If the agents' aim is to minimize the company's total costs, then also the impacts of their decision on the other departments have to be taken into account. If we denote the highest possible cost in the i-th house with $\bar{c}^{(i)}$, and the costs occuring in the i-th department with $c^{(i)}$, then the fitness value of the rule recently chosen in the j-th department becomes[1]

$$f = \frac{\sum_{i=j}^{4} \bar{c}^{(i)} \cdot f_b^{i-j} - \sum_{i=j}^{4} c^{(i)} \cdot f_b^{i-j}}{\sum_{i=j}^{4} \bar{c}^{(i)} \cdot f_b^{i-j}} \ . \tag{4.6}$$

The only difference to formula (4.5) is that we consider a weighted sum of all the costs in the recent house plus all the following houses. The factors f_b in the above formula are used to adjust the trade-off which has to be made between local and global optimization. A selfish, suboptimizing decision maker will choose $f_b = 0$, while a decision maker only interested in the success of the company as a whole will choose $f_b = 1$. Nevertheless, not all the costs occuring in subsequent steps are caused by the decision at step i. Thus, we expect that choosing a level of f_b somewhere between 0 and 1 will yield the best results.

[1] In formula (4.6) $c^{(i)}$ and $\bar{c}^{(i)}$ express costs or boundaries for costs in house i, while f_b^{i-j} means that the weight f_b is taken to the power $(i - j)$.

Strength - apportionment of credit

While training the classifier system, all the fitnesses gained by a rule during a particular iteration are cumulated. Thus, rules which are chosen more frequently also gain a higher strength. After processing all the inputs of the training set through the system, the strength is updated according to

$$strength = (1 - \alpha)\ oldstrength + \alpha\ newstrength\ . \tag{4.7}$$

This is done several times. In our simulation we defined the variable r_p, determining how often the whole set of data has to be processed through the system within one generation.

4.2.5 The genetic algorithm - rule discovery system

After r_p iterations a genetic algorithm is used to adapt the rules in the classifier system. More details about training a classifier system using genetic algorithms can be found in chapters 2 and 3, and for instance in (Goldberg, 1989). Our aim is to find rules that enable the departments to find a "good" response to every input.

Selection

In order to select those rules that we use to generate the next population we rank all the rules with respect to their strength received in the past. Then we choose the best 50%. Thus, the successful rules of the last period always occur again in the next period. However, at the beginning we start with a very small population. Thus, during the start we choose all the rules in order to increase the number of individuals. A small example of the selection process is given in figure 4.4.

The rules that are selected cannot be changed by the following crossover operation but only by the mutation.

Crossover

First we build pairs with the selected individuals (rules and actions). Then a crossover mask is generated to create two new individuals. Finally, we assign the average fitness as starting fitness value to the new rules. The crossover probability χ is 1, i.e. the crossover process always takes place. The example in figure 4.5 illustrates the crossover operator, when it is applied to the first and the third rule out of the pool of selected rules (recall figure 4.4).

Mutation

The last operator in our genetic algorithm is the mutation. The aim of this operator is to avoid converging towards a local minimum. Therefore, we first create mutation masks containing only zeros and ones. The probability of

population

rules	actions	strength
# 4 1	0 2 4	0.75
3 4 #	2 3 1	0.67
1 3 0	4 1 3	0.84
2 4 1	1 0 2	0.34
# 0 1	3 4 0	0.82

selecting the 3 fittest rules

rules	actions	strength
# 4 1	0 2 4	0.75
1 3 0	4 1 3	0.84
# 0 1	3 4 0	0.82

Figure 4.4: Selecting the rules

crossover mask

1 1 0 1 0 1

new individuals built with # 1 and # 3

rules	actions	strength
# 4 1	0 4 4	0.785
# 0 1	3 2 0	0.785

Figure 4.5: The crossover operator

ones is the mutation probability μ, which is between 0.001 and 0.004 in our simulation. Then we create the random strings containing only $0, 1, 2, 3$, and 4. Finally, we replace the old contents of our population with the random numbers with probability μ, as it is shown in figure 4.6.

mutation mask

0 0 0	0 0 1
0 0 0	0 0 0
0 0 0	0 0 0
0 1 0	0 0 0
0 0 0	0 0 0

new population

rules	actions
# 4 1	0 2 2
1 3 0	4 1 3
# 0 1	3 4 0
# 4 1	0 4 4
# 0 1	3 2 0

Figure 4.6: The mutation operator

Afterwards the second phase of the mutation operator takes place. If we'd finish our genetic search as described until here, as a result of the mutation operator the number of don't care symbols would decrease from one generation to the next one. To avoid this we apply a special mutation only to write new don't care symbols into the existing rules. This works exactly like the general mutation described above. Another effect of having an extra mutation operator only for don't care symbols lies in the fact that this enables us to control the specifity of our rules. The parameter determining the probability of writing a don't care symbol into the population is called d_p. In our experiments we tried values between 0 and 0.003.

4.2.6 Brief model summary

Summarizing our model is based on the following assumptions:

- All the relevant information is encoded in vectors $\vec{x} \in \{0, 1, 2, 3, 4\}^3$.
- The agents take one unique decision at each time step in response to a unique and known input signal.
- The costs are a polynomial function of order 2 of the input and output vector.

- The agents do not know the cost functions but receive unbiased information about the recent costs.
- In response to this information the agents may update their rule–base, i.e. we assume that the examined firm is a learning organization.

4.3 Simulation Results

The objective of the departments is to find replies that yield low costs. As a first approach the fitness of a rule is assumed to depend only on those costs that arise in that particular department. That means, all the departments try to find a strategy that minimizes their own local costs without considering the indirect effects of their decision. However, it would also be interesting what happens if the fitness assigned to a certain rule is also influenced by the costs in the subsequent departments. The numerical parameters used for these simulations are listed in table 4.1. In total there are 432 ($3 \cdot 3 \cdot 3 \cdot 4 \cdot 4$) possible combinations that we use for our experiments.

variable	description	values
α	strength update factor	0.1, 0.2, 0.3
r_p	CS cycles before invoking GA	1, 2, 3
$despop$	number of classifiers (population size)	36, 72, 108
μ	mutation probability	0.001, 0.002, 0.003, 0.004
d_p	probability of inserting a #	0, 0.001, 0.002, 0.003

Table 4.1: Simulation parameters

The generations from 1 to 40 are used to train the classifier system in all the four houses. Therefore, the 105 input messages of the training set are used. From generation 41 to 55 we use the remaining 20 messages to check if the classifiers are capable to deal with new inputs which they did not experience during the training period. If we observe the development of the average costs in each department over time when the decision makers only get information about their own costs, the time series may look like in figure 4.7. The values at the abscissas indicate the generations. The vertical lines at generation 40 mark the change from training data to validation data. The according set of simulation parameters is given in the lower right corner of the figure.

The graph in the first row on the left-hand side shows the situation in the first house. The dashed line shows the number of don't care symbols within the rule base. This number increases dramatically at the beginning of the learning process and quickly stabilizes at approximately 150. That means at the beginning the more general rules are more likely to be chosen, and so their frequency within the population increases. Since the population consists of 72 classifiers, each of them on average contains about two don't care symbols. That means in the first step of the process — when the data

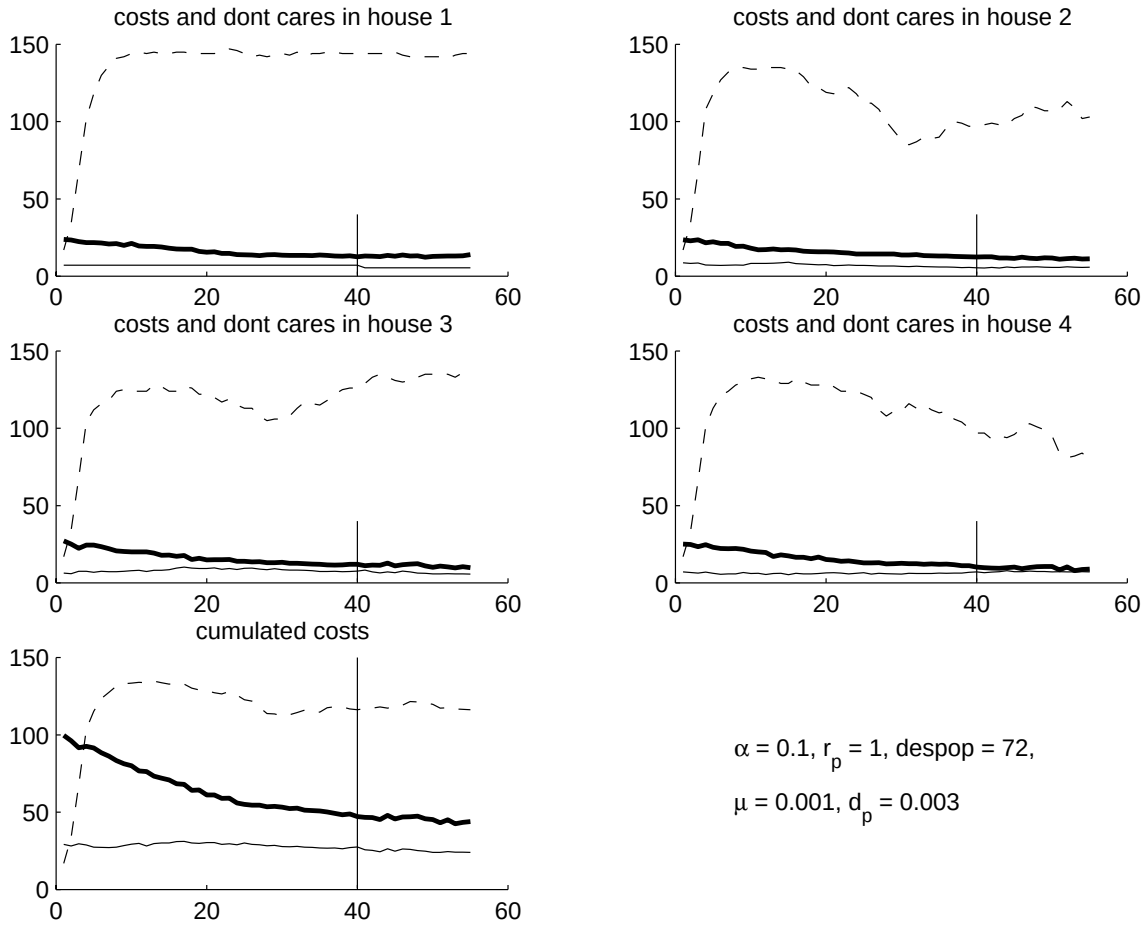


Figure 4.7: Product development - simulation results

from the marketing department are used for parts deployment — very general rules are applied. The thick solid line shows the actual costs. Beginning at a level of 23.8 they decrease until 12.5 at generation 40 (end of the training period). Now the inputs of the validation set must be handled by classifiers trained with the training set. This results in a modest increase of the costs to 13.1. During the following generations the system adopts to the new inputs, which results in a decline of the costs to 12.3. Since the changes within the last 15 generations are very modest, we can conclude, that the system gained during the first 40 generations is general enough to be able to handle new inputs. The thin solid line shows the minimum costs that can be achieved with the given input and cost structure. Minimum means we assume the decision makers know the cost function and can choose their response such that the minimum costs arise. Those costs were computed with full enumeration. The cost structure is the same for all 60 generations, while the input is modified once, between generation 40 and 41. This results in a small step in the graph between generation 40 and 41.

The graph in the first row on the right-hand side shows the same for the second house. The inputs for the second house are the outputs of the first house. Therefore, in contrast to the situation described in the previous paragraph, the inputs change during the learning period as the rule base in the first house changes. As a result, the number of don't care symbols is more fluctuating than before. The number of possible inputs is less or equal to the number of rules in the first house. Therefore, within each generation at most 72 different inputs are possible. Thus, the second house is confronted with a smaller variety of inputs, which results in a smaller frequency of don't care symbols, i.e. the rules are more specific. The change from generation 40 to generation 41 is extremely small now, because the signals delivered by the first house do not change completely when the set of input data is changed.

The second row in figure 4.7 shows the same for the third and fourth house. The results are similar to those in the second house. Finally, the picture in the third row shows the cumulated actual and minimum costs of all the four houses together. The number of don't care symbols was divided by four before plotting. As a result, the dashed line indicates the average number of don't care symbols in the four houses.

If we look at figure 4.7, we can see easily, that the algorithm is indeed capable of finding better rules in terms of cost reduction. The minimum costs of the first house do not change, because the input remains the same. On the other hand, the minimum costs of the other three houses change, because they depend on the output of the previous department as suggested in figure 4.1. In some situations the minimum costs even increase as time elapses due to the suboptimizing behaviour of the preceding departments.

The reason why the cost reduction in the first house is smaller than in the other houses lies in the fact that the first house has to find a policy for 105 different inputs with a classifier system containing only 72 different rules. With this classifier system it can only produce 72 different outputs, which

facilitates the others' decision problems. To analyze this effect in more detail we ran the simulation with population sizes of 36, 72, and 108 individual rules in each classifier system. Unfortunately doubling the population size also means doubling the required computing time.

Certainly, it would be possible to implement a classifier system which always finds the optimal solution by increasing the population size until one rule for each possible input is available. However, real-world individuals must be able to react in situations never encountered before. Therefore, the adaptive agents in the simulation are required to find responses to many different inputs with a limited set of rules. In turn, the purpose of the classifier systems is finding a generalization among the encountered inputs, and establishing a simple policy with satisfying results.

If we analyse the behaviour after the training period — i.e. from generation 41 to 55 — we find out, that this requirement is fulfilled. The departments get completely new inputs, and the costs remain reasonable small, moreover, only moderate adaptions are made in the sequel. To judge the success of different sets of parameters we compare the cumulated costs during the validation period. The best ten parameter settings in terms of low costs are listed in table 4.2.

cumulated costs	α	r_p	$despop$	μ	d_p
492.1223	0.3	1	36	0.001	0.002
538.7303	0.1	3	36	0.003	0.002
548.9550	0.1	1	72	0.001	0.002
549.6529	0.2	1	36	0.003	0.001
554.4548	0.1	1	36	0.001	0.001
560.4413	0.2	1	36	0.001	0.003
561.8771	0.1	2	36	0.001	0.002
568.5375	0.2	3	36	0.002	0
571.3693	0.2	1	36	0.004	0.003
572.6556	0.1	1	36	0.003	0.003

Table 4.2: Cumulated costs during validation

At the first view it seems surprising that classifier systems with only 36 rules outperform setups with 72 or 108 rules. Maybe the bigger populations would require more iterations (higher r_p) before invoking the genetic algorithms, and more generations. However, the ranking in table 4.2 does not consider how well a system learns the training data, but only how well it can get along with the new data during validation. Therefore, the smaller systems, which are forced to generalize while learning the training data, lead to better results when new inputs arise.

By comparing the different results it turns out, that a high level of the strength update factor α (in particular in combination with a high rate of repeats r_p) results in very fissured cost curves (see figure 4.8). Thus, the

genetic algorithm does not always move to lower costs as time elapses. The same holds for high mutation probabilities μ and d_p.

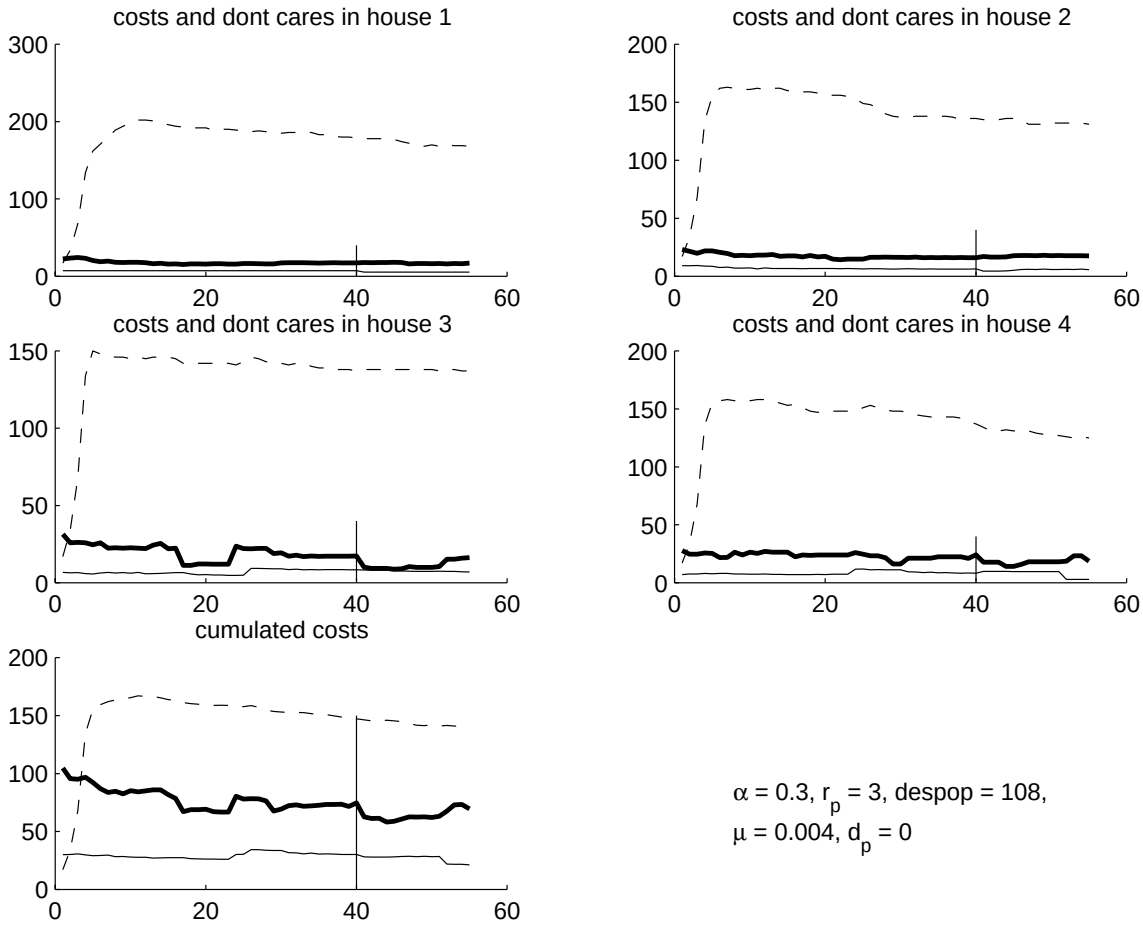


Figure 4.8: Product development - simulation results

Table 4.2 lists successful sets of parameters. Now let's have a look at the single parameters. In table 4.3 for each parameter value the minimum and the maximum cumulated costs during the validation period are listed. Moreover, the ratio $\frac{maximum}{minimum}$ and the average cumulated costs are listed.

When the max/min–ratio is high it means that the results in this row are not predictable when only the value of the parameter under consideration is known. In such a situation the other parameters or chance have a high influence on the result. This is the case when $\alpha = 0.3$ (ration = 2.78), $despop = 36$ (2.78), and $\mu = 0.001$ (2.75). A very small max/min–ratio, on the other hand, appears when $\alpha = 0.1$ (1.64), $r_p = 1$ (1.8), and $r_p = 2$ (1.89).

Yet, predictability alone has no value. What is also of great importance is the level of the costs. The lowest minimum costs appear in the rows $\alpha = 0.3$, $r_p = 1$, $despop = 36$, $\mu = 0.001$, and $d_p = 0.002$. The highest maximum

parameter	value	minimum	maximum	ratio	average
α	0.1	538.7	884.2	1.64	703.3
	0.2	549.7	1120.7	2.04	747.0
	0.3	492.1	1366.0	2.78	852.7
r_p	1	492.1	884.2	1.80	715.9
	2	561.9	1061.8	1.89	742.1
	3	538.7	1366.0	2.54	845.0
$despop$	36	492.1	1366.0	2.78	732.8
	72	549.0	1258.5	2.29	775.0
	108	644.1	1353.5	2.10	795.2
μ	0.001	492.1	1353.5	2.75	751.3
	0.002	568.5	1366.0	2.40	765.0
	0.003	538.7	1304.8	2.42	784.2
	0.004	571.4	1180.7	2.07	770.2
d_p	0.000	568.5	1366.0	2.40	761.5
	0.001	549.7	1353.5	2.46	770.1
	0.002	492.1	1258.5	2.56	756.1
	0.003	560.4	1355.0	2.42	782.9

Table 4.3: Minimum costs, maximum costs, max/min–ration, and average costs for each parameter value

costs appear when $\alpha = 0.3$, $r_p = 3$, $despop = 36$, $\mu = 0.002$, and $d_p = 0.000$. Favourable are those numerical values that lead to low minimum costs and small max/min–ratios. Such combinations can be found in the rows for $\alpha = 0.1$ and $r_p = 1$.

Another insight deliverd by table 4.3 is the change of the cumulated costs when the value of a parameter is modified. If we look at the minimum costs in the first three rows we see there is a modest increase from $\alpha = 0.1$ to $\alpha = 0.1$ and a sudden cost reduction when moving to $\alpha = 0.3$. Looking at the maximum costs shows that they increase monotonically. Thus, it is difficult to estimate which values are optimal. The following three rows show a tendency to higher costs when r_p increases. Again this tendency is not completely clear when we look at the minimum costs, which slightly decrease from $r_p = 2$ to $r_p = 3$. For the other three parameters there is no clear tendency either. What we can see immediately is that for $r_p = 1$ and for $d_p = 0.002$ the minimum and the maximum costs are the lowest within their category.

4.4 Possible extensions

As already indicated we would like to observe what happens if the information set is enlarged or reduced. In economic terms this can be interpreted as a change in the firm's organization structure, or a change of the internal

incentive system. The incentive is represented by the weigths f_b in formula (4.6). When $f_b = 0$, the agents only consider their local costs. Thus, they decide themselves to discard the information about the other departments. When $f_b \in (0, 1)$, the costs in the following departments is also considered for the optimization, but with a smaller weigth. For example $f_b = 0.5$ means that the costs in the next department get assigned 0.5 times the weight of the own costs, the costs in the next but one department get assigned 0.25 times the weight, and so on. Finally, $f_b = 1$ means that all the costs have the same weight. Thus, modifying the incentive scheme by changing the weigths f_b influences the information set considered by the agents. The performance of such different agents can then be observed in different environments. For instance, the inputs can be constant, changing periodically, or changing randomly. Since the model analysed in chapter 5 operates with a more complex classifier system, such an analysis is left out here, but will be done in chapter 5. Moreover, it would be interesting to observe the changes of the reactions to one particular input message and the related costs. Another interesting aspect could be to compare different learning algorithms or just have a look at what happens when some of the genetic operators are added, changed, or removed. Another possible extension would be to create a central classifier system responsible for all the decisions in the whole company.

Chapter 5

A model of product placement

To launch a new product the marketing department has to decide about the kind of customer attributes they would like to meet. In a heterogenous market with different customers' tastes and several competitors taking the decision about product placement is rather complex.

Customers usually choose that product which best fits their desire, as indicated for instance by Kotler et al. (1996, p. 7) *"They therefore want to choose products that provide the most satisfaction for their money."* To make sure that one particular customer buys, a producer could decide to customize his/her offer according to the wishes of that customer. However, this might lead to a product that no one else would like to buy. Certainly this is not a very favourable situation for a supplier, except in case this one customer has such a great purchasing power that indeed designing a product for one particular individual can still yield a good profit. Some practical examples for this situation are custom-made suits, paper-making machines, or power stations.

To avoid dependence on one customer the producer could decide to place her/his product such that the distance to most customers' requirement profiles is as small as possible. Again, this might not always be the optimal strategy. If all the competitors already try to launch such a mass product that represents the average of all the customers' wishes, then offering another average product might not lead to great success. Thus, for deciding what kind of product to supply one has to be aware of the customers' desires and the competitors' products as well. In the following we will elaborate on a simulation model featuring many customers with equal purchasing power.

Similar investigations were done by Polani and Uthmann (1999). They provide a distributed simulation environment that allows to insert different types of customer and firm agents. The products may not be substituted for each other. The customers have a demand for a certain bundle of goods and buy them for the lowest price offered by the firm agents. The firms aim to maximize their profit — i.e. the difference between turnover and costs. The customers, on the other hand, gain a profit from the difference between the maximum price they are willing to pay and the actual market price.

However, in this chapter we will analyze how adaptive agents, who use classifier systems to take the product placement decision and learn by using genetic algorithms, would place their products in a dynamic and heterogenous market. Therefore, we assume there might be m customers and n suppliers in a market of goods that may be substituted for each other[1]. The products are assumed to have only two different attributes (this assumption is made to facilitate visualization) and each attribute can take 10 different values. Each firm is allowed to offer one particular product and each customer buys one product. Thus, the consumers and the vendors both have 100 alternatives. A typical situation with $m = n = 5$ (i.e. there are 5 customers who can choose one out of 5 different products) is illustrated in figure 5.1, where the x^i-symbols denote the customers and the y^j-symbols the suppliers.

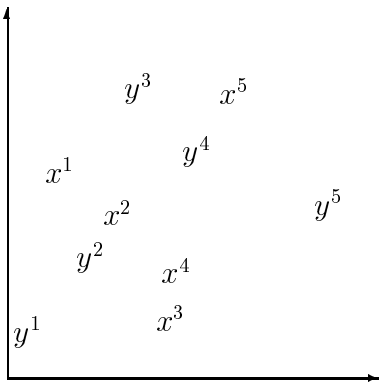


Figure 5.1: A typical market situation

5.1 A simple model

5.1.1 The market mechanism

After all the customers have declared their wishes and all the suppliers have made their offers, the customers choose those products with the smallest euclidian distance between the ideal product and the actual offers. In case of two or more products with the same distance one of them is chosen randomly. If we denote $\vec{x^i}$ the $i-$th customers wish and $\vec{y^j}$ the $j-$th vendors offer, then the decision a^i of the $i-$th customer might be

$$a^i = \arg \min_{j \in \{1 \ldots n\}} \left\{ \|\vec{x^i} - \vec{y^j}\|_2 \right\}. \tag{5.1}$$

To keep the model simple we make the following assumptions:

1. All the customers have the same purchasing power.

[1] In most markest it holds that $m \gg n$.

2. All the producers have the same internal cost structur.

3. Whenever a product is chosen the vendor receives a fixed profit p.

4. Each product in the set $\{1, \ldots, 100\}^2$ causes the same costs.

5. All the producers are capable to manufacture each possible product in $\{1, \ldots, 100\}^2$ in any quantity.

Thus, π^j the profit of supplier j within one particular time period equals p times the number of customers who decided to buy the product offered by the supplier j, i.e. $\pi^j = p \cdot \#\{i|a^i = j\}$. In the example in figure 5.1 consumers x_1, x_2, x_3, and x_4 would buy the product offered by firm y_2 and consumer x_5 would buy the product of firm y_4. As a result firm y_2 makes a profit of $4\,p$, firm y_4 gets p, and the other firms get no profit.

5.1.2 The buying agents

At the initial state of our simulation the preferences of the customers are placed randomly somewhere in the twodimensional set. In this simple model we assume that the customers' wishes do not change as time elapses. Later on, in section 5.3 we will also observe markets where the customers' tastes change.

5.1.3 The selling agents

Like before we use a uniform random distribution to define the initial offers of all the suppliers, and collect the data in the $2 \times n$ matrix

$$S = \begin{pmatrix} y_1^1 & y_1^2 & \cdots & y_1^n \\ y_2^1 & y_2^2 & \cdots & y_2^n \end{pmatrix}.$$

We will observe two classes of agents at the supply side of our artificial market.

First class agents

The first class of selling agents are using classifier systems as described in chapter 2. We use classifier systems with a condition part containing at least three and at most five conditions. In the following we use the variable n_{cond} to refer to the number of conditions. The incoming messages are the data about the offered products of all the sellers in the last period plus an additional gene which contains π^j, the recent success of the offers. Thus, M_t the list of incoming messages at time t becomes

$$M_t = \begin{pmatrix} y_{1,t-1}^1 & y_{2,t-1}^1 & \pi_{t-1}^1 \\ \vdots & \vdots & \vdots \\ y_{1,t-1}^n & y_{2,t-1}^n & \pi_{t-1}^n \end{pmatrix}. \tag{5.2}$$

Certainly it does not make sense to make a decision based on the information about only one competitor. Therefore, in this model the rules contain several conditions and, hence, only those rules are fulfilled that are activated by several suppliers.

The genetic algorithm

Selection

The selection operator has a very high influence on the dynamics of the population. It is used to determine which individuals' offsprings may occur in the next generation, and which get discarded. In the present model we use a ranking procedure. First, the rules have to be ordered according to their fitness values. Then, those rules belonging to the best 80% are selected, and the others discarded. Finally, those rules belonging to the best 20% are written into the list a second time. This increases the chance of the very successful rules to remain in the rule base of the next time step.

Crossover

After selecting the rules we produce offsprings by either copying the rows of the present rule-base into the new one, or by combining two rules. First we build pairs of rules randomly. After that with a probability of $\chi = 0.5$ we create new rules by combining the strings, otherwise both strings remain.

Mutation

At the beginning of the iteration process it is very important to avoid striving to a local optimum. Therefore, a mutation operator is used to place random numbers somewhere into the population. This happens with a probability of μ, which we assigned the values 0, 0.001, and 0.002. In order to control the strictivity of the rule-base we use another mutation operator, which only writes don't care symbols (#) into the condition part of the rules. This is done with a probability of d_p.

Second class agents

The second class of selling agents are the simpler ones. They just make small random movements discarding any available information about the market. The purpose of these agents is just to find out if the agents using classifier systems are indeed capable to find intelligent strategies, i.e. to outperform the second class agents.

If we have n^1, the number of first class agents, and $n^2 = n - n^1$, the number of second class agents, then the $2 \times n^2$ matrix ΔS^2 with all its components uniformly distributed on $\{-1, 0, 1\}$ determines the movements of the second class agents. The decisions of the agents in class i are collected in the $2 \times n^i$

matrix S^i, which leads us to

$$S_t = \left(S_t^1, S_t^2\right) = \left(S_t^1, S_{t-1}^2 + \Delta S_t^2\right). \tag{5.3}$$

This in turn is the transpose of the first two columns of the matrix M in equation (5.2) for the next time step $t + 1$.

5.2 Simulation results

Typical time series resulting from the previously described simulation are shown in figure 5.2. We use setups with $m = 50$ customers, $n_1 = 5$ first class selling agents, and $n_2 = 5$ second class selling agents. Assuming $p = 1$, i.e. each sale is worth one monetary unit, the total sales in the market add to $P = m\,p = 50$.

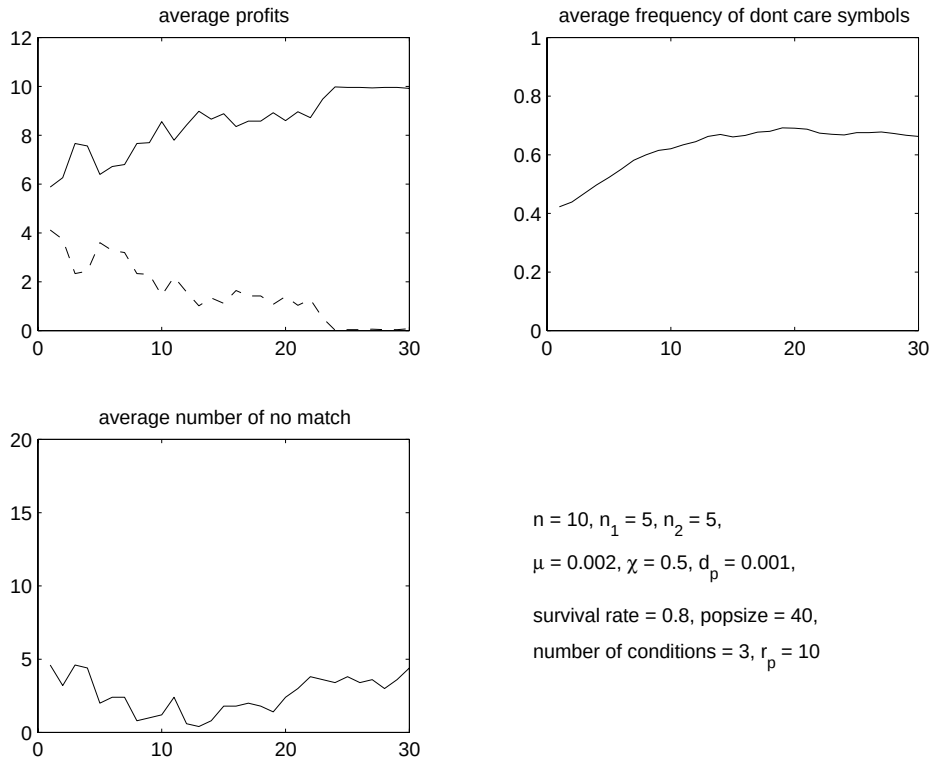


Figure 5.2: Simulation results

In the graph in the first row on the left-hand side the average profits of the first class agents are plotted as a solid line and the average profits of the second class agents are indicated by a dashed line. Since the total profit available in the market is 50, an average profit over 5 agents of 10 monetary units means that those 5 agents get all the sales. At the beginning of the simulation the market shares differ only slightly, but in the sequel the share of the intelligent agents increases. Finally, after 24 generations, the

first class agents occupy the whole market, and their average profit becomes $P/n_1 = 10$. This shows that the learning process induced by the genetic algorithm succeeded in producing useful rules. Moreover, we can conclude, that a decision need not to be based on all the information available. In the present simulation illustrated in figure 5.2 the classifier systems contain 3 conditions. Thus, the decisions taken by the first class agents are based only on information about 3 different suppliers.

On the right-hand side we see the frequency of don't care symbols (#) in the classifer systems of the first class agents. At the initial state only about 40% of the entries in the condition parts are #, but later on their share increases significantly until it stabilized slightly above 60 %. Thus, we can conclude that the very general rules are the more successful ones, and thus survive the selection process.

The graph in the second row shows how often it happend that some of the first class agents had no rule fulfilled. In that case their decision had to be taken randomly. This curve always remains between 0 and 5. Since $r_p = 10$, the classifier systems are activated 10 times within one generation. Thus, an average number of randomly taken decisions of at most 5 means that in the worst case every second decision was taken randomly.

In total we ran 648 experiments. In table 5.1 all those parameters are listed, whose value was fixed in all the simulations.

χ	crossover probability	0.5
m	number customers	50
n	number of suppliers	10
n_1	number of 1^{st} class agents	5
n_2	number of 2^{nd} class agents	5
P	cumulated profit	50
p	profit obtained when selling to one customer	1
s_{rate}	survival rate used by the selection operator	0.8
T	number of generations	30

Table 5.1: Fixed parameters

In table 5.2 all those parameters are listed, which got assigned different values. In some of the experiments the general mutation operator also could produce #, thus leading to the expressions $f(\mu)$ in the row describing the parameter d_p in table 5.2.

In all the experiments the intelligent agents (i.e. those using classifier systems rather than random walk) outperformed their competitors. To compare the tested parameter settings quantitatively we derive the profit gained by the first class agents during the last 10 generations. Thus, 500 is the best result that can be achieved. In table 5.3 all those sets of parameters are listed, that actually led to a profit of 500 for the intelligent agents from generation 21 to generation 30.

p_{size}	size of the populations of rules in the CSs	40, 80, 160
n_{cond}	number of conditions in the CSs	3, 4, 5
r_p	number of repetitions before invoking the GA	5, 10, 20
α	factor of fitness updates	0.1, 0.2
μ	mutation probability	0, 0.001, 0.002
d_p	probability of # used by the mutation operator	$0, 0.001, f(\mu)$, $0.001 + f(\mu)$

Table 5.2: Variable parameters

p_{size}	n_{cond}	r_p	α	μ	d_p
40	3	20	0.2	0.001	$0.001 + f(\mu)$
40	4	10	0.2	0.000	$0.000 + f(\mu)$
40	4	20	0.1	0.000	$0.000 + f(\mu)$
40	4	20	0.2	0.001	$0.001 + f(\mu)$
80	4	20	0.2	0.000	$0.000 + f(\mu)$
80	4	20	0.2	0.002	$0.001 + f(\mu)$
80	5	10	0.2	0.001	$0.001 + f(\mu)$
80	5	20	0.1	0.001	$0.000 + f(\mu)$
160	4	20	0.2	0.002	$0.000 + f(\mu)$
40	3	20	0.2	0.000	0.000
40	4	20	0.1	0.000	0.001
40	4	20	0.2	0.001	0.000
40	5	20	0.1	0.002	0.001
80	3	20	0.1	0.000	0.001
80	3	20	0.2	0.000	0.000
80	5	20	0.1	0.001	0.001
160	3	20	0.2	0.001	0.001
160	3	20	0.2	0.002	0.001
160	5	10	0.2	0.002	0.001
160	5	20	0.1	0.000	0.001
160	5	20	0.2	0.001	0.000

Table 5.3: Best performing parameter settings

From table 5.3 it follows, that choosing $r_p = 5$, i.e. iterating the classifier systems five times between two calls of the genetic algorithm, failes completely and $r_p = 20$ performs significantly better than $r_p = 10$. Thus, we can conlude that the classifier systems need about 20 or more iterations to produce stable deterministic results. This corresponds perfectly with the findings in section 3.3.

While, $r_p = 10$ occurs in the list only in combination with $n_{cond} = 4$ or $n_{cond} = 5$, $r_p = 20$ also occurs in combination with $n_{cond} = 3$. Since n_{cond} is the number of conditions, it determines how many strings containing information about one particular supplier can be taken into consideration by one individual classifier. That means classifier systems that carry out more iterations (r_p), i.e. they follow a more deterministic search, require less information to produce successful decisions. From a practical point of view this means that managers who are better in interpreting information about their own and their competitors sales need less information to take the right decisions. This is certainly a very trivial statement, but is shows that the simulation results of this model are consistent with common expectations.

Looking at the parameters p_{size}, α, μ, and d_p shows that all the tested values are more or less equally successful. Consequently, no meaningful interpretations with respect to these parameters are possible.

5.3 An extended model

In this section we examine the importance of information about the market and the exploration of this information under different environmental conditions. This is achieved by inserting two new types of selling agents into the simulation framework described in section 5.1. The first class and second class agents are the same like before.

The first new type is some kind of superior agent. Like the first class agents, these superior suppliers also use classifier systems to take their decisions. However, the input is not the information about the own and the competitors' sales, but instead they know exactly the positions of the customers. Although they do not know the recent desires of the customers but only their desires of one time step before this may be a reasonable competitive advantage. The term superior refers to the direct observation of their customers which is in contrast to the indirect information of the first class agents.

Another additional type of agent included in this extended model applies an extremely simple strategy. It observes the sales of all the competitors in the previous period — i.e. the input is the same as for the first class agents — and then it moves to the position of the most successful agent. In economic terms this means that this supplier never tries any new products but always imitates products that have sold well a short time ago. Such a behaviour can for instance be observed in the textile industry. A few weeks after the fashion shows where the designers present their new collections some

very similar products already can be found in the shelfes of the cheap retail chains. Another example of such an imitating strategy is the car industry. While some decades ago each brand and perhaps even each type had its own and very particular body, nowadays many cars of different brands look very similar.

The second extension of this marketing model refers to the behaviour of the customers. While in the simple model in section 5.1 the demands of the customers are assumed to be static, here we will investigate three different scenarios.

1. In the first scenario the customers' wishes are initialized with random numbers and then remain constant forever.

2. In the second scenario there are two completely different sets of customers' demands that are computed initially before the learning process starts. Then, the environment switches periodically between these two states. This change is carried out at the end of each cycle of the classifier system. Thus, the environment changes r_p times within one generation of the genetic algorithm.

3. Finally, the third scenario is based on a random walk. Again the demands of the customers are initialized with random numbers. Then, their movement follows a random walk with small steps. Both components may be increased or decreased by one. Thus, the demand side of the market is not completely static, but changes slightly. If the changes were too big, it would not make sense for the suppliers to build decision rules based on their experience.

   In mathematical terms we can say that the demand side is represented by a $2 \times m$ matrix

   $$D = \begin{pmatrix} x_1^1 & x_1^2 & \dots & x_1^m \\ x_2^1 & x_2^2 & \dots & x_2^m \end{pmatrix}.$$

   The initial state D_0 is a $2 \times m$ matrix with all the components uniformly distributed on the set $\{0, \dots, 100\}$. The state D_t at any time t can be given as

   $$D_t = D_{t-1} + \Delta D_t, \tag{5.4}$$

   where ΔD_t is a $2 \times m$ matrix with all its components uniformly distributed on the set $\{-1, 0, 1\}$. Hence, regions that are very profitable at time $t-1$ can become unfavourable within one time increment. This random walk is also done for every cycle of the classifier system.

To keep the simulation small we include $n_0 = 3$ superior agents, $n_1 = 3$ first class agents, $n_2 = 3$ second class agents, and $n_3 = 1$ imitating agent. All the simulation parameters that were fixed in section 5.2 remain in this version. Moreover, based on the findings of the previous section we also fix

$p_{size} = 40$, $n_{cond} = 4$, $r_p = 30$, and $\alpha = 0.2$. The only variable parameters are the mutation probability and the probability of inserting a don't care symbol. We use $\mu, d_p = 0, 0.0005$, and 0.001. For simplicity the same parameters are applied for the superior agents and for the first class agents. Since there are $3 \cdot 3 = 9$ different parameter sets and 3 different scenarios we get 27 different variants. To exclude random results each of these variants is run 5 times. This leads to 135 independent simulations. An example for the time series obtained from such a simulation in case of a static environment and $\mu = d_p = 0$ is given in figure 5.3.

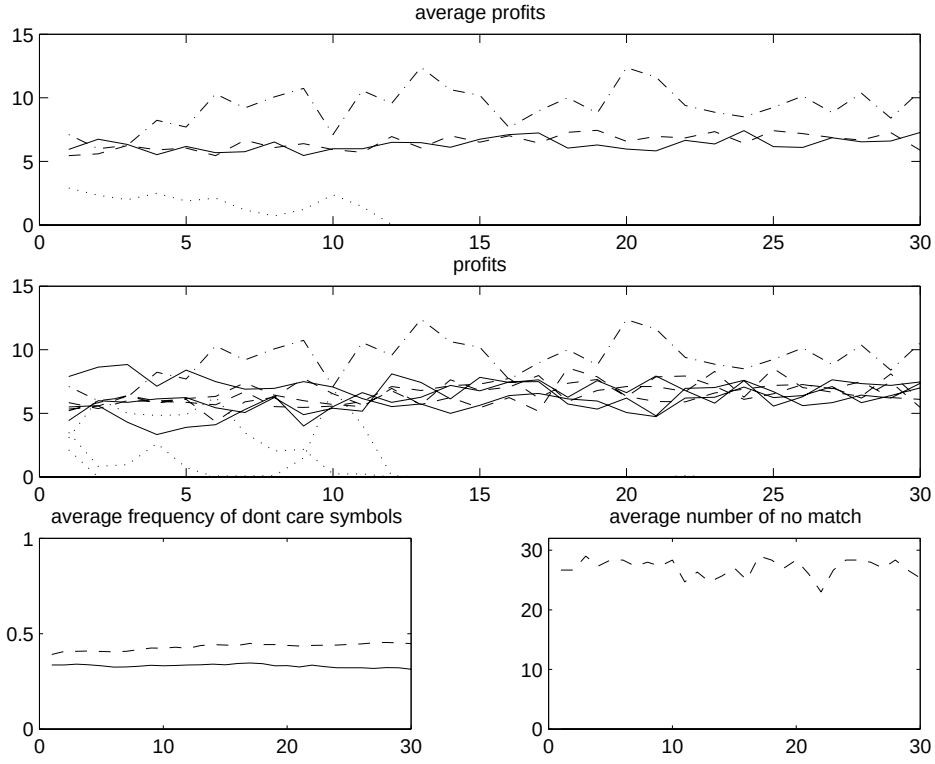


Figure 5.3: Simulation results of the extended model

The graph in the first row shows the average profits earned by the four different groups of marketers. Here and in the remainder of this section a solid line always indicates the superior agents, a dashed line the first class agents, a dash-dotted line the imitating agent, and the dotted line the second class agents. In case of the imitating agent this is actually not an average profit but a total profit because there is only one supplier of that type.

Like in the simple model the market share of the second class agents vanishes quickly. The market share of the superior agents and the first class agents is approximately the same. Thus, in this setting it does not make a big difference whether the wishes of the buyers can be observed directly. Although it is an advantage to know the demands of the customers it does not lead to better decisions when the classifier system is limited to rules containing only

four conditions. The first class agents use information about the suppliers and their sales. The position of a very successful supplier allows conclusions about the positions of several customers sice high sales only happen when there are many customers in the neighbourhood. The superior agents, on the other hand, base their decision on information about four customers. Increasing the number of conditions would presumably improve their competitiveness.

The most remarkable difference to the previous model can be found when looking at the profit of the imitating agent, which is significantly higher than the competitors' profits. However, if there were several agents applying this strategy, they would be at the same position at each time step. As a result, they would have to share their sales and the average profit would be lower than the average profits of the agents using classifier systems. Moreover, later it will be shown that the imitating agent is not that competitive in every environment.

The graph in the second row illustrates the profits of each particular agent. It turns out that the imitating agent does not only sell better than the average learning agent but it even outperforms all its competitors. In this particular simulation beginning at generation 6 the sales of the imitating agent are always greater or equal than the sales of each individual competitor.

The graph on the rigth-hand side of the third row shows how often it happened that no classifier had all its conditions fulfilled. In such a situtation the selling agent concerned places its product at a random position. Again the solid line represents the superior agents and the dashed line the first class agents. The solid line cannot be seen because it is always zero. The dashed line moves around a level of 25. The explanation why these curves differ so much is pretty simple. Both types of agents — superior and first class — have the same number of classifiers and conditions in their rule-base. While the inputs of the superior agents are the positions of all the 50 customers, the inputs of the first class agents are the positions and the sales of the 10 suppliers. A bigger amount of inputs certainly increases the chance that at least one classifier has all its conditions fulfilled. Nevertheless, in some simulations the superior agents also had to apply random decisions. This is illustrated in figure 5.4 showing the results obtained in a static environment with $\mu = 0$ and $d_p = 0.001$. Anyway, the average number of no match for the superior agents was always much lower than for the first class agents.

In the graph on the left-hand side of the third row the frequencies of the don't care symbols are plotted. The solid line represents the classifiers of the superior agents and the dashed line the first class agents. It turns out that the first class agents' rule bases contain more don't care symbols, which means they apply more general rules than the superior agents. This again is due to the smaller supply of input messages. This result was obtained in all the simulations.

For getting a more general view of the results figure 5.5 shows the average profits of the four types of agents in the three different environmental scenarios. Now average does not only mean an average over all the agents of

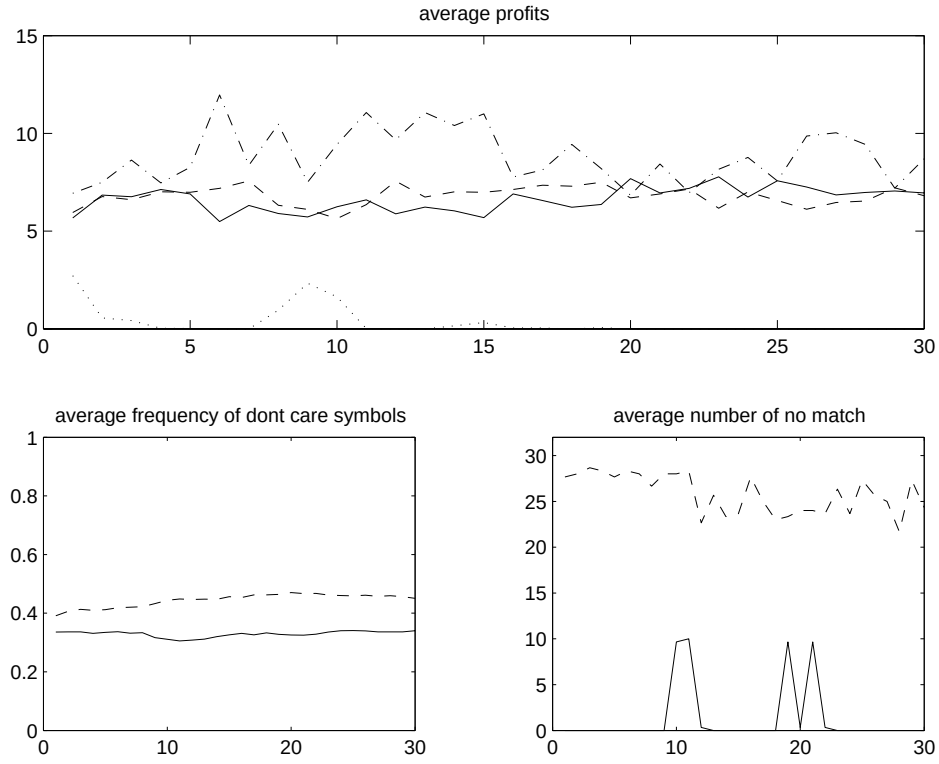


Figure 5.4: Simulation results of the extended model

the same type but also an average over all simulations of the same scenario. Therefore, the curves become smoother.

In the static scenario the imitating agent performs best. The suprerior and first class agents make a lower profit but are both at approximately the same level. The second class agents make almos no profit.

If the demands change periodically, the sales of the imitating agent fall below the sales of the superior and the first class agents. This is due to the sudden change of the demands in every single time step. The imitating agent always moves to a position that was excellent in the previous period and will be excellent again in the following period. Moreover, the superior agents now perform better than the first class agents. Thus, in this environment direct information has a higher value.

Finally, when the customers follow a random walk, the competitiveness is inverted. Both types of learning agents loose a big part of their market share and they even fall behind the second class agents, who were the losers in the other two scenarios. Again the level of superior and first class agents is nearly the same.

Figure 5.6 exhibits the average number of don't care symbols within the classifiers and how often the learning agents take random decisions because no rule is fulfilled. The only thing remarkable is the solid line in the graph on the right hand side in the third row. This curve differs significantly from the other

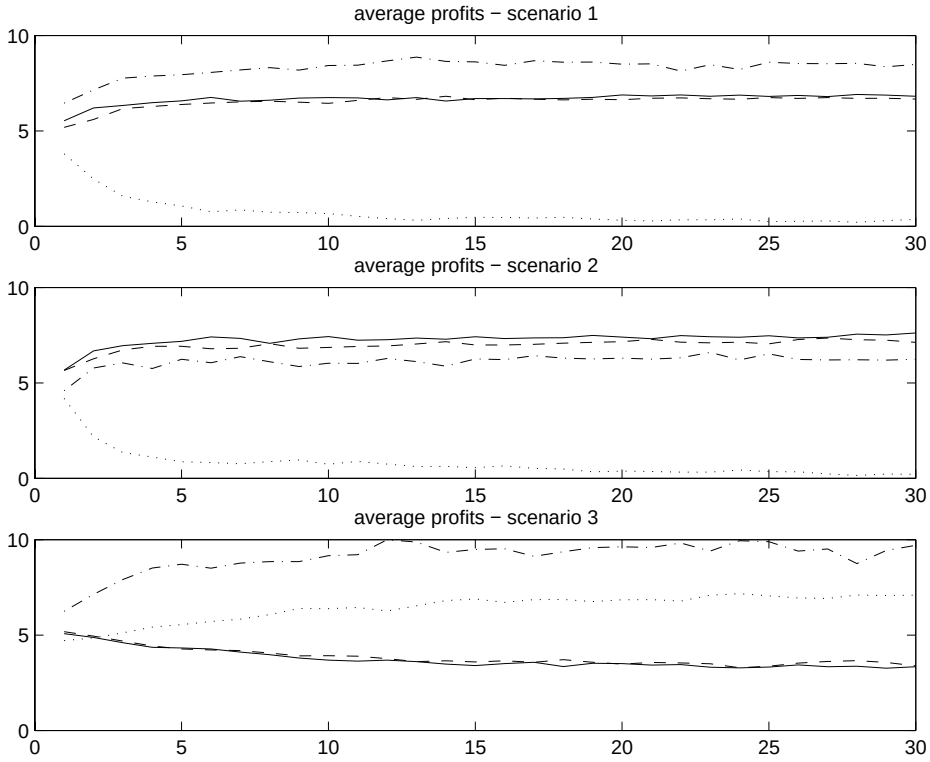


Figure 5.5: Average sales in three different scenarios

two scenarios. This means that the classifier systems of the superior agents learn the recent situations very quickly but fail when they are confronted with a new set of demands.

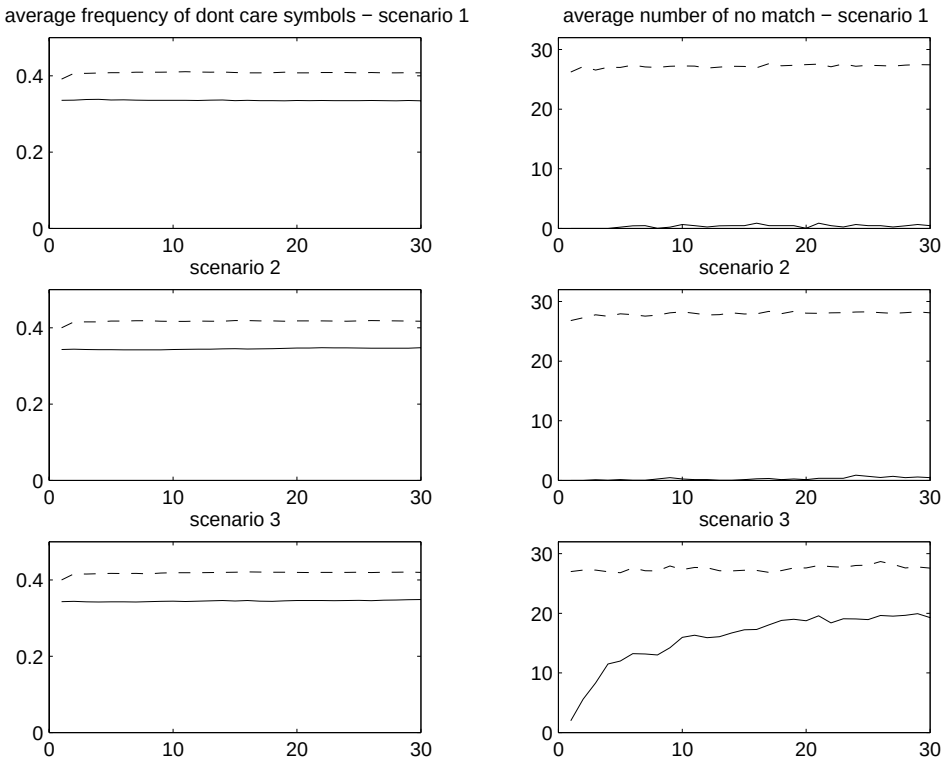


Figure 5.6: Average don't cares and no match in three different scenarios

Chapter 6

Summary and outlook

The goal of this work is to introduce complex adaptive systems as a valueable tool for modelling economic phenomena that cannot be controlled by means of equilibrium based analytical models. Special emphasis is dedicated to learning classifier systems and genetic algorithms, which are used to implement and train the rule base of the adaptive agents in two economic models.

Chapter 1 starts with a brief discussion about situations where analytical models — an approach of economic modelling which is also well-known under the term *greek letter economics* — do not succeed in illustrating the processes going on in reality. Further, the main ideas and characteristics of complex adaptive systems are illustrated. As an example exhibiting the recent economic changes, the second industrial revolution and its result — the homo informaticus — are described. Finally, the increasing importance of knowledge management (which plays a major role in the models in chapters 4 and 5) is elaborated.

The following chapter 2 provides detailed information about classifier systems and genetic algorithms. The construction of the components of various types of classifier systems is shown. The problem of apportionment of credit, which becomes difficult to handle when rule coupling occurs, and possible methods for solving it are discussed. Genetic algorithms are introduced as a possible method of improving the rule base of a classifier system. Several different genetic operators can be found in the literature. Section 2.2 provides a comprehensive survey of methods that can be applied for selection, crossover, and mutation. Those methods that are applied in the examples in chapter 3 and in the models in chapters 4 and 5 are discussed in detail. The chapter is concluded with a general summary of machine learning. This section includes some thoughts about gaining knowledge, an overview of popular ways of representing machine learning problems and search strategies, and a general classification of learning methods.

Chapter 3 contains three examples pointing out the range of possibilities provided by the methods explained in chapter 2. The first two examples only deal with gentic algorithms, while the following two examples are about

classifier systems combined with genetic algorithms.

Beginning with section 3.1 it is shown that genetic algorithms can quickly find the optimum of a smooth function with a reasonable big area of well performing feasible solutions in the neighbourhood of the optimum. It is also shown that a deliberate choice of the parameters determining the dynamics of the genetic algorithm has a high influence on the speed of the convergence and on the quality of the solution as well.

The difficulty of the task that has to be handled by the genetic algorithm is increased in section 3.2. Now the maximum of a function with several local maxima not too distant from the global optimum has to be found (see figure 3.2 on p. 49). It turned out that in this example the choice of the genetic operators and their parameters has to be done more carefully than in the first example. In particular the optimum was not found exactly within 80 generations when arithmetic crossover was applied, while when heuristic crossover was applied the optimal soultion was found in 42 % of the simulations (again within 80 generations).

In section 3.3 a classifier system equipped with an initial rule base that has been filled with random numbers has to generate those conditions and actions that are required to perform the simple task of shifting the components of a vector $\vec{a} \in \{0,1\}^3$ to the left. Thus, the input (a_1, a_2, a_3) should lead to the output (a_2, a_3, a_1). Before the simulation parameters are fixed, some considerations about the required number of cycles of the classifier system within one generation of the genetic algorithm are formulated. Moreover, a popular technique for modifying the probability of crossover and mutation during the simulation is explained. This enables the genetic algorithm to move very quickly within the search space at the beginning of the process to avoid a rapid convergence toward a suboptimal local optimum. Later on, when the genetic algorithm is supposed to be already in the neighbourhood of the global optimum, the movement is damped to stabilize the search such that the already found solutions do not get discarded again. Based on these findings more than 8000 simulations — each containing 150 generations of rules — are undertaken. In 18 simulations the system succeeded in fining the correct rule base within 140 generations and produced only correct responses during the remaining 10 generations.

The special problem of classifier systems with memory is given attention in section 3.4. The memory can only be implemented by using internal messages and rule coupling. There arises the problem that one rule can lead to a correct response in combination with a certain other rule but the same rule can also result in a wrong response in combination with another rule. Therefore, assigning strengths to the rules in a meaningful way becomes very tough. Since the economic applications in the chapters 4 and 5 do not make use of classifier systems with memory the aim of this section is not to solve this problem in detail. For the sake of completeness this section provides some reflections concerning the problems that may arise when learning classifier systems are applied in other situations, but simulation results are left out.

Chapter 4 is devoted to a simulation of a firm that designs a new product. Managing the communication between the involved departments may become a challenge in some cases. Thus, Hauser and Clausing (1988) developed a communication scheme which they called the *"House of Quality"*. The model introduced in this chapter is based on that communication scheme. It is assumed that the decision making process is distributed among four consecutive layers. The decision taken at one layer influences the problem at all the following layes. Moreover, each decision causes implementation costs and opportunity costs. The latter measures how well the solution meets the requirements. Minimizing only one of these two cost components does not lead to an optimal outcome. In the simulations each of these four decision makers is represented by a classifier system. To examine the parameter space 432 simulations with different sets of numerical parameters are done. A statistical analysis shows which numerical values should be chosen to quickly generate a well-performing rule-base (see table 4.2 on p. 76 and table 4.3 on p. 78).

Finally, chapter 5 provides a simulation of a market of products that can be substituted for each other. There are four types of agents that compete in placing their products in the market — also the agents of the same type compete among each other. Two out of the four types are learning agents that are modelled by a classifier system. One group of learning agents base their decisions on the desires of the customers, the other group on the offers and the sales of all the vendors. Another group of sellers places their products following a random walk and one agent just imitates the offer of the previously most successful supplier. These four groups of sellers are tested in three different environmental scenarios — static, cyclic, and random walk. In the first and in the last scenario the imitating agent is the most competitive one, while in the periodic scenario the learning agents making use of the information about the customers perform best.

The models in chapters 4 and 5 assume very simple products. In chapter 4 it is supposed that in every stage of the decision making process all the relevant information can be encoded in a vector $\vec{x} \in \{0,1,2,3,4\}^3$, and in chapter 5 the whole product is identified by a vector $\vec{y} \in \{0,\ldots,100\}^2$. The purpose of these restrictions is to keep the model simple in order to obtain the simulation results quickly. Due to technical progress the computing power of standard personal computers increases rapidly. Therefore, in the near future it might be possible to blow up the learning domains. Thus, it will become possible to simulate realistic market situations and insert data about real products. Thus, the results could be the basis of decision support systems helping decision makers involved in designing and releasing products in a competitive situation.

Bibliography

Aoki, M. (1996). Horizontal vs. vertical information structure of the firm. *The American Economic Review*, 76(5):971–983.

Aspesi, C. and Vardhan, D. (1999). Brilliant strategy, but can you execute. *The McKinsey Quarterly*, (1):89–99.

Baghai, M. A., Coley, S. C., and White, D. (1999). Turning capabilities into advantages. *The McKinsey Quarterly*, (1):101–109.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming, An Introduction, On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, San Francisco, California.

Bateson, G. and Bateson, M. C. (1972). *Steps to an Ecology of Mind: Collected Essays in Anthropology, Psychiatry, Evolution, and Epistemology*. Ballantine Books, New York.

Beinhocker, E. D. (1997). Strategy at the edge of chaos. *The McKinsey Quarterly*, (1):24–39.

Brenner, T. (1996). Learning in a repeated decision process: A variation-imitation-decision model. Papers on Economics & Evolution #9603, Max-Planck-Institute for Research into Economic Systems.

Brenner, T. (1998). Can evolutionary algorithms describe learning processes? *Journal of Evolutionary Economics*, 8(3):271–283.

Carley, K. and Lee, J.-S. (1998). Dynamic organizations: Organizational adaption in a changing environment. *Advances in Strategic Management*, 15:269–279.

Carley, K. and Svoboda, D. (1996). Modeling organizational adaption as a simulated annealing process. *Sociological Methods & Research*, 25:138–168.

Caspary, W. and Wichmann, K. (1994). *Lineare Modelle: algebraische Grundlagen und statistische Anwendungen*. Oldenbourg, Wien.

Chase, R. B. and Aquilano, N. J. (1995). *Production and Operations Management, Manufacturing and Services*, chapter 5, pages 173–180. D. Irwin, Inc., Chicago, 7th edition.

Churchman, C. W. (1971). *The Design of Inquiring Systems, Basic Concepts of Systems and Organization*. Basic Books, New York.

DeCanio, S., Dibble, C., and Amir-Atefi, K. (2000). The importance of organizational structure for the adoption of innovations. *Management Science*, 46(10):1285–1299. Informs.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). Positive feedback as a search strategy. Technical Report No. 91-016, Politecnico di Milano, Italy.

Drucker, P. F. (1970). *Managing for Results, Economic Tasks and Risk–Taking Decisions*. Harper and Row, London.

Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proc. Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan*, Piscataway, NJ. IEEE Service Center.

Fraunhofer Institut für Arbeitswirtschaft und Organistion, Deutsche Bank AG (1999). *Wettbewerbsfaktor Wissen: Leitfaden zum Wissensmanagement*. Frankfurt, Main.

Friedberg, R. (1958). A learning machine, part i. *IBM Journal of Research and Development*, 2:2–13.

Geyer-Schulz, A. (1995). Holland classifier sytems. *APL Quote Quad*, 25(4):43–55. ACM SIGAPL.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Goldberg, D. E., Horn, J., and Kalyanmoy, D. (1992). What makes a problem hard for a classifier system? Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL Report No. 92007.

Göppert, J. and Steinbrecher, M. (2000). Modulare Produktentwicklung leistet mehr. *Harvard Business Manager*, 22(3):20–30.

Hargadon, A. and Sutton, R. I. (2000). Building an innovation factory. *Harvard Business Review*, 78(3):157–166.

Hauser, J. R. and Clausing, D. (1988). The house of quality. *Harvard Business Review*, 66(3):63–73.

Holland, J. H. (1976). Adaption. In *Progress in Theoretical Biology IV*. ed. Rosen, R. F. New York, Academic Press.

Holland, J. H. (1995). *Adaption in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Bradford Books edition, Cambridge, Massachusetts, 1. MIT Press ed. 1992, 4. print. edition. MI.

Holland, J. H., Holyoak, K. J., Nisbeth, R. E., and Thagard, P. R. (1997). *Induction, Processes of Inference, Learning, and Discovery*. The MIT Press, Cambridge, Massachusetts, London, England.

Houck, C. R., Joines, J. A., and Kay, M. G. (1995). A genetic algorithm for function optimization: A matlab implementation. North Carolina State University, NCSU-IE Technical Report 95-09.

Johnston, J. and DiNardo, J. (1997). *Econometric Methods*. McGraw-Hill, New York, 4th edition.

Joines, J. and Houck, C. (1994). On the use of non-stationary penalty functions to solve constrained optimization problems with genetic algortihms. *IEEE International Symposium Evolutionary Computation*, pages 579–584.

Kennedy, J. (1997). The particle swarm: Social adaption of knowledge. In *IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana*, Piscataway, NJ. IEEE Service Center.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *IEEE International Conference on Neural Networks, Perth, Australia*, Piscataway, NJ. IEEE Service Center.

Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *International Conference on Systems, Man, and Cybernetics*.

Kennedy, J. and Eberhart, R. C. (1999). The particle swarm: Social adaption in information–processing systems. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, chapter 25, pages 379–387. McGraw-Hill, London.

Kollman, K., Miller, J., and Page, S. (2000). Decentralization and the search for policy solutions. *Journal of Law, Economics, and Organization*, 16:102–128.

Korzybski, A. (1950). *Manhood of humanity*. International Non-Aristotelian Library Pub. Co., Lakeville, Conn.

Kotler, P., Armstrong, G., Saunders, J., and Wrong, V. (1996). *Principles of Marketing, The European Edition*. Prentice Hall Europe, Campus 400, Maylands Avenue, Hemel Hampstead, Herfordshire, HP2 7EZ.

Malhotra, Y. (2000). Knowledge management for e-business performance: advancing information strategy to "internet time". *Information Strategy, The Executive's Journal*, 16(4):5–16.

Marengo, L. (1992). Coordination and organizational learning in the firm. *Journal of Evolutionary Economics*, 2:313–326. Springer-Verlag.

Michalewicz, Z. (1994). Genetic algorithms + data structures = evolution programs. *AI series, Springer-Verlag, New York.*

Nonaka, I. and Takeuchi, H. (1995). *The Knowledge-Creating Company.* Oxford University Press.

Page, S. and Ryall, M. (1998). Does strategy need computer experimentation? *Advances in Strategic Management*, 15:299–326.

Paul, D. L., Butler, J. C., Pearlson, K. E., and Whinston, A. B. (1997). Computationally modeling organizational learning and adaptability as resource allocation: an artificial adaptive systems approach. *Computational & Mathematical Organization Theory*, 2(4):301–324. Kluwer Academic Publishers.

Polani, D. and Uthmann, T. (1999). DMARKS: Eine verteilte Umgebung für agentenbasierte Simulationen von Marktszenarien. In Hohmann, G., editor, *Simulationstechnik, 13. Symposium in Weimar*, volume 3 of *Frontiers in Simulation*, pages 391–394. SCS - The Society for Computer Simulation International in cooperation with ASIM - Arbeitsgemeinschaft Simulation.

Über den Autor …

Thomas Fent studied Applied Mathematics at the Vienna University of Technology. During his diploma study he was involved in several research projects at the Institute of Econometrics, Operations Research and Systems Theory at the Vienna University of Technology. There he worked essentially in the field of intertemporal optimisation. After graduating diploma study he worked for four years at the Center for Business Studies at the University of Vienna. At this institution he was busy in science and teaching. Among others he participated in the integrated research program "Adaptive Information Systems and Modelling in Economics and Management Science". A part of his results gained in this program is collected in this volume. Currently Thomas Fent is employed at the Austrian Academy of Sciences at the Institute for Demography.t

---

Über diesen Band …

In this volume the necessity and the practicality of the employment of complex adaptive systems for describing recent economic happenings is discussed. They are compared with common analytical modelling techniques to give an idea about the advantages and shortcomings of both approaches. Then some methods that are qualified to implement complex adaptive systems are being explained. Special emphasis is dedicated to learning classifier systems and genetic algorithms.
In the following some examples are provided to illustrate possibilities and also restrictions of the usage of such procedures. Based on this elaborations two comprehensive economic models are formulated and analysed.
The interpretation of the simulation results delivers valuable hints for determining successful business strategies.

Über diese Reihe …

Die Bände dieser neuen ASIM - Reihe Fortschrittsberichte Simulation konzentrieren sich auf neueste Lösungsansätze, Methoden und Anwendungen der Simulationstechnik (Ingenieur-wissenschaften, Naturwissenschaften, Medizin, Ökonomie, Ökologie, Soziologie, etc.).
ASIM, die deutschsprachige Simulationsvereinigung (Fachausschuss 4.5 der GI - Gesellschaft für Informatik) hat diese Reihe ins Leben gerufen, um ein rasches und kostengünstiges Publikations-medium für derartige neue Entwicklungen in der Simulationstechnik anbieten zu können.
Die Fortschrittsberichte Simulation veröffentlichen daher: * Monographien mit speziellem Charakter, wie z. B. Dissertationen und Habilitationen * Berichte zu Workshops (mit referierten Beiträgen) * Berichte von Forschungsprojekten * Handbücher zu Simulationswerkzeugen (User Guides, Vergleiche, Benchmarks), und Ähnliches.
Die Kooperation mit den ARGESIM Reports der ARGESIM vermittelt dabei zum europäischen Umfeld und zur internationalen Publikation.