Fraunhofer Institut für Produktionsanlagen und Konstruktionstechnik
Bereich Automatisierungstechnik

Von der Fakultät für Verkehrs- und Maschinensysteme
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation

# Entwicklung eines lernfähigen Bildverarbeitungssystems unter Einsatz von Verfahren des Soft Computing

vorgelegt von
Diplom-Physiker
Mario Köppen

Tag der wissenschaftlichen Aussprache: 22. Juni 2005

*Promotionsausschuss*

| | |
|---|---|
| Vorsitzender: | Prof. Dr.-Ing. Joachim Herrmann |
| Gutachter: | Prof. Dr.-Ing. Jörg Krüger |
| | Prof. Dr.-Ing. Javier Ruiz-del-Solar (UdC) |

Berlin 2005

# Vorwort

Die vorliegende Dissertation ist das Ergebnis der langjährigen Mitarbeit in der Abteilung Sicherheitstechnik, ehem. Mustererkennung des Fraunhofer Institutes für Produktionsanlagen und Konstruktionstechnik. In diesen Jahren boten sich mir einmalige, vielfältige und ausgezeichnete Möglichkeiten, daß sich gerade entwickelnde Repertoire lernfähiger Verfahren des Soft Computing ständig an praktischen Anwendungen der digitalen Bildverarbeitung und Mustererkennung kennenzulernen, zu erproben und zu erweitern.

Ich hatte jedoch so auch die ausgezeichnete Möglichkeit, das Wachstum dieser Disziplin mitzuerleben, und mit meinen bescheidenen Fähigkeiten zu einem kleinen Teil auch mitzugestalten. Für die Unterstützung, das Verständnis für dieses Bestreben, die enge Zusammenarbeit bei meinen ersten Schritten in einer internationalen wissenschaftlichen Gemeinschaft, den mir gebotenen Möglichkeiten, dies auch kontinuierlich weiter zu betreiben, und nicht zuletzt dem entscheidenden thematischen Anstoß möchte ich ganz besonders Dr.-Ing. Bertram Nickolay danken. Mein Dank ebenso an die Betreuer dieser Arbeit, Professor Dr.-Ing. Jörg Krüger, und Professor Dr.-Ing. Wolfgang Adam. Aber auch meinen Kollegen aus der Zeit am Fraunhofer Institut möchte ich in diesem Zusammenhang danken. Ohne deren Kreativität, Geduld, deren Praxisgeist und deren wissenschaftliche Neugierde wäre nichts so, wie es heute ist: ganz vorneweg Dr. Katrin Franke, mit der mich die meisten Jahre einer sehr fruchtbaren Zusammenarbeit verbinden, aber auch all meine anderen Kollegen und Mitstreiter, insbesondere Manfred Busch, Christoph Nowack, Dr.-Ing. Christian Westendorf, Dr.-Ing. Martin Teunis, Dr.-Ing. Lutz Lohmann, Masoumeh Mohammadirah, Dr. Pierre Soille, Steffen Großert, Stephan Rudlof, Thorsten Sy, Dr.-Ing. Aureli Soria Frisch, Daniel Kottow, Dörte Waldöstl, Dr.-Ing. Gert-Steffen Rösel, Xiufen Liu, Christian Veenhuis, Anna Ukovich, Elnas Masandarani und Raul Vicente-Garcia.

Doch auch den vielen Forschern, die meine Entwicklung wahrgenommen, sie aufmerksam verfolgt und in wichtigen Phasen unterstützt haben, und mich so an einer sich entwickelnden internationalen Forschungsgemeinschaft zum Soft Computing auch teilhaben liessen, sei hier mein ausdrücklicher Dank ausgesprochen. Stellvertretend für viele andere mein Dank an Prof. Takeshi Furuhashi von der Nagoya Universität, Prof. Nikhil Pal vom Indian Statistical Institute, Prof. Rajkumar Roy von der Cranfield University, Prof. Javier Ruiz-del-Solar von der Universidad de Chile, Prof. Takeshi Yamakawa vom Kyushu Institute of Technology,

Prof. Yasuhiko Dote vom Muroran Institute of Technology, Dr. Ajith Abraham von der Chung-Ang University und Prof. Nikola Kasabov von der Auckland University.

Mein besonderer Dank an Frau Professor Dr. Kaori Yoshida vom Kyushu Institute of Technology, die der Arbeit den letzten Anstoß zur Vollendung gab. Dies ist auch ein ganz persönlicher Dank an Kaori, verbunden mit der Hoffnung und dem Wunsch, daß wir unsere privaten und wissenschaftlichen Wege in Zukunft weiterhin und noch enger gemeinsam beschreiten werden.

Schließlich: ein besonders lieber Dank an meine Eltern für all ihre beispiellose Unterstützung und all ihren Rat über die vielen Jahre, und die alle Erfolge mit mir geteilt haben. Sowohl von der seelischen als auch der praktischen Perspektive aus, ohne sie wäre diese Arbeit niemals zustande gekommen.

Berlin, im Februar 2006 Mario Köppen

# Inhaltsverzeichnis

# Liste der Akronyme

| | |
|---|---|
| **2D** | zwei-dimensional |
| **2DHL** | 2D-Histogramm Lookup |
| **AAAS** | American Association for the Advances of Science |
| **BCS/FCS** | Boundary Contour System/Feature Contour System |
| **CCD** | Charged Coupled Devices |
| **CD** | Compact Disc |
| **CI** | Computational Intelligence |
| **CMOS** | Complementary Metal Oxide Semiconductor |
| **GA** | Genetischer Algorithmus |
| **GP** | Genetische Programmierung |
| **IIZUKA** | Iizuka-Konferenzakronym |
| **IQ** | Intelligenzquotient |
| **KI** | Künstliche Intelligenz |
| **LPS** | Linear Pixel Shuffling |
| **LUCIFER** | Lookup-Compositional Inference |
| **MBPN** | Multilayer Backpropagation Neural Network |
| **MIT** | Massachusetts Institute of Technology |
| **MP3** | MPEG(Motion Pictures Expert Group) 1 layer 3 |
| **MTD** | Multi-Threshold Device |
| **NFL** | No Free Lunch (Theorem) |

| | |
|---|---|
| **OWA** | Ordered Weighted Averaging |
| **OWM** | Ordered Weighted Minimum |
| **PS** | Parameterstruktur |
| **RBF** | Radial Basis Function Neural Network |
| **ROI** | Region of Interest |
| **SCA** | Sense-Compute-Act (Paradigm) |
| **TFK** | Textilfehlerkatalog |
| **TN** | Texture Number (auch LBP - Local Binary Pattern) |

# Nomenklatur

| | |
|---|---|
| $\alpha_i$ | Verbindungsgewicht im MTD |
| $\phi_i$ | Aktivierung MTD-Neuron |
| $\theta$ | Schwellwert im MTD |
| $\Psi$ | Ausgabewert MTD |
| $A, B, C, \ldots, X$ | Mengensymbole |
| $a, b, c, d, x, y, z$ | Variablensymbole |
| $f, g, h$ | Funktionssymbole |
| $\mu(.)$ | Fuzzy-Zugehörigkeitsfunktion |
| $c_{\lambda\{,w\}}$ | Fuzzy-Komplement |
| $\tilde{g}(.)$ | Pseudoinverse Funktion zu $g(.)$ |
| $T(x, y)$ | T-Norm, triangulare Norm |
| $S(x, y)$ | S-Norm, T-Conorm, triangulare Conorm |
| $a(.)$ | Algorithmus |
| $\{(d_m^x(i), d_m^y(i))\}$ | Testsequenz von $m$ Paaren $(x, y)$ |
| $\delta(x, y)$ | KRONECKER-Symbol |
| $E[x]$ | Erwartungswert der Variablen $x$ |
| $\lambda$ | Parametersymbol |
| $[i_1, i_2, \ldots, i_k]_n$ | Darstellung von $i$ zur Basis $n$ mit $k$ Ziffern |
| 0110 | Bitstring, z.B. der Bitfolge 0,1,1,0,1 |
| $w_{ij}$ | Gewicht der Maskenposition $(i, j)$ |

| | |
|---|---|
| $I(x, y)$ | Bildfunktion |
| $R(x, y)$ | Ergebnisbild |
| $g_i$ | Grauwerte |
| $r_i$ | Referenzwerte |
| $M_w$ | gewichtete Filtermaske |
| $v$ | Vektorsymbol |
| $\Pi$ | Permutation |
| $H(g_1, g_2)$ | 2D-Histogramm |

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Position

Der Prozeß der zunehmenden Verbreitung digitaler Technologien und Infrastrukturen ist heutzutage nicht mehr umkehrbar. Sowohl der technologische Fortschritt in der Produktion von elektronischen Bauelementen bei gleichzeitigem Verfall der Endverbraucherpreise, als auch die universelle Möglichkeit der Repräsentation von Vorgängen, Zuständen und Abläufen auf Basis einer binären Logik sind die hauptsächlichen Motoren dieser Entwicklung.

Der eigentliche Fortschritt innerhalb dieses Prozesses findet jedoch nicht durch Ausweitung des technologischen Trägers dieses Prozesses, sonden durch den Ersatz von immer mehr Technologien und Infrastrukturen durch ihr digitales *Alter Ego* statt. Negroponte, der Leiter des MIT Media Labs, benutzt hier gern die Allegorie des Wettbewerbs von "Atomen" und "Bits"[131]: Schallplatte kontra Audio-CD oder MP3-Datei, Fernseher kontra Computer, Schmierzettel kontra Handheld-Computer, Drehscheibe kontra Tastentelefon oder Modem, Magnetband kontra Digital Video, Mont Blanc Füllfederhalter kontra digitale Signatur,... die Auflistung ließe sich endlos fortsetzen.

Digitale Technologien und Infrastrukturen (die Bits) können bestehende Technologien oder Infrastrukturen (die Atome) nur durch das Angebot einer geeigneten digitalen Repräsentation des neuen Einsatzgebietes ergänzen oder ersetzen.

Digitale Repräsentationen der technologischen oder infrastrukturellen "Wirklichkeit" finden sich heute in der Mehrzahl der Anwendungen in den folgenden Formen vor:

- digitalisierte Abtastwerte (Audio-CDs, CCD- oder CMOS-Kamerabilder)
- Datenbankindizes (Kundennummern, Zugangskodes)
- Regeln (*Event-driven* Nutzerinterface, Programmabläufe)

Eine Zunahme der Komplexität führt bei allen diesen Arten von Repräsentationen zu erheblichen Problemen. Dies kann auf den gemeinsamen Nenner gebracht werden, daß sie die Komplexität des Repräsentierten bei Verbreitung der Einsatzgebiete schlicht und einfach multiplizieren. Dies ist teilweise für den Anwender heute kaum mehr zu übersehen. Davon kann man sich zum Beispiel anhand der

wachsenden Anzahl der von ihm zu verwaltenden Datenbankindizes (Kontonummer, Passnummer, diverse Geheimworte, Bestellnummern, KFZ-Kennzeichen, Bearbeitungsnummern, Typen von Haushaltsgeräten, Softwareversionen, Softwareupdateversionen, Hausnummern, Termine,...) leicht selbst überzeugen.

Dies sind Anzeichen einer zunehmenden Entfremdung des Menschen von seinem Unternehmen, digitale Technologien und Infrastrukturen in einer immer größeren Anzahl von Lebensbereichen einzusetzen.

Die in der Praxis zum Einsatz kommenden digitalen Repräsentationen haben wenig mit der Art zu tun, wie etwa der Mensch seine Umwelt kognitiv repräsentiert. Auch wenn hier noch keine ultimativen Forschungsergebnisse vorliegen, ist es dennoch sicher, daß etwa die Netzhaut des Auges Bilder der Umwelt nicht pixelweise abtastet, oder daß unser Gedächtnis nicht in Form einer relationalen, geschweige denn einer mittels Schlüsselzahlen referenzierenden Datenbank organisiert ist.

Eine These soll daher an den Anfang dieser Arbeit gestellt werden:

*Fortschritt ist hier nur noch erzielbar auf der Basis der besseren Repräsentation einer höheren Komplexität.*

Der Ansatz besteht also in der Benennung von erweiterten Repräsentationsmöglichkeiten für die Objekte und für die Beziehungen zwischen den Objekten, die in Technologien und Infrastrukturen wesentlich mit eingehen.

In der vorliegenden Arbeit werden die methodischen Ansätze des *Soft Computing* als solch eine verbesserte Repräsentation ermöglichend eingeführt. Dies wird im nächsten Abschnitt näher erläutert.

## 1.2 Soft Computing

Der Begriff *Soft Computing* wurde zu Beginn der 90er Jahre von Lotfi Zadeh, auf den auch der kontroverse Begriff *Fuzzy Logic* zurückgeht, eingeführt. Zadeh wollte den Eigenheiten der gerade stattfindenden Entwicklung auf dem Gebiet der lernfähigen Verfahren Rechnung tragen und sie unter einem Dach zusammenfassen. Mit der Renaissance des Neurocomputing Mitte der 80er Jahre lebten auch eine Reihe anderer Verfahrensgruppen wieder auf, ohne daß es einen auf den ersten Blick offensichtlichen Zusammenhang zwischen ihnen gab. So erschien 1989 David Goldbergs Buch über genetische Algorithmen [54], durch das eine Vielzahl junger Wissenschaftler erstmalig auf einfache und anschauliche Weise in das Gebiet der evolutionären Verfahren eingeführt wurde (Hollands' Buch von 1975 [68] war hier viel zu abstrakt). Ähnliches gilt für ein Buch von Sugeno, aus dem insbesondere viele japanische Forscher die Grundlagen der Fuzzy-Logik lernten [166].

In diesem Zeitraum probierten sich Forscher aus dem asiatischen Raum auch erstmalig an einer Kombination aus neuronalen Netzen und Fuzzy-Logik. Im Jahre 1988 fand in der japanischen Stadt Iizuka, international kaum wahrgenommen,

ein Workshop über Neuro-Fuzzy-Systeme statt. Gemeinsam mit Zadeh und dem Physiologen Walter Freeman beschloß man, der nächsten Veranstaltung in Iizuka einen internationalen Rahmen zu geben. Der zweijährige Zyklus der IIZUKA-Konferenzen, der 1990 begann, war damit geboren. Der Schwerpunkt der Konferenzen wurde stets auf die gemeinsame Behandlung von neuronalen Netzen und Fuzzy-Logik gelegt. Bei der Eröffnungsrede von Zadeh zur ersten IIZUKA-Konferenz 1990 sprach Zadeh diese Gemeinsamkeit noch als *interpolative reasoning* an. Erst 1994 wurde dann offiziell *Soft Computing* zum Schwerpunkt der IIZUKA-Konferenz erklärt, wohl als der ersten internationalen Konferenz überhaupt.

Alternativ wird in der Forschung auch die Bezeichnung *computational intelligence* (CI) verwendet. Durch diesen Begriff soll eine bessere Abgrenzung zum Begriff der künstlichen Intelligenz (*artificial intelligence*) erreicht werden.

Es dauerte jedoch einige Jahre, bevor der Begriff *Soft Computing* in der Form gefaßt wurde, in der er heute im Gebrauch ist, nämlich als einer losen Allianz aller Verfahren, die auf neuronalen Netzen, Fuzzy-Logik, evolutionären Algorithmen und deren Hybridformen beruhen.

In [203] faßt Zadeh den Begriff so zusammen: „*. . . to mimic the ability of the human mind to effectively employ modes of reasoning that are approximate rather than exact. In traditional –hard– computing, the prime desiderata are precision, certainty, and rigor.*“ Zusätzlich gibt er die Grundformel jeden *Soft Computing* Verfahrens an: *exploitation of the tolerance for imprecision and uncertainty.*

In [204] betont Zadeh dann auch die Interoperabilität der unter dem Begriff *Soft Computing* zusammengefaßten Verfahren: „*What is important about soft computing is that it suggests the possibility of employing fuzzy logic, neurocomputing and genetic algorithms in combination rather than in isolation.*“

Die hier interessierende neue Form der Repräsentation durch *Soft Computing* findet sich in der von Zadeh als zentral betrachteten Thematik der Ungenauigkeit (*imprecision*), und der formalen Methoden, damit umzugehen. Der Begriff Ungenauigkeit weist hier auf die defizitiäre Ausprägung verschiedener Arten von Repräsentationen hin:

- Meßungenauigkeit oder numerische Ungenauigkeit, wie etwa beim Runden von Zahlenwerten,
- linguistische Unschärfe, wie etwa beim Abbilden von Begriffen auf Zahlenwerte,
- Vagheit, als Ausdruck des Unwissens über eine Sache,
- Wahrscheinlichkeit oder die
- Möglichkeit einer Sache, die von der Wahrscheinlichkeit ihres Eintretens zu unterscheiden ist.

Der Kerngedanke ist, daß eine Verringerung solcher Ungenauigkeiten einen wie auch immer gearteten höheren Aufwand erfordert. Man muß entweder genauer messen, genauer erläutern, mehr Wissen einholen, eine größere Stichprobe wählen, oder den Kontext einer Sache stärker abgrenzen. Dies steht dann im Gegensatz zu dem eigentlichen Nutzen der höheren Genauigkeit. Unter einer Reihe von Umständen ist diese nämlich gar nicht erforderlich, notwendig, gewünscht oder möglich:

- Die Kosten für den zusätzlichen Aufwand zur Erlangung einer höheren Genauigkeit stehen in keinem Verhältnis zu dem sich aus der höheren Genauigkeit ergebenden Vorteil.
- Eine hohe Genauigkeit ist nicht zweckmäßig, wie z.B. bei der Regelung einer Klimaanlage, bei der die Raumtemperatur sicher nicht auf 0.001 K genau geregelt sein muß.
- Eine hohe Genauigkeit kann sogar fehlerhaft sein, da ein System dadurch unflexibel werden kann.
- Eine hohe Genauigkeit ist nicht möglich, da das System gar keine *„ground truth“* besitzt oder diese nicht erlangt werden kann.
- Abweichungen vom Ideal sind nicht direkt modellierbar, wie z.B. beim Ausbalancieren einer Kugel auf einer Auflage auf einem umgekehrten Pendel so, daß die Kugel um maximal $\delta x$ von der Mittellage abweicht (es gibt keine „Differentialungleichungen“).

Alle unter dem Dachbegriff *Soft Computing* zusammengefaßten Methoden haben daher einen entscheidenden Aspekt gemeinsam: sie erlauben die direkte Umsetzung eines höheren Aufwands zum Erzielen genauerer Ergebnisse. Beispiele dafür sind:

- Die Verwendung von mehr linguistischen Termen und damit von mehr Regeln bei einem Fuzzy-Controller.
- Die Erhöhung der Anzahl von Neuronen bei einem neuronalen Netz.
- Die Vergrößerung der Population eines genetischen Algorithmus, und die Erhöhung der Anzahl von Generationen.

Dies ist eine empirische Tatsache. Offen bleibt jedoch hier erst einmal, wie genau diese Verfahren im Endeffekt werden können, oder wie sicher sie die formal gegebenen Ziele überhaupt erreichen. Die (nicht immer positive) Antwort darauf geben eine Reihe von mathematischen Theoremen, deren Diskussion einen Teil der vorliegenden Arbeit ausmacht:

- Das "Ugly Duckling" Theorem beweist die Unmöglichkeit, Objekte lediglich auf der Basis von logischen Attributen zu unterscheiden.
- Das "No Free Lunch" Theorem beweist, daß alle Optimierungsverfahren im Mittel über alle Optimierungsprobleme gleichgut performieren.
- Das Schemata-Theorem beweist, daß bei einem evolutionäre Verfahren überdurchschnittlich gute Individuen sich proportional zu dem Wert ihrer Überdurchschnittlichkeit in der Population anhäufen.
- Das Stone-Weierstrass Theorem beweist die universelle Approximationsfähigkeit von neuronalen Netzen und Fuzzy-Systemen.
- Das Kolmogorov-Theorem beweist die universelle Repräsentationsfähigkeit von neuronalen Netzen und Fuzzy-Systemen.

Jedes dieser Theoreme hat seine eigene Geschichte, und unterscheidet sich auch in der Art und Weise, wie es in das Repertoire des *Soft Computing* aufgenommen wurde. Während so das Kolmogorov-Theorem quasi rückwirkend als Garant dafür, daß zu jeder Systemfunktion $f$ ein neuronales Netz existiert, daß diese Funktion genau repräsentieren kann, wiederentdeckt wurde, war die Einführung des "No Free Lunch" Theorems Mitte der 90er Jahre von einer der heftigsten Kontroversen unter den Wissenschaftlern auf diesem Gebiet begleitet, die es jemals gegeben hat.

## 1.3 Mustererkennung

Der digitalen Repräsentation von Objekten und der Beziehungen zwischen Objekten, wie sie verstärkt technologisch angestrebt wird, steht die Repräsentation durch Muster, wie sie im wesentlichen alle Lebensformen praktizieren, gegenüber. Natürlich entsteht hierbei die Frage, ob *Soft Computing* mehr zur Mustererkennung beitragen kann.

Dazu ist es notwendig, den Begriff der Mustererkennung selbst klar abzugrenzen. In der Literatur finden sich jedoch überraschend wenig Versuche, sowohl den Begriff des Musters als auch die Gesamtheit aller Aktivitäten, die unter dem Begriff Mustererkennung zusammengefaßt werden zu definieren. Viele wissenschaftliche Disziplinen nehmen in der einen oder anderen Form auf Muster Bezug, ohne sie als wesentlichen Schwerpunkt ihrer Forschungen anzusehen. So spielen Muster und die Prozesse ihrer Entstehung und Herausbildung in der Physik eine Rolle, wie z.B. in den synergetischen Ansätzen von Haken, in der Mathematik (Beispiel: Entschlüsselung von Chiffrierverfahren), in der Chemie, der Stochastik, der Signaltheorie, der Kontrolltheorie, der Biologie, der Meteorologie, der Geologie, der Informatik, den Designwissenschaften, der Paläontologie, in der Psychologie (der Rorschach-Test oder die Gestaltpsychologie nehmen direkt Bezug auf Muster)

und natürlich in der Physiologie, die die Mechanismen der Musterwahrnehmung direkt untersucht. Mustererkennung versteht sich jedoch nicht als Teilzweig einer der benannten Disziplinen, sondern als Gruppe gemeinsamer Aktivitäten [184].

Genauso unscharf wie der Begriff Mustererkennung ist auch der Begriff Muster selbst. Schon die etymologische Herkunft ist sehr vielschichtig. Das seit dem 14. Jahrhundert bezeugte deutsche Wort *Muster* stammt vom italienischen Wort *mostra* für „zeigen, weisen, hinweisen, bezeichnen“ ab, welchem wiederum das lateinische Wort *monstrare* mit ähnlicher Bedeutung zugrunde liegt [40]. *Muster* steht im Deutschen für „Vorlage, Modell; Vorbild; Probestück; Zeichnung, Figur“, also etwas, von dem auf eine größere oder allgemeinere Gesamtheit geschlossen werden kann. Etymologisch ist im Deutschen Muster mit *Monster*, *Monstrum* und *monströs* verwandt, wobei letztere auf den Aspekt des „Gezeigt-werdens“ verweisen, d.h. etwas ist nur dann *monströs*, wenn wir es jemand anderem zeigen und nicht einfach nur darauf hinweisen. Die Verwandschaft mit *Monster* ist hier deshalb interessant, weil das englische Wort *monster* dieselbe Wurzel hat wie der deutsche Begriff *Monster*, das englische Wort *pattern*, das dem deutschen *Muster* entspricht, jedoch nicht. Im Unterschied zu vielen kategorialen Bezeichnungen wie Struktur, Organisation, oder Information weichen die deutsche und englische Bezeichnung deutlich voneinander ab. Das gilt sogar in inhaltlicher Hinsicht. Das englische Wort *pattern* geht auf das französische Wort *patron* zurück, also im Sinne von „Meister; Beschützer“, oder, wie Watanabe schreibt ([184], S.6): „*. . . anything that was supposed to be followed or imitated.*“ Das französiche *patron* wiederum, welches auf das lateinische *patronus* für „(väterlicher) Schutzherr, Schirmherr“ zurückgeht, bildet auch die Wurzel der deutschen Worte *Patron* und *Patrone*. Die ursprüngliche und übertragene Vorstellung hier ist, daß das Verhalten des Vaters Vorbild für den Charakter des Sohnes ist. Das deutsche Wort *Musterung*, wie z.B. die einer Textilie, spiegelt mehr diese englische Bedeutung wieder.

In diesem Zusammenhang ist auch der Modus des Erkennens von Mustern im Deutschen und Englischen ursprünglisch anders gefaßt, nämlich induktiv im Deutschen und deduktiv im Englischen. Heutzutage konvergieren diese Bedeutungen zwar im allgemeinen Sprachgebrauch, können jedoch nicht die Tatsache unterdrücken, daß Muster an sich nicht objektiv real sind, sondern eher eine Art „Äther“ darstellen, in dem unsere subjektive Sicht auf unsere Umwelt abläuft. Oder anders gesagt: daß erkannte Muster in jedem Fall Ergebnis einer Ableitung aus realen Fakten sind, die in einer anderen „Sprache“ formuliert ist als die, die durch die Qualität dieser Fakten direkt vorgegeben ist. Genauer gefaßt, finden sich alle Momente des Musterbegriffs schon im dialektischen Gegensatzpaar von Inhalt und Form wieder, da Muster auf einen formalen Aspekt einer Sache Bezug nimmt, der auf einen inhaltlichen Aspekt verweist. Muster werden aus der Form abgeleitet, jedoch auf Inhalte angewendet.

Damit soll im folgenden unter Mustererkennung die Beschreibung der Vorgänge verstanden werden, die eine Transduktion von Form zu Inhalt auf Basis einer

unspezifizierten Zwischenstufe, eben den Mustern, durchlaufen. Ein Muster hat damit immer zwei unvereinbare Aspekte, den formalen und den inhaltlichen. Ein Beispiel für dieses „janusköpfige“ Wesen des Musters finden wir in der Merkmalsklassifikation, in der der formale Aspekt des Musters als Merkmal und der inhaltliche Aspekt als Klasse aufgefaßt wird (siehe Abschnitt 2.1). Ziel der Merkmalsklassifikation ist dann die Umrechnung von quantifizierten Merkmalen in Klassennummern. Das Muster selbst tritt bei diesem Vorgang niemals in irgendeiner „jungfräulichen“ Form zutage.

Da das Wechselspiel von Inhalt und Form in nahezu allen Wissenschaften eine grundlegende Rolle spielt, wundert es auch nicht, daß auch dieser spezielle Modus dieses Wechselspiels sich in so vielen Disziplinen wiederfinden läßt und in so vielfältiger Hinsicht untersucht wird, ohne jedoch ausschließlicher Gegenstand dieser Disziplinen zu werden.

## 1.4 Mustererkennung und Soft Computing

Die ursprüngliche Ausprägung der Momente der heute unter dem Dachbegriff *Soft Computing* zusammengefaßten Methoden und Algorithmen fand ihren Anfang bereits in den 60er Jahren, geriet dann aber in den 70er Jahren eher aus dem Blickfeld der aktiven Forschung. Erst gegen Ende der 80er Jahre kam es zu einer plötzlichen Wiederaufnahme der Untersuchungen zu diesen Verfahren. Im wesentlichen kann man dafür die rasche Verbreitung einfach verfügbarer, leistungsfähiger Computerhardware verantwortlich machen, die heute noch anhält. Erst in den 90er Jahren wurde diese Entwicklung dann durch eine gewisse theoretische Breite ergänzt, die auf die beginnende Formulierung der theoretischen Grundlagen des *Soft Computing* hinauslief.

So wurden die ersten Vorschläge zur Anwendung der evolutionären Idee bei der Lösung von Optimierungsaufgaben bereits um 1960 ins Spiel gebracht [19]. Dieses waren jedoch aus heutiger Sicht einfache Mutationsansätze, die später [67] um die Rekombination unter dem Titel zielorientierte Mustererkennung (*goal-directed pattern recognition*) ergänzt wurden. Was jedoch den weiteren Einsatz evolutionärer Algortihmen verhinderte, war klar die nicht zur Verfügung stehende Rechenleistung, die zur Durchführung solcher populationsbasierter Ansätze notwendige Voraussetzung war. Erst die viel später einsetzende Verbreitung von *Personal Computern* machte dies möglich.

Ähnliches läßt sich zu den neuronalen Netzen sagen, deren interne Berechnungen relativ umfangreich sind, und eine gewisse numerische Genauigkeit erfordern. Die ersten Arbeiten zum Perzeptron im Zusammenhang mit der Mustererkennung wurden Ende der 50er Jahre durchgeführt (siehe Abschnitt 2.2.1). Sie stießen auf besonderes Interesse, zumal das Perzeptron ja nicht nur ein Konzept auf dem Papier war, sondern bereits in Form des *Mark I Perceptrons* als konstruierte Maschine existierte.

Man muß hier unbedingt festhalten, daß es ursprünglich keine solche Trennung von Neuroinformatik und Computerarchitektur gab, wie sie uns heute selbstverständlich erscheint. Die vorherrschende Idee war, daß sich die weitere Entwicklung der Computerhardware in Anlehnung an die biologischen Prinzipien vollziehen werde [120]. Wie weiter unten näher diskutiert wird (siehe S. 17), war das Perzeptron hier eher eine Art "Schnellschuß", um die vermeintlich zu langsamen Fortschritte auf diesem Gebiet zu kompensieren. Vor allem fehlte dem Perzeptron eine genuine mathematische Modellierung. Diese wurde von Minsky und Papert [127] später nachgereicht, jedoch wurden die dabei aufgezeigten prinzipiellen Probleme eines solchen neuronalen Musterlerners voreilig zum Anlaß genommen, sich der Idee des biologischen Ansatzes in der Computerentwicklung nahezu vollständig zu entledigen. In den folgenden zwanzig Jahren fanden Forschungen zur Neuroinformatik zwar weiter statt, aber eher abseits vom allgemeinen wissenschaftlichen und praktischen Interesse.

In den 70er Jahren dann finden wir beide Momente, Computer und Mustererkennung, bereits fast sauber getrennt vor: die Computerforschungen waren durch die Künstliche Intelligenz (KI) geprägt, während die Mustererkennung statistischen und wahrscheinlichkeitstheoretischen Ansätzen folgte. Der Umstand, daß KI nicht auch Einzug in der Mustererkennung gehalten hatte, war auf die Präsentation eines Theorems von Satosi Watanabe, heute unter dem Namen "Ugly Duckling" Theorem bekannt, zurückführbar (siehe [184]). Dies stellte sicher, daß Objektunterscheidung alleine auf Basis von logischen Attributen nicht durchführbar ist. Objekte und ihre Beziehungen untereinander sind reichhaltiger als die bloße logische Konsternierung ihrer Existenz. Nach wie vor war das Konzept des Musters, dem jedoch in der KI keine Heimat geboten werden konnte, Dreh- und Angelpunkt intelligenter Leistungen.

Die Geschichte der jahrzehntelangen Ächtung der Fuzzy-Logik ist hinreichend bekannt und soll hier nicht noch einmal erzählt werden [124]. Dabei gab es mehr Motivationen als genug, den starren Rahmen einer numerisch-binären Logik eher früher als später zu verlassen (worauf Zadeh gerade mit der Zielstellung Mustererkennung in [202] auch hinweist): Spencer-Brown [162] beklagt vom wissenschaftstheoretischen Standpunkt das Fehlen einer BOOLEschen Arithmetik (siehe Abschnitt 2.2.2). Hätte man dem Ende der 50er Jahre entdecktem KOLMOGOROV-Theorem auch nur einen Bruchteil der Aufmerksamkeit wie dem Perzeptron gewidmet, wäre aufgefallen, daß die universelle Repräsentation einer stetigen Funktion im KOLMOGOROV-Theorem über eine heute unter dem Namen Fuzzy-Singletons bekannte Strukturierung erfolgt. Dies wurde, weitgehend unbeachtet, von Albus zur Konzeption eines vom Perzeptron verschiedenen neuronalen Netzes genutzt [3] und erst in den 90er Jahren von Kosko in Form der Fuzzy-Patches thematisiert [96]. Auch das seit Beginn der 60er Jahre bekannte "Ugly Duckling" Theorem, wie bereits benannt, weist als eine Voraussetzung, unter der dieses Theorem *nicht* gilt, auf die sogenannte *Ponderation* von logischen Attributen hin, also einer numerischen Quantifizierung des Grads ihrer Ausprägung.

Ungeachtet solcher Signale glaubte man jedoch für viele Jahre (Jahrzehnte), auf reellwertige Logiken verzichten zu können, und war eher bemüht, die logische Unmöglichkeit dieses Ansatzes zwingend nachzuweisen [42].

Wie diese kursorhaften Ausführungen zeigen, sind die Entwicklung von *Soft Computing* und Mustererkennung sehr komplex miteinander verflochten. Eine detailierte Darstellung würde den Rahmen dieser Arbeit jedoch sprengen. Wir können festhalten, daß *alle* Momente des *Soft Computing* sich bereits Ende der 50er bis Anfang der 60er Jahre vorfanden und auf das Gebiet der Mustererkennung wesentlich verwiesen. Jedoch wurden diese Ansätze aus einer Reihe von Gründen, wie vor allem mangelnder Rechenleistung und fehlender theoretischer Fundierung, wieder aufgegeben. Unabhängig davon war eines der primären Targets dieser Untersuchungen jedoch stets die Mustererkennung gewesen. Wir können daher heute, nach dem Wiederaufleben all dieser Methoden und der Wiederentdeckung grundlegender Arbeiten, jedoch auch auf Basis eines breiteren Verständnis der Natur des *Soft Computing*, wichtige Impulse für die Mustererkennung erwarten. Dies herauszustellen ist ein wesentliches Ziel dieser Arbeit. Dabei ist der Ansatz des *Soft Computing* in der Mustererkennung nicht eindeutig. Es soll in dieser Arbeit sowohl demonstriert werden, wie die Gestaltung eines mustererkennenden Systems mittels *Soft Computing* abläuft, aber es sollen genauso die wesentlichen Tatsachen zusammengetragen werden, auf deren Basis die Gestaltung dieses und anderer mustererkennender Syteme abzulaufen hat.

## 1.5 Überblick

In dieser Arbeit wird ein neues lernfähiges Bildverarbeitungssystem unter Einsatz der Methoden des *Soft Computing* entwickelt. Um den Einsatz des *Soft Computing* hier besser zu motivieren, liegt der Schwerpunkt auch auf den theoretischen Grundlagen des *Soft Computing*, die in dieser Weise bisher noch nicht zusammengestellt worden sind. Dies ist jedoch notwendig, um wesentliche Gestaltungsprinzipien zum allgemeinen Einsatz von *Soft Computing* in der Mustererkennung herauszuarbeiten.

In Kapitel 2 werden die digitale Bildverarbeitung thematisiert und wesentliche Verfahren des *Soft Computing* vorgestellt, insbesondere die neuronalen Netze Perzeptron und *Multilayer Backpropagation Neural Network*, Fuzzy-Logik, genetische Algorithmen und genetische Programmierung. Diese Verfahren gehen wesentlich in das im Kapitel 6 beschriebene System LUCIFER ein, die Einführung der Verfahren erfolgt jedoch unter dem Aspekt der weiteren mathematischen Behandlung, und der Präzisierung der Aufgabenstellung hinter der vorliegenden Arbeit in Kapitel 3.

In Kapitel 4 werden dann die für das *Soft Computing* wesentlichen mathematischen Theoreme vorgestellt. Es wird eine kurze Darstellung des "Ugly Duckling" Theorems gegeben, nach dem Objekte sich nicht auf Basis von logischen Attri-

buten unterscheiden lassen. Auf dessen Basis läßt sich die Frage nach der Repräsentation von Sensordaten zugunsten des datenorientierten Ansatzes klären. Als weiteres Theorem wird das "No Free Lunch" Theorem vorgestellt, nach dem alle Optimierungsverfahren im Durchschnitt genauso gut sind, und es wird ein neuer Beweis präsentiert, der gegenüber dem von Wolpert und Macready gegebenen einfacher ist und die Denkweise hinter diesem Theorem besser illustriert. Diese Sichtweise wird exemplarisch an einigen Beispielen aus der Bildverarbeitung dargelegt.

Ein weiterer Teil von Kapitel 4 stellt die Problematik der Funktionsapproximation und -repräsentation heraus. Zwei außerhalb des *Soft Computing* entstandene Theoreme, das Stone-Weierstrass-Theorem und das Kolmogorov-Theorem, garantieren den Verfahren des *Soft Computing* zumindest stets die Existenz einer beliebig genau approximierenden, oder perfekt repräsentierenden Lösung.

Als letztes wesentliches Theorem des *Soft Computing* wird auf das Schemata-Theorem eingegangen, welches insbesondere bei genetischen Algorithmen eine wichtige Rolle spielt und diese Algorithmen von Zufallsalgorithmen unterscheiden hilft.

Aus den Darstellungen von Kapitel 4 werden wesentliche Gestaltungsprinzipien für Anwendungen des *Soft Computing* speziell in der Bildverarbeitung und Mustererkennung in Kapitel 5 herausgestellt. Insbesondere den dort vorgestellten methodischen Gerüsten kann eine hohe Versatilität in Bezug auf potentielle *Soft Computing* Anwendungen nachgewiesen werden.

Diese Prinzipien gehen dann auch in das in Kapitel 6 im Detail beschriebene SystemLucifer zur automatischen Generierung von Texturfiltern mit ein. Dieses System erlaubt auf Basis einer einfachen ikonischen Nutzervorgabe (Ausgangsbild und binäres Zielbild) die Konfiguration eines zweidimensionalen methodischen Gerüstes. Die internen Komponenten des Systems werden im Detail erläutert und an Beispielen wird die Gesamtfunktionalität des Systems illustriert.

InLucifer kommt der 2D-Lookup Algorithmus der mathematischen Morphologie, einer Teildisziplin der digitalen Bildverarbeitung zum Einsatz. Dieser benötigt zur Durchführung die Festlegung dreier Komponenten: zwei Bildoperationen und die binäre 2D-Lookup Matrix. Die Bildoperationen, die aus dem Ausgangsbild ein dem Zielbild möglichst ähnliches Ergebnisbild generieren, werden mittels genetischer Programmierung gesucht. Dies beinhaltet auch eine Konzeption der allgemeinen Beschreibung von Bildoperatoren durch die Baumstrukturen des genetischen Programmes.

Die 2D-Lookup Matrix läßt sich, als ein wesentlicher Vorzug des Systems, bei gegebenen Bildoperationen aus diesen direkt fusionieren und muß nicht extra adaptiert werden. Es zeigt sich jedoch, daß diese Matrix wesentlich für die Generalisierung der Texturfilter zuständig ist. Bestimmte, die Generalisierung begünstigende Strukuren können durch ein optionales Vorverarbeitungsmodul oder eine nachgeschaltete Auswertung mittels eines speziellen neuronalen Netzes hervorge-

hoben werden.

Das Sytem LUCIFER erweist sich als konsistente Umsetzung der Methoden des *Soft Computing* zur Gestaltung eines lernfähigen Bildverarbeitungssystems.

Diese Arbeit schließt mit einer Zusammenfassung der erzielten Ergebnisse.

# 2 Stand der Erkenntnis

## 2.1 Digitale Bildverarbeitungssysteme

Die digitale Bildverarbeitung beinhaltet die Digitalisierung und anschließende rechnerbasierte Weiterverabeitung von Bildern. Ziel ist dabei die Verbesserung bildlicher Information zur Interpretation durch den Menschen als auch die automatische Verarbeitung von Bilddaten mit dem Ziel der Überführung der reinen Bildinformation in eine andere Wissensdomäne [56]. Typische Aufgabenstellungen dabei sind

- Bildverbesserung: Ziel ist die Verringerung von Bildstörungen (Rauschen) und die Hervorhebung von Bildeigenschaften (Kontrast).
- Bildkompression: Die Verringerung des mit dem Speichern und Übertragen von Bilddaten verbundenen Datenaufkommens bei möglichst hoher Wahrung der relevanten Bildinformationen.
- Bildrestauration: Dies beinhaltet eine Modellierung von Bilddegradationen, um Bilder in einen ungestörten Zustand zurückzuführen.
- Bildsegmentierung: Ist die flächenhafte (spatiale) Einteilung eines Bildes in "sinntragende Bereiche".

Die digitale Bildverarbeitung ist ein interdisziplinäres, technologieorientiertes Wissensgebiet, das hauptsächlich von Verfahren und Methoden der Signalverarbeitung und Mustererkennung getragen wird. Sie verfügt über wenig eigene Grundlagen (meist werden diese von anderen Wissensgebieten übernommen), und sehr viel anwendungsorientierten, empirischen Herangehensweisen. Als eigenes Wissensgebiet entstand die digitale Bildverarbeitung gemeinsam mit den technischen Mitteln zur Bildakquisition (insbes. der CCD-Technologie) in den 60er Jahren.

In nahezu allen heute existierenden Anwendungen der digitalen Bildverarbeitung wird der in Abb. 2.1 dargestellten linearen Verarbeitungskette der Merkmalsklassifikation gefolgt (siehe z.B. [61] [187] [186] [142] [41] [123] [8] [141] [114] [38] [39] [37], aber siehe Abschnitt 5.1.2 für einen generalisierten Gesichtspunkt). Der primäre Schritt ist die Bildakquisition, oder Bildgewinnung. Hierbei werden sensoriell erfaßte Daten in eine für die rechnergestützte Weiterverabeitung

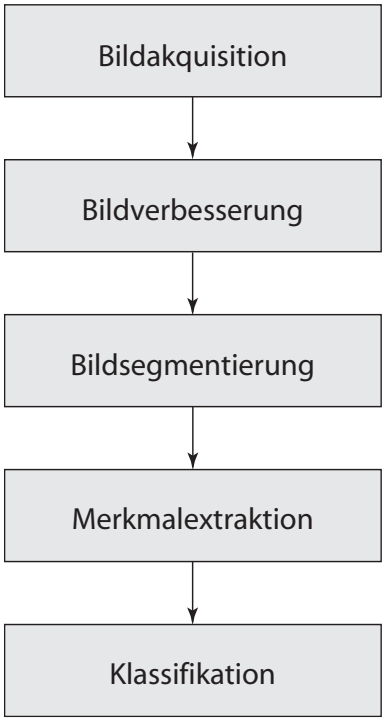



Abbildung 2.1: Stufen der digitalen Bildauswertung.

geeignete Repräsentation überführt. Dies beinhaltet i.d.R. zwei Arten von Digitalisierung eines analogen Eingangssignals: die *Rasterung*, bei der das analoge Eingangssignal an einer Menge von Stützstellen unter möglicher Wahrung des ursprünglichen geometrischen Zusammenhanges des Signals abgetatstet wird, und die *Quantisierung*, bei der das abgetastete Signal in eine endliche Anzahl von Digitalisierungsstufen (i.d.R. intensitätsbezogen) abgebildet wird. Ergebnis der Bildakquisition ist ein *digitalisiertes Bild*, das der weiteren Verarbeitung zugeführt werden kann. Die Bildakquisition erfolgt in einem komplexen, offenen Umfeld von Beleuchtungsverhältnissen, Akquisitions- und Digitalisierungsgeräten, Zeitregime und Entwicklungsstand der genutzten Technologie. Prinzipiell beeinflussen alle diese Faktoren die Qualität der nachfolgenden Bildauswertung. Einigen Mängeln der Bildakquisition kann in der zweiten Bearbeitungsstufe, der Bildverbesserung, Abhilfe geschaffen werden. So können Verfahren festgelegt werden, die eine ungleichmäßige Belegung der Quantisierungsbereiche wieder ausgleichen oder andere visuelle Strukturen des Bildes stärker herausstellen. Ebenso zu nennen sind Rauschfilter, die zufällige Störungen des physikalischen Akquisitionsprozesses unterdrücken.

In der *Bildsegmentierung* erfolgt eine Einteilung des Wertebereiches des Bildes in zusammenhängende Bereiche, die evtl. getrennt weiterverarbeitet werden. Dies kann eine fixe Zuordnung sein (wie z.B. bei Texturfenstern, siehe weiter unten), aber auch adaptive, bildabhängige (also funktionale) Verfahren können zum

Einsatz kommen (wie z.B. bei der Vordergrund/Hintergrund-Trennung). Die Bereiche können dabei aus ihrem flächenhaften Zusammenhang abgeleitet werden (Ergebnis einer Histogrammanalyse) oder durch die sie begrenzenden Konturen (Verfahren der Konturverfolgung).

In den für die weitere Auswertung relevanten Bildbereichen (Segmenten) werden dann Merkmale erhoben. Dies bedeutet eine Überführung der Quantisierungswerte als Funktion der erhobenen Stützstellen (die sog. Bildfunktion) in einen anderen Wertebereich. Es gibt eine hohe Anzahl möglicher Methoden zur Berechnung von Merkmalen. Typische Verfahren basieren auf statistischen Eigenschaften der Werte der Bildfunktion in dem jeweils betrachteten Segment, oder beinhalten bereits modellhafte Informationen in ihrer Spezifikation der Berechnungsvorschriften, wie etwa bei der Festlegung von Objektmerkmalen. Wesentlich ist, daß diese Merkmale für den folgenden Schritt der Klassifikation signifant sein sollen, und untereinander wenig korrelieren. Eine konkrete Instanz von Merkmalen wird dabei als Punkt in einem $n$-dimensionalen Merkmalsraum verstanden. *Signifikanz* von Merkmalen zur Lösung einer bestimmten Bildverarbeitungsaufgabe äußert sich dann in der gelungenen Anhäufung von Merkmalen, die von Objekten derselben Klasse stammen, in einem separierbaren Bereich des Merkmalsraumes (Cluster). Aufgabe der Klassifikation ist dann "lediglich" die Abtrennung solcher Bereiche von anderen Bereichen im abstrakten Merkmalsraum. Die Klassifikation beinhaltet auch die *Interpretation* der ermittelten Klasse, z.B. in Bezug auf Fehlerrelevanz bei der Qualitätssicherung, oder dem Einbezug erlaubter Toleranzen in die Gesamtentscheidung.

Einen besonderen Stellenwert im Repertoire der digitalen Bildverarbeitung nimmt die Texturanalyse ein. Texturen sind Muster, die man in natürlichen oder synthetischen Bildszenen vorfindet. Durch die Wiederholung von Feinmustern entsteht dabei der subjektive Eindruck von Uniformität. Texturwahrnehmung spielt eine sehr wichtige Rolle im menschlichen Sehen. Sie dient zur Erkennung und Unterscheidung von Objekten, zur Ableitung von Oberflächenorientierung und Perspektive, und zur Wahrnehmung der Form dreidimensionaler Objekte. Interessant in diesem Zusammenhang ist, daß Texturen durch den Menschen nicht objektiv und präzise, sondern nur subjektiv und unter Benutzung unscharfer Charakterisierungen beschrieben werden können [90] [132].

Texturanalyse ist ein Teilgebiet der Bildverarbeitung, die das Konzept von bildhaft gegebenen Texturen zur Lösung zahlreicher visueller Inspektionsaufgaben einsetzt. Typische Aufgaben, die mittels Methoden der Texturanalyse gelöst werden, sind: Textursegmentierung, Texturfehlererkennung, Textursynthese und Texturbereinigung (siehe Abbildung 2.2).

Zielstellung der Texturfilterung ist es, Bildoperationen auf Basis pixelweiser Berechnungen zu finden, die bestimmten Regionen in Bildern bestimmten Klassen zuordnen. Üblicherweise entstammen die Pixel, deren Grau- oder Farbwerte[1] für

[1] In Folge als Pixelwerte bezeichnet.

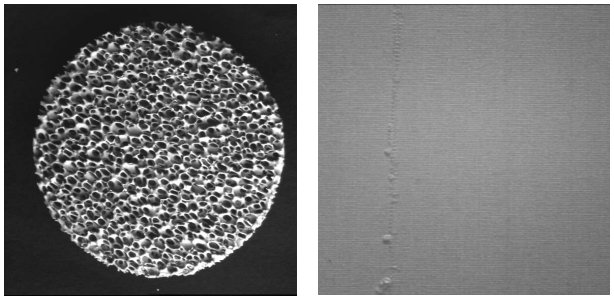



Abbildung 2.2: Zwei Beispiele für Texturen: Keramikfilter (a) und Textilfehler (b).

diese Berechnungen genutzt werden, genau aus den Bildregionen, die klassifiziert werden sollen. Typische Regionen sind dabei

- lokale Nachbarschaften (d.h. ein Pixel und seine direkten Nachbarpixel im Bild),
- ROIs (*regions of interest*) als zusammenhängende, kompakte Teilmengen der Menge aller Pixel,
- „Texturfenster“, also rechteckig begrenzte Teilbereiche des Bildes oder
- das gesamte Bild (globale Berechnungen).

Die Berechnungen aus den Pixeln selbst nutzen dabei eine breite Palette mathematischer oder informationstheoretischer Konzepte, wie z.B. statistische Ansätze, strukturelle Ansätze, dimensionstheoretische (fraktale) und spektrale Ansätze.

Über die Jahre mußte die Bildverarbeitung Lösungen in immer komplexeren Themengebieten entwickeln, für die sich das starre Schema der Merkmalsklassifikation oft als nicht mehr ausreichend erwies. So machte es z.B. die wachsende Anzahl möglicher Optionen bei der Umsetzung der Einzelschritte immer schwerer möglich, eine optimale (hinsichtlich Auswertequalität, Bearbeitungszeit oder Flexibilität) Konfiguration zu finden. In Anlehnung an das biologische Vorbild der Mustererkennung und Signalverarbeitung bei Lebewesen kam es so zum Einbezug von *Lernfähigkeit* in den Prozeß der Bildverarbeitung. Die hierzu existierenden Methoden und Ansätze sollen im folgenden Abschnitt näher vorgestellt werden.

## 2.2 Soft Computing Verfahren

In diesem Abschnitt werden die grundlegenden Verfahren des *Soft Computing* vorgestellt, insbesondere das Perzeptron, das *Multilayer Backpropagation Neural*

*Network* , Fuzzy-Logik, der genetische Algorithmus und die genetische Programmierung. Die Einführung dieser Verfahren erfolgt unter dem Aspekt der Vorbereitung ihrer mathematischen Diskussion, und nicht, um Vollständigkeit zu erlangen. Die Möglichkeiten der Anpassung des Aufwandes zur Steuerung der Genauigkeit oder Erlangbarkeit einer Lösung werden diskutiert.

## 2.2.1 Neuronale Netze

### Das Perzeptron

Ende der 50er Jahre entwickelte eine Gruppe um Frank Rosenblatt und Charles Wightman von der Universität Buffalo das *Mark I Perceptron*, und damit zwar nicht den ersten Neurocomputer überhaupt, aber den ersten Neurocomputer, der nützliche Funktionen (insbesondere zur Mustererkennung) ausführen konnte. In [144] findet sich eine theoretische Darstellung des Perzeptrons. Eine wichtige, dort diskutierte Frage ist die nach dem Zusammenspiel von Neuronen beim Abrufen von Erinnerungen auf Grund einer bestimmten perzeptionellen Aktivierung durch die Sinne. Rosenblatt favorisiert eindeutig den behaviouristischen Standpunkt in Form eines *Stimulus–Response* Systems, das in geeigneter Weise vorkonfiguriert ist, um bestimmte Erkennungsleistungen zu vollbringen, und analysiert die mathematischen Implikationen solch eines Systems.

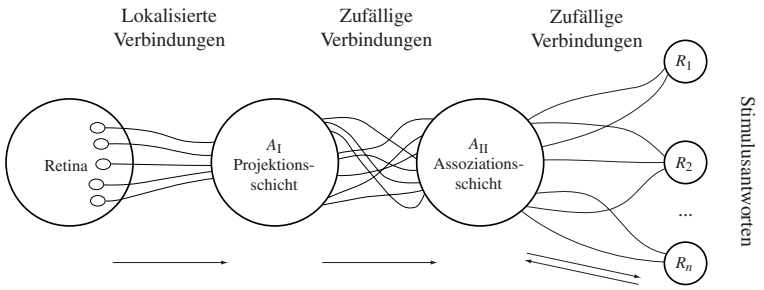



Abbildung 2.3: Organisation des Perzeptrons (nach [144]).

Das ursprüngliche Perzeptron, wie in Abbildung 2.3 dargestellt, besteht aus vier Schichten, in denen folgende Schritte ablaufen:

1. Ein Stimulus gelangt auf die Retina, die aus einzelnen sensorischen Einheiten, den $S$-Punkten gebildet wird.

2. Der dadurch ausgelöste Impuls gelangt zu assoziativen Zellen ($A$-Zellen) in einer „Projektionsschicht“ ($A_I$). Jede $A$-Zelle empfängt dabei solche Stimuli von mehreren Verbindungen zu den $S$-Punkten in der Retina, den sog. Ursprungspunkten. Jede Verbindung kann dabei hemmend oder aktivierend (inhibitorisch oder exhibitorisch) wirken. Die Ursprungspunkte sind in Gruppen nahe beieinanderliegender $S$-Punkte organisiert (was man

heutzutage als „rezeptives Feld“ bezeichnet). Übersteigt die arithmetische Summe von Aktivierungen und Hemmungen einen bestimmten Schwellwert, „feuert“ die entsprechende $A$-Zelle.

3. Zwischen Projektionsschicht $A_I$ und Assoziationsschicht $A_{II}$ liegen Verbindungen zufällig vor. Damit sollte der Tatsache Rechnung getragen werden, daß neuronale Netze in Lebewesen zwar stets ähnlich funktionieren, die Vernetzung der Neuronen untereinander jedoch in jedem einzelnen Lebewesen stets verschieden ist. Ansonsten funktionieren die Verbindungen von $A_I$- und $A_{II}$-Schicht genauso wie die der $A$-Zellen zu den $S$-Punkten.

4. Auch die Antwortzellen (*Response-Units*) $R_1$ bis $R_n$ sind mit $A_{II}$ zufällig miteinander verbunden, jedoch gibt es hier auch eine Rückopplung der Antwortzellen zu den Assoziativzellen.

Ohne auf die mathematische Analyse dieses Modells hier näher einzugehen, fallen in der Arbeit von Rosenblatt immerhin fünf Weichenstellungen auf, die die späteren Forschungen zur Neuroinformatik erst wieder im Rahmen der Etablierung des *Soft Computing*, also gut dreizig Jahre später, und da auch nur zum Teil wieder rückgängig machen sollten:

1. Rosenblatt will nicht vordergründig ein biologisches Modell entwickeln, sondern ein technisches System, welches in der Lage ist, Funktionen vergleichbar zur menschlichen Wahrnehmung nachzubilden. Der akzeptierte Preis hierfür ist, daß solch ein Perzeptron unter Umständen überhaupt nichts mehr mit einem biologischen System gemeinsam haben muß und damit auch nicht mehr viel zu dessem Verständnis beiträgt.

2. Die wichtige Rolle der Statistik als einzige „zuverlässige“ Instanz der Kognition: die in jedem Individuum andere Neuronenvernetzung wird als Indiz dafür genommen, daß die eigentliche Musterwahrnehmung nicht auf der konkreten Form der Vernetzung beruhen kann, sondern für jede Art der Vernetzung gleichermaßen funktionieren muß. Dies geht nur, wenn die Vernetzung selbst wirklich rein zufällig erfolgt ist, der Prozeß der Wahrnehmung jedoch auf einer Art strukturellen Ankopplung an die leichten Abweichungen der konkreten Verteilung von einer idealen zufälligen Verteilung beruht. In diesem Sinne wundert es auch nicht, wenn Rosenblatt am Ende feststellen muß, daß die Ausgaben seines Perzeptrons immer mehr Zufallswerten gleichen, je mehr Klassen es lernen sollte.

3. Anstelle der holistischen Sichtweise der Gestaltpsychologie wird hier einem blinden Pragmatismus zuliebe das Kontrollparadigma wieder eingeführt, welches ein Mustererkennungssystem nur als Eingabe-Ausgabe-System unter Kontrolle der Umgebung begreifen kann. Die Fähigkeiten des Perzeptrons verlagern sich so in den instruktorischen Bereich: richtig konfiguriert, kann es im Prinzip alles.

4. Die aktuell erreichbaren Erkennungsleistungen des Perzeptrons werden als wichtiger erachtet als die Klärung der Frage, was überhaupt mit solch einem Modell theoretisch darstellbar ist. Lediglich der Vorgang des Akquirierens der Fähigkeit zur Wiedererkennung wird (als Zufallsprozeß) näher betrachtet. Damit steht das Perzeptron deutlich im Gegensatz bspw. zum KOLMOGOROV-Netzwerk (siehe Abschnitt 4.4.1), also einer neuronalen Architektur, von der bekannt ist, welche Funktionen sie approximieren kann.

5. Der digitale Computer wird dem analogen als überlegen angesehen, für Satosi Watanabe resultierend aus einem „Cluster von Halbwahrheiten“ ([184], S. 439): die Erkenntnis, daß Neuronen im wesentlichen die zwei Zustände feuernd und nichtfeuernd besitzen; die erfolgreiche Nutzung von Dualzahlen bei der Entwicklung digitaler Computer und die Entdeckung der BOOLE-schen Algebra führten zu dem historischen Vorurteil, daß sich mittels binärer Logik alles auf der Welt erklären, begreifen und nachbilden lassen muß. Auch das Perzeptron wurde als ein binäres System konzipiert.

In den folgenden Jahren wandelte sich das Konzept des Perzeptrons zum *Multi-Threshold-Device* (MTD), dessen eigentliche Urheberschaft zu Anfang der 60er Jahre heute nicht mehr erkennbar ist. Sah man im Perzeptron eher die theoretische und exemplifizierte Basis zur Gestaltung weiterer intelligenter Maschinen, und seitdem auch Rosenblatt als Vater der Neuroinformatik, so war das MTD eine Reminiszenz an die etwas unbefriedigend behandelte Frage der Konfigurierung des Perzeptrons.

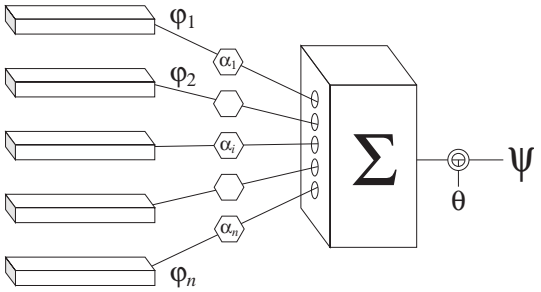



Abbildung 2.4: Architektur des Multi-Threshold-Devices (nach [127], dort als *Perceptron* bezeichnet).

Abbildung 2.4 stellt die Architektur des MTD mit einem Ausgabeneuron dar. Es besteht aus Eingabeneuronen mit den Aktivierungen $\varphi_i$, die direkt aus der Retina stammen und (*sic!*) binärer Natur sind: $\varphi_i \in \{0, 1\}$. Jedes Ausgabeneuron ist nun mit allen Eingabeneuronen verbunden, jeder Verbindung $i$ ist dabei ein Verbindungsgewicht $\alpha_i$ (eine reelle Zahl) zugeordnet. Jedes Ausgabeneuron erhält dadurch eine Aktivierung $\sum_i \alpha_i \varphi_i$. Dieser Wert wird mit einem Schwellwert $\theta$ verglichen. Ist er größer, „feuert“ das Ausgabeneuron, d.h. es ergibt sich

als Ausgabewert $o = 1$, ist er kleiner, der Wert 0. Die Funktionalität des MTD, die also von den Verbindungsgewichten $\alpha_i$ und einem Schwellwert $\theta$ abhängt, läßt sich somit in einer einfachen Formel zusammenfassen:

$$o = \begin{cases} 1 & \text{wenn } \sum_i \alpha_i \varphi_i > \theta \\ 0 & \text{wenn } \sum_i \alpha_i \varphi_i \leq \theta \end{cases} \tag{2.1}$$

Die Vereinfachungen zum Perzeptron sind hier deutlich. So wurde auf die Projektionsschicht verzichtet (was auch schon von Rosenblatt vorgeschlagen worden war), und an Stelle zufälliger Verbindungen zwischen zwei Gruppen von Neuronen wurden hier alle Verbindungen einer Gruppe von Neuronen zu einem Neuron, jedoch dafür gewichtet, eingeführt. Eine Rückopplung von der Antworteinheit gibt es nicht mehr. Alle grundlegenden Paradigmen des Perzeptrons bis auf das statistische Lernen sind nach wie vor erfüllt: technisches (also kein biologisches) Modell, Kontrollparadigma, Leistung ist gleich Erkennungsrate und binäres System.

Dennoch konnte man mit Gleichung 2.1 die Frage stellen, was ein solches neuronales Netz eigentlich alles „in Wirklichkeit“ kann. Die ernüchternde Antwort auf diese Frage kam von Marvel Minsky und Seymour Papert in ihrem Buch „Perceptrons“ [127].

Aus rein geometrischen Überlegungen heraus konnten Minsky und Papert zeigen, daß ein MTD prinzipiell nicht in der Lage ist, zum Beispiel für eine Menge von Punkten zu unterscheiden, ob sie zusammenhängend ist oder nicht.

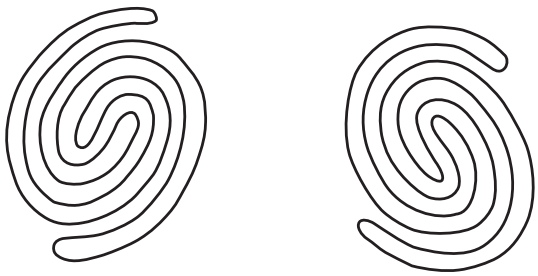



Abbildung 2.5: Das *Intertwined Spirals* Problem: Ein MTD kann nicht unterscheiden, daß die Figur in (a) aus zwei, die Figur in (b) dagegen nur aus einem spiralförmigen Linienzug besteht.

Offenbar hängt dies eng mit dem Problem zusammen, daß *zusammenhängend* auf eine globale Eigenschaft einer Menge von Punkten verweist, während das MTD nur lokale Eigenschaften auswerten kann. Neben diesem Problem lieferten Minsky und Papert auch noch weitere Probleme, die auch heute zum Teil noch nicht gelöst sind. Einige sollen hier benannt werden:

- Das *Intertwined Spirals* Problem (siehe Abbildung 2.5 und die Titelseite des Buches „Perzeptrons“), bei dem es darum geht, zu unterscheiden, ob

es sich bei einem spiralförmigen Linienzug um zwei ineinandergebettete Spiralen oder nur eine handelt (eine auch für den Menschen nicht ganz leichte Aufgabe).

- Das XOR-Problem, für das Minsky und Papert nachweisen konnten, daß alle nicht linear separierbaren logischen Funktionen (von denen XOR nur die einfachste und elementarste ist) durch ein MTD nicht nachgebildet werden können.

- Das Paritätsproblem, bei dem für eine Menge $X$ entschieden werden soll, ob die Anzahl seiner Elemente gerade oder ungerade ist.

- Das Symmetrieproblem, bei dem ein MTD nicht entscheiden kann, ob eine Menge eine Symmetrieachse besitzt oder nicht.

- Das Und/Oder-Problem: Zu jedem $n$ läßt sich ein Paar $(\psi_1, \psi_2)$ von Prädikaten der Ordnung 1 finden, für das die Ordnung der Prädikate $\psi_1 \wedge \psi_2$ und $\psi_1 \vee \psi_2$ größer als $n$ ist. Ein Beispiel dafür ist das „Maskierungsprädikat“: Wenn $A$, $B$ und $C$ Mengen mit $A \cup B \cup C = R$ sind, ist zu entscheiden

$$\begin{array}{l}(\|X \cap A\| > \|X \cap C\|) \wedge (\|X \cap B\| > \|X \cap C\|) \\ \quad \text{oder} \\ (\|X \cap A\| > \|X \cap C\|) \vee (\|X \cap B\| > \|X \cap C\|).\end{array} \tag{2.2}$$

  Auch dies kann durch ein MTD nicht gelöst werden.

Das XOR-Problem fand in der Zwischenzeit seine Lösung durch das *Multilayer Backpropagation Neural Network*, auf das im nächsten Abschnitt näher eingegangen wird. Das erste Problem ist bis heute nicht einmal in seiner „Vorform“, der Erkennung einer Spirale an sich, zufriedenstellend gelöst. Das Paritätsproblem kann heute z.B. durch genetische Programmierung gelöst werden, daß Und/Oder-Problem dagegen scheint vollkommen in Vergessenheit geraten zu sein, auch wenn Minsky und Papert ihm gerade im Zusammenhang von Mengenapproximationen eine sehr große Bedeutung beimessen. Als möglicher Lösungsansatz wurde von Minsky und Papert selbst das mehrschichtige Perzeptron von Gamba [51] benannt. Das Symmetrieproblem kann ebenfalls als bis heute ungelöst betrachtet werden.

### Das Multilayer Backpropagation Neural Network

Blickt man aus der Perspektive Ende der 90er Jahre zurück auf die Entwicklung und Herausprägung der ursprünglichen Momente des *Soft Computing*, fällt besonders der große Einfluß der Publikationen von Rumelhart und McClelland [148] in der *„Parallel and Distributed Computing“* (PDP) Buchreihe der University of California in San Diego zur Thematik *„Backpropagation Network“* (BPN) auf.

Bis in die Mitte der 90er Jahre gehörten diese Publikationen mit zu den meistzitierten Arbeiten der wieder einsetzenden Forschungen zu neuronalen Netzen. Doch nicht nur dadurch zeigte sich der enorme Einfluß dieser Arbeiten. Die Vorstellung des Backpropagation-Algorithmus führte faktisch „über Nacht“ zu einer großen Euphorie unter Wissenschaftlern und Ingenieuren vieler Disziplinen über die Vielzahl sich neu abzeichnender Anwendungen und gab den Forschungen auf dem Gebiet der neuronalen Netze einen neuen Aufschwung[2].

An diesem plötzlich einsetzenden Prozeß war die Demonstration des NETalk-Systems (beschrieben in [152]) nicht ganz unbeteiligt. NETalk war ein System, welches mittels des Backpropagation-Algorithmus lernte, Worte vorzulesen. Das Abspielen einer Tonaufzeichnung des „Anlernens“ zeigte, wie sich sinnloses Gestammel über kindliches Geplapper langsam zu einer ordentlichen Intonation des gesprochenen Wortes wandelte. Das Publikum und die Presse waren „galvanisiert“ ([95], S. 197). Auch heute noch fehlt in keinem Lehrbuch über neuronale Netze die Erwähnung von NETalk.

Was war nun so neu am *Multilayer Backpropagation Neural Network* (MBPN)? Historisch gesehen nichts. Das MBPN war im Prinzip die Subsumption verschiedener Konzepte, die im Zusammenhang mit dem Perzeptron einzeln alle bereits einmal aufgetreten waren. Im wesentlichen waren dies

- Die Einführung einer versteckten Schicht von Neuronen (*hidden layer*), die weder zur Eingabe noch zur Ausgabe direkte Verbindungen unterhält. Auf Seite 16 wurde bereits bemerkt, daß selbst daß ursprüngliche Perzeptron versteckte Schichten besaß. Anfang der 60er Jahre untersuchte Gamba ebenfalls mehrschichtige Perzeptrons [51].

- Die Einführung eines *Bias*-Neurons, welches stets den konstanten Wert 1 hält und die Repräsentation affiner Transformationen ermöglicht. Einen Bias gab es bereits im ADALINE [189].

- Die in der MTD-Gleichung auftretende Vergleichsfunktion

$$compare(x, y) = \begin{cases} 1 & \text{wenn } x > y, \\ 0 & \text{sonst.} \end{cases} \tag{2.3}$$

  wurde durch Einführung der nichtlinearen und vor allen Dingen differenzierbaren Sigmoidalfunktion

$$sig(x) = \frac{1}{1 + e^{-cx}} \tag{2.4}$$

  durch $sig(x - y)$ ersetzt ($y$ ist dabei der *Bias*, der den Schwellwert $\theta$ des MTD ablöst). Damit konnte dann der *Mean Squared Error* Ansatz direkt

[2] Kosko spricht von einer *„second wave“*, wobei die *„first wave“* die Forschungen zum Perzeptron Mitte der 60er Jahre waren.

zur Aufstellung eines Lerngesetzes genutzt werden. Auch das hatte bereits Widrow 1962 für das (lineare) ADALINE getan [189], der Ansatz des MBPN ist hier lediglich eine Verallgemeinerung.

Der aus heutiger Sicht wesentliche Beitrag des Backpropagation-Algorithmus bestand dabei in einer Eigenschaft, die kurz nach seiner Publikation eigentlich Basis für den schwersten Angriff seitens seiner Kritiker wurde: wieder „emanzipierte“ sich ein neuronales Verfahren von der Rolle, die biologischen neuronalen Netze, d.h. die Nervensysteme höherer Organismen und deren Lernverhalten zu modellieren. So stellte Grossberg in einer unmittelbar darauffolgenden Studie folgende Unterschiede zur empirischen Erfahrung des Lernens dar [57]:

1. Der Backpropagation-Algorithmus lernt nicht stabil in komplexen Umgebungen.

2. Der Backpropagation-Algorithmus leitet weder Prototypen noch Kategorien aus den Eingangsdaten ab.

3. Die Gewichte, im biologischen Sinne als Synapsenstärken interpretiert, müssen im Backpropagation-Algorithmus auf „magische Weise“ zwischen neuronalen Schichten transportiert werden. Dies ist der wichtigste Kritikpunkt an diesem Verfahren (auch als Nicht-Lokalität der Berechnungen bezeichnet), da nicht klar ist, welche Instanz z.B. im menschlichen Gehirn für solch einen Gewichtstransport zuständig sein könnte.

4. Daß Erwartungen auf Erkennungsvorgänge einen wesentlichen Einfluß haben, ist sowohl aus der Erfahrung her bewußt, als auch physiologisch in vielfacher Hinsicht bewiesen worden. Im präattentiven Modus der Arbeitsweise biologischer neuronaler Netze wird dies als *Priming* bezeichnet. Eine Umsetzung dieses für die Modellierung von physiologischen Wahrnehmungsprozessen grundlegenden Mechanismus fehlt im Backpropagation-Algorithmus vollständig.

Das Fazit von Grossberg lautet: *„Despite the appeal of this demonstration [gemeint ist das NETalk-System — M.K.], the BP model does not model a brain process ... This shortcoming does not limit the model's possible value in technological applications which can benefit from a steepest descent algorithm, but it undermines the model's usefulness in explaining behavioral or neural data.“*

Im Jahre 1989, also etwa zeitgleich mit Hornik [69] im Zusammenhang mit der Approximierbarkeit in drei Schichten, bewies Hecht-Nielsen ein grundlegendes Theorem über das MBPN.

**Theorem 1 (Hecht-Nielsen).** *Zu jedem $\epsilon > 0$ und einer beliebigen $L_2$-Funktion $f : [0,1]^n \to \mathbf{R}^m$ existiert ein dreischichtiges MBPN, welches $f$ mit einem mittleren quadratischen Fehler kleiner als $\epsilon$ approximieren kann.*

Der Beweis geht über die FOURIER-Zerlegung von $f$ (deshalb die Voraussetzung, daß $f$ zu $L_2$ gehört). Da gezeigt werden kann, daß sich die Sinusfunktion beliebig genau durch ein MBPN approximieren läßt, folgt daraus dann letztlich das eben genannte Theorem. Die Güte der Approximation wächst dabei mit der Anzahl versteckter Neuronen. In praktischen Fällen wird diese Anzahl jedoch extrem groß. Die Einführung mehrerer versteckter Schichten kann hierbei sehr nützlich sein.

An dieser Stelle sei nochmals auf die von Minsky und Papert benannten Probleme verwiesen. So wurde festgestellt, daß ein MTD prinzipiell nicht *zusammenhängend* von *unzusammenhängend* unterscheiden kann. Ist die Bildebene beschränkt, z.B. durch Bilder der Größe $128 \times 128$ Pixel, so ist *zusammenhängend* eine diskrete Funktion $z$ von $128^2 = 16384$ Veränderlichen. Diese Funktion $z$ läßt sich in eine $L_2$-Funktion $Z$ so einbetten, daß sie an den diskreten Gitterpunkten genau die Werte von $z$ annimmt (z.B. durch eine hochdimensionale Splineapproximation). Damit gibt es also mit Sicherheit ein MBPN, welches zusammenhängend erkennen kann, obwohl jedes Eingabeneuron nur auf einen einzelnen Punkt des Bildes zugreift. Das Problem ist eben einfach nur, daß die Anzahl der benötigten versteckten Neuronen extrem groß ist. Und vor allem: die Genauigkeit der Approximation wird nur dadurch besser, daß weitere versteckte Neuronen hinzugenommen werden.

Aus solchen Überlegungen ergibt sich mehr oder weniger, daß neuronale Netze wie das MBPN oder das KOLMOGOROV-Netzwerk prinzipiell alle geometrischen Prädikate ermitteln können, wenn das notwendige Vorgehen auch nicht unbedingt praktisch realisierbar ist.

Dennoch sind diese neuronalen Netze nicht zur Kognition fähig. Dies läßt sich wieder am Beispiel der Spirale sehr gut illustrieren. Wird die Spirale nur anhand einer Menge von Trainingsdaten vorgegeben, kann die Erwartung nicht nur darauf gesetzt werden, daß dieser Datensatz zu hundert Prozent richtig klassifiziert wird, sondern auch, daß Punkte jenseits der Grenzen der vorgegebenen Spirale im Sinne einer *fortgesetzten* Spirale klassifiziert werden. Ein dreischichtiges MBPN zerlegt jedoch den Merkmalsraum (in diesem Falle also die Koordinatenebene) durch Geraden in einzelne Segmente und weist dann jedem Segment eine konkrete Klasse zu. Durch diese Zerlegung der Ebene durch Geraden gibt es jedoch unbeschränkte Segmente, die von jeder Spirale nach einer genügend großen Anzahl von Umläufen geschnitten werden *müssen* (siehe Abbildung 2.6). Damit wird, egal wie gut die Spirale für $k$ Umläufe durch ein MBPN approximiert wurde, diese Approximation bei wachsendem $k$ irgendwann nicht mehr richtig erfolgen. Das MBPN „erkennt“ die Spirale niemals an sich.

### 2.2.2 Fuzzy-Logik

Regeln beschreiben das Operieren mit Wahrheitswerten. Mit dem Wahrheitswert der Regelvoraussetzungen sind auch die Wahrheitswerte der Regelkonklusionen

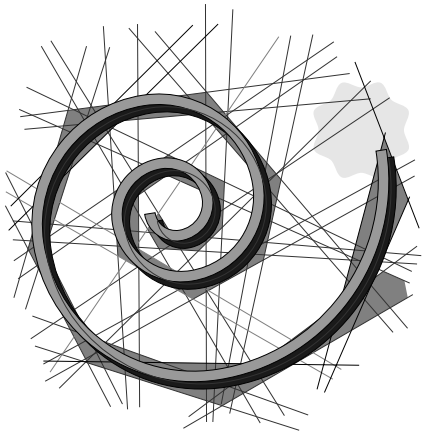


Abbildung 2.6: Theoretische Approximation eines Spiralabschnitts durch ein MBPN. Jeder der 39 Geraden entspricht einem versteckten Neuron. Die Spirale kann nur dadurch in ihrem weiteren Verlauf approximiert werden, daß weitere versteckte Neuronen hinzugenommen werden. Es gibt jedoch keine Konstellation von endlich vielen Geraden, die die Spirale bis ins Unendliche fortgesetzt richtig approximieren könnte.

gegeben. Seit dem Erscheinen des Buches *„Laws of Thought“* von George Boole im Jahre 1854 [18] ist die nach dessen Autor benannte BOOLEsche Algebra der Grundstein jeglichen mathematischen Operierens mit den Wahrheitswerten von Aussagen. In dem Versuch, eine mathematische Theorie zu formulieren, die für Ideen und Gedanken dasselbe leisten kann wie die gewöhnliche Algebra für Zahlen, der auch bereits von Leibnitz unternommen worden war, führte George Boole die Wahrheitswerte WAHR und FALSCH auf eine Algebra der Zahlen 1 und 0 zurück. Die logischen Operationen $\wedge$ und $\vee$ wurden dabei durch die arithmetischen Operationen minimum und maximum ersetzt, die Negation von $a$ durch $1 - a$. Es zeigte sich, daß durch solch eine *zweiwertige Logik* eine konsistente Algebra von sog. Prädikaten gestaltet werden kann.

Der Ansatz, der heutzutage eine fundamentale Rolle in der Computertechnik und der Gestaltung von Schaltkreisen in der Elektrotechnik, und auch lange Zeit bei der Modellierung neuronaler Netze spielte, blieb in der Folge nicht ohne Kritik. Schon die Entwicklung der Quantenphysik in den 30er Jahren dieses Jahrhunderts zeigte, daß zumindest die Natur nicht der BOOLEschen Algebra folgt. Auch der Satz von Gödel, nach dem kein Axiomensystem widerspruchsfrei ist und es stets eine wahre Aussage enthält, die in diesem Axiomensystem nicht beweisbar ist, war mit der scheinbaren Evidenz der zweiwertigen Logik schwer vereinbar. Berühmte Mathematiker wie Charles Sanders Peirce, Bertrand Russell, Max Black und Jan Lukasiewicz haben sich in Folge mit mehrwertigen Logiken beschäftigt. Die Ergebnisse dieser Arbeiten haben jedoch nie ihre „mathematische Spielwiese“ verlassen.

Ende der 60er Jahre machte Spencer-Brown, ein Schüler von Russell, eine wichtige Feststellung im Zusammenhang mit der Fundierung der BOOLEschen Algebra. Er wies auf das vollständige Fehlen einer dazu gehörenden Arithmetik hin. In [162] schreibt er: *„Every algebra has an arithmetic, but Boole designed his algebra to fit logic, which is a possible interpretation of it, and certainly not its arithmetic.“*

Eine Algebra (nicht zu verwechseln mit dem speziellen mathematischen Begriff der Multiplikationsalgebra) liefert die Regeln zur Manipulation einer bestimmten Klasse von Objekten, wie z.B. der natürlichen Zahlen. Vor allem beinhaltet eine Algebra die Beschreibung des Gebrauchs von *Variablen*. Die dazugehörige Arithmetik dagegen befaßt sich mit den Objekten selbst und beschreibt die Beziehungen, die ein solches Objekt eingehen kann. So gehört z.B. das Assoziativgesetz für natürliche Zahlen $a$, $b$ und $c$:

$$(a + b) + c = a + (b + c) = a + b + c \tag{2.5}$$

zur Algebra der natürlichen Zahlen, Aussagen wie: *„7 ist eine Primzahl“* jedoch zur Arithmetik der natürlichen Zahlen. Algebra und Arithmetik stehen also in einem ähnlichen dualen Verhältnis zueinander wie Organisation und Struktur von Systemen.

Die BOOLEsche Algebra liefert nun keine dazu passende BOOLEsche Arithmetik. Operationen wie $\sqrt{\text{WAHR}}$ oder Relationen wie „WAHR läßt bei der Teilung durch FALSCH den Rest FALSCH“ geben mathematisch keinen Sinn.

Eine praktisch einsetzbare numerische Arithmetik logischer Prädikate wurde 1965 von Lotfi Zadeh unter dem Namen Fuzzy-Logik eingeführt [202]. In ihr wird die Funktion, die einer Aussage den Wert 0 oder 1 als BOOLEschen Wahrheitswert zuordnet, nicht von vornherein auf diesen zweielementigen Wertebereich eingeschränkt. Betrachtet werden also allgemeine Wahrheitswertfunktionen $\mu(x)$, die auch mehr als nur zwei Werte annehmen können (in der Regel Werte aus dem Intervall $[0, 1]$).

Dies hat jedoch eine sehr dramatische Konsequenz: der Satz vom ausgeschlossenen Dritten, seit Aristoteles ein Axiom der formalen Logik, kann dann nicht mehr gelten! Formal lautet dieses Axiom, daß für jedes $a$ gelten muß

$$\max(\mu(a), 1 - \mu(a)) = 1. \tag{2.6}$$

Dies kann jedoch nur stimmen, wenn $\mu(a) = 0$ oder $\mu(a) = 1$ ist.

Diese Eigenheit der Fuzzy-Logik ist sehr lange kontrovers diskutiert worden. Aus praktischer Sicht ist es akzeptabel, auf diese Art mit *Nichtwissen* zu operieren.

Eine Menge $\mathcal{M}$ wird in der Fuzzy-Logik durch ihre Zugehörigkeitsfunktion $\mu(x)$ beschrieben. Der Wert von $\mu(x)$ für ein Objekt $x$ beschreibt dabei den (nicht auf 0 oder 1 beschränkten) Wahrheitswert der Aussage, daß $x$ zu $\mathcal{M}$ gehört. Die Interpretation von Werten, die dabei verschieden von 0 oder 1 sind, muß jedoch

nicht immer für eine inherente „Unschärfe“ in der Mengenspezifikation stehen. Andere mögliche Interpretationen sind z.B.:

- Die *Möglichkeit*, daß $x$ zu $\mathcal{M}$ gehören kann (*possibility*).
- Der Grad, zu dem $x$ *typisch* für $\mathcal{M}$ ist (*typicality*).
- Die *Wahrscheinlichkeit*, daß $x$ zu $\mathcal{M}$ gehört (*probability*).
- Herausstellung der Ambiguität der Aussage, daß $x$ zu $\mathcal{M}$ gehört (*ambiguity*).
- Der Grad an Glauben oder Überzeugung daran, daß sich $x$ in $\mathcal{M}$ befindet (*belief*).
- Der eingesparte Aufwand bei der Überprüfung der Aussage, daß sich $x$ tatsächlich in $\mathcal{M}$ befindet (*verification*).

Fuzzy-basierte Regelsysteme erlauben es, mit diesen Formen von Nichtwissen oder unvollständigem Wissen zu operieren. Natürlich erfolgt dies nicht mit dem Ziel, das fehlende Wissen zu ergänzen.

Eine Operation $c(a)$ von $[0,1]$ in $[0,1]$ wird Fuzzy-Komplement genannt, wenn sie folgende Eigenschaften aufweist: $c(0) = 1$, $c(1) = 0$ und für alle $a, b \in [0,1]$ gilt, daß aus $a \leq b$ $c(a) \geq c(b)$ folgt. Zusätzliche Forderungen wären noch, daß $c(a)$ stetig ist und involutiv, d.h. für alle $a$ aus $[0,1]$ gilt $c(c(a)) = a$. Neben $c(a) = 1 - a$ erfüllen auch das SUGENO-Komplement

$$c_\lambda(a) = \frac{1-a}{1+\lambda a} \text{ mit } \lambda > -1, \tag{2.7}$$

welches für $\lambda = 0$ in das Standardkomplement übergeht, und das YAGER-Komplement

$$c_w(a) = (1 - a^w)^{1/w} \text{ mit } w > 0, \tag{2.8}$$

das für $w = 1$ in das Standardkomplement übergeht, diese Forderungen. Beide lassen sich zusammenfassen im $(\lambda, w)$-Komplement

$$c_{\lambda,w}(a) = \left[\frac{1-a^w}{1+\lambda a^w}\right]^{1/w} \tag{2.9}$$

mit $\lambda > -1$ und $w > 0$, das für $\lambda = 0$ dem YAGER-Komplement und für $w = 1$ dem SUGENO-Komplement entspricht.

Es gibt sogar ein Repräsentationstheorem, nach dem sich durch eine sog. ansteigenden Generatorfunktion $g(a)$ stets ein Fuzzy-Komplement definieren läßt. Eine ansteigende Generatorfunktion ist eine streng monoton wachsende, stetige Funktion $g(a)$ mit $g(0) = 0$. Die Pseudoinverse $\tilde{g}(a)$ von $g(a)$ ist definiert als

$$\tilde{g}(a) = \max\left[0, \min\left[1, g^{-1}(a)\right]\right], \tag{2.10}$$

wobei $g^{-1}(a)$ die Umkehrfunktion von $g(a)$ ist.

Eine abfallende Generatorfunktion $f(a)$ ist ähnlich definiert: $f(a)$ ist streng monoton fallend, stetig, und es ist $f(1) = 0$. Die Pseudoinverse $\tilde{f}(a)$ ist genauso definiert wie für $g(a)$.

Ist $g(a)$ eine ansteigende Generatorfunktion, dann ist

$$c_g(a) = \tilde{g}\,[g(1) - g(a)] \tag{2.11}$$

ein Fuzzy-Komplement.

Die logischen Operationen $\wedge$ und $\vee$ werden in der Fuzzy-Logik durch die sogenannten T- und S-Normen (oder T-Konormen) ersetzt. Eine T-Norm ist eine Abbildung von $[0,1] \times [0,1]$ in $[0,1]$ mit folgenden Eigenschaften ($a, b, c$ beliebig aus $[0,1]$):

1. Gebundenheit: $T(a,1) = a$.
2. Monotonie: $b \leq c \rightarrow T(a,b) \leq T(a,c)$.
3. Kommutativität: $T(a,b) = T(b,a)$.
4. Assoziativität: $T(a,T(b,c)) = T(T(a,b),c)$.

Desweiteren kann aus praktischen Gründen gefordert werden:

1. $T(a,b)$ ist stetig.
2. $T(a,a) \leq a$.
3. Aus $a_1 < a_2$ und $b_1 < b_2$ folgt auch $T(a_1,b_1) < T(a_2,b_2)$ (strikte Monotonie).

Die Definition einer S-Norm erfolgt analog. Es ändert sich nur Forderung 1 zu $S(a,0) = a$ und optionale Forderung 2 zu $S(a,a) \geq a$.

Ist $T(a,a) = a$ bzw. $S(a,a) = a$, wird eine solche Norm idempotent genannt. Bisher sind nur die Standardnormen als idempotente Normen bekannt.

Tabelle 2.1 gibt eine Auflistung typischer T- und S-Normen und wird durch die bereits seit 1963 bekannten ursprünglichen vier Definitionen von Schweizer und Sklar in Tabelle 2.2 ergänzt.

Weitere T- und S-Normen lassen sich ebenfalls über Generatorfunktionen bestimmen. Wenn $f(a)$ eine abfallende Generatorfunktion ist, dann ist

$$T_f(a,b) = \tilde{f}\,[f(a) + f(b)] \tag{2.12}$$

eine T-Norm. Ebenso ist mit der ansteigenden Generatorfunktion $g(a)$ die Funktion

$$S_g(a,b) = \tilde{g}\,[g(a) + g(b)] \tag{2.13}$$

| T-Norm | S-Norm |
|---|---|
| Standard | |
| $\min(a,b)$ | $\max(a,b)$ |
| Algebraisches Produkt / Algebraische Summe | |
| $ab$ | $a+b-ab$ |
| Beschränkte Summe / Differenz | |
| $\max[0,a+b-1]$ | $\min[1,a+b]$ |
| Drastischer Durchschnitt / Drastische Vereinigung | |
| $T(a,b)=\begin{cases} a \text{ für } b=1 \\ b \text{ für } a=1 \\ 0 \text{ sonst} \end{cases}$ | $S(a,b)=\begin{cases} a \text{ für } b=0 \\ b \text{ für } a=0 \\ 1 \text{ sonst} \end{cases}$ |
| Hamacher, 1978 | |
| $\frac{ab}{\gamma+(1-\gamma)(a+b-ab)}$ | $\frac{a+b+(\gamma-2)ab}{1+(\gamma-1)ab}$ |
| Bem.: $\gamma > 0$ | |
| Frank, 1979 | |
| $\log_s\left[1+\frac{(s^a-1)(s^b-1)}{s-1}\right]$ | $1-\log_s\left[1+\frac{(s^{1-a}-1)(s^{1-b}-1)}{s-1}\right]$ |
| Bem.: $s>0,\ s\neq 1$ | |
| Dubois und Prade, 1980 | |
| $\frac{ab}{\max[a,b,\alpha]}$ | $1-\frac{(1-a)(1-b)}{\max[1-a,1-b,\alpha]}$ |
| Bem.: $\alpha \in (0,1)$ | |
| Yager, 1980 | |
| $1-\min\left[1,[(1-a)^w+(1-b)^w]^{1/w}\right]$ | $\min\left[1,[a^w+b^w]^{1/w}\right]$ |
| Bem.: $w>0$ | |
| Dombi, 1982 | |
| $\frac{1}{1+\left[\left(\frac{1}{a}-1\right)^\lambda+\left(\frac{1}{b}-1\right)^\lambda\right]^{1/\lambda}}$ | $\frac{1}{1+\left[\left(\frac{1}{a}-1\right)^{-\lambda}+\left(\frac{1}{b}-1\right)^{-\lambda}\right]^{-1/\lambda}}$ |
| Bem.: $\lambda>0$. Für $ab=0$ ist die T-Norm 0, für $(1-a)(1-b)=0$ ist die S-Norm 1. | |
| Weber, 1983 | |
| $\max\left[0,\frac{a+b+\lambda ab-1}{1+\lambda}\right]$ | $\min\left[1,a+b-\frac{\lambda ab}{1+\lambda}\right]$ |
| Bem.: $\lambda>-1$ | |
| Yu, 1985 | |
| $\max[0,(1+\lambda)(a+b-1)-\lambda ab]$ | $\min[1,a+b+\lambda ab]$ |
| Bem.: $\lambda \leq 1$ | |

Tabelle 2.1: Zusammenstellung verschiedener Definitionen für T- und S-Normen für Werte $a,b \in [0,1]$.

eine S-Norm. Gilt für $g(a)$ auch noch $g(1) = 1$, dann ist mit $T(a, b)$ T-Norm auch

$$T^+(a, b) = \tilde{g}\,[T(g(a), g(b))] \tag{2.14}$$

ebenfalls eine T-Norm (allgemeines Repräsentationstheorem für T-Normen). Der in Gleichung 2.12 gegebene „Bauplan“ für T-Normen ist ein Spezialfall dieser allgemeinen Beziehung, wenn als Ausgangsnorm die beschränkte Summe genommen wird (und das Äquivalent des Repräsentationstheorems für S-Normen). Normen nach diesem Schema oder auf Basis des algebraischen Produktes bzw. der algebraischen Summe werden auch als ARCHIMEDIsche Normen bezeichnet.

Es gibt also eine Vielzahl von Möglichkeiten, T- und S-Normen aus Generatorfunktionen oder aus anderen T- und S-Normen zu definieren. Die mathematischen Aspekte der damit zusammenhängenden Fragestellungen sind heute noch lange nicht geklärt. Nicht alle T- und S-Normen gehen aus Generatorfunktionen hervor. Manchmal ist auch, wie im Falle der etwas mystisch wirkenden Definitionen von Dubois und Prade, einfach ein „genialer Blick“ notwendig.

Vorteile beim Einsatz alternativer Normen liegen vor allem in ihrer Anpaßbarkeit an konkrete Systeme begründet. Eine exzellente Darstellung zu T- und S-Normen jüngeren Datums findet sich in [45].

| T- und S-Normen Schweizer und Sklar, 1963 |
|---|
| $T(a,b) = \max[0, a^p + b^p - 1]^{1/p}$<br>$S(a,b) = 1 - \max[0, (1-a)^p + (1-b)^p - 1]^{1/p}$<br>Bem.: $p \neq 0$ |
| $T(a,b) = 1 - [(1-a)^p + (1-b)^p - [(1-a)(1-b)]^p]^{1/p}$<br>$S(a,b) = [a^p + b^p - (ab)^p]^{1/p}$<br>Bem.: $p > 0$ |
| $T(a,b) = \exp\left[-(\mid \ln a \mid^p + \mid \ln b \mid^p)^{1/p}\right]$<br>$S(a,b) = 1 - \exp\left[-(\mid \ln(1-a) \mid^p + \mid \ln(1-b) \mid^p)^{1/p}\right]$<br>Bem.: $p > 0$ |
| $T(a,b) = \frac{ab}{[a^p+b^p-(ab)^p]^{1/p}}$<br>$S(a,b) = 1 - \frac{(1-a)(1-b)}{[(1-a)^p+(1-b)^p-[(1-a)(1-b)]^p]^{1/p}}$<br>Bem.: $p > 0$ |

Tabelle 2.2: Die vier Definitionen von Schweizer und Sklar für T- und S-Normen aus dem Jahre 1963 für Werte $a, b \in [0, 1]$.

Mit dieser Vielzahl von Definitonen, die die Booleschen Maximum- und Minimumoperationen ersetzen können, stellt sich der Zusammenhang zum Satz vom ausgeschlossenen Dritten neu. Überraschenderweise gibt es auch S-Normen, für die dieser Satz weiterhin gültig bleibt! Speziell sind es die beschränkte Summe, die drastische Vereinigung, die S-Norm von Weber und die S-Norm von Yu (wobei die letzteren beiden als Verallgemeinerungen der beschränkten Summe angesehen

werden können). Fuzzy-Logik bietet also ganz nebenbei auch den Freiheitsgrad an, dieses logische Axiom beizubehalten.

Alle Definitionen für T- und S-Normen von zwei Werten $a$ und $b$ können durch das Assoziativgesetz auf die Ermittlung der T- und S-Norm von $n$ Zahlen $a_1, a_2, \ldots, a_n$ aus $[0, 1]$ übertragen werden. Einige Ausdrücke lassen sich dabei auch in geschlossener Form darstellen. So beträgt die algebraische Summe von $n$ Zahlen

$$S_{AS}(a_1, a_2, \ldots, a_n) = 1 - \prod_{i=1}^{n}(1 - a_i) \tag{2.15}$$

oder die allgemeine Form der T-Norm von Frank

$$T_{Frank}(a_1, \ldots, a_n) = \log_s \left[1 + \frac{1}{(s-1)^{n-1}} \prod_{i=1}^{n}(s^{a_i} - 1)\right]. \tag{2.16}$$

Für andere Normen, wie der von Dubois und Prade, läßt sich keine einfache geschlossene Form finden.

T- und S-Normen können zur Definition lokaler Bildoperatoren genutzt werden. Ein durch die Bildfunktion $I(x, y)$ gegebenes Grauwertbild mit einer Bittiefe von 8 (d.h. ein Bild mit $I(x, y) \in \{0, 1, \ldots, 255\}$) wird dabei in ein Grauwertbild $R(x, y)$ umgewandelt, in dem an jeder Position $(x, y)$ des Bildes $I$ die T- oder S-Norm der Grauwerte der vier benachbarten Positionen inkl. der Position $(x, y)$ berechnet wird (die Grauwerte werden dabei vorher durch 255 geteilt, die Normen anschließend mit 255 multipliziert und gerundet). Ebenso kann ein Fuzzy-Komplement zur Definition einer Bildoperation genutzt werden. Abbildung 2.7 zeigt die Ergebnisse solcher Operationen. Dies illustriert, wie unterschiedlich die verschiedenen T- und S-Normen die zu normierenden Daten behandeln, aber auch, welche Rolle diese BOOLEsche Arithmetik in der Bildverarbeitung spielen kann. Die so definierten Bildverarbeitungsoperationen sind der Fuzzy-Morphologie zuzuordnen [85].

### 2.2.3 Genetische Algorithmen

Allgemein bezeichnet man mit evolutionäre Algorithmen eine Familie von Computermodellen, die auf den Mechanismen der natürlichen Auslese und Genetik basieren [188]. Die erste Erwähnung solch eines Algorithmus geht bis auf das Jahr 1957 zurück, als Box den Begriff evolutionäre Operation einführte [19]. Andere folgten diesem Ansatz, der sich jedoch stets auf das Schema *zufällige Modifikation und Auswahl der Besten* zurückführen ließ [49] [50] [20]. Die Rolle der Rekombination findet sich erstmalig in den Arbeiten von Holland [66] und führte, wie bereits erwähnt, im Laufe der nächsten Dekade zur Ausprägung des Konzeptes genetischer Algorithmen.


Abbildung 2.7: T- und S-Normen als Bildverarbeitungsoperationen. Im Originalbild wurde die S-Norm für den Grauwert jeden Pixels und seiner vier Nachbarn als neuer Grauwert genommen (linke Bilder). Dann wurde auf diesem Ergebnis dieselbe Prozedur mit der zur S-Norm dualen T-Norm wiederholt (rechte Bilder). Im unteren Teil sind einige pixelweisen Fuzzy-Komplemente nach Gleichung 2.9 dargestellt.

Typischerweise löst man mittels genetischer Algorithmen Optimierungsprobleme, also der Art durch eine Abbildung $f : X \to Y$ gegebenen Aufgabenstellungen in der Form, daß ein $x$ gesucht ist, dessen zugeordneter Wert $y = f(x)$ eine bestimmte Eigenschaft erfüllt. Genetische Algorithmen kodieren eine potentielle Lösung des Problems (hier also einen $x$-Wert) in eine Datenstruktur, die sich an die Struktur eines realen Chromosoms anlehnt, in der Regel als Bitstring repräsentiert, und wenden genetische Operatoren wie Crossover und Mutation auf diese Strukturen an. Die sich ergebenden neuen Lösungen werden dekodiert zu den $x$-Werten und es wird $y = f(x)$ ermittelt. Der so erhaltene Wert wird in Bezug auf das Optimierungsproblem als Qualitätsmaß oder Fitnesswert (manchmal auch als Kostenwert bezeichnet) betrachtet. Einige genetische Operatoren, wie die Selektion, werden von diesen Fitnesswerten kontrolliert, andere, wie die Mutation (ein kontingentes Ereignis) in der Regel nicht.

Die Implementierung eines genetischen Algorithmus beginnt mit einer Population von „Chromosomen“ (Bitstrings), also der Generation 1. Die einzelnen Elemente werden oft als Individuen bezeichnet.

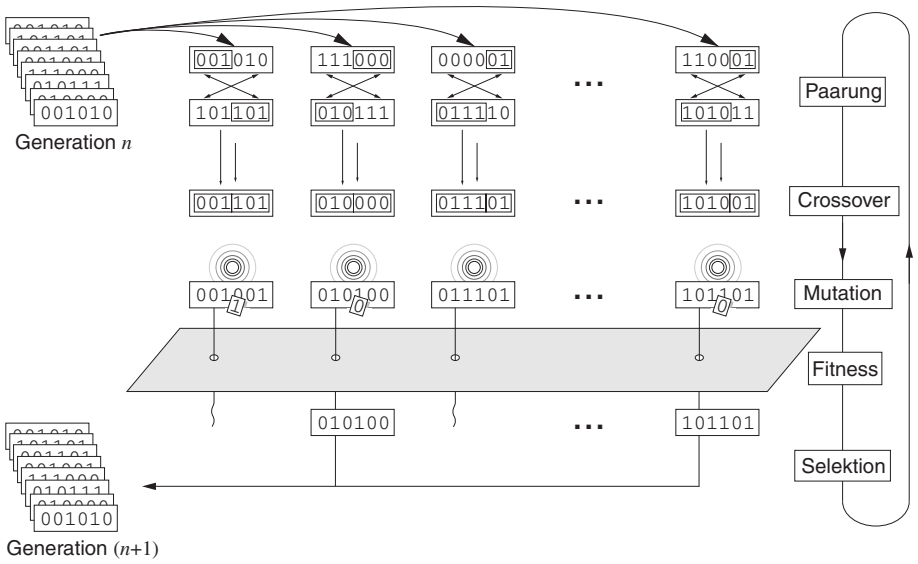



Abbildung 2.8: Schematischer Ablauf eines genetischen Algorithmus.

Danach wird ein Satz von genetischen Operatoren in fester Reihenfolge auf die Individuen der Population angewandt und so eine neue Generation von Individuen erzeugt. Dies wird wiederholt, bis die Folge der Zyklen ein bestimmtes Abbruchkriterium erreicht hat (wie Anzahl der Generation, Fitnesswert der besten Lösung, Änderung des Fitnesswertes in den letzten $n$ Generationen). Schließlich wird aus der Population das Individuum mit der besten Fitness ausgewählt und als Lösung des Optimierungsproblems angesehen.

Abbildung 2.8 stellt den Wechsel von Generation $n$ zur Generation $(n + 1)$ in einem genetischen Algorithmus dar. Zuerst werden $2m$ Bitstrings aus dem Be-

stand der Generation $n$ von $k$ Individuen ausgewählt. Diese Auswahl erfolgt in der Regel durch die Fitnesswerte der auszuwählenden $m$ Paare beeinflußt, also ist z.B. die Wahrscheinlichkeit für ein Individuum, für den Schritt der Paarung ausgewählt zu werden, proportional zu seiner Fitness im Verhältnis zur Fitness aller anderen Individuen. Je besser die Fitness, destso höher die Wahrscheinlichkeit für ein Individuum, sein „genetisches Material“ (also einige seiner Bits) auch in die nächste Generation mit einzubringen.

Sobald die $m$ Paare ausgewählt sind, erfolgt die Rekombination in Form des Crossover. Ein gemeinsamer Trennpunkt wird bei jedem Paar zufällig ausgewählt, und es wird ein neuer Bitstring aus der ersten Hälfte des einen und der zweiten so abgeteilten Hälfte des zweiten Bitstring kombiniert. Danach wird das neu entstandene Individuum mutiert, d.h. jedes seiner Bits wird mit einer bestimmten (in der Regel sehr kleinen Wahrscheinlichkeit) invertiert. Dadurch hat man nun die $m$ *Kinder* der Generation $n$ aus $k$ Eltern gebildet.

Nun wird die Fitness dieser Kinder bestimmt. Einige von ihnen mögen bessere Fitnesswerte aufweisen als die Eltern. Aus den $k$ Eltern der Generation $n$ und den $m$ Kindern werden nun die $k$ besten Individuen ausgewählt (Elitistenselektion), um die Generation $(n + 1)$ zu bilden.

### 2.2.4 Genetische Programmierung

Ursprünglich als „Ausbrüten von Programmen“ bezeichnet, führte zu Beginn der 90er Jahre John Koza mit seinen beiden Büchern [98] und [99] die genetische Programmierung als einen scheinbaren Ableger der genetischen Algorithmen anhand einer Vielzahl von detailliert untersuchten Anwendungsbeispielen ein.

Bei der genetischen Programmierung gibt es vieles von dem, was es bei genetischen Algorithmen auch gibt: Populationen aus Individuen, Generationen, genetische Operatoren wie Crossover und Mutation, und eine Fitnessfunktion. Als wesentlicher Unterschied werden jedoch nicht Bitstrings evolviert, sondern *Computerprogramme*. In Anlehnung an die Computersprache LISP werden dabei diese Programme durch grammatikalische Bäume (kurz: Bäume) repräsentiert. Jeder Baum besteht dabei aus den elementaren Programmzutaten Knoten und Terminalen. Ausgehend von einem Basisknoten verzweigt jeder Knoten des Baums in jeder Tiefe in eine Zahl von weiteren Knoten oder Terminalen. Ein Terminal verzweigt dabei nicht weiter.

Abbildung 2.9 (a) stellt einen solchen Baum dar, der die arithmetische Operationsfolge $2 \times 3 + 5$ repräsentiert. Die Knoten bilden hierbei die arithmetischen Operationen Multiplikation und Addition, die Terminale die Operanden 2, 3 und 5. Das Ergebnis der „Ausführung“ des Baumes ist die Zahl 11. Der Baum in Abbildung 2.9 (b) dagegen enthält als Terminal eine Variable $x$ an Stelle der 2. Damit wird hier die Funktion $3x+5$ dargestellt. In der Programmiersprache LISP hätten diese Bäume die Repräsentation `(+ (* 2 3) 5)` und `(+ (* x 3) 5)`.

Die *Fitness* solch eines Baums ergibt sich damit aus der Ausführung des Baums.

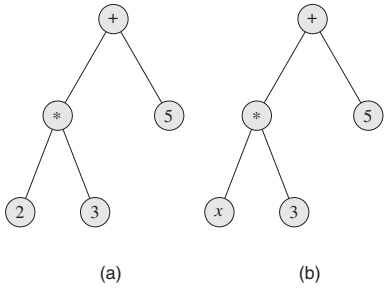



Abbildung 2.9: Grammatikalische Bäume, hier für die Ausdrücke $2\times3+5$ (a) und $3x+5$ (b), wie sie bei der genetischen Programmierung verwendet werden.

Nach der Initialisierung einer Menge zufälliger Bäume (also mittels Zufallszahlen selektierter Knotenfunktionen und zufallsbasierter Strukturierung von Bäumen) werden dann zyklisch die genetischen Operatoren Selektion und Crossover angewendet (seltener auch Mutation).

Selektion erfolgt hier nach dem Turnierprinzip: es werden zweimal jeweils zwei Bäume der Elterngeneration ausgewählt. Dann wird von den beiden Paaren jeweils das mit der besseren Fitness genommen. Man hat dann zwei rekombinationsfähige Elternindividuen.

Crossover dieser so ausgewählten Individuen (also Bäume) erfolgt dann durch zufällige Auswahl von jeweils einem Knotenpunkt in jedem Baum und Austausch der beiden Teilbäume unterhalb dieses Knotenpunktes (siehe Abbildung 2.10). Auf diese Art erhält man neue durch die Bäume repräsentierte Programme. Entsprechend ihrer Fitness, die sich durch ihre Ausführung ergibt, können sie in den Bestand der nächsten Generation mit eingehen.

Um die genetische Programmierung anzuwenden, gibt es kein Kodierungsproblem wie bei den genetischen Algorithmen. Die genetischen Programme besitzen denselben Ausführungsmodus wie das Problem, zu dessen Lösung sie eingesetzt werden. Jedoch ist in der Praxis die richtige Wahl der Knoten- und Terminalfunktionen eher entscheidend. Knotenfunktionen sollten eher komplexe, problemspezifische Funktionen als elementare Funktionen sein.

## 2.3 Anwendungen von Soft Computing in der Mustererkennung

Die im vorhergehenden Abschnitt herausgestellte Vielfalt der Methoden des *Soft Computing* führte natürlich bereits zu einer grossen Anzahl von Anwendungen dieser Verfahren für die gesamten Bandbreite von Aufgaben- und Problemstellungen der digitalen Bildverarbeitung und Mustererkennung. Der Versuch, hierzu eine halbwegs vollständige Übersicht zu geben, würde den Rahmen dieser Arbeit

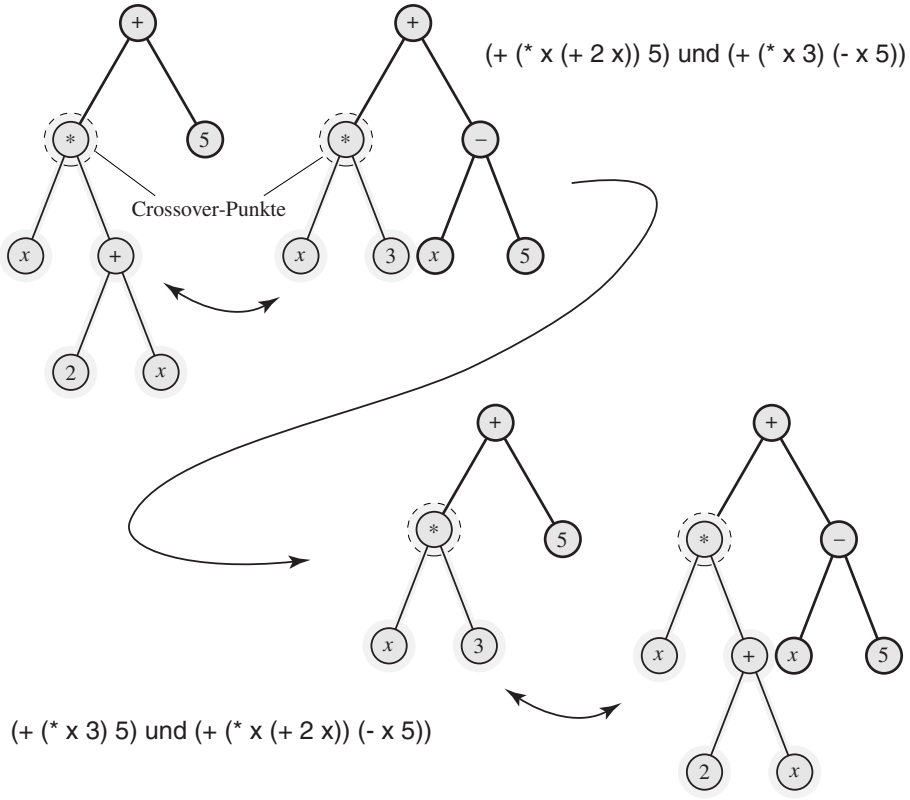



Abbildung 2.10: Crossover-Operation in der genetischen Programmierung: hier werden die Teilbäume unterhalb der markierten Knoten ausgetauscht. Aus den Ausdrücken $x^2 + 2x + 5$ und $4x - 5$ entstehen so die Ausdrücke $3x + 5$ und $x^2 + 3x - 5$.

sprengen, daher im folgenden nur eine grobe und partielle Übersicht zu den Themengebieten.

**Bildfilterung** Die in Abschnitt 2.2.2 angedeuteten Möglichkeiten zur Gestaltung von neuartigen Berechnungen auf Basis des Fuzzy-Kalküls wurden bereits in vielfältiger Weise zur Gestaltung von elementaren Bildoperatoren, speziell Bildfiltern genutzt. In [129] findet sich eine Sammlung von Beiträgen auf diesem Gebiet. Typische Einsatzgebiete dieser Filter sind Bildglättung und Rauschfilterung [35]. Allgemeines Filterdesign (z.B. zum Design von Wavelet-Filtern [73], Morphologischen Filtern [100][201] oder allgemeinen Bildoperatoren [130][6]) nimmt unter diesen Arbeiten einen breiten Raum ein. Fuzzy-Konzepte werden dabei meist zur formalen Spezifikation der Filter selbst genutzt, während evolutionäre Verfahren meist zur parametrischen oder strukturellen Anpassung gegebener Filter an bestimmte Bildtypen genutzt werden. Neuronale Netze können durch ihre Struktur zur direkten Repräsentation von Filtern genutzt werden.

**Bildkompression** Mit den Techniken zur Bildkompression hängen oft Optimierungsprobleme zusammen, die auch mit evolutionären Verfahren behandelt werden können [150][128][168].

**Bilddatenbanken** Hier finden sich Anwendungen in der Indizierung von Bildern, z.B. durch eine automatische Layouterkennung oder Erkennung von Bildkomponenten unter Einsatz neuronaler Netze [7], und für die Retrieval-Techniken selbst, in denen auf Fuzzy-Ansätzen basierende neuartige Konzepte für Ähnlichkeit (*similarity*) entwickelt wurden [29].

**Biometrie** Verfahren des *Soft Computing* kommen bei biometrischen Applikationen eher selten zum Einsatz. Unter den wenigen Ansätzen finden sich frühe, unspektakuläre Arbeiten zum Einsatz neuronaler Netze zur Klassifikation von Fingerabdrücken [190][53]. In [1] kommt ein evolutionäres Verfahren zur Vorverarbeitung von Fingerabdruckbildern zum Einsatz. Als wesentliche Ausnahme erfordern wegen der hohen Komplexität die mit der Handschriftenuntersuchung als spezieller Biometrie zusammenhängenden Aufgabenstellungen jedoch durchaus den intensiven Einsatz von *Soft Computing*, siehe z.B. [47] und [48] für eine Übersicht. Es scheint, dass 'Soft Biometrics' erst dann wirklich entstehen wird, wenn die Anforderungen an die biometrischen Technologien durch die heute üblichen, allzuoft simplifizierenden Algorithmen nicht mehr erfüllt werden können - und das wird erwartungsgemäß der praktische Einsatzfall in naher Zukunft sein.

**Clusterung und Segmentierung** Die Aufteilung von Daten in Gruppen ähnlicher Elemente (Clusterung), oder von Bildern in ähnliche Bereiche (Segmentierung) nimmt einen breiten Stellenwert in der Bildverarbeitung und Mustererkennung ein. Auch die vorliegende Arbeit wird einen Ansatz zur Bildsegmentierung präsentieren. Da die Spezifizierung dieser Aufgabenstellungen *per definitionem* Optimierungskriterien festlegt, finden sich hier auch viele Arbeiten zur Nutzung evolutionärer Verfahren [14] [5][107][115]. Die formalen Aspekte der Problemspezifikation, insbes. des Begriffes 'Ähnlichkeit', können jedoch auch unter Nutzung von Fuzzy-Konzepten bearbeitet werden [80][169][12].

**Objekterkennung und -detektion** In diesem Gebiet haben sich in letzter Zeit leistungsfähige Frameworks etabliert [112][178], die in der Regel interne Anpassungen benötigen. Dazu wurden auch evolutionäre Verfahren und neuronale Netze genutzt [71][52], in der Mehrheit finden sich hier jedoch klassische Verfahren des Maschinenlernens.

**Bewegtbildanalyse** Die Vielfalt der Aufgabenstellungen der Bewegtbildanalyse führte auch zu einer Reihe von Anwendungen des *Soft Computing*, wie z.B.

in [77] (Segmetierung von Videosequenzen), [31] (Detektion von Bewegungen), [15][26] (Targetverfolgung) und [108] (Bestimmung von Bewegungsabläufen) beschrieben.

**Merkmalsextraktion** Merkmalsauswahl und -transformation spielen eine wichtige Rolle bei der Vorbereitung einer Klassifikation. In [135][133] ist beispielhaft beschrieben, wie genetische Algorithmen zur Merkmalsextraktion eingesetzt werden können.

**Klassifikation** Neben der 'klassischen' Nutzung von neuronalen Netzen zur Klassifikation finden sich in letzter Zeit verstärkt auch Untersuchungen zur Kombination von Klassifikatoren. Hier leisten auch evolutionäre Verfahren nützliche Dienste [102][34][159].

**Bildrepräsentation** Eine Stärke der *Soft Computing* Verfahren liegt in ihrer Flexiblität der Datenrepräsentation. Dies ist hilfreich sowohl bei der Rekonstruktion eines Bildes durch einen Modelansatz [122][139][23], als auch zur Gestaltung neuer Ansätze in der Repräsentation von Bildern selbst [125] [174].

**3D-Bildverarbeitung** Wegen der höheren Komplexität der mit 3D-Bildverarbeitung verbundenen Aufgabenstellungen findet auch hier *Soft Computing* eine Vielzahl von Anwendungen, z.B. [173] (modelbasiertes Matching), [70] (Registration von 3D-Objekten), [149] (3D-Objekterkennung), [113][55] (Stereobildauswertung) und [198][106] (Matching von 3D-Kurven und 3D-Flächen).

**Kantendetektion** In der Kantendetektion, einem wichtigen Teilschritt bei der Bildanalyse, werden oft bekannte Algorithmen durch evolutionäre Verfahren adaptiert oder durch Fuzzy-Konzepte generalisiert [13][59].

**Farbbildverarbeitung** Vergleichbar zur 3D-Bildverarbeitung bietet auch die Farbbildverarbeitung die Vielfalt und Komplexität der Aufgabenstellungen, die den Einsatz von Verfahren des *Soft Computing* stimuliert: [116][151] (Quantisierung der Farbwerte), [157][60] (Bildverbesserung) und [110][177] (Farbbildsegmentierung), um nur einige wenige zu nennen.

**Kamerakalibrierung** Der adaptive Charakter einer Kamerakalibrierung bedingt auch Untersuchungen zur Einsetzbarkeit evolutionärer Verfahren auf diesem Gebiet: [72][62].

**Fuzzy-Geometrie** Die Idee zur Nutzung der Fuzzy-Logik zur Beschreibung von geometrischen Objekten wurde bereits von Rosenfeld in den 80er Jahren formuliert [146] [147]. In der Zwischenzeit ist aus der Fuzzy-Geometrie eine eigenständige, sehr fruchtbare Teildisziplin der Bildverarbeitung geworden.

**Dokumentenverarbeitung** Die digitale Auswertung von Dokumenten bieten viele Möglichkeiten zur Nutzung von *Soft Computing*, sowohl die Analyse des Dokumentenlayouts betreffend [7], als auch die Zeichen-, Schrift- und Logoerkennung [33][140][105][117][78].

**Formanalyse** Die Auswertung von Formen, die selbst Ergebnis z.B. einer Bildbinarisierung sind, nutzt oft *Soft Computing* Verfahren zur formalen Repräsentation oder Anpassung von bekannten Verfahren [158][170] [199].

**Affective Computing** Ein neueres Einsatzgebiet der Bildverarbeitung und Mustererkennung ist das *Affective Computing* (auch bekannt als *Emotional Information Processing* oder *Kansei Computing*), mit dem Ziel der Anpassung von informationsverarbeitenden Systemen an den menschlichen Nutzer. Eine typische Aufgabenstellung in diesem Gebiet ist die Erkennung menschlicher Gesichtsausdrücke, die auch mit *Soft Computing* erreicht werden kann[134][145].

**Andere Aufgabenstellungen** Weitere Ansätze, die auf Verfahren des *Soft Computing* basieren, finden sich in der Bildsynthese [32] (Suche von Bildkomponenten, die zur Synthese genutzt werden können), der Umsetzung des Konzeptes der visuellen Routinen mittels evolutionärer Verfahren [9], der Optimierung digitaler Wasserzeichen [155] [181], der Erkennung von Fehlern bei der Übertragung von Videodaten [156] oder der Verbesserung von Superresolution-Algorithmen [182].

Zusammenfassend läßt sich sagen, daß *Soft Computing* Verfahren in sämtlichen Gebieten der digitalen Bildverarbeitung und Mustererkennung zum Einsatz gekommen sind und zum Einsatz kommen. Die Herausstellung dieses Einsatzes selbst findet sich jedoch eher in Arbeiten aus den 90er Jahren, Arbeiten der jüngeren Vergangenheit verweisen weniger direkt auf die Nutzung von *Soft Computing*. Der generelle Anteil ist jedoch über die Jahre konstant geblieben, es findet also gegenwärtig weder eine Intensivierung der Arbeiten auf diesem Gebiet statt, noch eine Reduzierung. In der Regel finden evolutionäre Verfahren meist nur Gebrauch als Optimierungsverfahren, generische evolutionäre Verfahren in der Bildverarbeitung sind eher selten. Neuronale Netze werden in den meisten Fällen zur Klassifikation genutzt, während Fuzzy-Ansätzen die konzeptionellen Dinge vorbehalten sind (etwa zur Definition von Ähnlichkeit). Hybride Verfahren finden sich selten. Der Einsatz von *Soft Computing* nimmt eindeutig zu, wenn das Themengebiet komplexer wird und 'traditionelle' Techniken keine befriedigenden Lösungen mehr liefern.

Wesentlicher Punkt ist, daß das Potential der *Soft Computing* Verfahren in der Vergangenheit eigentlich wenig ausgeschöpft wurde. Nur eine Minderheit der Arbeiten betrachtet überhaupt eine mögliche Fusion aus Ansatz und Methodik. Es kommt nie zum Ausdruck, daß *Soft Computing* hier etwas anderes bieten

könnte als etwa Verfahren der Statistik, der künstlichen Intelligenz oder des Maschinenlernens. Ein Grund dafür liegt offenbar darin, daß *Soft Computing* in der Vergangenheit im wesentlichen nur 'benutzt' wurde. Den Modi des Einsatzes von *Soft Computing* selbst, z.B. den Rahmenbedingungen für die Einsetzbarkeit, den Richtlinien für die problemorientierte Gestaltung der Verfahren, oder den Formalisierungen zugrunde gelegter Objekte wie 'Suchraum', wurde bisher kaum Beachtung geschenkt.

# 3 Aufgabenstellung

## 3.1 Anforderungen an ein Bildverarbeitungssystem

Moderne Bildverarbeitungssysteme müssen einer Reihe von sich teilweise widersprechenden Anforderungen gerecht werden. Ein Schwerpunkt von vorbereitenden Entwicklungsarbeiten kommt dabei der *Konfiguration* des Systems zu, also der Menge von zeitnahen Tätigkeiten, die es in die Lage versetzt, die gestellte Aufgabe zeitfern optimal zu lösen. Die wesentlichen Optimalitätskriterien sind dabei:

- Hohe Systemleistung: numerisch festgelegte Verfahren werden zur Bewertung der Qualität der Lösung der gestellten Aufgabe herangezogen (Benchmarks, Erkennungsquoten, Fehlermaße).
- Hoher Systemdurchsatz: das wesentliche Merkmal hier ist die Verarbeitungszeit, und die Koordination von Abhängigkeiten zwischen verschiedenen Verarbeitungsschritten, aber auch die zeitgerechte Bereitstellung von Ressourcen und Bildkompression beeinflussen die Menge der in einem bestimmten Zeitabschnitt verarbeitbaren Bilddaten.
- Flexibilität der Lösung: Die mit möglichst geringem Aufwand verbundene Übertragung einer Bildverarbeitungslösung in eine verwandte Domäne stellt ebenfalls einen wichtigen Gradmesser für die Qualität dieser Lösung dar.
- Hoher Grad an Autonomie: Dies beinhaltet den effektiven Einbezug von Wissen und Erfahrung des Menschen und einen geringen Grad an direkter Interaktion mit dem Menschen während der Bearbeitung, sowie die Gestaltung eines geeigneten Nutzerinterfaces.

Die in der Systemkonfigurationen zusammengefaßten Tätigkeiten nehmen dabei mehr oder weniger direkt auf diese Optimierungsziele bezug. Aus der Perspektive des *Entwicklers* einer Bildverabeitungslösung erweist sich dies in der Regel als *Auswahltätigkeit* aus einem bekannten Repertoire an Möglichkeiten, die untereinander strukturell und semantisch miteinander vernetzt sind:

- Auswahl von Algorithmen und Verfahren,
- Auswahl geeigneter technischer Geräte,

- Auswahl von praxisrelevanten Bewertungskriterien,
- Auswahl von Regimes der Datenbehandlung.

Die Verkettung einer Reihe von Auswahltätigkeiten hat den direkten Effekt der kombinatorischen Explosion der Anzahl entstehender Optionen (Designs), die eine vollständige Untersuchung aller Möglichkeiten direkt vereiteln. Hat man, um ein sehr vereinfachtes Beispiel zu geben, in der klassischen Bildverarbeitungskette (siehe Abb. 2.1 in Abschnitt 2.1) jeweils zehn Möglichkeiten, eine Grauwerttransformation zur Bildverbesserung durchzuführen, dann zehn Verfahren zur Binarisierung zur Auswahl, ebenso wie zehn Merkmalsberechnungen und zehn Klassifikatoren[1] sind das alleine schon zehntausend Möglichkeiten, die zu untersuchen wären.

Optimierung kann also nur erreicht werden, wenn dem Repertoire an Operationen und Verfahren eine gewisse globale Struktur verliehen werden kann, die es z.B. erlaubt, aus dem Testen einer Option auch etwas bezüglich anderer Optionen abzuleiten. Die Suche innerhalb dieses Repertoires muss dann dieser Strukturierung angepaßt erfolgen, wozu sich natürlich die extrem flexiblen Verfahren des *Soft Computing* besonders eignen.

## 3.2 Aufgabenstellung und Herangehensweise

Ziel dieser Arbeit ist *die Entwicklung eines flexiblen, lernfähigen Systems zur effizienten Konfiguration von Bildverbeitungssystemen als entscheidende Tätigkeit des Entwicklers und unter Beachtung der genannten Optimierungsziele hohe Systemleistung, hoher Systemdurchsatz, hohe Flexibilität der Gestaltung und hoher Grad an Autonomie*.

Dieses Ziel bedingt eine Auseinandersetzung mit der Frage nach den prinzipiellen Möglichkeiten eines solchen Systems. Hierzu sind einige quasi "Naturgesetze" der Lernfähigkeit zu beachten, die z.B. die generelle Erfüllbarkeit des ersten Optimierungszieles, der hohen Systemleistung *für alle Anwendungsfälle gleichzeitig* ausschließen, aber die möglichst hohe Erfüllung anderer Optimierungsziele speziell durch "Schematisierung" und universelle Repräsentationsfähigkeit ermöglichen.

Weiterhin ist eine möglichst generalisierte Repräsentation einer hohen Zahl von Bildverarbeitungsoperationen zu finden, die es unter möglichst einfacher Nutzervorgabe erlaubt, die geeignete Bildverarbeitungslösung nahezu automatisch zu entwickeln und zu verifizieren. Dies benötigt natürlich eine eher abstrakte, formale Betrachtung des Konzeptes einer "Bildverarbeitungsoperation" als Bildoperator, wie sie speziell in der mathematischen Morphologie vorgenommen wird (die

[1] Die Anzahl der über die Jahre alleine in der Literatur vorgestellten Verfahren geht in jedem dieser vier Fälle jedoch schon in die Hunderte.

mathematische Morphologie als Teildisziplin der digitalen Bildverarbeitung führt die Behandlung von Bildfunktionen auf die Mengentheorie zurück). Speziell wird eine Operation der mathematischen Morphologie betrachtet, der 2D-Lookup, der Bildoperatoren selbst als Parameter besitzt und damit in hohem Grade generisch ist. Der generische Charakter erlaubt die direkte Bereitstellung von *Soft Computing* Verfahrens zur Ermittlung einer optimalen Operation: evolutionäre Verfahren, speziell die genetische Programmierung zur Optimierung der Systemleistung, Fuzzy-Operatoren zur Erhöhung der Flexibilität der eingesetzten Operatoren, und der Reduzierung der Nutzerabhängigkeit (=Autonomie) der Lösung sowie neuroanle Netze zur Verbesserung des Systemdurchsatzes.

# 4 Umsetzung der Theoreme des Soft Computing

In diesem Kapitel werden die mathematischen Theoreme vorgestellt, auf deren Basis sich die Möglichkeiten der in Kapitel 2 vorgestellten Verfahren einschätzen lassen. Es wird eine kurze Darstellung des "Ugly Duckling" Theorems gegeben, nach dem Objekte sich nicht auf Basis von logischen Attributen unterscheiden lassen. Auf dessen Basis läßt sich die Frage nach der Repräsentation von Sensordaten zugunsten des datenorientierten Ansatzes klären. Als weiteres Theorem wird das "No Free Lunch" Theorem vorgestellt, nach dem alle Optimierungsverfahren im Durchschnitt genauso gut sind, und es wird ein neuer Beweis präsentiert, der gegenüber dem von Wolpert und Macready gegebenen einfacher ist und die Denkweise hinter diesem Theorem besser illustriert. Diese Sichtweise wird exemplarisch an einigen Beispielen aus der Bildverarbeitung dargelegt.

Ein weiterer dieses Kapitel stellt die Problematik der Funktionsapproximation und -repräsentation heraus. Zwei außerhalb des *Soft Computing* entstandene Theoreme, das Stone-Weierstrass-Theorem und das Kolmogorov-Theorem, garantieren den Verfahren des *Soft Computing* zumindest stets die Existenz einer beliebig genau approximierenden, oder perfekt repräsentierenden Lösung.

Als letztes wesentliches Theorem des *Soft Computing* wird auf das Schemata-Theorem eingegangen, welches insbesondere bei genetischen Algorithmen eine wichtige Rolle spielt. Aus dem Schemata-Theorem heraus läßt sich verstehen, daß die Gestaltung der Fitnessfunktion nicht unbedingt mit dem Ziel erfolgen muß, die Fitnesswerte optimal zu erhalten. Es können auch Anwendungen genetischer Algorithmen gestaltet werden, deren Erkennungsziel die Schemata selbst sind.

## 4.1 Das „Ugly Duckling“ Theorem

### 4.1.1 Formulierung

In diesem Abschnitt wird ein Theorem mit dem etwas illustren Namen „Ugly Duckling“ (zu deutsch: „Theorem vom häßlichen Entlein“, bezugnehmend auf das Märchen von Hans Christian Andersen) vorgestellt. Dieses Theorem wurde 1961 erstmals von Satosi Watanabe während eines Vortrags auf dem jährlichen Treffen der *American Association for the Advances of Science* (AAAS) erwähnt

und dann 1965 in [183] publiziert. Die hier gegebene Darstellung folgt der aus [184].

Betrachtet wird dabei die Fragestellung, wie man mittels Prädikaten Dinge voneinander unterscheiden kann. Man betrachte zwei begrifflich unterschiedene Dinge, wie z.B. die sprichwörtlichen Äpfel und Birnen. Um sie auch logisch zu unterscheiden, wählt man bestimmte Prädikate. So sind Äpfel rund und Birnen länglich (nicht rund), oder Äpfel sind fest, Birnen nicht unbedingt. Die Prädikate lauten also *Rundheit* $R$ und *Festigkeit* $F$.

Je mehr Prädikate sich finden lassen, um zwei Dinge voneinander zu unterscheiden, um so verschiedener sind diese Dinge auch voneinander. Das ist der intuitive Standpunkt.

Aus den gewählten „Basisprädikaten“ *Rundheit* und *Festigkeit* lassen sich nun andere Prädikate zusammensetzen, wie etwa „nicht rund und fest“ ($\neg R \wedge F$), welches weder von Äpfeln noch von Birnen, aber z.B. von Autos angenommen wird. Es gibt nun keinen Grund, dieses Prädikat als wichtiger oder unwichtiger als die anderen beiden Prädikate zur Bewertung des Unterschieds von Äpfeln und Birnen anzunehmen, oder um eine dritte Klasse von Dingen, wie etwa Autos, von Äpfeln und Birnen zu unterscheiden. Es gibt, zumindest formal, noch komplexere Prädikate wie „fest und nicht rund, oder rund und nicht fest“ ($(F \wedge \neg R) \vee (R \wedge \neg F)$). Auch solche Prädikate sind, ohne zusätzliche Kriterien zur Unterscheidung einzuführen, vollkommen gleichberechtigt zu $F$ und $R$.

Wieviele solcher Prädikate gibt es nun? Tabelle 4.1 listet alle Möglichkeiten für die Prädikate *Festheit* und *Rundheit* bis auf logische Äquivalenzen (eine solche Äquivalenz wäre z.B. $(F \wedge \neg R) \vee (F \wedge R) \equiv F$) auf. Dabei stehen $\emptyset$ für das *leere* Prädikat, welches von keinem Objekt angenommen wird, und $U$ für das *universelle* Prädikat, welches von jedem Objekt angenommen wird (das „Ding an sich“).

In dieser Tabelle tauchen nun Äpfel und Birnen jeweils genau acht mal auf. Äpfel und Birnen sind also bezüglich dieser Unterscheidung alleine aus Prädikaten vollkommen gleichwertig, da sie beide acht von sechzehn Prädikaten gemeinsam haben. Das „Ugly Duckling“ Theorem ist nun genau die Verallgemeinerung dieser Aussage für eine beliebige Anzahl von Prädikaten. Der Name des Theorems rührt daher, daß im Endeffekt es keine Menge von Prädikaten gibt, die es schlüssig erlaubt, daß häßliche Entlein aus dem berühmten Märchen von dem stolzen Schwan zu unterscheiden, in den es sich am Schluß verwandelt. Der mathematische Hintergrund dafür ist nicht schwer einzusehen, wenn man sich den Sachverhalt mittels sogenannter Euler-Diagramme veranschaulicht.

In Abbildung 4.1 sind die Prädikate $F$ und $R$ als Mengen dargestellt. Äpfel gehören dann zum Durchschnitt der Mengen $F$ und $R$, $F \cap R$, Birnen zur Menge $\neg(F \cup R)$. Offensichtlich gibt es in diesem Euler-Diagramm vier Bereiche, die durch Mengenoperationen nicht weiter verkleinert werden können, sogenannte Atome $\alpha_1, \ldots, \alpha_4$. Diese Atome lassen sich durch $F$ und $R$ folgendermaßen beschreiben:

| | |
|---|---|
| $\emptyset$ | - |
| $F$ | A |
| $R$ | A |
| $\neg F$ | B |
| $\neg R$ | B |
| $F \wedge R$ | A |
| $F \vee R$ | A |
| $F \wedge \neg R$ | - |
| $\neg F \wedge R$ | - |
| $\neg F \wedge \neg R$ | B |
| $\neg F \vee R$ | A, B |
| $F \vee \neg R$ | A, B |
| $\neg F \vee \neg R$ | B |
| $(F \wedge R) \vee (\neg F \wedge \neg R)$ | A,B |
| $(F \wedge \neg R) \vee (R \wedge \neg F)$ | - |
| $U$ | A, B |

Tabelle 4.1: Zusammengesetzte Prädikate aus *Rundheit* $R$ und *Festigkeit* $F$ und die Erfüllung dieser Prädikate durch Äpfel (A) und Birnen (B).

- $\alpha_1 = \neg F \wedge \neg R$,
- $\alpha_2 = F \wedge \neg R$,
- $\alpha_3 = F \wedge R$ und
- $\alpha_4 = \neg F \wedge R$.

Jedes Prädikat aus Tabelle 4.1 entspricht nun genau einem bestimmten Satz von solchen Atomen. So ist z.B. das Prädikat $\neg F \vee \neg R$ darstellbar durch $\alpha_1 \cup \alpha_2 \cup \alpha_4$. Da es 16 Kombinationen der vier Atome gibt, ergeben sich auch genau 16 Prädikate (bis auf logische Äquivalenzen). Die Anzahl der Atome, die ein Prädikat

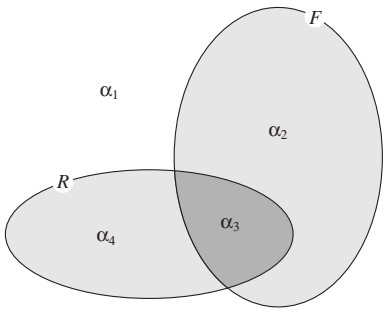



Abbildung 4.1: Euler-Diagramm für die Prädikate $F$ und $R$ sowie die atomaren Teilmengen $\alpha_1, \ldots, \alpha_4$, die durch $F$ und $R$ abgeteilt werden.

beschreiben, wird als *Rang* des Prädikats bezeichnet. Das eben gegebene Beispiel ist also ein Prädikat vom Rang 3.

Auf der anderen Seite sind durch ein gegebenes EULER-Diagramm mit den Atomen $\alpha_1, \alpha_2, \ldots, \alpha_m$ zwei Objekte genau dann unterscheidbar, wenn sie mit genau zwei Atomen $\alpha_i$ und $\alpha_j$ korrespondieren. Insgesamt kann also eine Menge von Prädikaten soviele Objekte unterscheiden, wie es Möglichkeiten gibt, Paare von Atomen aus dem entsprechenden EULER-Diagramm auszuwählen (sozusagen die „Kapazität“ einer Menge von Prädikaten). Im Beispiel sind dies genau sechs Objekte. In Abbildung 4.1 entsprechen z.B. Äpfel dem Atom $\alpha_3$ und Birnen dem Atom $\alpha_1$. *Ähnlichkeit* zweier Objekte kann nun daran gemessen werden, wieviel Prädikate von ihnen gemeinsam geteilt werden. Diese Anzahl kann folgendermaßen hergeleitet werden: Es gibt bei $m$ Atomen kein Prädikat vom Rang 1, das von $\alpha_i$ und $\alpha_j$ geteilt werden kann. Es gibt ein Prädikat vom Rang 2, nämlich $\alpha_i \cup \alpha_j$. Für Rang 3 stehen zwei von den drei Atomen fest ($\alpha_i$ und $\alpha_j$), das dritte kann aus den verbleibenden $m-2$ Atomen frei ausgewählt werden. Dafür gibt es $\binom{m-2}{1} = m-2$ Möglichkeiten. Allgemein gilt für den Rang $r$, daß zwei Prädikate feststehen und die restlichen $r-2$ aus den verbleibenden $m-2$ Atomen frei ausgewählt werden können, wofür es $\binom{m-2}{r-2}$ Möglichkeiten gibt. Damit folgt für die Gesamtzahl der gemeinsamen Prädikate

$$\sum_{r=2}^{m} \binom{m-2}{r-2} = (1+1)^{m-2} = 2^{m-2}. \tag{4.1}$$

Im Beispiel mit den Äpfeln und Birnen gibt es also $2^{(4-2)} = 4$ gemeinsame Prädikate, wie durch Tabelle 4.1 bestätigt wird.

Damit hängt jedoch die Anzahl der gemeinsamen Prädikate nicht von der Wahl von $\alpha_1$ und $\alpha_2$ ab! Das ist der Inhalt des „Ugly Duckling“ Theorems von Satosi Watanabe:

**Theorem 2 (Watanabe).** *Bei Verwendung einer endlichen Menge von Prädikaten, die in der Lage sind, alle Objekte einer vorgegebenen Menge von Objekten voneinander zu unterscheiden, ist die Anzahl der Prädikate, die irgendwelche zwei Objekte aus dieser Menge gemeinsam haben, konstant, unabhängig von der Wahl dieser Objekte.*

### 4.1.2 Diskussion des „Ugly Duckling“ Theorems

Die zentrale Aussage von Theorem 2 ist also, daß sich Objekte niemals eindeutig durch ihre Eigenschaften beschreiben lassen, da die Anzahl der Prädikate, die sie von beliebigen anderen Objekten unterschieden, unabhängig von diesen anderen Objekten ist. Die wesentliche Voraussetzung dafür ist jedoch, daß alle zusammengesetzten Prädikate als gleichbedeutend betrachtet werden können. Der Ausweg besteht nun darin, die Prädikate zu *wichten*, ein Prozeß, den Watanabe

*Ponderation* nennt. Dieser Ponderation liegt jedoch stets eine gewisse Willkür zugrunde. Um ein Objekt von einem anderen anhand von Prädikaten zu unterscheiden, wertet man unter allen möglichen Prädikaten genau die höher, die eine solche Unterscheidung erst ermöglichen. Es zeigt sich also, daß Unterscheidungen zwischen Objekten stets gradueller und nicht logischer Struktur sind, und wir finden hier also den primären Ansatz der Fuzzy-Logik wieder.

## 4.2 Das „No Free Lunch“ Theorem

### 4.2.1 Hintergrund

Das „No Free Lunch“ Theorem (NFL-Theorem) wurde 1995 von David H. Wolpert und William G. Macready, damals Wissenschaftler am Santa Fe Institut, aufgestellt. Es stellt im wesentlichen die Behauptung auf, daß alle Algorithmen zur Lösung von Optimierungsaufgaben im Durchschnitt genauso gut geeignet sind. Das NFL-Theorem wurde ursprünglich nur in Form eines technischen Berichts publiziert [193], später dann folgte eine stark überarbeitete Fassung in einer Fachzeitschrift [194]. Es basiert auf vorangehenden Arbeiten derselben Autoren im Zusammenhang mit dem generalisierenden Lernen aus Testdaten (siehe [192] und [191]). Unter dem Namen „No Free Lunch“ Theorem erlangte es jedoch durch eine Podiumsdiskussion während der *International Conference on Genetic Algorithms* in Pittsburg, PA im Juli 1995 und der nachfolgenden, über das Internet geführten intensiven Diskussion eine sehr große Verbreitung.

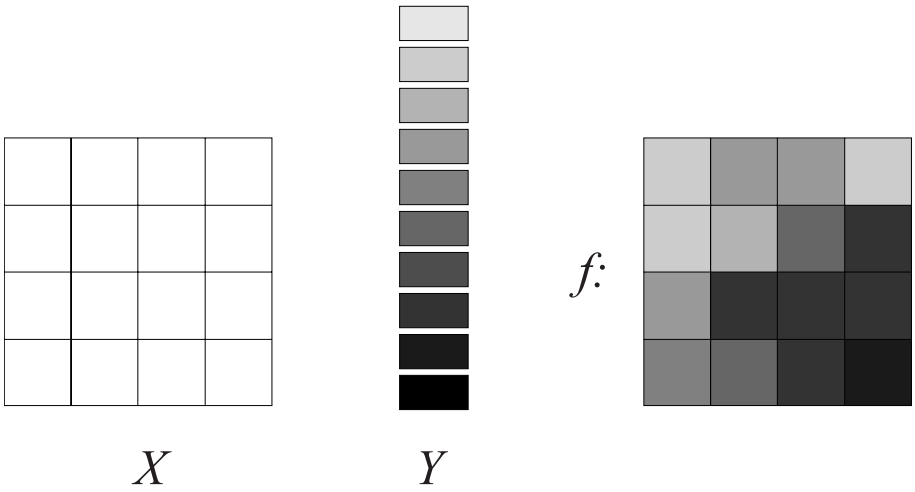



Abbildung 4.2: Zur Definition des Begriffes Suchraum am Beispiel einer Bildfunktion: $X$ ist eine Menge von 16 Punkten, die eine Teilmenge eines zweidimensionalen Gitters bilden, und $Y$ ist die Menge von 10 möglichen Grauwerten.

Der Inhalt des NFL-Theorems wurde schnell zu einem wissenschaftspolitischem

Faktum. Nachdem über mehrere Jahre hinweg die „Verfahrensingenieure“ im wesentlichen das Bild der methodischen Neuentwicklungen des *Soft Computing* geprägt hatten, und alle diese Entwicklungen aus dem Bestreben heraus erfolgt waren, bekannte Verfahren nur verbessern zu wollen, schien nun, im Zwielicht dieses Theorems, jede weitere Bemühung um die Verbesserung der bekannten Verfahren einfach nur fragwürdig. *„In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.“* ([193], S. 1).

### 4.2.2 Formulierung

In den folgenden Ausführungen wird der in [193] und [194] verwendeten Notation so weit wie möglich gefolgt, die dort fehlenden Definitionen sollen jedoch hier ergänzt werden. Ebenso wird hier eine vereinfachte und neue Version des Beweises des NFL-Theorems präsentiert, die ohne Bezug auf bedingte Wahrscheinlichkeiten auskommt.

Gegeben seien zwei finite Mengen $X$ und $Y$. Die Menge $X$ soll hier als Suchraum bezeichnet werden, die Menge $Y$ als Kostenraum.

**Definition 1.** Ein Kostenfunktion $f$ ist eine Abbildung $f : X \rightarrow Y$, deren Projektion in $X$ gleich $X$ ist.

Die vorab genannte Unterspezifikation des Suchraumes entspricht hier der Tatsache, daß an die Kostenfunktion $f$ keine weitere Forderung erfolgt außer daß es sich um eine beliebige Abbildung von $X$ in $Y$ handelt. In [193] wird dies als Uniformität von $P(f)$ bezeichnet und soll die Eigenschaft, kein *a priori* Wissen zu benutzen, mathematisch umsetzen.
Sei nun $m$ eine natürliche Zahl kleiner oder gleich $|X|$.

**Definition 2.** Eine Testsequenz $d_m$ der Länge $m$ von $f$ ist eine Teilmenge $\{(d_m^x(i), d_m^y(i))\}, \quad i = 1, \ldots, m$ und $1 \leq m \leq |X|$ des Graphen von $f$. Dabei sind alle Elemente $d_m^x(i)$ paarweise voneinander verschieden. Speziell sei die leere Menge eine Testsequenz der Länge 0.

(Siehe Abbildungen 4.2 und 4.3 zur Erläuterung der Definitionen.) Mit $d_m^x$ wird im folgenden die Projektion von $d_m$ in die Menge $X$ und mit $d_m^y$ die Projektion von $d_m$ in die Menge $Y$ bezeichnet.

**Definition 3.** Ein (datengetriebener) Algorithmus $a$ ist eine Abbildung, die jeder Testsequenz $d_m$ der Länge $m$ von $f$ ein Element aus $X \setminus d_m^x$ zuordnet (siehe Abb. 4.4).

In dieser Darstellung wird also durch die Anwendung eines Algorithmus $a$ auf eine durch $f$ vorgegebene Kostenfunktion nach $m$ Schritten eine Folge von Paaren $(d_m^x(i), d_m^y(i))$ generiert. Der $(m + 1)$-te Schritt besteht dann in der Auswahl

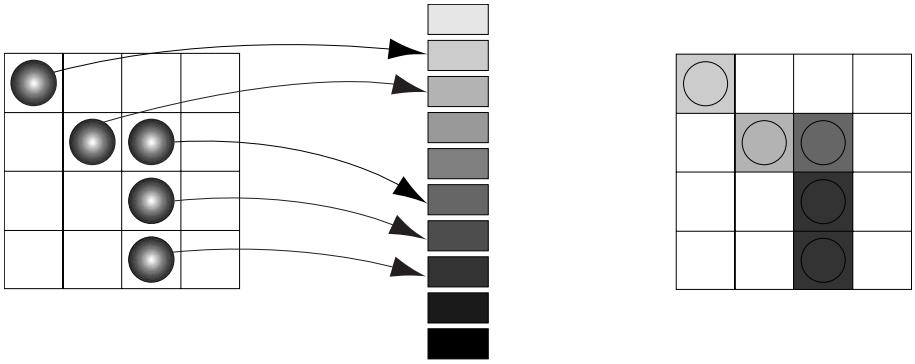



Abbildung 4.3: Zur Definition des Begriffes Testsequenz: im Beispiel der Abbildung 4.2 wurden fünf $x$-Werte ausgewählt. Außer den durch $f$ zugeordneten $y$-Werten ist durch die Testsequenz nichts weiter über $f$ bekannt.

eines weiteren Elementes $x$ aus $X$, welches noch nicht in $d_m^x$ aufgetreten ist. Durch Hinzunahme des Paares $(x, f(x) = y)$ zur Testsequenz $d_m$ entsteht eine Testsequenz $d_{m+1}$ der Länge $m+1$ von $f$. Diese wird auch als Ergebnis der $(m+1)$-maligen Anwendung des Algorithmus $a$ auf die Kostenfunktion $f$ bezeichnet.
Für $m = |X|$ liegt eine vollständige Suche vor, der Algorithmus $a$ bricht hier notwendigerweise ab.
Im folgenden soll $\delta(d_m^y, (f, m, a))$ den Wert 1 annehmen, wenn die Projektion der $m$-maligen Anwendung des Algorithmus $a$ auf die Kostenfunktion $f$ in $Y$ gleich $d_m^y$ ist, 0 sonst.
Mit diesen Definitionen ist das „No Free Lunch“ Theorem von Wolpert und Macready in [194] formuliert als:

**Theorem 3.** *Für zwei beliebige Algorithmen a und b gilt:*

$$\sum_f \delta(d_m^y, (f, m, a)) = \sum_f \delta(d_m^y, (f, m, b)) = |Y|^{|X|-m}.$$

Der Beweis findet sich im Anhang A.
Aus diesem Theorem ergibt sich noch folgendes Korollar:

**Korollar 1.** *Zu jedem Algorithmus a und jeder Menge $d_m^y$ von m Werten aus Y (Wiederholungen erlaubt) existiert mindestens eine Kostenfunktion f, so daß a auf f angewendet nach m Schritten genau eine Testsequenz erzeugt, deren Projektion in Y gleich $d_m^y$ ist.*

Der Beweis findet sich im Anhang A.
Das NFL-Theorem läßt sich weiter spezifizieren, indem auf $Y$ eine Indizierung vorgegeben wird, durch die jedem $i$ aus $\{1, \ldots, |Y|\}$ in eineindeutiger Weise ein $y_i \in Y$ zugeordnet wird.

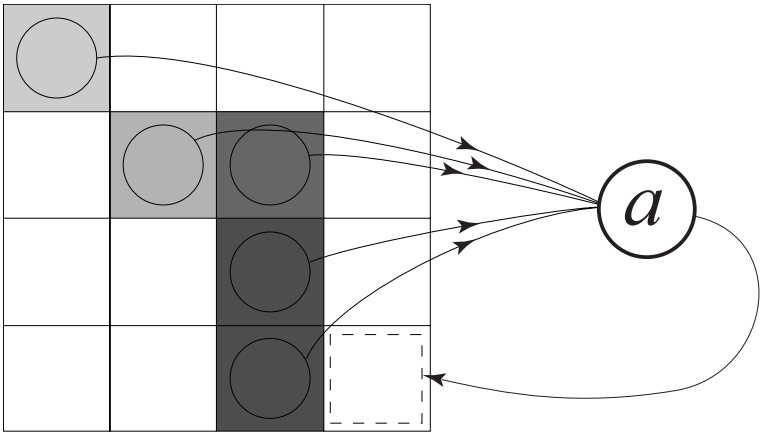



Abbildung 4.4: Zur Definition des Begriffes Algorithmus: In Abhängigkeit von den bisher untersuchten $(x, y)$-Paaren trifft ein Algorithmus eine Auswahl für den nächsten zu untersuchenden und von allen bisher untersuchten verschiedenen $x$-Wert.

**Definition 4.** Bei gegebenem Algorithmus $a$ und Kostenfunktion $f$ ist der Performance-Vektor $c$ nach $m$ Schritten definiert als das Histogramm der $d_m^y$ (siehe Abbildung 4.5):

$$c_i = |\{j \mid d_m^y(j) = y_i\}|$$

Der Performance-Vektor kann besser benutzt werden als quantitatives Maß für die Leistung eines Algorithmus als die Testsequenzen. Ist insbesondere das durch die Kostenfunktion definierte Suchproblem dadurch gegeben, daß ein Wert $x$ zu finden ist, für den $f(x) = y_{opt}$ optimal in einem definierten Sinn ist (z.B. möglichst groß zu sein, wenn $Y$ eine finite Teilmenge der reellen Zahlen ist), dann kann die Leistung eines Algorithmus $a$ an der Größe der Komponente $c_m(y_{opt})$ gemessen werden.
Ähnlich wie bei der Formulierung des Theorems 3 nehme $\delta(c, (f, m, a))$ den Wert 1 an, wenn der Algorithmus $a$ auf die Kostenfunktion $f$ angewendet nach $m$ Schritten eine Testsequenz $d_m$ liefert, aus der sich der Performance-Vektor $c$ ergibt, 0 sonst; $\delta(c, d_m^y)$ sei 1, wenn $c$ genau der Performance-Vektor von $d_m^y$ ist, 0 sonst.
Aus Theorem 3 ergibt sich dann [193]

**Satz 1.** *Für zwei beliebige Algorithmen a und b gilt:*

$$\sum_f \delta(c, (f, m, a)) = \sum_f \delta(c, (f, m, b)).$$

Der Beweis findet sich im Anhang A.
Das heißt aber, daß die relative Häufigkeit, mit der man einen bestimmten Vektor $c$ erhält, wenn man einen Algorithmus $a$ $m$-mal auf alle möglichen Kostenfunktionen $f$ anwendet, nicht von der Gestaltung des Algorithmus $a$ selbst abhängt!

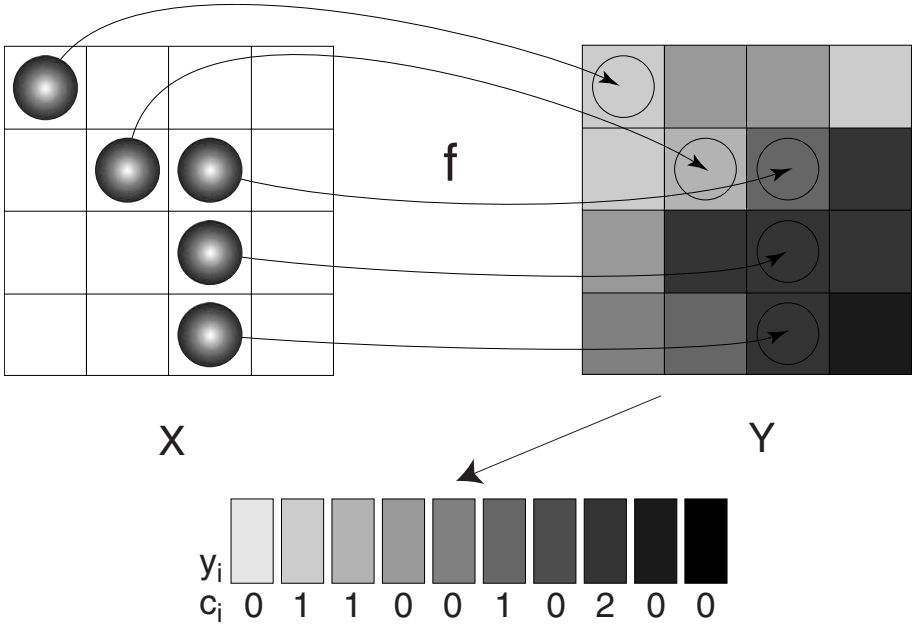



Abbildung 4.5: Bestimmung des „Performance“-Vektors $c$ aus $f$, $m$ und $a$. Der Algorithmus $a$ wählt bei gegebener Kostenfunktion $f$ $m = 5$ Werte aus $X$ aus, denen $f$ fünf Werte aus zehn möglichen aus $Y$ zuordnet. Dann ist $c$ der Häufigkeitsvektor der zugeordneten $y$-Werte, für diesen Fall also $(0, 1, 1, 0, 0, 1, 0, 2, 0, 0)$.

Damit hängt aber auch der Erwartungswert $E_f[c]$ nicht von $a$ ab, d.h. Algorithmen sind bezüglich dieser „Perfomance“, wenn sie an allen möglichen Problemen $f$ gemessen wird, nicht unterscheidbar. Anders gesagt, im Mittel sind alle solchen Algorithmen gleich gut beim „Abtasten“ des Suchraums $\mathcal{X}$, egal wie elaboriert ihre internen Rechnungen sind. *Random sampling* ist so gut wie *smart sampling*.

### 4.2.3 Der Grundgedanke des NFL-Theorems

Der Grundgedanke des NFL-Theorems läßt sich klar veranschaulichen: da der Algorithmus kein *a priori* Wissen nutzt, ist nach $m$ Schritten jede Testsequenz gleichwahrscheinlich. Damit ist auch, gemessen an allen möglichen Kostenfunktionen, auch jeder Kostenwert im $(m+1)$-ten Schritt gleichwahrscheinlich, egal, wie der Algorithmus $a$ auch vorgehen mag. Für bestimmte Probleme (d.h. bestimmte Funktionen $f$) mag es sein, daß ein besserer Kostenwert gefunden wird. Dafür gibt es dann aber gezwungenermaßen auch Kostenfunktionen $f$, für die der nächste Schritt des Algorithmus $a$ einen schlechteren Kostenwert liefern *muß*! Oder, um es anders zu formulieren: egal, wieweit der Algorithmus $a$ bereits die Kostenfunktion „erforscht“ hat, für jeden folgenden Versuch kann diese Kostenfunktion nach wie vor einen beliebigen Wert liefern. Das ist der eigentliche Inhalt des „No Free

Lunch“ Theorems.

An dieser Stelle sollen noch einmal alle Voraussetzungen klar benannt werden, auf denen der Beweis beruht:

1. Der Suchraum $X$ und der Raum der Kostenwerte $Y$ sind finit. Daß NFL-Theorem kann nicht direkt auf den Fall infiniter Such- oder Kostenräume übertragen werden.

2. Es wird kein *a priori* Wissen über $f$ genutzt. Das ist mit Sicherheit eine wesentliche „Schwachstelle“ für die Gültigkeit des Theorems. Jede Gruppe von Suchverfahren macht Annahmen über den Charakter der Lösung, seien sie auch noch so unscheinbar. Die Identifikation von *a priori* Wissen in einem Suchproblem erlaubt also nach wie vor die Qualifikation der Leistungsfähigkeit von Suchverfahren. In [193] wird dieser Fakt als Nicht-Uniformität der Verteilungsfunktion $P(f)$ betrachtet. In der darauffolgenden Diskussion wird jedoch unterschieden, ob der Algorithmus $a$ diese Nicht-Uniformität nutzen kann oder nicht. Ein hinreichendes und notwendiges Kriterium für die Gültigkeit des NFL-Theorems in Bezug auf die Verteilung der Testfunktionen wurde in [44] vorgestellt.

3. Jeder Punkt des Suchraumes taucht höchstens einmal in der Sequenz $d_m$ auf. Dies stellt keine wesentliche Einschränkung für den Beweis dar, da, wie in [193] bemerkt wird, dann lediglich die um alle mehrfachen Erscheinungen von Suchpunkten „bereinigten“ Folgen $d_m$ so betrachtet werden müssen, als ob sie die Ergebnisse einer Variante $a'$ des Algorithmus $a$ wären.

4. Es wird keine vollständige Suche durchgeführt. Zwar wäre in diesem Falle das NFL-Theorem zwar auch richtig, würde aber nur eine Tautologie darstellen.

Die Kritik an dem NFL-Theorem hat sich unabhängig von den hier gemachten Bemerkungen auf folgende Punkte konzentriert:

- Es sind nicht alle Funktionen $f$ praktisch relevant. Eigentlich taucht sogar nur eine Minderheit aller konstruierbaren Kostenfunktionen in praktischen Aufgabenstellungen auf.

- Die Natur hat den Evolutionsprozeß als optimale Strategie entwickelt, die mittels natürlicher Selektion jedes Anpassungsproblem lösen kann. Daher scheint es also einen überperformierenden Algorithmus zu geben.

- Die im Beweis eingeführten bedingten Wahrscheinlichkeiten sind keine, da die entsprechenden Terme, wie oft genug in [193] bei Zwischenrechnungen benutzt wurde, stets 0 oder 1 ergeben. Überhaupt sind die eingeführten Maße (z.B. $c$ als „Performance“-Maß) weder als solche verifiziert worden, noch entsprechen sie irgendwelchen in der Mathematik anwendbaren Konstrukten.

- Es widerspricht dem gesunden Menschenverstand, anzunehmen, daß das Raten einer Lösung in der Praxis genauso gut funktionieren soll wie der Einsatz elaborierter Suchverfahren.

Abschließend sei auf einen Aspekt hingewiesen, der die eigentliche Rolle des NFL-Theorems am stärksten herausstellt. Generell hat die Aussage des NFL-Theorems eigentlich keine große praktische Bedeutung, da nahezu alle Suchverfahren *a priori* Wissen einzusetzen erlauben und auch einsetzen. Was das NFL-Theorem jedoch ermöglicht, ist eine eher pauschale Sichtweise auf Algorithmen an sich. Gemäß dieser Sichtweise sind zuallererst einmal *alle Algorithmen gleich*. Damit liefert das NFL-Theorem den entscheidenden Rückhalt für viele der folgenden Betrachtungen, bei denen es ja darum geht, prinzipielle Möglichkeiten von Verfahren an sich aufzuzeigen.

### 4.2.4 Beispiele für die Anwendung des NFL-Theorems

In der Mustererkennung, speziell in der Bildverarbeitung, werden sehr oft Algorithmen eingesetzt. Hier weicht jedoch der Begriff des Algorithmus etwas von dem in Definition 3 ab, da nicht primär die Suche nach einem optimalen Kostenwert im Vordergrund steht. Dennoch läßt sich der formale Apparat des NFL-Theorems auch auf Bilder anwenden. Dies soll im folgenden anhand einiger Beispiele illustriert werden.

#### Merkmalsklassifikation

Sei $D$ eine beschränkte Teilmenge aus $Z \times Z$ und $G$ eine beschränkte Teilmenge von $N$ (inklusive 0).

**Definition 5.** Ein Bild $B$ ist eine Abbildung von $D$ in $G$.

In diesem Sinne werden Elemente von $D$ auch als Koordinaten oder Bildpositionen bezeichnet, und die Elemente von $G$ auch als Grau- oder Intensitätswerte. Die Paare aus Bildpositionen $(x, y)$ und zugeordneten Grauwerten $g$ nennt man auch Pixel (von engl. *picture element*).

Betrachten wir nun als erstes Beispiel die Klassifikation von Bildern mittels Merkmalen, also den klassischen Ansatz in der Bildverarbeitung (siehe Abschnitt 2.1). Ziel ist die Bestimmung der Zugehörigkeit eines Bildes zu einer bestimmten Klasse aus einem Satz vorgegebener Klassen, die als *a priori* bekannt angenommen werden. Aus dem Bild $B$ werden auf Basis aller $[(x, y), g]$-Paare nach einem vorgegebenem und auf jedes Bild anwendbarem Schema $M$ ein Satz von $m$ numerischen Werten ermittelt, die sogenannten Merkmale. Die Literatur ist voll von Ansätzen zur Ermittlung solcher Merkmale, als illustrierendes Beispiel sei hier nur der Mittelwert aller Grauwerte über alle Bildpositionen genannt. Ziel der Klassifikation ist die Zuordnung von Bildern via der Merkmale zu Klassen.

Mittels einer weiteren Vorschrift, dem sog. Klassifikator $C$, werden aus den Merkmalen Klassennummern abgeleitet. Dies kann formal auch als direkte Berechnung aus dem Bild selber angesehen werden. Wesentlich ist dabei die Annahme der Existenz einer vollständigen Klassifikation (also der Existenz einer *ground truth*). Das verfolgte Schema ordnet prinzipiell jedem Bild eine Klasse zu, und damit würde die Bewertung der Klassifikation auch davon ausgehen müssen, daß sie auf der Basis der Bewertung der korrekten Klassifikation *aller* Bilder erfolgt - jede andere Einschränkung würde die Bewertung zu einer Funktion dieser Einschränkung machen.

In der formalen Übertragung entspricht die Menge aller $f$ hier der Menge aller möglichen Klassifikationen aller Bilder $B$. Es zeigt sich dann nach dem NFL-Theorem:

**Satz 2.** *Die mittlere Bewertung eines Klassifikators über alle möglichen Klassifikationen aller Bilder ist für jeden Klassifikator gleich.*

Vereinfacht drückt dies die Tatsache aus, daß ein Klassifikator zwar eventuell zwei Bildklassen immer unterscheiden kann, aber nicht absolut die eine Menge von Bildern etwa Klasse 1 und nicht Klasse 2 zuordnen kann. Anders gesagt: die Syntax der Klassifikation kann sich nicht aus der reinen Datenvorgabe erschliessen. Ein Austausch z.B. der Klassenbezeichner selbst ist stets möglich.

Damit entsteht das eigentlich kaum wahrgenommene Problem, daß sich die Leistung eines Klassifikators eigentlich gar nicht bewerten läßt. Die übliche Praxis zur Messung solch einer Leistung besteht ja darin, die relative Anzahl richtiger Klassifikationen in einer vorgegebenen Menge von Bildern zu ermitteln. Diese Performance hängt jedoch eindeutig von der Auswahl der Bilder selbst ab. Interpretiert wird diese Messung jedoch in der Regel als Prognose für die relative Häufigkeit bei einer noch größeren Anzahl von Bildern (z.B. wenn die OCR-Software einem Kunden verkauft wird, der sie dann täglich einsetzt). Das kann jedoch nicht stimmen, denn mit der Vergrößerung der Anzahl von Bildern wird auch der Grad an Gleichverteilung zunehmen, und das richtige Klassifikationsergebnis nur noch mit einer Wahrscheinlichkeit von $1/N$ (mit $N$ der Anzahl möglicher Klassen) eintreten. Die Frage nach einer optimalen Auswahl von Bildern zur möglichst objektiven Bestimmung der Leistung eines Merkmalklassifikators ist also mathematisch nicht formalisierbar: entweder man wählt nur „kooperative“ Beispiele aus, dann wird die Leistung sehr hoch sein, oder man wählt *beliebige* Beispiele, dann ist die Leistung gering, ihr Wert trivial, und ebenfalls nicht relevant für den Einsatzfall.

### Vordergrund-Hintergrund-Trennung

Die Vordergrund-Hintergrund-Trennung ist eine positionsweise Klassifikation der Pixel eines Bildes in zwei Klassen, eben dem Bildvordergrund und dem Bildhinter-

grund. Beispiel hierfür ist z.B. die Separation von Handschriften auf Dokumenten, oder die Separation von Prägeschrift auf Prägehintergrund.

Die Entscheidung, daß ein Bildpunkt $(x, y)$ zum Bildhintergrund gehört, geschieht in der Regel algorithmisch auf Basis der Auswertung einer Menge von benachbarten Bildpositionen (inkl. des Bildpunktes selber). Wie im Falle der von einem Algorithmus $a$ generierten Sequenzen $d_m^y$ ist auch hier jede Sequenz generierter Grauwerte, betrachtet über alle möglichen Bilder, gleichwahrscheinlich, und damit auch die Zuordnung eines Punktes zum Vorder- oder Hintergrund.

Das Problem entsteht hier folgendermaßen: zu einem gegebenem Vordergrund $V$ betrachte man die Menge aller Bilder $B_v$ mit beliebigen Grauwerten an den Hintergrundpositionen (also dem Komplement der Vordergrundpositionen im Bild). Dann gilt der

**Satz 3.** *Es gibt keinen Algorithmus, der in allen Bildern in $B_v$ stets die Positionen des gegebenen Vordergrunds $V$ und nur diese der Klasse Vordergrund zuordnet.*

Betrachtet man Fälle, in denen der „ergänzte“ Hintergrund Kopien von Teilen von $V$ enthält (und solche Bilder gehören zu $B_v$), ist dies offensichtlich (gemäß dem Sprichwort: „*Es ist schwierig, einen schwarzen Kater im Dunkeln zu fangen*“).

Ansonsten gelten die im letzten Unterabschnitt gemachten Bemerkungen hier genauso.

### Multivariates Ranking

Bei einer Reihe von Bildoperationen wird auf Ordnungsrelationen in den Grauwerten, z.B. in der Nachbarschaft eines Pixels, Bezug genommen. Ein Beispiel hierfür sind die Rankoperatoren, bei denen aus $B$ ein neues Bild generiert wird, welches an der Position $(x, y)$ den Grauwert vom Rank $n$ aus der Nachbarschaft des Pixels $(x, y)$ im Bild $B$ erhält.

Bei der Übertragung solcher Operatoren auf Farbbilder stößt man jedoch auf das Problem der Nichtexistenz einer kanonischen Ordnungsrelation auf Vektoren. In einem Farbbild ist jeder Bildposition nicht ein Grauwert $g$ sondern ein Farbwert in Form eines Tupels von Intensitätswerten (z.B. (Rotanteil, Grünanteil, Blauanteil)) zugeordnet. Diese Tupel entsprechen formal Vektoren im $R_n$.

Der verbreitete Ansatz zum Sortieren von Vektoren ist die sog. reduzierte Ordnung. Dazu benötigt man eine skalare Funktion $f$, die jedem Farbwert mittels der Werte seines Tupels einen reellen Wert zuordnet. Farbwert $f_1$ wird als größer als Farbwert $f_2$ festgelegt, genau dann wenn $f(f_1) > f(f_2)$ ist. Die Fälle, in denen verschiedene Farbwerte denselben Wert von $f$ erhalten, müssen dabei durch Sonderfestlegungen so behandelt werden, daß zwei Farbwerte nur dann gleich sind, wenn sie in allen Elementen ihres Tupels übereinstimmen (auch *totale Ordnung* genannt).

Ein Beispiel: sind die Farbwerte durch ihre Rot, Grün- und Blauanteile gegeben (jeweils Werte aus $G$) als $(r, g, b)$, dann könnte zum Sortieren die Funktion $f(r, g, b) = r + g + b$ genutzt werden. In den Fällen, in denen die Summe der Farbkomponenten gleich ist, wird der Wert genommen, der den höheren Rotanteil hat. Sind auch diese gleich, wird nach dem Grünanteil entschieden.

Der Punkt ist, daß wenn es keine Einschränkung bezüglich der Spezifikation einer solchen Funktion $f$ gibt (woher sollte sie kommen?), auch alle Sortierungen möglich sind.

**Satz 4.** *Zu jeder vorgegeben Permutation einer Menge paarweise verschiedener Farbwerte existiert eine skalare Funktion $f$, die diese Farbwerte genau in dieser Folge mittels reduzierter Ordnung sortiert.*

Dies folgt gerade aus dem Korollar 1. Man sollte sich also im Falle von Rankoperatoren für Farbbilder auch nach anderen Ansätzen umsehen (siehe [89] für ein multivariates Rankingschema, welches nicht auf eine reduzierte Ordnung zurückführbar ist).

### Bildnachbarschaftsoperatoren

Die Anwendung von Bildnachbarschaftsoperatoren (kurz: Bildoperatoren) in der Bildverarbeitung geht bis auf die frühen Arbeiten von Rosenfeld et al. zurück. Dazu wird, wie oben schon im Fall des Rankoperators kurz beschrieben, jeder Bildposition mittels der Grauwerte seiner unmittelbaren Nachbarn ein neuer Grauwert zugeordnet. Ein Bildoperator besteht also aus einer Vorschrift zur Auswahl der Nachbarn (üblicherweise die 4er- oder 8er-Nachbarschaft in einem rechteckigen Bildgitter inkl. des Bildpunktes selber) und einer Funktion $f$, die den Grauwerten dieser Nachbarschaft einen neuen Grauwert zuordnet, der an dieser Position dann im Ergebnisbild eingetragen wird. Beispiele hierfür sind der Laplace-, Sobel-, Robertson- oder Kirsch-Operator, die Grauwertdilatation und -erosion, oder auch die Rankoperatoren.

Natürlich hat auch für diese Operatoren das NFL-Theorem, insbes. in Form des Korollars 1, eine Interpretation. Dazu betrachten wir ein Bild $B_1$ als *kompatibel* zu Bild $B_2$, wenn an je zwei Positionen, an denen die Grauwerte der Nachbarschaften in Bild $B_1$ übereinstimmen, auch die Grauwerte an den korrespondierenden Positionen in Bild $B_2$ übereinstimmen.
Dann gilt

**Satz 5.** *Zu jedem zu einem Bild $B_1$ kompatiblen Bild $B_2$ existiert ein Bildoperator, der aus Bild $B_1$ Bild $B_2$ generiert.*

Dabei sollte erwähnt werden, daß die geforderte Kompatibilität von Bildern in der Praxis keine relevante Einschränkung darstellt. Bilder, die z.B. mit CCD-Kameras akquiriert wurden, weisen selbst in homogen wirkenden Bereichen genügend Schwankungen innerhalb benachbarter Grauwerte auf, um schon überhaupt das Vorhandensein zweier gleicher Nachbarschaften im Bild auszuschließen.

Die Spezifikation solch eines Operators erfolgt genauso wie im Beweis des NFL-Theorems. Man kann dazu beide Bilder als eine Art Lookup-Funktion benutzen: um den Wert von $f$ für eine Menge von Grauwerten zu erhalten, versuche man zuerst, diese Menge von Grauwerten als eine Nachbarschaft in Bild $B_1$ zu finden. Existiert eine Position $(x, y)$ mit genau solch einer Nachbarschaft, ist der Wert von $f$ genau der Grauwert an dieser Position im Bild $B_2$. Existiert er nicht, kann der entsprechende Grauwert frei gewählt werden.

Das NFL-Theorem selbst besagt in diesem Fall, daß im Durchschnitt über alle Bildoperatoren jedes zu einem Bild $B$ kompatible Bild als Ergebnis der Anwendung des Bildoperators gleichwahrscheinlich ist.

## 4.3 Funktionsapproximation und -repräsentation

Auch die Approximation und Repräsentation von Funktionen stellt ein Suchproblem dar. Hierbei steht jedoch mehr die Frage nach der Repräsentierbarkeit einer unbekannten (also zu suchenden oder zu modellierenden) Funktion durch die Art der Approximation oder dem Schema der Repräsentation im Vordergrund. Eine wichtige Entwicklung auf diesem Gebiet waren die neuronalen Netze, die sich ursprünglich im Zusammenhang mit der Lösung des 13. Hilbertschen Problems durch Kolmogorov ergaben (siehe Abschnitt 4.4). Es zeigt sich, daß sich neuronale Netze direkt aus der Theorie der Funktionsapproximation und -repräsentation ableiten lassen, d.h. ohne Anleihen bei biologischen Modellen wie z.B. der Funktionsweise des Gehirns zu machen. Der hier betonte Vorteil dieser Herangehensweise liegt in der unmittelbaren Möglichkeit, grundlegende Aussagen über die Leistungsfähigkeit neuronaler Netze auf Basis grundlegender Theoreme der Theorie der Funktionsapproximation, wie dem Stone-Weierstrass-Theorem und dem Kolmogorov-Theorem treffen zu können. Einige wesentliche Grundtatsachen zur Approximation von Funktionen sind im Anhang B zusammengestellt.

### 4.3.1 Repräsentierbarkeit von Funktionen

Die Problemstellung der *Repräsentation* von Funktion ist allgemeiner als die der Approximation von Funktionen. Dabei geht es um die Frage nach hinreichend genauer *Berechenbarkeit* einer Funktion $f$ nach einem bestimmten, vorgegebenem Schema.

Als ein solches Schema soll hier die Berechenbarkeit (*computability*) in Schichten betrachtet werden. Sei $f$ eine stetige Abbildung von $R_n$ in $R$.

**Definition 6.** Die Funktion $f$ wird als *in einer Schicht berechenbar* bezeichnet, wenn einer der folgenden Bedingungen erfüllt ist:

- $f$ ist eine lineare Funktion $f(x_1, \ldots, x_n) = w_0 + w_1 x_1 + \ldots + w_n x_n$ mit reellen Werten $w_0, w_1, \ldots, w_n$; oder

- $f$ ist eine Funktion $f(x)$ von einer Variablen.

**Definition 7.** Die Funktion $f$ wird als *berechenbar in m Schichten* bezeichnet ($m \geq 2$), wenn eine der folgenden Bedingungen erfüllt ist:

- $f$ kann als Linearkombination der Form $w_0 + w_1 f_1(x_1, \ldots, x_n) + \ldots + w_k f_k(x_1, \ldots, x_n)$ mit stetigen Funktionen $f_i$, die in $m-1$ Schichten berechenbar sind, dargestellt werden; oder
- $f$ kann in der Form $g(h(x_1, \ldots, x_n))$ dargestellt werden, wobei $g$ eine stetige Funktion einer Variablen ist und $h$ in $m-1$ Schichten berechenbar.

Zwei Fragen stehen hier im Mittelpunkt: die Frage nach der minimalen Anzahl von Schichten, um beliebige stetige Funktionen berechnen zu können, und die Frage nach der minimalen Anzahl von Schichten, um Funktionen berechnen zu können, die beliebige stetige Funktionen $f$ approximieren.

Die zweite Frage findet ihre Antwort im Stone-Weierstrass-Theorem. Der konkrete Beweis wurde von Hornik [69] gegeben und es erwies sich, daß hier $m = 3$ gilt:

**Satz 6.** *(Hornik) Jede stetige Funktion $f : [0,1]^n \rightarrow R$ kann durch Funktionen approximiert werden, die in 3 Schichten berechenbar sind.*

Ergänzend dazu gilt

**Satz 7.** *Es existiert eine Funktion $f : [0,1]^n \rightarrow R$, die nicht durch in 1 oder 2 Schichten berechenbare Funktionen approximiert werden kann.*

Damit kann jede beschränkte stetige Funktion durch Funktionen, die in genau drei Schichten berechenbar sind, approximiert werden.

Zur Beantwortung der ersten Frage haben wir somit schon den Hinweis, daß sicher mehr als drei Schichten benötigt werden, um jede stetige Funktion auf $[0,1]$ zu berechen. Es kann gezeigt werden, daß die Funktion $x_1 x_2 \cdot \exp((x_1^2 + x_2^2)/2)$ *nicht* in drei Schichten berechenbar ist. Es sind also mindestens vier Schichten notwendig. Der Frage, ob dies auch *ausreichend* ist, wird im nächsten Abschnitt durch die Betrachtung des Kolmogorov-Theorems nachgegangen.

## 4.4 Das Kolmogorov-Theorem

Im Jahre 1900 formulierte der Mathematiker David Hilbert anläßlich einer Rede auf dem Internationalen Mathematikerkongreß in Paris dreiundzwanzig damals noch ungelöste mathematische Probleme, die aus seiner Sicht durch die zukünftige Entwicklung der Mathematik im 20. Jahrhundert gelöst werden sollten [65]. Diese Probleme, die unter dem Namen „Hilbertsche Probleme“ in die Geschichte eingingen, wurden zum größten Teil in den nächsten zwanzig bis dreißig Jahren

gelöst. Andere Probleme, wie z.B. die Klärung der Kontinuumshypothese (Problem Nummer 1: *Gibt es eine Menge, die weder zur Menge der natürlichen Zahlen noch zur Menge der reellen Zahlen gleichmächtig ist?*) erfolgte erst Ende der 60er Jahre (durch den Beweis, daß es nicht bewiesen werden kann). Bei einigen Problemen sind selbst heute noch nicht alle Aspekte vollständig geklärt.

Zu jedem der Probleme gab Hilbert einen Kommentar ab und versuchte, die zu erwartende Antwort auf dieses Problem auch zu prognostizieren. Speziell bei seinem Problem Nummer 13 sollte er sich mit seiner Prognose irren, wenn es auch lange Zeit nicht so aussah. Um es vorwegzunehmen: hätte Hilbert Recht behalten, würde heute kein Mensch sich ernsthaft mit neuronalen Netzen, Fuzzy-Logik oder Agenten beschäftigen.

Um zu erklären, worum es bei diesem dreizehnten Hilbertschen Problem geht, betrachte man z.B. die Lösungsformel für quadratische Gleichungen der Form $x^2 + px + q = 0$ einmal etwas anders. Sie lautet, wie bekannt ist

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}. \tag{4.2}$$

Da $p$ und $q$ die freien Parameter der quadratischen Gleichung darstellen, hat man also so die Darstellung des Wertes für $x$ als Funktion der *beiden Veränderlichen* $p$ und $q$ gewonnen: $x = x(p, q)$. Wesentlich ist hierbei der formale Aufbau der Lösungsformel, die zur Ermittlung von $x$ aus $p$ und $q$ führt: es ist eine Sequenz von Operationen, die nur von *einer* Veränderlichen abhängen, und zusätzlichen Additionen von jeweils zwei Zwischenwerten. Noch einmal in Teilschritten:

1. $t_1 = -\frac{p}{2}$ ist eine Funktion von $p$ allein.
2. $t_2 = \frac{p^2}{4}$ ist ebenfalls eine Funktion nur von $p$.
3. $t_3 = -q$ ist eine Funktion von $q$.
4. $t_4 = t_2 + t_3$ ist eine einfache Addition von zwei Werten.
5. $t_5 = \sqrt{t_4}$ ist eine Funktion von nur einer Veränderlichen.
6. $x_{1,2} = t_6 = t_1 \pm t_5$ ist eine einfache Addition (oder Subtraktion).

Diese Form von Ausdrücken, also die Superpositionen der Grundrechenarten und Funktionen einer Veränderlichen der Form $\sqrt[n]{t}$ sind sog. Radikale. Auch für Gleichungen dritten und vierten Grades lassen sich die Wurzeln durch aus ihren Koeffizienten gebildeten Radikalen allgemein beschreiben, wenn auch so umständlich, daß diese Lösungsformeln keine praktische Rolle spielen. In den in der Nacht vor dem tödlichen Duell gemachten Aufzeichnungen des gerade mal 23 Jahre alten Emile Galois fand sich später der über zweihundert Jahre von den Mathematikern gesuchte Beweis, daß sich Gleichungen fünften und höheren Grades nicht mehr durch Radikale lösen lassen.

Dennoch, und damit nähert man sich dem 13. Hilbertschen Problem, blieb die Frage, ob dies vielleicht gehen könnte, wenn man sich nicht nur auf die Wurzelfunktion einschränken würde. Angenommen, man würde also eine größere Menge von Funktionen nur *einer* Veränderlichen zulassen (wie z.B. $\sin x$): gibt es dann Lösungsformeln von Gleichungen höheren als vierten Grades, die durch endliche Superposition der Grundrechenarten und diesen „erweiterten" Radikalen entstehen?

Mittels der TSCHIRNHAUS-Transformation läßt sich eine algebraische Gleichung $n$-ten Grades stets auf die Form:

$$t^n + a_4 t^{n-4} + \ldots + a_{n-1} t + 1 = 0 \tag{4.3}$$

bringen, d.h. das zweite, dritte und vierte Glied lassen sich stets eliminieren. Für eine Gleichung fünften Grades ergibt dies dann die vereinfachte Form:

$$x^5 + tx + 1 = 0 \tag{4.4}$$

die nur noch von einem Parameter $t$ abhängt. Damit hat man jedoch gerade eine implizite Darstellung der Nullstellen dieses Polynoms als einer Funktion von nur einer Veränderlichen $t$ (auch wenn sie sich nicht auf bekannte Weise durch elementare Funktionen beschreiben läßt) als $x = x(t)$.

Die Vermutung lag nahe, daß man Gleichungen sechsten Grades auf ähnliche Art auf eine Funktion zweier Veränderlicher, siebenten Grades auf eine Funktion dreier Veränderlicher usw. reduzieren kann. Zur Zeit von Hilberts Rede hatte man sich gerade auf Gleichungen siebenten Grades eingeschossen, und Hilbert dreizehntes Problem lautete dann auch: *Unmöglichkeit der Lösung der allgemeinen Gleichung 7. Grades mittels Funktionen von nur 2 Argumenten.*

In den folgenden Jahren häuften die Mathematiker Satz um Satz an, von denen jeder immer näher zum Beweis der Hilbertschen Vermutung führte. Jedoch selbst nach fünfzig Jahren war der endgültige Beweis immer noch nicht gefunden. Dann die Überraschung: als Folge eines kleinen Wettstreits zwischen den beiden russischen Mathematikern Kolmogorov und Arnold folgten von 1956 bis 1957 Schlag auf Schlag die Beweise, daß sich *jede* stetige Funktion als Superposition von Funktionen *dreier* Veränderlicher darstellen läßt, als Superposition von Funktionen *zweier* Veränderlicher (damit war Hilbert widerlegt) und schließlich als Superpositionen von Funktionen nur *einer* Veränderlichen [81]. Die genaue Formulierung des KOLMOGOROV-Theorems lautet:

**Theorem 4 (Kolmogorov).** *Es existieren fest gewählte, stetige und wachsende Funktionen $\alpha_{i,j}(x)$ mit $x \in I = [0,1]$, so daß sich jede stetige Funktion $f$ auf $I^n$ in der Form*

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{2n+1} \phi_i \left( \sum_{j=1}^{n} \alpha_{i,j}(x_j) \right) \tag{4.5}$$

*darstellen läßt, wobei die $\phi_i$ geeignet gewählte Funktionen einer Veränderlichen sind.*

Anfang der 60er Jahre wurde das KOLMOGOROV-Theorem dann von Sprecher auf eine „handlichere“ Form gebracht [163].

**Theorem 5 (Sprecher).** *Es existieren Konstanten $\lambda_p$ und fest gewählte, stetige und wachsende Funktionen $\Phi_q(x)$ auf $I = [0, 1]$, so daß sich jede stetige Funktion $f$ auf $[0, 1]$ in der Form*

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} g_q \left( \sum_{p=1}^{n} \lambda_p \Phi_q(x_p) \right) \tag{4.6}$$

*darstellen läßt, wobei die $g$ geeignet gewählte stetige Funktionen einer Veränderlichen sind.*

Damit klärt sich auch die erste im vorhergehenden Abschnitt gestellte Frage nach der minimalen Anzahl von Schichten $m$, in denen jede beschränkte stetige Funktion berechenbar ist. Diese Anzahl ist 4, da das *Sprecher*-Theorem genau eine Berechnung in vier Schichten repräsentiert: Schicht 1 gibt die Funktionen einer Veränderlichen $\Phi_q(x_q)$, Schicht 2 die Linearkombination $\sum_{p=1}^{n} \lambda_p \Phi_q(x_p)$ solcher Funktionen, Schicht 3 die Anwendung von $g_q$ als Funktion einer Veränderlichen auf diese Linearkombinationen, und Schicht 4 schließlich die Summe der $g_q$ als eine Linearkombination.

Die Darstellungen der letzten beiden Abschnitten lassen sich also in folgendem Satz zusammenfassen:

**Satz 8.** *Jede stetige Funktion $f : [0, 1]^n \to R$ ist in genau vier Schichten und nicht weniger berechenbar und durch Funktionen $g$, die in genau drei Schichten und nicht weniger berechenbar sind, approximierbar.*

### 4.4.1 Das Kolmogorov-Netzwerk

Ungeachtet des mathematischen Erfolgs von Kolmogorov und Arnold fand diese Erkenntnis jedoch außerhalb der Mathematik keine große Beachtung. Es war einfach nur das 13. Hilbertsche Problem gelöst worden, und zur großen Überraschung anders, als man es hundert Jahre geglaubt hatte.

Erst im Jahre 1987 richtete Robert Hecht-Nielsen die Aufmerksamkeit der Fachwelt auf dieses „alte“ und scheinbar sehr abstrakte Theorem [63]. Denn im Prinzip stellte es das erste jemals formulierte neuronale Netz dar! Betrachtet man die Funktionen einer Veränderlichen als Knoten in einem Netz mit jeweils einem Eingang, einer Eingangsfunktion, einer Übertragungsfunktion und einem Ausgang, verbindet diese Knoten und ordnet Verbindungen Werte zu, so läßt sich die Formel in Theorem 5 für den Fall $n = 2$ grafisch wie in Abbildung 4.6 darstellen. Die Knoten „Neuronen“ zu nennen und die den Verbindungen zugeordneten Werte „Gewichte“ ist jetzt reine Formsache. Es ist im Grunde lediglich eine grafische Repräsentation einer Formel, die jedoch ihren Existenzbeweis „frei

Haus“ mitliefert. Mit dem in Abbildung 4.6 dargestellten, nach Hecht-Nielsen KOLMOGOROV-Netzwerk benannten neuronalen Netz läßt sich jede noch so komplizierte (wenn wenigstens stetige und beschränkte) Funktion $f$ (hier von zwei Veränderlichen) exakt auf Basis einfacher Rechenelemente (den Neuronen) und ihrer Vernetzung darstellen.

Neuronale Netze sind also grafische Repräsentationen der Berechenbarkeit in $m$ Schichten. Dabei ist zu beachten, daß die Schichten eines neuronalen Netzes nicht ganz den Schichten einer Berechnung entsprechen. Ein neuronales Netz geht immer von einer Alteration der beiden möglichen Fälle von Berechenbarkeit (Linearkombination $C$ oder Funktion einer Veränderlichen $L$) aus. Jede Berechnung in $m$ Schichten entspricht also einer Sequenz aus den $C$ oder $L$, wie z.B. $LLCL$. Ein neuronales Netz hat jedoch immer eine solche Sequenz der Form $LCLCL\ldots$ (dabei werden die $L$ auch als Transferfunktionen bezeichnet). Hat man jedoch in der Berechnung einer Funktion $f$ in $m$ Schichten auch ein Auftreten von Teilsequenzen $LL$ oder $CC$, so lassen sich diese leicht umwandeln: $LL$ entspricht stets einem $L$, da hier zwei Funktionen einer Veränderlichen nacheinander angewendet werden, was effektiv einer Funktion einer Veränderlichen entspricht; und $CC$ läßt sich in $CLC$ umwandeln, in dem für $L$ die identische Funktion gewählt wird.

Somit läßt sich jede Berechnung in $m$ Schichten tatsächlich als ein neuronales Netz darstellen. In [84] findet sich eine Darstellung der Vorgehensweise zur Belehrung solch eines neuronalen Netzes.

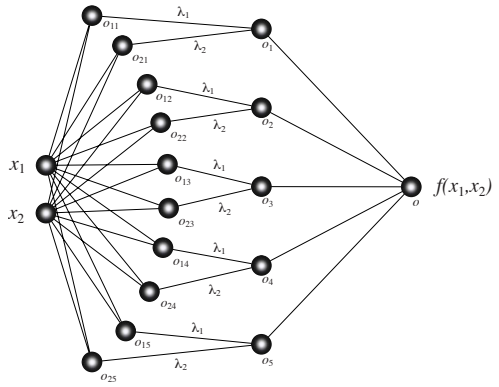



Abbildung 4.6: Grafische Repräsentation des Theorems von Sprecher für zwei Veränderliche $x_1$ und $x_2$ in Form des sog. KOLMOGOROV-Netzwerkes. Dabei gelten die Beziehungen $o_{pq} = \Phi_q(x_p)$, $o_q = g_q\left(\sum_p \lambda_p o_{pq}\right)$ und $o = \sum_q o_q$.

## 4.4.2 Universelle Repräsentation von Bildfunktionen

Üblicherweise, besonders bei maschinennahen Programmiersprachen wie C, wird ein Bild im Arbeitsspeicher eines Rechners in Form eines sogenannten *Buffers*

abgelegt. Dies entspricht einer zeilenweisen Abwicklung des Bildes, beginnend mit der Speicherzelle des Pixels $(0, 0)$. Befindet sich also ein Bild in einem Buffer beginnend ab Adresse $S$, dann findet man den Grauwert des Pixels $(i, j)$ an der Adresse $S + j \cdot \lambda + i$, wobei $\lambda$ für die Breite des Bildes steht. Also formal

$$f(i, j) = g[S + i + \lambda \cdot j]. \tag{4.7}$$

Das ist aber genau die Formel für eine Berechnung in zwei Schichten! Mehr ist nicht notwendig, um Bilder mit einer bestimmten Auflösung zu repräsentieren.

Gegeben sei eine Funktion $\phi : I^n \to I$ und der BANACH-Raum $S_\phi$ aller Funktionen $g[\phi(x)]$, wobei $g$ eine nicht notwendig stetige, beliebige Funktion ist, die $I$ auf $I$ abbildet. Eine Funktion $\phi_1$ heißt *effizienter* als eine Funktion $\phi_2$, wenn $S_{\phi_1} \supset S_{\phi_2}$ ist. Eine Funktion, für die $S_\phi$ gleich dem BANACH-Raum $S$ *aller* Funktionen über $I$ ist, heißt *maximal effizient*, da sie zu jeder Funktion aus $S$ eine beste Approximation in Form einer Berechnung in zwei Schichten liefern kann.

In diesem Sinne ist $\phi(i, j) = i + \lambda \cdot j$ eine maximal effiziente Funktion im Raum aller Bilder der Breite $\lambda$. Durch diesen Sachverhalt wird also die pixelweise Repräsentation von Bildern gerechtfertigt, da man mittels einer finiten „Lookup-Tabelle“ $g$ in der Lage ist, *jede* (beschränkte) Bildfunktion zu erzeugen. Es gibt jedoch auch andere maximal effiziente Funktionen für Bildfunktionen. Jede totale Ordnung aller Bildpunkte liefert solch eine Funktion.

Die Pixel sind aber in einem anderen Sinne auch das größte Übel in der Bildverarbeitung. Ein wesentlicher Aspekt ist die mangelnde Repräsentation von Nachbarschaftsbeziehungen in Bildern durch Pixel. Während der linke und der rechte Nachbarpixel im Bild genau eine Speicheradresse entfernt abgespeichert sind, finden sich der obere und untere Nachbarpixel genau um $\lambda$ Adressen entfernt wieder. Dieses Mißverhältnis kann durch eben benannte totale Ordnungen, z.B. entlang endlicher Approximationen von PEANO-Kurven, verringert werden.

In [164] diskutiert David Sprecher die Frage, wie sich das KOLMOGOROV-Theorem ändert, wenn man auf die Forderung nach Stetigkeit der Funktionen $\psi$ und $\chi$ verzichtet. Es zeigt sich dann, daß statt $(m+1)$ Termen nur noch einer benötigt wird:

**Theorem 6.** *Zu jeder natürlichen Zahl $n \geq 2$ existieren reellwertige, streng monoton wachsende Funktionen $h_p(x)$ mit $1 \leq p \leq n$, die von $n$ abhängen, so daß gilt:*
*(i) Die Funktion*

$$h(x) = \sum_{p=1}^{n} h_p(x_p)$$

*separiert alle Punkte von $I^n$:*

$$\sum_{p=1}^{n} h_p(x_p) = \sum_{p=1}^{n} h_p(y_p)$$

*dann und nur dann, wenn $x_p = y_p$ für alle zulässigen Werte von p gilt.*
*(ii) Jede stetige Funktion von n Veränderlichen $f(x_1, \ldots, x_n)$ mit dem Definitionsbereich $I^n$ kann in der Form*

$$f(x_1, \ldots, x_n) = g\left[\sum_{p=1}^{n} h_p(x_p)\right] \tag{4.8}$$

*mit einer in der Regel nicht stetigen Funktion g repräsentiert werden.*

Der Beweis beruht auf einer einfachen Konstruktion. Hier soll lediglich die Konstruktion vorgestellt werden, der Beweis ist dann mehr oder weniger evident. Sei mit $[i_1, i_2, \ldots, i_k]_\gamma$ die Ziffernfolge zur Basis $\gamma$ bei der Darstellung einer Zahl $d_k \in I$ mit $k$ Stellen nach dem Komma bezeichnet. Unter Ausschluß von Folgen $[i_1, \ldots]$, die ab einer Position verschieden der ersten nur noch aus Ziffern $\gamma - 1$ bestehen, läßt sich so auch jede irrationale Zahl eindeutig repräsentieren.

Nun wird im wesentlichen die Darstellung einer Zahl aus $I$ zur Basis $n^n$ in $n$ Darstellungen von Zahlen aus $I$ zur Basis $n$ zerlegt. Wenn gilt

$$x = \sum_{r=1}^{\infty} \frac{i_r}{n^r} \tag{4.9}$$

dann lautet

$$h_p(x) = \sum_{r=1}^{\infty} \frac{i_r}{n^{nr-(p-1)}} = \sum_{r=1}^{\infty} \frac{i_r \cdot n^{p-1}}{(n^n)^r}. \tag{4.10}$$

Jede Ziffer $i_r$ aus $\{0, \ldots, n^n - 1\}$ der Darstellung einer Zahl $x \in I$ zur Basis $n^n$ besitzt eine Darstellung zur Basis $n$ mit maximal $n$ Stellen:

$$i_r = \sum_{s=0}^{n-1} i_{rs} n^s, \tag{4.11}$$

wobei für jedes $i_{rs}$ $0 \leq i_{rs} < n$ gilt. Dann gilt ebenfalls

$$[i_1, \ldots, i_k]_{n^n} = h([i_{11}, \ldots, i_{k1}]_n, \ldots, [i_{1n}, \ldots, i_{kn}]_n) \tag{4.12}$$

und dies ist genau die Bijektion zwischen $I^n$ und $I$.

**Beispiel 4.4.1.** Sei $n = 2$ und wir wollen das Bild des Punktes mit der Dezimaldarstellung $(0.625, 0.75)$ bestimmen. Zur Basis 2 hat dieser Punkt die Darstellung $(0.101_2, 0.11_2)$ und es folgt

$$\begin{aligned} h_1(0.101_2) &= \frac{1}{4} + \frac{1}{4^3} \\ h_2(0.11_2) &= \frac{2}{4} + \frac{2}{4^2}. \end{aligned}$$

Damit ist

$$h[(0.625, 0.75)] = \frac{3}{4} + \frac{2}{16} + \frac{1}{64} = 0.890625.$$

Umgekehrt soll das Urbild von $0.78125 = 0.302_4$ bestimmt werden. Dazu findet man aus $3_4 = 11_2$, $0_4 = 00_2$ und $2_4 = 10_2$ durch Konstruktion von zwei Folgen von Dualzahlen aus dem ersten bzw. zweiten Element dieser Darstellungen

$$h^{-1}(0.78125) = (0.101_2, 0.100_2) = (0.625, 0.5).$$

Die Konstruktion von $g$ bei gegebenem $f$ ist simpel: man setze für $y \in I$ den Wert $g(y) = f[h^{-1}(y)]$.

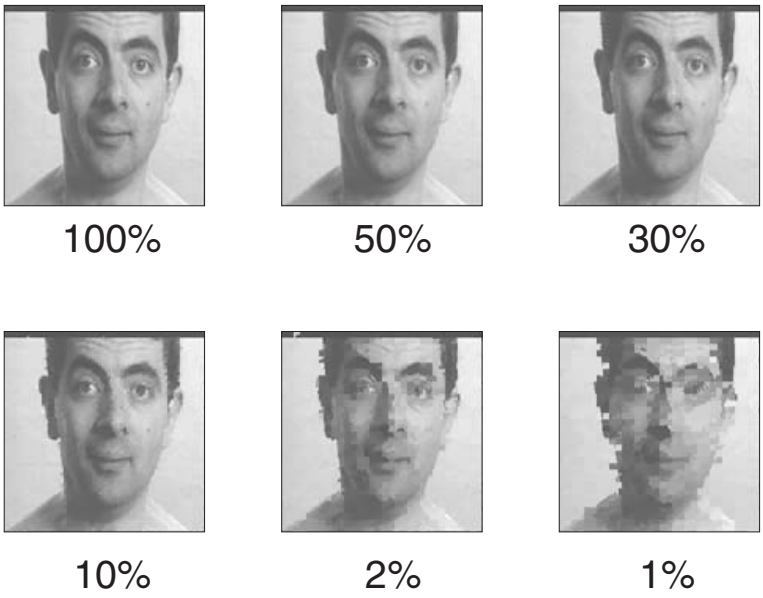



Abbildung 4.7: Qualität der Kompression bei Weglassung eines Teils der Datenpunkte $g(y)$ nach der SPRECHER-Transformation und Regeneration des Originalbildes (links oben) aus dieser reduzierten Funktion.

Abbildung 4.7 zeigt eine sehr interessante Eigenschaft dieser Transformation. Bestimmt man $g(y)$ für ein Bild, streicht danach einen bestimmten Anteil der Einträge (d.h. ersetzt alle Werte in einem Intervall durch einen einzelnen Wert dieses Intervalls) und regeneriert daraus wieder ein Bild, erhält man eine dekomprimierte Darstellung des Ausgangsbildes. Wie man sieht, bietet die SPRECHER-Transformation auch bei einer Kompression von 1 : 3 noch sehr gute Ergebnisse. Dies zeigt, daß hier lokale Nachbarschaftsbeziehungen der Pixel zueinander sehr gut repräsentiert werden.

In diesem Ansatz liegt jedoch noch eine weitere entscheidende Möglichkeit, nämlich die maximal effiziente Repräsentation von Bildern nicht auf Basis von Pixeln sondern auf Basis von *Umgebungen von Pixeln*. Betrachtet man z.B. einen

Pixel $(x, y)$ eines Bildes und seine vier nächsten Nachbarn, so können die diesen fünf Punkten zugeordneten Grauwerte (normiert auf den Bereich $[0, 1]$) auf eine reelle Zahl $y \in I$ abgebildet werden. Diese Zahl wiederum kann als Grauwert an der Stelle $(x, y)$ im Bild eingetragen werden (eventuell unter Nutzung einer Repräsentation von Grauwerten durch mehr als die heute üblichen 8 Bit). Dann hat man ebenfalls eine Repräsentation eines Bildes in Form der punktweisen Repräsentation von Nachbarschaften[1].

Hier müssen benachbarte Nachbarschaften nicht mehr kompatibel sein wie bei der üblichen Pixelrepräsentation, bei der zum Beispiel der Grauwert eines Punktes immer auch gleich dem Grauwert des rechten Nachbarn des linken Nachbarn sein muß. Die Techniken der Bildakquisition beinhalten keine solche Kompatibilität. So akkumuliert die einzelne Zelle eines CCD-Chips Lichtintensität aus einem Bereich, der zum Teil auch benachbarte Zellen überdeckt. Natürlich wird diese Überlappung technisch möglichst gering gehalten, aber es kann hier dennoch zu typischen Kamerafehlern wie dem *Blurring* kommen, bei dem die Aktivierung einer Zelle auf benachbarte übergreift. Dennoch ist diese Beeinflußung von Zellen durch Nachbarzellen nicht notwendig symmetrisch, und eine Repräsentation von Bildern durch Nachbarschaften kann dem besser gerecht werden.

Ebenso lassen sich Bildoperatoren effizienter repräsentieren, da man sie nur für das Einheitsintervall $I$ spezifizieren muß. Es kann jedoch zum Verlust analytischer Eigenschaften der Operatoren wie Stetigkeit kommen.

## 4.5 Das Schemata-Theorem

### 4.5.1 Formulierung

Den Schlüssel zum Verständnis der Funktionsweise eines genetischen Algorithmus stellen die Schemata dar. Eine aus einer Menge von Bitstrings bestehende Population enthält mit diesen Bitstrings auch alle die Schemata, die von diesen Bitstrings realisiert werden. Jede Manipulation an den Bitstrings stellt auch eine Manipulation der Schemata dar.

Um Fragen im Zusammenhang mit genetischen Algorithmen diskutieren zu können, betrachtet Holland in [68] ein Modellproblem, den $n$-armigen Banditen.

Der 2-armige Bandit ist ein Glückspielautomat mit zwei Hebeln. Das (zu bezahlende) Betätigen eines Hebels liefert einen bestimmten Gewinn. Jeder Hebel $i$ weist dabei einen festen Erwartungswert $\mu_i$ (und eine feste Standardabweichung $\sigma_i$) für das Ergebnis seiner Betätigung auf. Das Modellproblem lautet nun einfach so: der Spieler kennt zwar $(\mu_1, \sigma_1)$ und $(\mu_2, \sigma_2)$, er weiß jedoch nicht, welche Verteilung zu welchem Hebel gehört. Die Frage ist nun die nach einem Plan zur Betätigung der Hebel, der nach einer vorgegebenen Anzahl von Hebelbetätigungen $N$ einen möglichst hohen Gewinn sichert.

[1] Sollte man diese Elemente *Nexel* nennen?

Wäre die Zuordnung der Hebel zu den Verteilungen bekannt, würde der optimale Plan darin bestehen, stets nur den Hebel mit dem größeren Wert für $\mu$ zu betätigen (eine typische *greedy strategy*). Ein Ansatz für einen Plan besteht also darin, zuerst einmal in Erfahrung zu bringen, welcher der Hebel den höheren Erwartungswert aufweist. Dazu führt der Spieler eine Reihe von „Messungen“ für beide Hebel durch. Nach einer bestimmten Anzahl von Zügen sind dann die Verteilungen für beide Hebel hinreichend sicher ermittelt, und in Folge betätigt man dann nur noch den Hebel, für den sich der höhere Erwartungswert ergab.

Dies illustriert zwei Grundbegriffe von Suchverfahren: der *Modus* der Auswahl durch einen Plan läßt sich in zwei Klassen einteilen, Exploration (*exploration*) und Exploitation (*exploitation*). Zur Exploration gehören Strukturauswahlen, die die Exploitation vorbereiten. Die Exploration beim Problem vom 2-armigen Banditen besteht darin, eine Schätzung für den Hebel mit dem größeren Erwartungswert zu erhalten. Die anschließende Exploitationsphase läuft dann „blind“ in Bezug auf Plankonsolidierung nach dem Schema einer “gierigen Strategie“ ab. Während der Exploitation findet jedoch auch eine weitere Exploration statt, in dem eine gewisse Anzahl an Versuchen auch dem schlechteren Hebel zugewiesen wird und, falls es die Umstände ergeben, die Richtung der Exploitation auch gewechselt wird (hin zur ausschließlichen Betätigung des anderen Hebels).

Die Frage nach einem optimalen Plan wird durch obige Argumente auf die Frage reduziert, zu jeder Anzahl von Hebelbetätigungen $N$ eine geeignete Anzahl $n^*$ zu bestimmen, die den Anteil der Versuche beschreibt, die dem scheinbar schlechteren Hebel insgesamt zugewiesen wird. In Bezug auf das Modellproblem macht es jedoch keinen Unterschied, ob diese Versuche sofort hintereinander am Anfang erfolgen oder irgendwie über alle $N$ Versuche verteilt werden[2]. Also: bei den ersten $2n^*$ Versuchen werden beide Hebel gleichermaßen zufällig betätigt (scheinbar „planlos“) und es wird das Ergebnis der Betätigung gemessen. Somit werden dem schlechteren Hebel auf jeden Fall $n^*$ Versuche zugewiesen, egal, welcher es tatsächlich ist. Dann werden für jeden Hebel die Erwartungswerte bestimmt. In Folge wird nun nur noch der Hebel betätigt, für den sich der höhere Erwartungswert ergab. Je kleiner $n^*$ als Funktion von $N$ gewählt wird, destso höher ist das Risiko, auf Grund statistischer Schwankungen sich für den falschen Hebel zu entscheiden. Je größer $n^*$ ist, destso mehr Verlust entsteht durch die häufige Mitbetätigung des (objektiv) schlechteren Hebels. Es geht also darum, ein optimales $n^*$ zu bestimmen. Anders gesagt: wieviel Versuche hat man, bevor man sich endgültig entscheiden muß?

Holland ermittelt in [68] zwar nicht den optimalen Wert für $n^*$, aber eine wichtige Charakterisierung dieses Wertes, die aussagt, daß bei wachsendem $N$ der Anteil von $n^*$ an $N$ exponentiell abfällt.

[2]Der Unterschied spielt dann eine Rolle, wenn nach $N$ Versuchen nicht bekannt wird, welcher Hebel nun der bessere war. Dann beschreibt $n^*$ tatsächlich den relativen Anteil für das Testen des anderen Hebels auch in zukünftigen Versuchen.

Nun lassen sich genetische Algorithmen leicht in Zusammenhang mit den $n$-armigen Banditen bringen, wenn man jedes Schema aus Bitstrings einem solchen Hebel gleichsetzt. Die Philosophie dahinter ist dann, daß ein genetischer Algorithmus nicht primär nach einer guten Lösung sucht, sondern überdurchschnittlich guten Schemata (Hebeln) mehr Versuche zuweist, also mehr Individuen der Population zu guten Schemata gehören.

Im folgenden soll anhand eines einfachen Beispiels illustriert und diskutiert werden, wie diese Manipulation von Schemata genau abläuft. Dazu wird als Testproblem die Funktion

$$f(x) = 16 - (x-3)^2 \tag{4.13}$$

betrachtet. Der genetische Algorithmus besteht aus vier Individuen á drei Bit. Die Kodierung erfolgt über die Interpretation des Bitstrings als Dualzahl $x$. Damit hat z.B. der Bitstring 010 den $x$-Wert 2 und die Fitness $16-(2-3)^2 = 15$. Das Maximum von $f_{max} = 16$ wird durch den Bitstring 011 (mit dem $x$-Wert 3) erreicht.

Angenommen, die Generation $t$ besteht aus den vier Bitstrings

101
110
001
000

mit den entsprechenden Fitnesswerten 12, 7, 12 und 7 und also einer mittleren Fitness von 9.5 und einer besten Fitness von 12. Tabelle 4.2 listet nun alle 27 möglichen Schemata $H$ auf, zu denen ein Bitstring der Länge 3 gehören kann.

Zu jedem Schema ist seine mittlere Fitness $\bar{f}_H$ über alle seine Realisierungen angegeben. Das Schema *00 besitzt so zwei Realisierungen: 000 mit der Fitness 7 und 100 mit der Fitness 15. Damit hat es die mittlere Fitness $(7+15)/2 = 11$. Dieses Schema taucht in der Generation $t$ genau einmal auf, als durch den Bitstring 000 realisiert. Insgesamt realisiert der Bitstring 000 genau acht Schemata:

000 00* 0*0 0** *00 *0* **0 ***.

Das Schema 00* wird dabei z.B. auch durch den in der Population vorhandenen Bitstring 001 realisiert.

Die mittlere Fitness eines in der Population vorhandenen Schemas wird also durch alle seine Realisierungen, die zur Generation $t$ gehören, geschätzt. Das Schema $H =$ *0* besitzt so drei Realisierungen in der Generation $t$, nämlich die Bitstrings 101, 001 und 000. Die vierte mögliche Realisierung des Schemas $H$ durch den Bitstring 100 tritt dagegen in der Population für das gewählte Beispiel nicht auf. Die mittlere Fitness der drei Realisierungen ist $(12+12+7)/3 = 10.\bar{3}$. Der Unterschied $\epsilon_{H,t}$ der so abgeschätzten Fitness des Schemas zum exakten Wert

$f_H = 11.5$ (siehe Tabelle 4.2) beträgt also etwa 1.2. Die Sprechweise hier ist, daß der genetische Plan für das Schema $H$ eine mittlere Fitness von $10.\bar{3}$ „gemessen“ hat. Bezogen auf den $n$-armigen Banditen (hier ist $n = 27$) bedeutet es, daß für den Hebel zu dem Schema `*0*` ein mittlerer Gewinn von $10.\bar{3}$ ermittelt wurde. Das Verhältnis dieses Wertes zu den Werten für alle anderen in der Population vorhandenen Schemata sollte den Anteil bestimmen, zu dem in Zukunft diesem Hebel Versuche zugewiesen werden. Die „Überdurchschnittlichkeit“ (*above-averageness*) dieses Hebels beträgt $10.\bar{3}/9.5 \simeq 1.1$ (oder 10%). Diesem Schema sollten in Folge mehr Versuche zugewiesen werden.

Das Schema `**0`, um ein anderes Beispiel zu wählen, wird durch zwei Individuen der Population realisiert: durch den Bitstring `110` mit der Fitness 7 und durch den Bitstring `000` mit derselben Fitness. Die durch den Plan gemessene durchschnittliche Fitness von 7 weicht von der exakten Fitness von 11 um 4 ab, dieses Schema wird also relativ schlecht gesampelt. Mit einer „Unterdurchschnittlichkeit“ von $7/9.5 \simeq 0.7$ sollten diesem Schema in Folge weniger Versuche zugewiesen werden.

Die durchschnittliche Abweichung der gemessenen von der echten mittleren Fitness für alle 27 Schemata liegt bei 2.1. Mit einer Varianz von ca. 1.7 muß für die Genauigkeit der durch den genetischen Plan vollzogenen Messung aller Schemata $2 \pm 2$ geschrieben werden.

Nun soll ein Generationswechsel simuliert werden, der lediglich aus der viermaligen Anwendung des Crossover-Operators und Elitistenselektion besteht. Dabei liefert das Crossover acht neue Individuen, und die Selektion wählt von ihnen die vier besten aus.

Zur Durchführung des Crossover werden jeweils zwei Individuen mit einer Wahrscheinlichkeit ausgewählt, die proportional zu ihren relativen Fitnesswerten sind. So wird der Bitstring `110` mit einer Wahrscheinlichkeit von 7 geteilt durch die Summe aller Fitnesswerte der Population, also $12+7+12+7 = 38$ ausgewählt. Zur technischen Umsetzung solch einer Auswahl wird eine Zahl gleichverteilt zufällig aus der Menge $\{1, 2, \ldots, 38\}$ ausgewählt. Je nach den folgenden Intervallagen der gewählten Zahl ergibt sich so dann der ausgewählte Bitstring:

$$\begin{aligned} 1 - 12 &: \texttt{101} \\ 13 - 19 &: \texttt{110} \\ 20 - 31 &: \texttt{001} \\ 32 - 38 &: \texttt{000} \end{aligned}$$

Eine zweite zufällig aus $\{0, 1\}$ ausgewählte Zahl bestimmt dann den Crossover-

| $H$ | $\bar{f}_H$ | $M_{H,t}$ | $f_{H,t}$ | $\epsilon_{H,t}$ | $f_{H,t}/\bar{f}$ | $M_{H,t+1}$ | $f_{H,t+1}$ | $\epsilon_{H,t+1}$ |
|---|---|---|---|---|---|---|---|---|
| `***` | 10.5 | 4 | 9.5 | 1 | 1.0 | $4_{\pm 0}$ | 12.75 | 2.25 |
| `**0` | 11 | 2 | 7 | 4 | 0.7 | $1_{-1}$ | 15 | 4 |
| `**1` | 10 | 2 | 12 | 2 | 1.3 | $3_{+1}$ | 12 | 2 |
| `*0*` | 11.5 | 3 | $10.\bar{3}$ | 1.2 | 1.1 | $4_{+1}$ | 12.75 | 1.25 |
| `*00` | 11 | 1 | 7 | 4 | 0.7 | $1_{\pm 0}$ | 15 | 4 |
| `*01` | 12 | 2 | 12 | 0 | 1.3 | $3_{+1}$ | 12 | 0 |
| `*1*` | 9.5 | 1 | 7 | 2.5 | 0.7 | $0_{-1}$ | - | - |
| `*10` | 11 | 1 | 7 | 4 | 0.7 | $0_{-1}$ | - | - |
| `*11` | 8 | 0 | - | - | - | $0_{\pm 0}$ | - | - |
| `0**` | 12.5 | 2 | 9.5 | 3 | 1.0 | $2_{\pm 0}$ | 12 | 0.5 |
| `0*0` | 11 | 1 | 7 | 4 | 0.7 | $0_{-1}$ | - | - |
| `0*1` | 14 | 1 | 12 | 2 | 1.3 | $2_{+1}$ | 12 | 2 |
| `00*` | 9.5 | 2 | 9.5 | 0 | 1.0 | $2_{\pm 0}$ | 12 | 2.5 |
| `000` | 7 | 1 | 7 | 0 | 0.7 | $0_{-1}$ | - | - |
| `001` | 12 | 1 | 12 | 0 | 1.3 | $2_{+1}$ | 12 | 0 |
| `01*` | 15.5 | 0 | - | - | - | $0_{\pm 0}$ | - | - |
| `010` | 15 | 0 | - | - | - | $0_{\pm 0}$ | - | - |
| `011` | 16 | 0 | - | - | - | $0_{\pm 0}$ | - | - |
| `1**` | 8.5 | 2 | 9.5 | 1 | 1.0 | $2_{\pm 0}$ | 13.5 | 5 |
| `1*0` | 11 | 1 | 7 | 4 | 0.7 | $1_{\pm 0}$ | 15 | 4 |
| `1*1` | 6 | 1 | 12 | 6 | 1.3 | $1_{\pm 0}$ | 12 | 6 |
| `10*` | 13.5 | 1 | 12 | 1.5 | 1.3 | $2_{+1}$ | 13.5 | 0 |
| `100` | 15 | 0 | - | - | - | $1_{+1}$ | 15 | 0 |
| `101` | 12 | 1 | 12 | 0 | 1.3 | $1_{\pm 0}$ | 12 | 0 |
| `11*` | 3.5 | 1 | 7 | 3.5 | 0.7 | $0_{-1}$ | - | - |
| `110` | 7 | 1 | 7 | 0 | 0.7 | $0_{-1}$ | - | - |
| `111` | 0 | 0 | - | - | - | $0_{\pm 0}$ | - | - |

Tabelle 4.2: Beispiel für den Einfluß der genetischen Operatoren auf die Anteile von Schemata in einer genetischen Population beim Wechsel von der Generation $t$ (mit der mittleren Fitness $\bar{f} = 9.5$) zur Generation $(t+1)$ (mit der mittleren Fitness $\bar{f} = 12.75$). Dabei bezeichnen $H$ das jeweilige Schema, $\bar{f}_H$ den Erwartungswert der Fitness für alle Realisierungen von $H$, $M_{H,t}$ die Anzahl für das Auftreten von Schema $H$ in der Generation $t$, $f_{H,t}$ die über alle Realisierungen des Schemas $H$ in der Generation $t$ „gemessene“ mittlere Fitness dieses Schemas, $\epsilon_{H,t}$ den entsprechenden „Meßfehler“, $M_{H,t+1}$ die Häufigkeit von Schema $H$ in der Folgegeneration $(t+1)$, $\bar{f}_{H,t+1}$ die entsprechende mittlere Fitness und $\epsilon_{H,t+1}$ den entsprechenden Fehler in der Generation $(t+1)$.

Punkt. Folgende Crossover-Operationen werden so z.B. ausgeführt:

$$\begin{aligned}
17-12-2 &: \boxed{11|0}+\boxed{10|1}=\boxed{111}+\boxed{100}\\
8-22-2 &: \boxed{10|1}+\boxed{00|1}=\boxed{101}+\boxed{001}\\
20-34-1 &: \boxed{0|01}+\boxed{0|00}=\boxed{000}+\boxed{001}\\
11-25-1 &: \boxed{1|01}+\boxed{0|01}=\boxed{101}+\boxed{001}
\end{aligned}$$

wobei die erste Zeile z.B. bedeuten soll, daß als Zufallszahlen 17 (steht für die Wahl von $\boxed{110}$), 12 (steht für die Wahl von $\boxed{101}$) und 2 (steht für die zweite Zwischenposition als Crossover-Punkt) genommen wurden. Im Ergebnis wird für das erste Individuum dann der linke Teil $\boxed{11}$ aus $\boxed{110}$ und der rechte Teil $\boxed{1}$ aus $\boxed{101}$ entnommen, für das zweite Individuum der linke Teil $\boxed{10}$ aus $\boxed{101}$ und der rechte Teil $\boxed{0}$ aus $\boxed{110}$. Die acht neuen Individuen (*children*) haben dann folgende Fitnesswerte:

$$\begin{aligned}
\boxed{111} &: 0\\
\boxed{100} &: 15\\
\boxed{101} &: 12\\
\boxed{001} &: 12\\
\boxed{000} &: 7\\
\boxed{001} &: 12\\
\boxed{101} &: 12\\
\boxed{001} &: 12
\end{aligned}$$

von denen dann die folgende Generation aus den vier besten Bitstrings

$$\boxed{100}\quad\boxed{001}\quad\boxed{001}\quad\boxed{101}$$

aufgebaut wird. Nun kann in Tabelle 4.2 ersehen werden, wie sich durch diesen Schritt zur Generation $(t+1)$ die Anteile der Schemata geändert haben. Das überdurchschnittlich gute Schema $\boxed{*0*}$ ist statt dreimal jetzt viermal in der Population vertreten (alle vier Bitstrings der Population besitzen eine 0 in der Mitte). Das unterdurchschnittliche Schema $\boxed{**0}$ ist statt zweimal nur noch einmal vertreten. Insgesamt wurden die Anteile von Schemata erhöht, die die mittleren Fitnesswerten von 9.5, 10.5 und 12 besaßen, und es wurden die Anteile der Schemata gesenkt, deren mittlere Fitness bei 7 lag.

Die Genauigkeit der neuen Generation bei der Messung der mittleren Fitness ist etwa gleich geblieben (Mittelwert 2.1, Varianz 1.9, also ebenfalls $2 \pm 2$).

Interessant ist es hier, auf das Verhältnis von Überdurchschnittlichkeit und Anteilsänderung eines Schemas hinzuweisen. Ermittelt man

$$\lambda_H = \frac{f_{H,t}/\bar{f}}{M_{H,t+1}/M_{H,t}} \tag{4.14}$$

für alle Fälle, bei denen so nicht durch 0 dividiert wird, hat $\lambda_H$ den Wert $1.0 \pm 0.2$. Im wesentlichen ist also die Änderung des Anteils eines Schemas *proportional* zu seiner Überdurchschnittlichkeit. Wenn sich dies in jeder Generation so wiederholt, wird also der Anteil überdurchschnittlich guter Schemata exponentiell anwachsen.

Eine weitere wichtige Beobachtung, die Tabelle 4.2 entnommen werden kann: die Anzahl aller Schemata, die durch Bitstrings der Population realisiert werden, hat von 21 auf 16 abgenommen. In der Generation $t$ kann dadurch nur der Bitstring `011` nicht durch Crossover aus zwei anderen erzeugt werden. In Generation $(t+1)$ sind es bereits vier Bitstrings, die „verlorengehen“: `010`, `011`, `110` und `111` (da kein Individuum der Population eine 1 in der Mitte besitzt). Bedauerlicherweise ist `011` aber genau der optimale Bitstring. Die einzige Möglichkeit, auch diesen Bitstring zu erzeugen, liegt in der Anwendung der Mutation. Genau darin liegt auch ihre Rolle bei genetischen Algorithmen, im Unterschied zu anderen evolutionären Verfahren: das potentielle Wiedereinbringen von Schemata, die im Laufe der Evolution aus der Population verschwunden sind. Durch Mutation wird gewährleistet, daß ein Schema niemals tatsächlich aus einer Population verschwindet.

Die eben dargestellten Änderungen am Anteil von Schemata in gemäß einem genetischen Plan evolutionierenden Populationen lassen sich auch formal verifizieren und führen zu dem berühmten Schemata-Theorem als der theoretischen Grundlegung der Tatsache, daß genetische Algorithmen keine rein zufallsbasierten Suchverfahren darstellen.

Die Frage steht hierbei nach einer Abschätzung des Anteils eines Schemas $H$ in der Generation $(t+1)$, wenn es in der Population $t$ den Anteil $M_{H,t}$ hatte. Wie groß ist die Wahrscheinlichkeit, daß ein Schema durch die Crossover-Operation „zerstört“ wird?

Dazu wird die sog. definierende Länge $\delta_H$ eines Schemas $H$ eingeführt: sie ist der Abstand zwischen dem ersten von * verschiedenen Element eines Schemas und dem letzten. Das Schema `***1**0*101**` der Länge 13 hat eine definierende Länge von 7 als Differenz aus der letzten von * verschiedenen Position (11) und der ersten (4).

Fällt der Crossover-Punkt zwischen die erste und die letzte von * verschiedene Position eines Schemas, wird dieses in der Regel zerstört (es sei denn, der vom anderen Bitstring übernommene Teil ist kompatibel zu dem fehlenden Schemateil). Die pessimistische Annahme hier ist, daß es *stets* zerstört wird. Außerdem wird

angenommen, daß das Schema $H$ nicht durch andere Crossover-Operationen in die Population eingeführt wird. Damit erhält man also eine Abschätzung des Anteils $M_{H,t+1}$ *nach unten*. Ein Schema wird jedoch auch dann nicht zerstört, wenn der zweite Bitstring zu demselben Schema gehört, egal, wo der Crossover-Punkt liegt. Dieser Fall tritt mit der Wahrscheinlichkeit $P_{H,t} = M_{H,t}/N$ ein, wobei $N$ die Größe der Population ist. Damit läßt sich $M_{H,t+1}$ wie folgt nach unten abschätzen:

$$M_{H,t+1} \geq M_{H,t} \cdot \frac{f_{H,t}}{\bar{f}} \cdot \frac{\delta_H}{L-1}(1 - P_{H,t}). \tag{4.15}$$

Dabei ist $L$ die Länge eines Bitstrings. Zur Abschätzung des Einflusses der Mutation wird mit $o(H)$ die Ordnung eines Schemas als die Anzahl der Positionen, die verschieden von * sind, eingeführt. Dann ist die Wahrscheinlichkeit, daß eine positionelle Mutation, die an jeder Position mit der Wahrscheinlichkeit $p_m$ das entsprechende Bit im Bitstring kippt, ein Schema nicht zerstört

$$(1 - p_m)^{o(H)}. \tag{4.16}$$

Insgesamt ergibt sich damit eine Formulierung des Schemata-Theorems.

**Theorem 7 (Schemata).** *Ein Schema $H$, welches in der Generation $t$ mit der relativen Häufigkeit $P_{H,t}$ auftrat, wird in der Generation $(t+1)$ mit einer Häufigkeit nicht geringer als*

$$P_{H,t+1} \geq P_{H,t} \cdot \frac{f_{H,t}}{\bar{f}} \cdot \frac{\delta_H}{L-1}(1 - P_{H,t}) \cdot (1 - p_m)^{o(H)} \tag{4.17}$$

*auftreten.*

Dies ist nur eine Form des Schemata-Theorems. Je nach der konkreten Vorgehensweise des genetischen Plans wird sich die rechte Seite dieser Beziehung ändern. So wird in der ursprünglichen Formulierung [68] noch eine Wahrscheinlichkeit $p_c$ für das Crossover eingeführt. Damit ändert sich dann die rechte Seite z.B. zu

$$\begin{aligned} P_{H,t+1} \quad \geq \quad & P_{H,t} \cdot \frac{f_{H,t}}{\bar{f}} \times \\ & \times \left[1 - p_c \frac{\delta_H}{L-1}(1 - P_{H,t})\right] \\ & \times (1 - p_m)^{o(H)}. \end{aligned} \tag{4.18}$$

Für andere Fälle kann es formal sogar sehr schwierig werden, eine solche Abschätzung vorzunehmen, wie z.B. bei der Verwendung von Zweipunkt- oder uniformen Crossover (siehe [74] [2] [167]). Wesentlich ist in allen Fällen, daß der Anteil in erster Ordnung mindestens proportional zu seiner Überdurchschnittlichkeit $f_{H,t}/\bar{f}$ wächst. Damit wird jedoch genau verifiziert, was quasi-optimale Pläne beim $n$-armigen Banditen beinhalten: der Anteil der Hebelbetätigungen wächst proportional mit der gemessenen Überdurchschnittlichkeit der bisherigen Betätigungen des Hebels. Dabei entspricht jedes Schema solch einem Hebel.

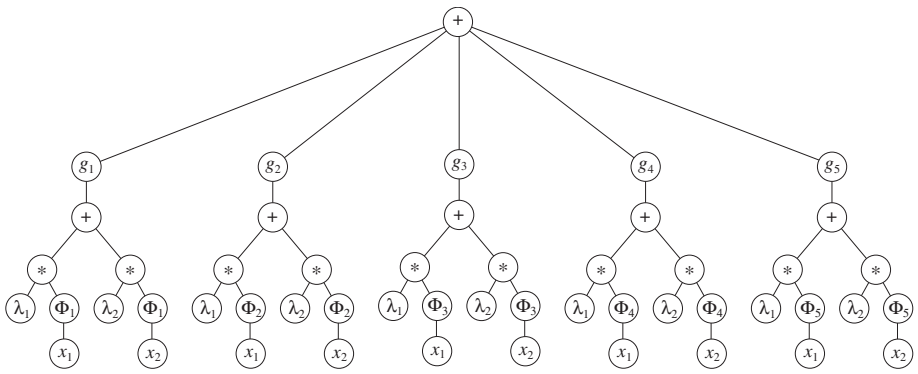



Abbildung 4.8: Darstellung des KOLMOGOROV-Theorems für Funktionen zweier Veränderlicher $x_1$ und $x_2$ in Form eines grammatikalischen Baumes.

### 4.5.2 Genetische Programmierung

Trotz großer Ähnlichkeit zu genetischen Algorithmen gibt es bei der genetischen Programmierung wesentliche Unterschiede bzgl. des Schemata-Theorems. Die Verifizierung dieses Theorems wird z.B. durch die Existenz redundanter Teilbäume erheblich erschwert. So würde ein in LISP Notation gegebener Teilbaum `(- 1 1)` sich stets zu 0 berechnen. Als Teil eines +-Knotens hätte dieser Teilbaum keinen Einfluß auf das Gesamtergebnis. Das Problem hier ergibt sich gerade daraus, daß die Gesamtzahl solcher Redundanzen, die das Sampeln von Schemata natürlich einschränkt, vom konkreten Problem abhängt und in der Regel auch nicht berechenbar oder sehr schwer abschätzbar ist. Es ist bis heute nicht gelungen zu zeigen, daß der Einfluß solcher Redundanzen keinen bedeutenden Einfluß auf den exponentiell wachsenden Anteil an Versuchen hat, die besseren Schemata zugewiesen werden. Nur bei speziellen Konfigurationen wurden bisher Ergebnisse in Bezug auf die Ableitung eines Schemata-Theorems erzielt, wie z.B. in [143]. Dort wird die Auswahl der Knotenpunkte vor dem Crossover auf Punkte eingeschränkt, bei denen bestimmte Maße für die unter diesen Knoten liegenden Teilbäume gleich sind (sog. *one point crossover*). Andere Betrachtungen zum Schemata-Theorem für genetische Programmierung finden sich ebenfalls in [143].

Ein weiteres wichtiges theoretisches Ergebnis in Bezug auf genetische Programmierung ist das sog. *fitness bloating*. Nach mehreren Jahren des Einsatzes der genetischen Programmierung wurde oft festgestellt, daß die sich ergebenden optimalen Lösungen oft von recht einfacher Struktur sind (die sog. Parsimonieeigenschaft), daß die genetische Programmierung jedoch dazu tendiert, immer größere und komplexere Bäume aufzustellen. In [104] dann konnte gezeigt werden, daß es der Fitnessdruck selbst ist, der die Individuen dazu bringt, sich „aufzublähen“ (daher die Bezeichnung *fitness bloating* für diesen Effekt). Ihr muß durch Restriktion der Suche auf kleine Baumtiefen vorgebeugt werden.

Die mathematischen Fragestellungen hinter der genetischen Programmierung

sollten jedoch gar nicht im Licht des Schemata-Theorems betrachtet werden sondern in Bezug auf das KOLMOGOROV-Theorem. Dazu betrachte man z.B. den Ausdruck, dessen Gültigkeit für stetige Funktionen $f(x_1, x_2)$ zweier Veränderlicher vom KOLMOGOROV-Theorem gewährleistet wird:

$$\begin{aligned} f(x,y) &= g_1(\lambda_1\Phi_1(x_1) + \lambda_2\Phi_1(x_2)) + \\ &\quad +g_2(\lambda_1\Phi_2(x_1) + \lambda_2\Phi_2(x_2)) + \\ &\quad +g_3(\lambda_1\Phi_3(x_1) + \lambda_2\Phi_3(x_2)) + \\ &\quad +g_4(\lambda_1\Phi_4(x_1) + \lambda_2\Phi_4(x_2)) + \\ &\quad +g_5(\lambda_1\Phi_5(x_1) + \lambda_2\Phi_5(x_2)) \end{aligned} \tag{4.19}$$

Dies läßt sich nicht nur als KOLMOGOROV-Netzwerk wie in Kapitel 4.4 grafisch repräsentieren, sondern auch als Baumstruktur, wie bei der genetischen Programmierung benutzt (siehe Abbildung 4.8). Die Generalisierung dieser Herangehensweise führt dann auf die in [176] diskutierten Operatorenalgebren. Bisher ist die universelle Zerlegbarkeit von Funktionen mehrerer Veränderlicher in Funktionen von weniger Veränderlichen nur in der Form bekannt, wie sie von Kolmogorov, Sprecher und Lorentz angegeben wurde. Es ergeben sich jedoch eigentlich damit auch weitere Fragen:

- Gibt es auch Superpositionen auf Basis anderer Operationen statt der Addition?
- Lassen sich auch anders strukturierte Terme als im KOLMOGOROV-Theorem aufstellen?
- Was ist mit dem Einbezug von Operationen von zwei- oder mehreren Veränderlichen (aber weniger, als die Anzahl Veränderlicher der zu approximierenden Funktion)?
- Lassen sich, falls eine universelle Approximation mittels eines Ausdrucks nicht möglich ist, zumindest Aussagen treffen über die Menge der Funktionen, die so approximierbar sind?
- In welchem Verhältnis stehen die Anzahlen der Möglichkeiten zueinander, dieselbe Funktion mittels verschiedener Ausdrücke zu approximieren?

Wie man sieht, steht die genetische Programmierung in engem Zusammenhang mit gerade solchen Fragestellungen. Unabhängig davon werden die operationellen „Zerlegungen“ der zu approximierenden Funktion mittels Beispielen gefunden.

Grundlegende mathematische Aussagen auf diesem Gebiet, von einigen Isomorphiebeweisen abgesehen, fehlen heute vollständig. Einzig das KOLMOGOROV-Theorem ragt wie der Olympus Mouns auf dem Mars aus diesem unwegbaren Terrain weit sichtbar heraus. Und selbst ihm fehlt eine versatile Herangehensweise an die Möglichkeit, die von ihm gewährleistete Zerlegung in praktischen Anwendungen überhaupt zu finden.

# 5 Gestaltung von Soft Computing Systemen zur Bildverarbeitung

In diesem Kapitel soll die Gestaltung eines konkreten, auf *Soft Computing* Verfahren basierenden Bildverarbeitungssystem vorbereitet werden. Dazu sind die im vorhergehenden Abschnitt benannten Theoreme in ihrer Bedeutung für solche Systeme erneut herauszustellen. Wichtige Fragen hier sind die möglichen Formen der Repräsentationen der Bearbeitungsobjekte solcher Verfahren (Bilder, Operationen), der Gestaltungsziele solcher Systeme, und nach den Möglichkeiten, ihre Leistungsfähigkeit zu quantifizieren.

## 5.1 Repräsentationen

### 5.1.1 Repräsentation der Bilder

Bei biologischen Systemen werden Muster den senso-motorischen Repräsentationen der Umwelt beigefügt. Für Systemlösungen, die auf *Soft Computing* basieren, steht die adäquate Frage nach einer geeigneten Repräsentation der sensorischen Eingangsdaten, insbesondere von sensoriell akquirierten Bildern. Die im vorhergehenden Abschnitt untersuchten Theoreme geben darauf eine Antwort.

Das "Ugly Duckling" Theorem weist dabei die Unmöglichkeit des z.B. bei Untersuchungen der Künstlichen Intelligenz favorisierten Ansatzes der aussagenlogischen Prädikate nach. Ausgehend von einer inhaltlichen Beschreibung z.B. eines Bildes durch "grüner Kegel steht neben schwarzem Obelisk" hat man letzlich keine Basis zur Unterscheidung dieses Bildes zu einem Bild "tomatenähnliches Lebewesen macht sich über den Kühlschrank her". Oder: auf Basis einer erweiterungsfähigen logischen Beschreibung eines Bildes kann das Bild selbst nicht mehr eindeutig singularisiert und in letzter Konsequenz von anderen differenziert werden.

Diese Erkenntnis hat sehr früh in der Mustererkennung zur Aufgabe des logischen Ansatzes geführt. Wir treffen ihn heute nur noch in isolierter Weise an [138]. Letztlich hat sich die merkmalbasierte Sichtweise durchgesetzt, bei der numerische Größen (die Merkmale) den Grad der Ausprägung eines logischen Attributes beschreiben.

Unabhängig davon stand den logischen Merkmalen auch immer das Problem an, ihre Gültigkeit aus der unmittelbaren Bildakquisition abzuleiten. Dies kann

sich jedoch schon bei einfach klingenden Formulierungen wie "ein grünes Auto" als sehr schwer handhabbares Problem erweisen, wenn etwa nur die Frontalansicht des Wagens im Bild zur Verfügung steht.

Ein wichtiger Ansatz in dieser Richtung, die sogenannten visuellen Routinen, wurde von Shimon Uhlman entwickelt [175]. Auch der Ansatz von David Marr, aus demselben Umfeld, zielte auf eine Hierarchie von schrittweiser Reduktion der konkreten Struktur hin zur Ableitung abstrakter Aussagen z.B. über ein Bild [120].

So wie das KOLMOGOROV-Theorem letztlich die universelle Möglichkeit der Repräsentation stetiger Funktionen durch Neuronen etabliert, könnten visuelle Routinen hier den eher dynamischen Aspekt herausstellen. Es geht also um eine universelle Repräsentation von logischen Aussagen über Objekte und deren Beziehungen untereinander durch Datentransformationen. Diese Transformationen sollen dann quasi Stellverterter der Bilder in einem Computer sein.

Bisher fehlen grundlegende mathematische Theoreme über solche visuelle Routinen vollständig. Den bisher fruchtbarsten Ansatz in dieser Richtung stellen die Agenten dar. Der Grundstein für diesen neuen Ansatz zur Organisation komplexer Systeme wurde 1985 von Marvin Minsky in seinem Buch *„The Society of Mind"* [126] gelegt. Die Frage, wie Maschinen Dinge in der Art und Weise vollführen können, wie sie bewußte Lebewesen vollführen, also wie Bewußtsein (*mind*) an sich funktioniert, wurde auf das Zusammenspiel „kleiner" Prozesse zurückgeführt, die für sich kein Bewußtsein aufweisen. Diese kleinen, einfachen Prozesse nennt Minsky *Agenten*, Bewußtsein selbst ist dann die Gemeinschaft solcher Agenten, die miteinander kommunizieren. Agenten folgen dem anthropomorphischen Prinzip, die Terme zur Beschreibung ihrer Funktionalität folgen also aus den Arten der Beschreibung menschlichen Handelns. In diesem Sinne ist es korrekt zu sagen, daß Agenten etwa sehen, handeln, entscheiden, planen oder sich irren.

Wesentlich für die Gestaltung von Agenten ist der *Sense-Compute-Act* Zyklus (SCA). Ein solcher SCA-Agent übernimmt dabei Daten aus der Umgebung nach einem sensorischen Prinzip (*sense*) und wertet diese selbständig aus (*compute*). In Folge der Auswertung kommt es zu einer Einflußnahme des Agenten auf seine Umgebung, die die sensorische Eingabe geliefert hatte (*act*). Neue sensorische Daten werden aufgenommen, ausgewertet und es wird erneut gehandelt usw. Ein Beispiel für den Einsatz von solchen Agenten in der Bildverarbeitung wurde in [87] vorgestellt. Dort wird die interne Berechnung eines SCA-Agenten mittels genetischer Programmierung anhand einer explizit vorgegebenen Zielstellung (dort das Auffinden von Rissen) evolutionär adaptiert. Ein solcher *Imagent* besitzt dabei stets eine Position im Bild und eine Orientierung (siehe Abbildung 5.1). Entsprechend der Orientierung nimmt der Imagent dabei Pixelwerte aus seiner lokalen Nachbarschaft auf und führt eine Berechnung mit diesen Werten aus. Das Ergebnis dieser Berechnungen kann dabei ein binärer Wert sein, z.B. 0 oder 1. Ist der Wert 1, „bewegt" sich der Agent um einen Pixel entsprechend seiner

Orientierung nach vorn, ist er 0, wechselt er seine Orientierung z.B. durch eine Drehung nach rechts. Dann werden wieder Pixelwerte aufgenommen, ausgewertet usw. In Folge vollführt der Imagent eine Bewegung entlang eines Pfades im Bild. Ein Imagent kann nun solche Aufgaben lösen wie die, eine Bildkante oder einen Riss zu finden und ihnen zu folgen, bestimmte Bildbereiche nicht zu verlassen oder gar Zeichen zu erkennen.

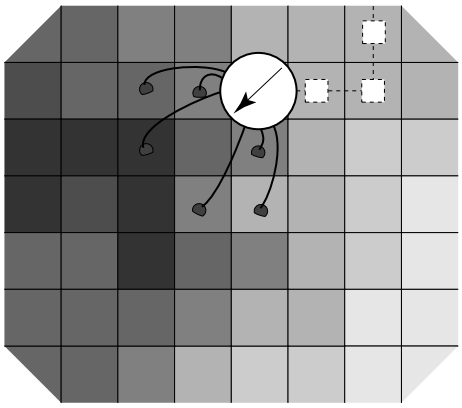


Abbildung 5.1: Ein Imagent hat eine Pixelposition und Orientierung in einem Bild. Nach einem bestimmten Schema erfaßt er die Pixelwerte der Umgebung und leitet aus ihnen mittels einer internen Berechnung eine Entscheidung über eine Folgebewegung ab. Auf diese Art absolviert ein Imagent einen Pfad in einem Bild, der einer Erkennungsaufgabe entsprechen kann.

Eine wichtige Frage in diesem Zusammenhang betrifft die Art, nach der eine komplexe Aufgabe in einfache Prozesse aufgeteilt wird. Durch das KOLMOGOROV-Theorem wird die Möglichkeit solch einer Aufteilung sichergestellt. Ein komplexes System kann durch Superposition einfacher Systeme ohne jeglichen Informationsverlust realisiert werden. Wie jedoch bereits dargestellt wurde, ist das Auffinden solch einer „Dekomplexisierung“ alles andere als einfach.

Ein Ausweg aus dieser Situation bietet die Organisation des Zusammenspiels zwischen den Agenten. In [21] werden die sogenannten *distinction networks* (*dnet*) als Beispiel dafür präsentiert, wie die globale Aktivität eines Ensembles von Softwareagenten unabhängig von der Spezifikation der einzelnen Mitglieder dieses Ensembles auf Basis lokaler Kommunikationen realisiert werden kann. Ein *dnet* als Netzwerk von Agenten, die sich nur lokal (also über die Verbindungen zwischen den Knoten des Netzes) beeinflussen, kann dabei als Ganzes einer komplexen Steuerung, einem mathematischen Theorem, einer logischen Schlußfolgerung oder einem Regelwerk entsprechen. Die einzige Sprache jedoch, die die Agenten verstehen, ist die der Modifikation dieser Verbindungen. Durch solches Kommunizieren jedoch, sozusagen implizit, beeinflussen sie jedoch gerade die globale Aktivität des Netzes und vereinfachen so z.B. eine durch sie repräsentierte logische

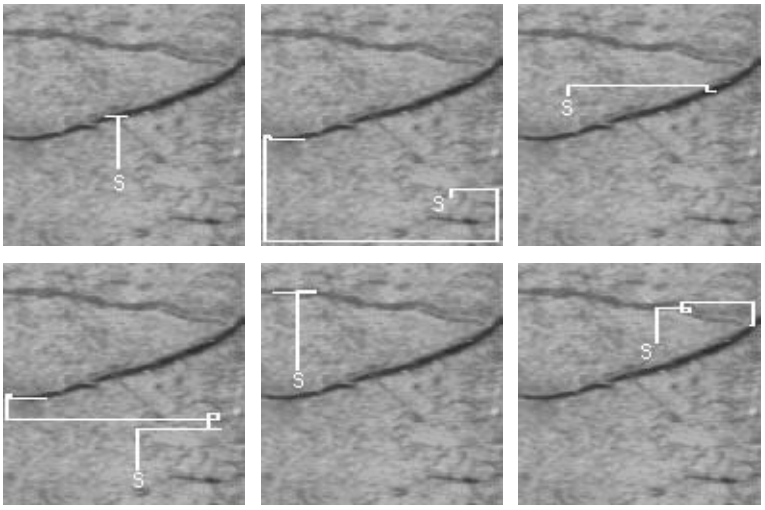



Abbildung 5.2: Pfade eines Imagent, der darauf trainiert wurde, Risse in Texturbildern zu finden, für verschiedene Startpunkte $S$ (aus [87]).

Schlußfolgerung, beweisen ein mathematisches Theorem, passen eine Steuerung an oder erweitern ein regelbasiertes Expertensystem. Bricken gibt als Beispiel die Umformung logischer Ausdrücke durch *dnets*, die sich selbst modifizieren.

Was allen diesen Ansätzen jedoch entgegensteht ist die universelle Repräsentation von Bildern durch Pixel. Wie bereits im Abschnitt 4.4.2 diskutiert wurde, ist die pixelweise Repräsentation von Bildern *maximal effizient*, d.h. umfassender als alle komplexeren Formen von Beschreibungen. Daher werden sich auch Agenten schwer dieser Form der Repräsentation entziehen können.

### 5.1.2 Repräsentation von Operationen

Neben den sensoriellen Daten muß ein *Soft Computing* System auch eine Repräsentation der Transformationen der Daten beinhalten. Wie z.B. bei der Merkmalklassifikation stößt man dabei stest auf Operationsketten, d.h. einer Reihe von Rechenschritten, die vom Ausgangsbild ausgehend zu einem bestimmten Ergebnisbild führen, aus dem das gesuchte Ziel leicht ableitbar ist.

Solch ein Ansatz kann als spezieller Fall der hier vorgeschlagenen allgemeinen Sichtweise als methodisches Gerüst (*framework*) angesehen werden [92]. Dabei sind einige Komponenten der zugehörigen Operationsfolgen oder der Relationen zwischen ihnen unspezifiziert und damit adaptiven Verfahren zugänglich. Man kann hier auch sagen, daß solch ein methodisches Gerüst erst *konfiguriert* werden muß (durch Festlegung von Parameterwerten, durch Operationsauswahl usw.), um eine voll funktionsfähige Datentransformation darzustellen.

Auf solche einfachen Operationsketten trifft zu, daß sie "niemals stärker sind als ihr schwächstes Glied". Um die Zuverlässigkeit und Robustheit solcher Systeme zu erhöhen, wird oft eine Rückoppelung von der letzten Stufe zu einer

Zwischenstufe eingeführt. Erweist sich das Ergebnis der Verarbeitungskette als unzureichend, kann die Verarbeitung an einer vorhergehenden Stufe ausgehend mit einer alternativen Konfiguration wieder aufgenommen werden. Diese Lösung ist jedoch nur begrenzt praktisch einsetzbar, da sich der Einfluß einer Änderung in einer Stufe auf das Endergebnis schwer vorhersagen läßt und da die Anzahl der Alternativen des Gesamtsystems proportional dem Produkt der Alternativen jeder einzelnen Stufe ist, mithin sehr schnell sehr groß werden kann.

Ein anderes Problem im Zusammenhang mit Ketten von Bildverarbeitungsoperationen ist nicht so offensichtlich: die hohe Dimension des Suchraums. Scheinbar gibt nur jeder freie Parameter einer der Operatoren der Kette eine weitere Dimension des Suchraums. Besteht eine Kette aus $n$ Operatoren mit jeweils $m$ freien Parametern, von denen jeder $k$ Werte annehmen kann, so besteht der so beschriebene Suchraum aus $k^{(nm)}$ Punkten. Tatsächlich jedoch ist der Suchraum durch die Menge aller möglichen Abbildungen des Eingangsbildes auf das Ausgangsbild gegeben. Die Anzahl der Abbildungen von $n$ Variablen mit $m$ möglichen Werten auf $k$ Werte ist $k^{(m^n)}$. Die Anzahl der durch die freien Parameter einer Kette *repräsentierbaren* Operationen ist von verschwindender Größe im Vergleich zur Anzahl aller möglichen Operationen, selbst in den einfachsten Fällen.

Dazu ein konkretes Beispiel: angenommen, daß Ziel der Bildverarbeitungskette besteht darin, aus dem Eingangsbild ein Bild zu generieren, welches einem vorgegebenen Zielbild so stark wie möglich ähnelt oder daß eine Reihe vorgegebener Eigenschaften besitzt. In so einem Fall ist das Konfigurierungsproblem ein Optimierungsproblem.

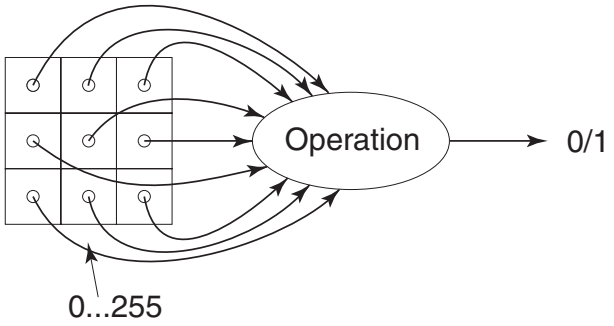



Abbildung 5.3: Eine Bildverarbeitungskette, die neun Grauwerte auf zwei abbildet.

Dazu betrachte man Abbildung 5.3, die eine einfache Bildverarbeitungskette darstellt. Die Berechnungen der Operatoren der Kette seien auf eine $3 \times 3$ Nachbarschaft jeden Pixels eingeschränkt, wobei der Grauwert jeder Bildposition ein Wert zwischen 0 und 255 sein kann. Das Ergebnis solch einer Operation soll ein Binärbild sein, also die Berechnungen für jeden Pixel lediglich Ergebniswerte aus z.B. $\{0, 1\}$ annehmen dürfen.

Ist eine Qualitätsfunktion $q$ für jede so mögliche Abbildung gegeben, besteht

dann die Aufgabenstellung darin, eine Abbildung zu finden, für den diese Funktion $q$ einen möglichst großen (oder kleinen) Wert annimmt. Für solch eine Abbildung sind die Funktionswerte von genau $256^9$ möglichen Aufpunkten zu spezifizieren. Dabei kann jeder zugeordnete Wert entweder 0 oder 1 sein. Die Anzahl möglicher Abbildungen ist also gleich der Anzahl möglicher $256^9$-stelliger Dualzahlen, also $2^{(256^9)}$ (da das Potenzieren nicht assoziativ ist, dürfen die Klammern niemals weggelassen werden).

Betrachtet man für dieses Beispiel eine Menge von Faltungsoperationen. Wenn eine gewichtete Maske durch folgende Menge gegeben ist

$$M_w = \begin{array}{|c|c|c|} \hline w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\ \hline w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\ \hline w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\ \hline \end{array}$$

mit $w_{ij} \in \{0, \ldots, 255\}$, dann ist das Ergebnis der Faltung des Bildes $I$ an der Position $(x, y)$ durch

$$R(x, y) = \sum_{(i,j) \in M_w} w(i, j) I(x + i, y + j). \tag{5.1}$$

gegeben. Um ein Binärbild zu erhalten, wird ein Schwellwert $\vartheta$ verwendet, der $m$ mögliche Werte annehmen soll. Man hat hier also $256^9 \cdot m$ mögliche Einstellungen der freien Parameter und damit auch lediglich $256^9 \cdot m$ repräsentierbare Operationen. Im Vergleich zur Größe des Suchraums von $2^{(256^9)}$ Elementen ist dies eine verschwindend geringe Anzahl. Um mit einer einfachen Bildverarbeitungskette den Suchraum gut auszufüllen, würde man $256^9/8$ freie Parameter benötigen!

Deshalb kommt der Frage, wie Suchräume von solcher Größe beim Einsatz von Verfahren des *Soft Computing* überhaupt einigermaßen effektiv gesampelt werden können, eine sehr große Bedeutung zu.

Das evolutionär entstandene primäre visuelle System der höheren Säugetiere gibt hier einen entscheidenden Hinweis, der in dem von Grossberg aufgestellten Modell des *Boundary Contour System / Feature Contour System* (BCS/FCS) [58] seinen Ausdruck findet. In diesem Modell verläuft die kortikale Informationsverarbeitung der Aktivierungen der Netzhaut entlang von *zwei* Pfaden. Bei der anschließenden Fusion der beiden Teilergebnisse kann selbst eine Abbildung einbezogen werden und damit die Anzahl repräsentierbarer Operationen dramatisch erhöht werden. Sollte es dann noch gelingen, diese Abbildungen unabhängig von den beiden Bildpfaden zu konfigurieren, kann die mit der Konfiguration des gesamten Systems zu lösende Aufgabenstellung weitaus effizienter angegangen werden.

Dies führt zu der Definition von $n$-dimensionalen methodischen Gerüsten. Dabei handelt es sich um die Zerlegung der Bildverarbeitung in $n$ parallel abgearbeitete Teile $op_1$ bis $op_n$ und einer abschließenden Fusion der Ergebnisse der $n$

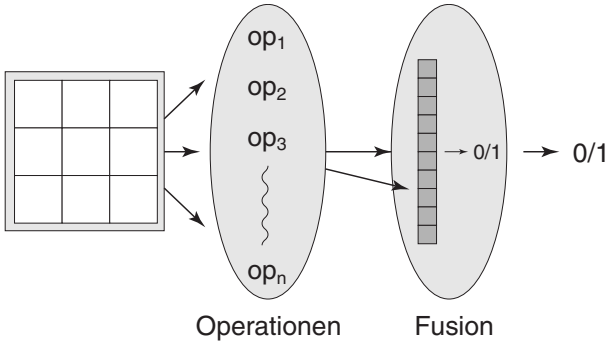



Abbildung 5.4: Ein $n$-dimensionales methodisches Gerüst, welches eine einfache Bildverarbeitungskette in $n$ parallel durchgeführte Operationen zerlegt, deren Ergebnisse abschließend fusioniert werden. Diese Fusion kann selbst eine Abbildung enthalten, die unabhängig von den Operationen spezifiziert werden kann.

Einzelverarbeitungen zum Ergebnisbild des gesamten Systems (siehe Abbildung 5.4).

Erfolgt die Fusion durch Spezifikation einer Abbildung von $k$ Variablen, die jeweils $m$ Werte annehmen können, auf die Menge $\{0, 1\}$, können so $2^{(m^k)}$ Abbildungen dargestellt werden. Wenn $m$ 256 ist (also bspw. ein Grauwert) und $k$ 9, ist es so möglich, alle Abbildungen wie im obigen Beispiel darzustellen. Jedoch sieht es jetzt so aus, daß man zur Spezifikation von $2^{(256^9)}$ möglichen Abbildungen des Eingangsbildes auf das Ausgangsbild $2^{(256^9)}$ Abbildungen innerhalb der Fusionseinheit festlegen müßte. So hätte man scheinbar nichts gewonnen.

Dies ist jedoch nicht so, da diese beiden Abbildungen qualitativ verschieden sind. Drei mögliche Ansätze hier sind:

- Die Abbildung, die die Fusionseinheit ausführt, kann direkt auf das Ziel bezogen aus den Ergebnissen der Einzeloperationen abgeleitet werden.
- Insofern die Einzelergebnisse *interpretierbar* sind, kann ein Regelsystem, z.B. auf Basis von Fuzzy-Regeln, die benötigte Abbildung aus *a priori* Wissen gestalten lassen.
- Es wird nicht direkt eine Abbildung zur Fusion eingeführt, aber ein bewährter Algorithmus verwendet.

In jedem dieser Fälle reduziert sich das mit der Konfiguration des Gesamtsystems zusammenhängende Optimierungsproblem auf die Wahl optimaler Einstellungen für die Einzeloperationen selbst (oder einer guten Wahl für die Operationen). Durch die intrinsische Spezifikation der Fusionseinheit kommt es damit zu einem „intrinsischen Parallelismus“ vergleichbar zur Suchstrategie genetischer Algorithmen. Mit jeder Konfiguration der Einzeloperationen werden alle durch die Fusionsabbildung möglichen Gesamtabbildungen gleichzeitig untersucht. Insofern die-

se Fusionsabbildung direkt aus den Einzeloperationen in Bezug auf das Gesamtziel abgeleitet wird, wird quasi eine gesamte Teilmenge des Suchraums anhand eines Repräsentanten gesampelt, wobei dieser eine Repräsentant von vornherein als bester aller Elemente dieser Teilmenge angenommen wird.

Darin liegt die grundlegende Idee mehrdimensionaler methodischer Gerüste begründet: anhand der Adaption von $n$ Teiloperationen die Adaption eines Gesamtsystems so vorzunehmen, daß der Suchraum für das Gesamtsystem einigermaßen repräsentativ gesampelt wird.

Ein solches System, auf Basis der Verwendung von Fuzzy-Regeln, bei denen zwei Einzeloperationen verwendet werden, ist in [111] dargestellt. Hierbei liefert eine Operation ein Kantenbild, eine zweite Operation ein Hintergrundbild. Anhand von Fuzzy-Regeln, die auf der physiologisch unterschiedlichen Wahrnehmung von Kontrasten auf hellen oder dunklen Bildhintergründen basieren, lassen sich so die Ergebnisse herkömmlicher Kantenoperatoren (die hier eigentlich nur das erste Einzelbild liefern) entscheidend verbessern.

Ein System auf Basis des Wasserscheidenalgorithmus der mathematischen Morphologie ist in [91] dargestellt. Durch die Wasserscheidenoperation lassen sich die Ergebnisse einer Kanten- und Segmentierungsoperation fusionieren, in einer Art, die es erlaubt, die Schwachstellen jeder dieser beiden Teiloperationen im Gesamtergebnis wieder auszugleichen.

Methodische Gerüste sind daher für die Einbringung von Ansätzen des *Soft Computing* in Bildverarbeitungssysteme besonders interessant. Sie ermöglichen eine umfassendere Repräsentation des Suchraumes durch die Verwendung interner Dekompositionen und Abbildungen (möglicherweise sogar vergleichbar zu den internen Karten des Gehirns) bei gleichzeitiger intrinsicher Spezifikation dieser Abbildungen unabhängig von den konkreten und zu prozessierenden Daten.

### 5.1.3 Fitnesssteuerung

Das Schemata-Theorem beschreibt die Möglichkeit für genetische Algorithmen, bei einem gegebenen Optimierungsproblem zu einer guten Lösung zu gelangen. Es operiert unter der Voraussetzung, daß es in der gewählten Repräsentation des Optimierungsproblems durch Bitstrings auch sog. *building blocks* gibt, also fixe Abschnitte des Bitstrings, die tendenziell zu einer besseren Lösung beitragen [165].

Dies hilft, die Anwendbarkeit genetischer Algorithmen in einem anderen Licht zu sehen als das einer *yet another* neuen Familie von heuristischen Optimierungsansätzen. Bei gegebenem Optimierungsproblem kann die Wahl zum Finden einer guten (oder optimalen) Lösung auf einen genetischen Algorithmus fallen, da dies etwa durch die leicht umsetzbare Problemrepräsentation motiviert ist. Ebenso ist natürlich, wie bei jedem *Soft Computing* Verfahren, der Aufwand des Algorithmus an den Nutzen anpaßbar, sei es durch die Aufwendigkeit der Kodierung und Dekodierung eines Bitstrings, durch die verwendeten genetischen

Operatoren, oder durch die Größe der Population und der Anzahl der absolvierten Generationen.

In diesem Abschnitt soll jedoch auf oben benannten anderen Aspekt eingegangen werden. Dieser besteht darin, den genetischen Algorithmus nicht als einen Algorithmus zu sehen, der zu einem bestimmten Zustand führt (der Problemlösung), sondern als einen Prozeß, der wesentliche Aspekte der Problemlösung herausstellt, die anderweitig genutzt werden können.

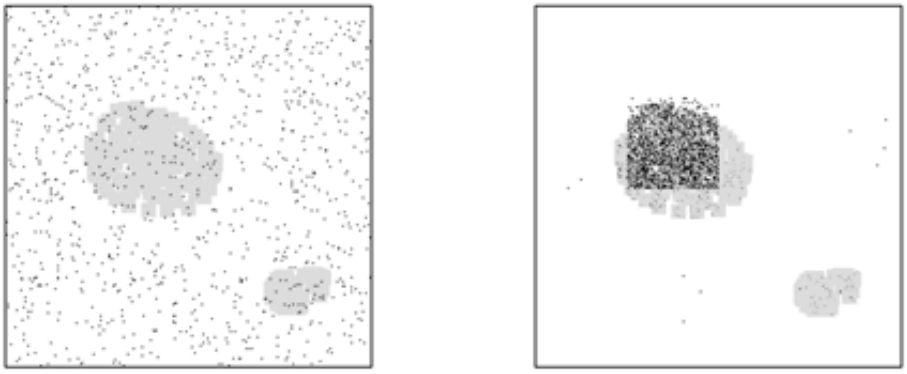



Abbildung 5.5: Schematasuche durch einen genetischen Algorithmus: (a) Zufallsinitialisierung, jeder Punkt entspricht einem Individuum, die grauen Bereiche sind das "Manna"; (b) Ergebnis nach zehn Generationen (nach http://www.caplet.com/MannaMouse.html).

In solchen Anwendungen ist nicht unbedingt ein guter Wert der Fitnessfunktion das Ziel, sondern die Anhäufung überdurchschnittlich guter Schemata in der Population. Dieser Aspekt läßt sich anhand von Abbildung 5.5 erläutern. Hier beschreibt jedes Individuum eines genetischen Algorithmus einfach eine $(x, y)$-Position im Einheitsquadrat. Die Fitnessfunktion ist durch die Vorgabe gegeben, daß diese Positionen sich innerhalb des sogenannten Manna befinden sollen, also bestimmten vorgegebenen Bereichen innerhalb des Einheitsquadrates. Einfach gesagt: die Individuen sollen ihr "Futter" finden. Wie man in Abbildung 5.5 sieht, liegt dieses in Form von zwei zusammenhängenden Clustern vor, die sich in der eingenommenen Fläche voneinander unterscheiden. Was nach zehn Generationen auffällt, ist, daß sich die gesamte Population ausschließlich auf das größere Cluster konzentriert. Dies entspricht natürlich der exponentiellen Zunahme überdurchschnittlich guter Schemata in der Population, wie durch das Schemata-Theorem beschrieben.

Damit operiert ein genetischer Algorithmus anders als klassische Verfahren. Diese würden die Individuen eher in einem Verhältnis auf die Cluster aufteilen, welches etwa durch die Flächenverhältnisse der Cluster gegeben ist.

Der hier interessierende Aspekt ist, daß dies auch als eine Volumenmessung in-

terpretiert werden kann. Der genetische Algorithmus hat die Lösung des Problems gefunden, unter allen existierenden Clustern das größte zu finden.

Dies motiviert andere Anwendungsmöglichkeiten genetischer Algorithmen. Eine solche Anwendung wurde in [93] und [94] beschrieben. Die Fragestellung war dort das Auffinden eines Tabellenbereiches im digitalen Bild eines Rechnungsformulars.

Durch Operatoren der mathematischen Morphologie und Histogrammauswertungen lassen sich alle Textzeilen des Dokumentes extrahieren. Innerhalb der Sequenz von Textzeilen kann nun der Tabellenbereich als Teilsequenz beschrieben werden, bei der eine höhere Ähnlichkeit der Zeilen zueinander besteht als zu anderen Zeilen, aber so gut wie niemals exakte Gleichheit. Damit läßt sich dann die benannte Strategie der *Fitnesssteuerung* hier praktisch leicht umsetzen. Ein genetischer Algorithmus wird hier so konfiguriert, daß er die Aufgabe lösen soll, ein Schema für eine Textzeile des Rechnungsformulars zu entwickeln, *das zu allen Textzeilen des Dokumentes gleich ist*. Natürlich besitzt diese Aufgabe keine exakte Lösung. Es gibt eine klassische optimale Lösung, wenn etwa ein mittlerer Abstand des gesuchten Schemas zu allen Textzeilen minimiert werden soll. Dies würde auf einen Mittelwert hinauslaufen.

Für den genetischen Algorithmus besteht die Lösung darin, daß die Individuen der Population überdurchschnittlich häufig eine hohe Ähnlichkeit zu den zur Tabelle gehörenden Zeilen aufweisen.

Bei der konkreten Umsetzung wurden dabei alle extrahierten Zeilen durch Bitstrings repräsentiert. Das digitale Teilbild jeder Zeile wurde in 40 gleichgroße Bereiche eingeteilt. Für jeden Bereich, in dem der Anteil schwarzer Pixel mehr als 30 Prozent betrug, wurde an der entsprechenden Stelle im Bitstring das Bit 1 gesetzt, sonst 0.

Ein Individuum des genetischen Algorithmus ist dann ein Bitstring der Länge 80. Zur Berechnung der Fitness solch eines Bitstrings wurde folgende Prozedur verwendet:

1. Zu einer Textzeile wird die HAMMING-Distanz zum linken Halbstring berechnet.

2. Ist dieser Wert größer als ein vorgegebener Zuverlässigkeitsparameter, wird stattdessen die HAMMING-Distanz zum rechten Halbstring berechnet. Der hier gewählte Wert betrug 0.9.

3. Die Fitness ist der Mittelwert aller so ermittelten HAMMING-Distanzen über alle Textzeilen.

Die Trennung des Bitstrings in zwei Hälften bringt hier den Vorteil, daß Überanpassungen in der Population unterdrückt werden.

Nach fünfzig Generationen mit 100 Individuen wird geprüft, ob ein Fitnesswert von 0.7 überschritten wurde. Erst hierdurch konnte sichergestellt werden, daß

sich im Bild der Rechnung tatsächlich eine sicher extrahierbare Tabelle befindet (so existieren auch Fälle von Leergutabrechnungen, die einer Dummytabelle mit beigefügt wurden). Dann wurden alle Zeilen mit hoher Ähnlichkeit zu dem linken oder rechten Halbstring des besten Individuums der Tabelle zugeordnet.

Eine Evaluierung des Gesamtsystems anhand von 12 Rechnungsformularen ergab unter 184 vorhandenen Tabellenzeilen, daß durch den genetischen Algorithmus nur 11 Zeilen ausgelassen und 12 zuviel hinzugefügt wurden. Solche Fehler ließen sich jedoch aus den Forderungen nach zusammenhängenden Textzeilen und der Konsistenz der Preiseinträge weiter korrigieren.

Ein andere Anwendung ist das *evolutionäre robuste Clustern* [179]. Hierbei werden evolutionäre Verfahren zum Clustern von Daten verwendet. Die Idee besteht dabei darin, mittels eines genetischen Algorithmus aus einer Reihe von Clusterungen durch *andere Verfahren* die konsistente Clusterung abzuleiten. Der evolutionäre Algorithmus operiert also nicht mit den zu clusternden Daten, sondern mit den Ergebnissen einer Reihe von Clusterverfahren auf diesen Daten.

Die Einteilung einer Gruppe von Daten in ähnliche Gruppen ist eines der ältesten und wichtigsten Aufgabenstellungen in vielen Gebieten von Wissenschaft, Technik und Wirtschaft. Es wurden bereits eine Vielzahl von Lösungen entwickelt, den sogenannten Clusterverfahren (siehe etwa [200], [28], [43], [11], [22], [196], [195], [27], [17], [180], [119], [76], [10], [24], [36], [109], [121]).

Die Motivation für die Verwaltung einer größeren Anzahl von Clusterverfahren liegt in den teilweise unterschiedlichen Anforderungen an die Ähnlichkeit von Daten. Abbildung 5.6 gibt dafür ein Beispiel. Hier wurde der Testdatensatz der *Intertwined Spirals* verwendet. Dieser Datensatz, der auf eines der von Minsky und Papert in [127] benannten Probleme verweist (siehe S. 20), wurde 1988 von Alexis Wieland von der MITRE Corp. im Internet erstmalig präsentiert und sofort von vielen Forschern aufgegriffen und studiert. In [103] findet sich die erste genaue Studie zu diesem Datensatz. John Koza ([98], Kapitel 17), Gail Carpenter [25] und Peter G. Anderson [4], um nur einige zu nennen, haben ihn in Folge genutzt, um ihre Lernmethodik zu exemplifizieren (siehe auch [46] [185] [101] [75] [97] [160]).

Die Abbildung zeigt die unterschiedliche Behandlung desselben Datensatzes durch zwei Clusterverfahren (*single linkage agglomerative clustering* und *average linkage agglomerative clustering*, siehe [179]). Der Unterschied beider Verfahren liegt in der unterschiedlichen Berechnung des Abstands zweier Cluster. Beim *average linkage* Verfahren wird dieser auf Basis der Schwerpunkte beider Cluster bestimmt. Beim *single linkage* dagegen erfolgt die Berechnung auf Basis zufällig ausgewählter Elemente der prospektiven Cluster. Die Folge ist, daß das *single linkage* Verfahren die Spiralen korrekt trennen kann, das *average linkage* Verfahren jedoch nicht.

Diesem Beispiel mögen Fälle gegenüberstehen, bei denen das *average linkage* Verfahren besser funktioniert, wie z.B. bei linear separierbaren zusammenhängenden Clustern. Wesentlich ist, daß es zu gegebenen Daten *passende* und *nicht*

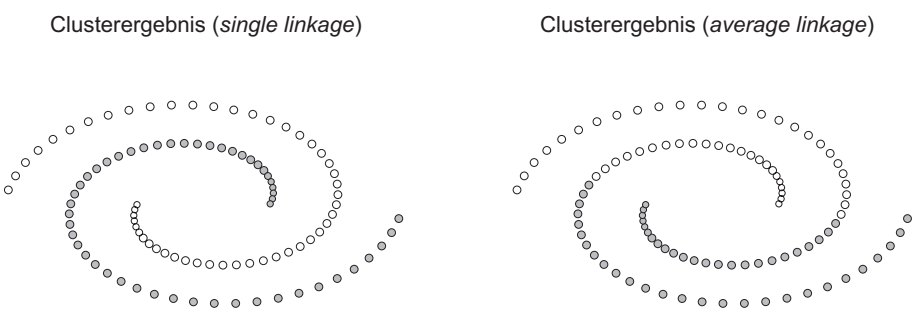



Abbildung 5.6: Clustern des *Intertwined Spirals* Datensatzes von Wieland: *average linkage agglomerative clustering* (oben) und *single linkage agglomerative clustering* (unten). Das *single linkage* Verfahren ist in der Lage, die beiden Spiralen zu trennen.

*passende* Clusterverfahren gibt.

Die Grundthese des robusten Clusterns besteht nun in folgendem: Wenn ein Clusterverfahren für einen gegebenen Datensatz nicht passend ist, wird man zu einem gegebenen Datensatz stärker variierende Ergebnisse erhalten, wenn die Konfiguration des Clusterverfahrens zufällig geändert wird, oder wenn die Daten selbst leicht zufällig modifiziert werden.

Sei $d = (d_1, d_2, \ldots, d_n)$ der Datensatz, der in zwei Cluster $C_1$ und $C_2$ einzuteilen ist. Weiter seien $M_1, M_2, \ldots, M_k$ verschiedene Clusterverfahren, die von initialen Konfigurationen $c_i$ abhängen. Mit $\tilde{c_i}$ werde eine zufällige Variation der initialen Konfiguration des Clusterverfahrens $M_i$ bezeichnet, und mit $\tilde{d}$ eine zufällige Variation der Daten (z.B. durch Addition eines $N(0, \sigma)$ normalverteilten Vektors).

Damit lassen sich eine Reihe von Einteilungen der Daten $d$ in zwei Cluster ermitteln: $C_{\tilde{d}M_i(\tilde{c_i})}$ ist dann das Ergebnis der Einteilung der $d$ in zwei Cluster unter der Konfiguration $\tilde{c_i}$ des Clusterverfahrens $M_i$ und der Datenvariation $\tilde{d}$. Jedes Ergebnis einer Clusterung läßt sich durch einen Bitstring repräsentieren, etwa Bit 1 an Position $i$, wenn Datum $d_i$ dem Cluster $C_1$ zugeordnet wurde, 0 sonst.

Man erhält also eine Anzahl von Bitstrings, die aus dem Operieren von Clusterverfahren auf $d$ resultieren. Haben wir bspw. fünf Variationen $\tilde{d}$ der Daten $d$ und fünf Clusterverfahren mit jeweils zwei Modifikationen der initialen Konfiguration, erhält man zu $d$ genau $5 \cdot 5 \cdot 2 = 50$ verschieden Bitstrings.

Obige These schlägt sich nun darin nieder, daß unter den so erhaltenen Bitstrings sich eine Untermenge von Bitstrings finden läßt, die untereinander ähnlicher zueinander sind als zu anderen Bitstrings. Dies läßt sich nun genau durch den Ansatz der Fitnesssteuerung umsetzen.

Wie bei der Tabellenfindung wird also ein genetischer Algorithmus verwendet, der einen Bitstring finden soll, der zu allen Cluster-Bitstrings gleich ist. Um es also noch einmal herauszustellen: der genetische Algorithmus operiert hier nicht mit den Daten direkt. Er ermittelt aus einer Reihe von Clusterergebnissen

eine möglichst konsistente Untergruppe. Diese Untergruppe dann verweist auf die zu diesem Datensatz passenden Clusterverfahren. Man erhält also sowohl durch Auswahl eines Bitstrings eine Clusterung der Daten (die jedoch von einem anderen Verfahren vorgenommen wurde), als auch das Verfahren selbst, das am besten zu den Daten paßt.

Durch Bereitstellung einer Gruppe von Clusterverfahren, die für eine Vielzahl von Daten angemessen sind, läßt sich so eine Art "blinde Clusterung" tatsächlich durchführen.

Eine Studie aus [179] untersuchte den *Intertwined Spirals* Datensatz näher und mittels der beschriebenen Methode des evolutionären Clusterns wurde tatsächlich die korrekte Einteilung des Datensatzes in zwei Spiralen gefunden. Bemerkt werden konnte noch, daß der beste Bitstring der Population aus 30 Individuen nach 300 Generationen die größte Ähnlichkeit zum unmodifizierten Datensatz $d$ selbst hatte.

Als letztes Beispiel sollen hier die *heuristischen Maße* erwähnt werden, die ebenfalls auf dem Konzept der Fitnesssteuerung basieren. In [88] wurde dies am Beispiel der Auswertung von KIRLIAN-Fotografien durchgeführt. Zielstellung war hierbei die Quantifizierung der visuellen Ähnlichkeit von KIRLIAN-Mustern derselben Person, die zu verschiedenen Zeiten aufgenommen wurden (siehe Abbildung 5.7).

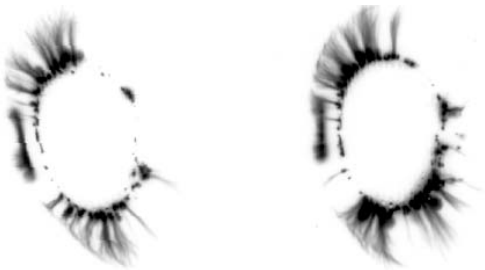


Abbildung 5.7: Ähnlichkeit von KIRLIAN-Mustern derselben Person.

Die Frage nach einer Intraindividualität von KIRLIAN-Mustern ist Gegenstand einer langjährigen Diskussion ([118], [30], [79], [82], [136], [137], [171], [172]). Im konkreten Fall wurden die nahezu zirkulären KIRLIAN-Muster polar abgewickelt, und mittels Schwellwertentscheidungen wurde dadurch ein Schema des KIRLIAN-Bildes eines einzelnen Fingers abgeleitet (siehe Abbildung 5.8). Diese Schemata lassen sich einfach durch Bitstrings beschreiben, und das Ziel ist es nun, die Ähnlichkeit von Schemata zu derselben Person quantitativ herauszustellen.

Auch hierzu kann ein genetischer Algorithmus verwendet werden. Bei dem Beispiel der Tabellensuche (siehe S. 85) wurde erwähnt, daß erst ein Fitnesswert des besten Individuums über 0.7 als Indiz für eine gefundene Tabelle genommen wurde. Dahinter versteckt sich auch die Idee eines *heuristischen Maßes*, also einer mittels eines heuristischen Verfahrens ermittelten Quantifizierung von Daten.

Genau wie bei einer klassischen Meßprozedur wird die zu messende Größe aus dem Antwortverhalten eines geeignet stimulierten Systems abgeleitet. Dies ist in der Regel ein deterministisches System, d.h. bis auf statistische Schwankungen wird stets dieselbe Antwort des Systems bei gleicher Stimulierung erwartet. Bei der heuristischen Messung jedoch zählt nicht der ermittelte Wert, sondern die Tatsache der Erzeugung ähnlicher Werte bei ähnlichen Stimulierungen. Das Ergebnis ist kein Messwert, sondern eine Klassifikation der Stimulierungen.

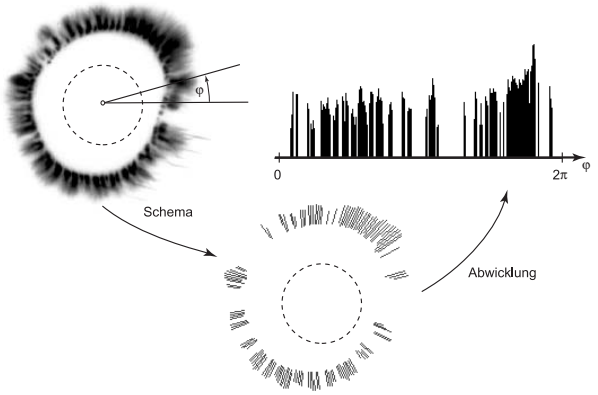



Abbildung 5.8: Abwicklung von KIRLIAN-Bildern zur Generierung eines Schemas.

Um mit dem Beispiel der aus den KIRLIAN-Mustern abgeleiteten Schemata fortzufahren: Ziel der Messung ist die Bestimmung des Grades der *schematischen* Ähnlichkeit zwischen Mustern derselben Person zu verschiedenen Zeiten. Die Motivation dafür liegt in der Möglichkeit, daß es im Muster zu Ausfällen ganzer Gruppen kommen kann, das gesuchte Maß also diese Art von Nichtübereinstimmung tolerieren können muß. Gesucht sind also genau *building blocks* in den Mustern, die auf Intraindividualität hinweisen.

Ein genetischer Algorithmus soll nun einen Bitstring suchen, der zu allen abgewickelten Mustern einer Gruppe von KIRLIAN-Bildern gleich ist. Durch die so gezielt genutzte Fitnesssteuerung wird sich so die schematische Ähnlichkeit einer Gruppe von KIRLIAN-Bildern ermitteln lassen. In der in [88] beschriebenen Studie wurden drei Gruppen von Bildern jeweils so untersucht:

- Gruppe 1 bestand aus Zufallsbildern, die also nicht aus KIRLIAN-Mustern entstanden waren. Hier wurden Fitnesswerte zwischen 0.2578 und 0.2930 erzielt.
- Gruppe 2 bestand aus KIRLIAN-Mustern verschiedener Personen. Hier ergaben sich Fitnesswerte zwischen 0.2815 und 0.3164.
- Gruppe 3 schließlich bestand aus KIRLIAN-Bildern derselben Person. Hier traten durchgehend Fitnesswerte über 0.34 (und bis zu 0.53) auf.

Man sieht also klar die Tendenz, bei gleichen Personen höhere Fitnesswerte der schematischen Gleichheit zu produzieren.

Diese Art der Vermessung mittels genetischer Algorithmen, die klar die durch das Schemata-Theorem verifizierte Herausstellung von *building blocks* einer Struktur nutzt, ist sicher sehr verschieden von einer klassischen Messung, liefert jedoch dennoch praktisch genausogut verwertbare Ergebnisse. Trotz Verwendung eines heuristischen Ansatzes ist sie, wie die Ergebnisse in [88] zeigen. dennoch reproduzierbar.

Generell werden also durch die Fitnesssteuerung Anwendungen genetischer Algorithmen (und anderer evolutionärer Verfahren) ermöglicht, die die Schemata-behandlung solcher Verfahren direkt ausnutzen. Die Anwendung beruht stets auf einer Fitnessfunktion, die den Grad der Übereinstimmung von Bitstring zu einer Gruppe von Bitstrings, die sich aus der jeweiligen Anwendung ergibt, beschreibt.

## 5.2 Gestaltungsziele

### 5.2.1 Optimierungsziele

Die Anpassung der internen universellen Repräsentationen eines *Soft Computing* Verfahrens an den konkreten Sachverhalt kann als eine Optimierungsaufgabe angesehen werden. Wie bereits bei den Ausführungen zum Schemata-Theorem in Abschnitt 4.5 dargelegt, ist der Nutzen des *Soft Computing* Verfahrens jedoch nicht ausschließlich in der Erreichung des Optimierungszieles begründet. Schließlich lassen sich auch Lösungsansätze formulieren, bei denen das gesuchte Optimum in den Daten gar nicht existieren kann.

Tatsächlich zählt für bspw. evolutionäre Verfahren nicht das Ziel, sondern der Weg zum Ziel. So stellt die Fitnessfunktion auch einen Kursor für den Verlauf der Evolution dar.

### 5.2.2 Nutzerschnittstelle

Ein weiterer Vorzug von *Soft Computing* Systemen sollte in keiner Anwendung unberücksichtigt bleiben: die möglichst einfache Gestaltung der Nutzerschnittstelle. Dies hängt mit der universellen (und durch das KOLMOGOROV-Theorem sanktionierten) Repräsentationsfähigkeit der im *Soft Computing* benutzten Ansätze zusammen. Damit wird sich stets eine Art "Black Box" Verhalten beim Umgang mit *Soft Computing* Verfahren aufzeigen lassen, da universelle Repräsentationen natürlich auch "blind" gegenüber individuellen Eigenheiten der modellierten Systeme sind.

So liegt die Nutzerschnittstelle bei einem MBPN (siehe Abschnitt 2.2.1) einfach in der Übergabe von Eingangs- und Trainingsdaten. Einem evolutionären

Algorithmus ist im wesentlichen nur die Kodierungs- und die Fitnessfunktion vorzugeben.

Natürlich darf hier nicht übersehen werden, daß in den Einzelfällen spezielle Anpassungen zur Verbesserung des Ergebnis beitragen können. Das entspricht der bereits benannten *exploitation of the tolerance for imprecision and uncertainty*.

Generell beinhalten auf *Soft Computing* basierende Lösungen jedoch stets dieses Moment der möglichst einfachen und vor allem voraussetzungslosen Nutzerschnittstelle. Anders gesagt: folgt man tatsächlich dem biologischen Vorbild, so ist etwa beim Menschen auch nie eine spezielle Konfiguration für z.B. das Sehen eines bestimmten Gegenstandes notwendig. Dieses Sehen geschieht stets authentisch, und ist voraussetzungslos.

Als einfache Form einer Nutzerschnittstellen, wie sie *Soft Computing* Verfahren benutzen sollten, wird hier speziell gesehen: Übergabe von Eingangs- und Trainingsdaten, zur Spezifizierung dessen, was man hat, und was man daraus abgeleitet haben möchte. Dies kann auch in ikonischer Weise geschehen, wie im nächsten Kapitel dargestellt wird.

## 5.3 Performancemessung

Das "No Free Lunch" gibt eigentlich eine ultimative Antwort auf die Möglichkeit der Messung der Leistungsfähigkeit von Verfahren. Hier wäre die Antwort, daß alle Verfahren gleich gut sind. Diese Aussage gilt natürlich immer nur unter Bezug auf *alle* möglichen Zielfunktionen einer Klasse. Im konkreten Fall hat man jedoch nur eine Zielfunktion, die zu verschiedenen Klassen gehören kann.

In diesem Zusammenhang ist das seit jeher praktizierte *Benchmarking* immer noch der beste Kompromiß. Das zu untersuchende Verfahren wird auf eine Suite von Testproblemen angewendet. Damit lassen sich mehrere Verfahren miteinander direkt vergleichen, in dem sie auf dieselbe Suite von Testproblemen angewendet werden, und es können durch die Wahl der Testfunktionen bestimmte Stärken und Schwächen des Ansatzes herausgestellt werden.

So kann die Leistung einer auf *Soft Computing* basierenden Lösung genauso wenig *exakt* gemessen werden, wie etwa die menschliche Intelligenz durch den IQ. Das heißt, sie kann nicht einfach auf einen Zahlenwert reduziert werden, und wenn doch, dann nur, um etwa einen direkten Vergleich zwischen Alternativen zu erlangen. Faktoren zur Beurteilung eines *Soft Computing* Ansatzes könnten etwa sein:

- Einfachheit der Nutzerschnittstelle,
- Implementierungsaufwand,
- Korrekturmöglichkeiten an den Lösungen,
- Intelligibilität der erlangten Lösungen und

- Versatilität des Ansatzes.

Wie man sieht, sind das alles für sich schwer quantifizierbare Eigenschaften.

# 6 Das LUCIFER System zur Texturanalyse

In diesem Kapitel wird eine Herangehensweise an Aufgabenstellungen der Texturfilterung vorgestellt, die den effektiven Einsatz von *Soft Computing* Verfahren ermöglicht. Die mehrdimensionalen Gerüste (siehe Abschnitt 5.1.2) stellen einen praktikablen Kompromiß (*trade-off*) zwischen der Größe der mit Bildverarbeitungsproblemen zusammenhängenden Suchräume und den Möglichkeiten von *Soft Computing* Verfahren dar. Dabei wird ein zweidimensionales methodisches Gerüst zur Adaption von Texturfiltern eingesetzt. Es basiert auf dem aus der mathematischen Morphologie bekannten 2D-Lookup Algorithmus. Mittels genetischer Programmierung wird eine optimierte Konfiguration dieses Algorithmus auf Basis einer ikonischen Nutzervorgabe (Zielbild) gesucht. Im Ergebnis erhält man sofort einsetzbare Texturfilter. Eine Verbesserung des Ansatzes auf Basis neuronaler Netze und eines optionalen Vorverbeitungsmoduls werden ebenfalls vorgestellt und diskutiert.

## 6.1 Gesamtübersicht

Das im folgenden ausführlich beschriebene System LUCIFER (*lookup compositional inference*) ist schematisch in Abbildung 6.1 dargestellt. Es besteht aus den Komponenten Eingangsbild (vom Nutzer bereitgestellt), Filtergenerator, Operationsbilder 1 und 2, Ergebnisbild, Zielbild (vom Nutzer gestellt), 2D-Lookup Matrix, Vergleichseinheit und Filtergenerierungssignal.

Das System kann auch als aus drei sich überlappenden Schichten zusammengesetzt verstanden werden:

1. Die *Instruktionsschicht*, die aus den vom Nutzer bereitzustellenden Komponenten Eingangsbild und Zielbild besteht. Der Nutzer kann jedoch das System auch via vorgegebener Operationen oder 2D-Lookup Matrix instruieren.

2. Die *Algorithmusschicht*, innerhalb der der eigentliche 2D-Lookup Algorithmus durchgeführt wird sobald alle seine notwendigen Komponenten (Eingangsbild, Operationsbilder 1 und 2, 2D-Lookup Matrix) konfiguriert sind.

3. Die *Adaptionsschicht*, die aus allen adaptierbaren Komponenten des Systems (Operationen 1 und 2, 2D-Lookup Matrix) und zur Durchführung

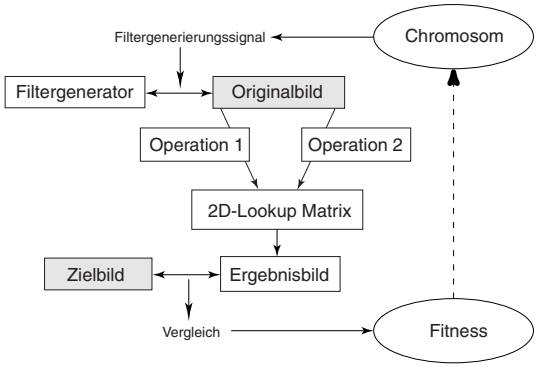



Abbildung 6.1: LUCIFER System zur automatischen Generierung von Texturfiltern.

der Adaption genutzter Komponenten (Vergleichseinheit, Filtergenerator) besteht.

In Bezug auf die Instruktionsschicht wurde das Nutzerinterface von LUCIFER so einfach wie möglich gestaltet. Das System wird durch ein manuell, z.B. mit Hilfe eines Fotobearbeitungsprogramms wie Photoshop erstelltes Zielbild dahingehend instruiert, was der zu generierende Filter leisten soll. In diesem Zielbild werden alle Bereiche des prospektiven Texturhintergrunds Weiß gemalt (Pixelwert 1, oder Grauwert 255), und alle Bereiche, die zum Vordergrund gehören (Texturfehler, Handschrift) Schwarz (Pixelwert 0 oder Grauwert 0). In Abbildung 6.2 ist die visuelle Erscheinungsform eines Schußfehlers in einer Textilie (a) in ein Zielbild (b) umgesetzt worden.

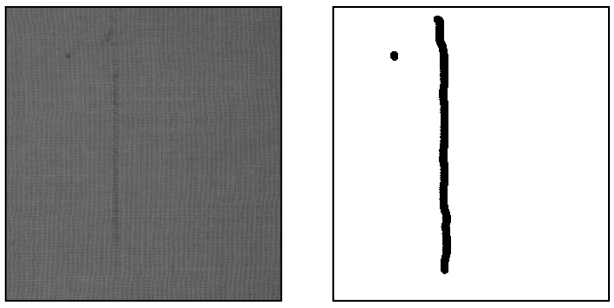



Abbildung 6.2: Beispiel für die Instruktion von LUCIFER zur Filterung eines Textilfehlers: Eingangsbild (a) und manuell erstelltes Zielbild (b).

Dadurch ist das System in seinem Ansatz datengetrieben. Kein spezielles Texturmodell muß dem Nutzer bekannt sein. Es gibt auch keine weiteren Forderungen an das Zielbild außer binär zu sein und in etwa die visuelle Repräsentation der Fehlererscheinung (bzw. des Vordergrundes der Hintergrundtextur) zu liefern.

In der Algorithmusschicht findet der im nächsten Abschnitt ausführlich dargestellte 2D-Lookup Algorithmus statt. Der Vorteil dieses Ansatzes (im Gegensatz zu einer allgemeinen Operationskette) liegt in der Zerlegung der Gesamtoperation in zwei parallel ablaufende Operationen, von denen jede für sich und unabhängig von der jeweils anderen an die konkrete Zielstellung angepaßt werden kann.

Damit handelt es sich bei diesem Ansatz um ein zweidimensionales methodisches Gerüst (siehe Abschnitt 5.1.2) zur Adaption von Texturfiltern. Wie weiter unten gezeigt wird (Abschnitt 6.4), ist das System so gestaltet, daß sich die Fusionsoperation direkt aus den Teilergebnissen der beiden Einzeloperationen ableiten läßt.

Adaption wird als ein Optimierungsproblem betrachtet, und evolutionäre Algorithmen, insbesondere die genetische Programmierung (siehe Abschnitt 2.2.4), kommen hier zum Einsatz. Die dazu benötigte Fitnessfunktion, deren Werte den Selektionsdruck der natürliche Evolution emulieren, wird in der Vergleichseinheit ermittelt und quantifiziert die Ähnlichkeit von einem aus einer aktuellen Konfiguration der Algorithmusschicht resultierenden Ergebnisbild und dem fest vorgegebenem Zielbild.

## 6.2 Der 2D-Lookup Algorithmus

Der 2D-Lookup Algorithmus ist ein Verfahren der mathematischen Morphologie [153] [154] [161] [64]. Er diente ursprünglich zur Segmentierung von Farbbildern, läßt sich jedoch genauso auf die Bearbeitung von Grauwertbildern übertragen.

Für die Ausführung des 2D-Lookup Algorithmus werden zwei Operationsbilder 1 und 2 gleicher Größe und Auflösung vorausgesetzt. Im LUCIFER System wird die Erzeugung der beiden Operationsbilder durch das Filtergenerierungssignal, das von einem Individuum eines evolutionären Algorithmus ausgeht, ausgelöst. Das Filtergenerierungssignal löst im Filtergenerator die Erzeugung zweier Bildoperationen aus. Diese Prozedur wird weiter unten im Detail beschrieben. Der 2D-Lookup Algorithmus läuft dann über alle gleichen Pixelpositionen beider Operationsbilder ab. Für jede Position liefern die entsprechenden Pixel an dieser Position in Operationsbild 1 und 2 jeweils einen Grauwert[1]. Das entstehende Paar von Grauwerten wird als Indexpaar für die 2D-Lookup Matrix betrachtet. Der Eintrag an dieser Stelle in der 2D-Lookup Matrix wird als Pixelwert an der Position, an der die beiden Grauwerte aus den Operationsbildern entnommen wurden, eingetragen. Wenn die 2D-Lookup Matrix binärwertig ist (wie die Zielbilder des LUCIFER Systems), ist das Ergebnisbild auch binär.

Formal beschrieben: seien $I_1$ und $I_2$ zwei Grauwertbilder (z.B. die Operationsbilder) gleicher Größe und Auflösung, definiert durch ihre Bildfunktionen $g_1$ und

[1]Farbbilder werden im folgenden nicht weiter betrachtet.

$g_2$ über ihren gemeinsamen Definitionsbereich $P \subseteq \mathcal{N} \times \mathcal{N}$:

$$g_1 : \quad P \to \{0, \ldots, g_{max}\} \tag{6.1}$$
$$g_2 : \quad P \to \{0, \ldots, g_{max}\} \tag{6.2}$$

Auch die 2D-Lookup Matrix kann in Form einer Bildfunktion $l$ vorgegeben werden, jedoch mit der Menge aller möglichen Paare von Grauwerten als Definitionsbereich,

$$l : \quad \{0, \ldots, g_{max}\} \times \{0, \ldots, g_{max}\} \to p \subseteq \{0, \ldots, g_{max}\}. \tag{6.3}$$

Dann bestimmt sich das Ergebnisbild durch

$$r : \; P \to \{0, \ldots, g_{max}\} \tag{6.4}$$
$$r(x, y) = l(g_1(x, y), g_2(x, y)). \tag{6.5}$$

In typischen Anwendungen der Bildverarbeitung sind Grauwerte durch 8 Bit kodiert, so daß $g_{max} = 255$ ist. Die Definitonsbereiche von Bildern sind in der Regel rechteckige Bereiche. In diesem Fall kann der 2D-Lookup Algorithmus durch folgenden (objekt-orientierten) Pseudokode beschrieben werden:

```
for x=0 to img width-1 do
begin
   for y=0 to img height-1 do
   begin
      g1 = g1(x,y)
      g2 = g2(x,y)
      out(x,y) = l(g1,g2)
   end y
end x
```

Um ein kleines Beispiel für diese Prozedur zu geben, sei im folgenden $g_{max} = 3$ (also 2 Bit Grauwerte). Sind dann

$g_1$ :

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 0 | 3 | 3 |

und $g_2$ :

| | | |
|---|---|---|
| 2 | 3 | 1 |
| 2 | 3 | 2 |

die beiden Operationsbilder und

$l$ :

| $g_2$ \ $g_1$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 2 | 2 |
| 2 | 1 | 2 | 3 | 3 |
| 3 | 2 | 3 | 3 | 2 |

die 2D-Lookup Matrix. Dann ist das Ergebnisbild des 2D-Lookup Algorithmus

$r$ :

| | | |
|---|---|---|
| $l(0,2)$ | $l(1,3)$ | $l(2,1)$ |
| $l(0,2)$ | $l(3,3)$ | $l(3,2)$ |

$=$

| | | |
|---|---|---|
| 1 | 3 | 2 |
| 1 | 2 | 3 |

## 6.3 Die Fitnessfunktion von Lucifer

Um Ergebnisbild und Zielbild nun miteinander zu vergleichen, muß eine Qualitätsfunktion festgelegt werden, die den Grad der Übereinstimmung beider Bilder beschreibt. Dabei geht es also um die Korrespondenz zweier positionell überlagerbarer Mengen (das vereinfacht den Vergleich erheblich). Zuerst die Definiton der verwendeten Fitnessfunktion. Dazu betrachte man Abbildung 6.3, in der zwei Mengen dargestellt sind: die Referenzmenge (entspricht den schwarzen Pixeln des Zielbildes) und die zu bewertende Mustermenge (entspricht den schwarzen Pixeln des jeweiligen Ergebnisbildes).

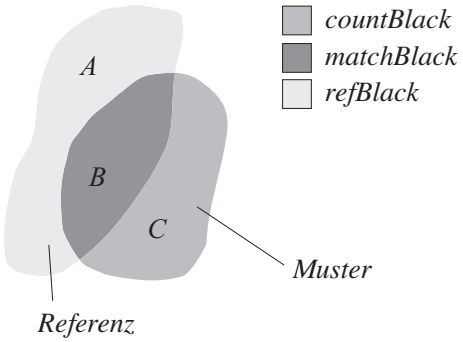



Abbildung 6.3: Zur Definition der verwendeten Fitnessfunktion.

Darin bezeichnen nun *countBlack* die Anzahl von Punkten der Mustermenge (also schwarzer Pixel des Ergebnisbildes), $(B+C)$ in Abbildung 6.3, *matchBlack* die Anzahl von Punkten des Durchschnitts von Referenz- und Mustermenge (also Punkte des Ergebnisbildes, die bezüglich des Zielbildes richtig gesetzt wurden), $B$ in Abbildung 6.3, und *refBlack* die Anzahl von Punkten der Referenzmenge (also der schwarzen Pixel des Zielbildes), $(A+B)$ in Abbildung 6.3. Daraus lassen sich die folgenden Verhältnisse berechnen:

$$r_1 = \frac{matchBlack}{refBlack} \tag{6.6}$$

ist der Anteil passender Musterpunkte an allen Referenzpunkten,

$$r_2 = 1.0 - \frac{countBlack - matchBlack}{N - refBlack} \tag{6.7}$$

ist der Anteil korrekt gesetzter Hintergrundpunkte des Musters (also korrekte weiße Punkte im Ergebnisbild), wobei $N$ die Gesamtzahl von Pixeln des Zielbildes (und damit auch Ergebnisbildes) ist, und

$$r_3 = \frac{matchBlack}{countBlack} \tag{6.8}$$

der Anteil passender Punkte an der Mustermenge (also des Zielbildes). Eine vierte Größe wäre hier noch der Anteil korrekter weißer Pixel am Musterhintergrund. Diese ist jedoch abhängig von den anderen drei Größen.

Das multiobjektive Optimierungsziel wäre es nun, alle drei Anteile gleichzeitig zu vergrößern. Es wird daher die folgende gewichtete Summe der drei Zielsetzungen als Fitnesswert benutzt:

$$f = 0.1r_1 + 0.5r_2 + 0.4r_3 \tag{6.9}$$

Durch diese Wahl der Gewichtungen der Zielsetzungen hat die Fitnessfunktion die folgenden Eigenschaften:

1. Sie nimmt die höchsten Werte für Teilmengen der Referenz an. Wenn alle Schwarzpixel des Ergebnisbildes in der Referenz liegen, ist $countBlack = matchBlack$, also $r_2 = r_3 = 1$ und somit $f \geq 0.9$.

2. Sie hat höhere Werte für Teilmengen der Referenz, die Übermengen anderer Teilmengen der Referenz sind.

3. Ein weißes Bild jedoch (die leere Teilmenge) gibt einen schlechten Fitnesswert von 0.5

Durch diese Eigenschaften ist die Fitnessfunktion für einen evolutionären Algorithmus geeignet, der auf der Fitnesssteuerung beruht (siehe Abschnitt 5.1.3). Solch eine Suche wird in Richtung der höher gewichteten Zielsetzung zuerst explorieren. Das ist hier das Maß $r_2$, mit 0.5 gewichtet. Das erste Subziel des evolutionären Algorithmus besteht dann also darin, so viele weiße Positionen wie möglich korrekt zu setzen. Entsprechend der Gewichtung von 0.4 für das Maß $r_3$ erfolgt die Suche dann leicht „zeitversetzt“ dahingehend, soviele Punkte der Referenz wie möglich zu setzen, wobei die bereits korrekten weißen Positionen erhalten bleiben. Sobald das Muster eine echte Teilmenge der Referenz geworden ist, ist eine Steigerung der Fitness nur noch dadurch möglich, daß diese Teilmenge *innerhalb* der Referenz vergrößert wird. Dies setzt ein, sobald die Fitness deutlich den Wert 0.9 überschreitet.

## 6.4 Ableitung der 2D-Lookup Matrix

Im folgenden sei wieder angenommen, daß die Operationsbilder 1 und 2 bereits bekannt sind, und auch ein Zielbild vorgegeben ist. Die Frage ist nun, ob es ein Verfahren gibt, das die 2D-Lookup Matrix für ein Ergebnisbild liefert, dessen Fitnessmaß, wie im letzten Abschnitt beschrieben, möglichst groß wird. In diesem Abschnitt wird solch ein Verfahren vorgestellt. Dazu wird die Berechnung der Fitnessfunktion in diesem Zusammenhang wiederverwendet.

Angenommen, eine 2D-Lookup Matrix würde bis auf eine einzige schwarze Position $(g_1, g_2)$ nur weiße Positionen beinhalten. Dann wird das aus der Anwendung des 2D-Lookup Algorithmus sich ergebende Bild nur an den Positionen $(x, y)$ schwarz sein, für die Operation 1 den Grauwert $g_1$ und Operation 2 den Grauwert $g_2$ ergab, also in der Regel relativ wenige Pixel. Nun ergab die Diskussion der Fitnessfunktion im letzten Abschnitt, daß dieses Maß den Wert 0.9 übersteigt, sobald die Menge aller schwarzen Pixel des Ergebnisbildes eine Teilmenge der schwarzen Pixel des Zielbildes geworden ist, unabhängig davon, wieviel Pixel dies sind. Somit können auch die kleinen Anteile schwarzer Pixel in den Ergebnisbildern des eben beschriebenen Typs relativ grosse Fitnesswerte liefern. Und was noch besser ist: für zwei solcher 2D-Lookup Matrizen, deren einzigen Schwarz-Einträge an verschiedenen Positionen vorliegen, sind die sich ergebenden Mengen an schwarzen Pixeln im Ergebnisbild disjunkt!

Formal ausgedrückt: bezeichnet $l_{(x,y)}$ eine 2D-Lookup Matrix, die nur an der Position $(x, y)$ schwarz ist, sonst weiß, und $l_M$ eine beliebige 2D-Lookup Matrix, in der genau die Positionen $M = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ schwarz sind, dann ist im Ergebnisbild $r_M$ die Menge aller schwarzen Pixel $M_r$ die Vereinigung der disjunkten Mengen $M_{(x_i,y_i)}$, die sich jeweils aus der Anwendung von $l_{(x_i,y_i)}$ ergeben:

$$M_r = \bigcup_{i=1}^{n} l_{(x_i,y_i)} \circ [op_1(I), op_2(I)]. \tag{6.10}$$

Da die „Wunschmenge“ $M_r$ durch das gegebene Zielbild bekannt ist, ergibt sich daraus eine Heuristik für die Besetzung der 2D-Lookup Matrix, in dem nur die Positionen $(x, y)$ Schwarz gesetzt werden, für die $M_{(x,y)}$ weitestgehend innerhalb der Menge aller schwarzen Pixel des Zielbildes liegt. Die unscharfe Formulierung „weitestgehend innerhalb“ wird dabei so interpretiert, daß die Fitnessfunktion für das Ergebnisbild der Anwendung von $l_{(x,y)}$ einen Wert größer als 0.88 liefern muß, also einen Wert etwas kleiner als 0.9. Die Menge $M$ in $l_M$ aller schwarzen Positionen der 2D-Lookup Matrix besteht dann aus den Punkten $(x, y)$, für die gilt:

$$M = \{(x, y) \mid f\left(l_{(x,y)} \circ [op_1(I), op_2(I)]\right) \geq 0.88\} \tag{6.11}$$

In dem Fall, daß $M_{(x,y)}$ leer ist, läßt sich keine Entscheidung treffen, ob dieser Punkt zu $M$ gehören sollte oder nicht. In diesem Fall erhält die 2D-Lookup Matrix an dieser Stelle einen „grauen“ Eintrag (z.B. den Grauwert 127, wenn man ansonsten den Grauwert 0 für schwarze und den Grauwert 255 für weiße Positionen gewählt hatte). Das mit den grauen Einträgen verbundene Generalisierungsproblem wird weiter unten diskutiert.

Liegt ansonsten eine nichtleere Menge an schwarzen Punkten im Ergebnisbild vor, und ist der Fitnesswert dieses Bildes kleiner als 0.88, wird die Position in der 2D-Lookup Matrix Weiß gesetzt.

Diese Prozedur stellt eine Relaxation dar: jede Position der 2D-Lookup Matrix wird „versuchsweise“ auf schwarz gesetzt. Verbessert dies den Wert eines glo-

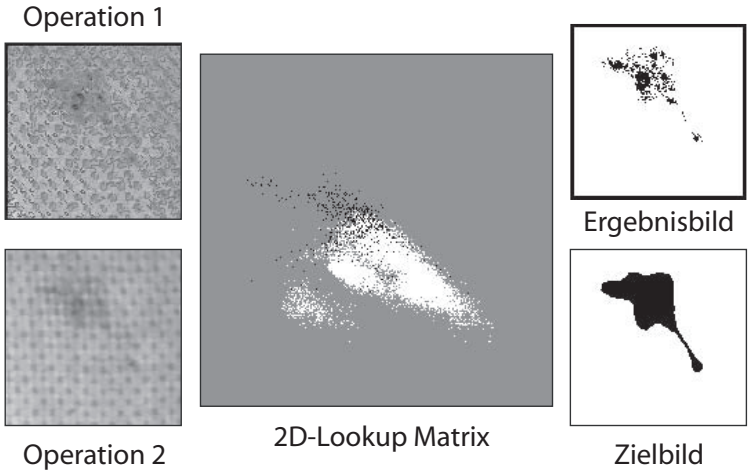



Abbildung 6.4: Relaxation zur Ableitung einer nahezu optimalen 2D-Lookup Matrix $l_{relaxation}(g_1, g_2)$, wenn die Operationsbilder $g_1$ und $g_2$bekannt sind.

balen „Energieterms“ (hier der Wert der Fitnessfunktion für das Ergebnisbild, die allerdings nicht unbedingt ein positiv definites Funktional und damit keine „echte“ Energie darstellen muß), wird die Änderung beibehalten. Auf Grund der vergleichsweise geringen Anzahl von möglichen Zustandsänderungen einer 2D-Lookup Matrix bei 256 Grauwerten ($256^2$) ist es jedoch möglich, hier auch *alle* Zustandsänderungen zu überprüfen und somit ein nahezu optimales Ergebnis für die Belegung der 2D-Lookup Matrix zu erlangen.

Abbildung 6.4 zeigt das Ergebnis einer solchen Relaxation für zwei Beispielbilder. Wie man sieht, ist alles in dem Moment ausreichend spezifiziert, in dem die beiden Ergebnisbilder $g_1$ und $g_2$ der Operationen 1 und 2 vorliegen. Unabhängig von der Art ihrer Gewinnung, also der Spezifikation der zu ihnen führenden Operationen, kann mittels dieser Relaxation sowohl eine 2D-Lookup Matrix $l_{relaxation}(g_1, g_2)$ als auch ein Fitnesswert (eben gerade der des mit $l_{relaxation}(g_1, g_2)$ gewonnenen Ergebnisbildes) diesen Operationsbildern zugewiesen werden. Damit reduziert sich die Frage nach der Anpassung des gesamten Systems auf die Frage nach der Spezifikation der beiden Operationen.

Hier liegt also genau der weiter oben beschriebene Idealfall eines 2D-Sytems vor (siehe Seite 82): die Spezifikation der Fusionsoperation kann *auf anderem Wege* erfolgen als die Spezifikation der einzelnen Operationsstränge. Das 2D-System LUCIFER ist entkoppelt.

## 6.5 Spezifikation der Operationen

Im folgenden wird die Struktur eines Individuums der einer genetischen Programmierung unterworfenen Population beschrieben. Durch Einsatz der im vorherge-

henden Abschnitts beschriebenen Relaxation genügt es dabei, die Gestaltung der zu den Operationsbildern $g_1$ und $g_2$ führenden Operationen vorzunehmen. Es ist also eine Operatorenalgebra festzulegen, die auf Bildoperationen angewendet werden kann.

Zielstellung ist es dabei natürlich, eine möglichst große Vielfalt an Operatoren für die Suche nach optimalen Operationsbildern bereitzustellen. Dies kann z.B. durch explizite Vorgabe einer großen Menge an bekannten Operatoren mit unterschiedlichen Parametereinstellungen und anschließender zufälliger Auswahl daraus erreicht werden. Der Nachteil ist jedoch, daß die Menge verfügbarer Operationen dann begrenzt ist. Besser ist eine generische Prozedur, bei der Operatoren aus der Kombination von Basisoperatoren gewonnen werden. Hierzu bieten die in der genetischen Programmierung genutzten Syntaxbäume genau die ideale Form für eine solche Kombination an.

Wenn wir jedoch die Repräsentation von Operatoren durch Superposition anderer Operatoren erreichen wollen, ist ein weiterer Umstand zu beachten: die wahllose Vermengung von Operatoren würde mit hoher Wahrscheinlichkeit Operatoren produzieren, die immer weniger Selektivität z.B. für bestimmte Bildstrukturen aufweisen (sozusagen eine variante Formulierung des No-Free-Lunch Theorems). Dies umso mehr, je elementarer die gewählten Knotenfunktionen sind. Es gibt also neben der Repräsentierung einer Vielfalt von Operatoren weitere Zielstellungen in der Operatorkombination: die Basisoperationen sollten nicht so elementar wie z.B. die vier Grundrechenoperationen sein, sie sollten in relativ geringer Anzahl kombiniert werden, und unter den repräsentierten Operatoren sollten möglichst viele bereits studierte Operatoren auftreten.

Dies kann erreicht werden, wenn als Knotenfunktionen der Syntaxbäume generische Bildoperatoren verwendet werden, also Bildoperatoren, die jeweils eine Menge bekannter Bildoperatoren zusammenfassen, und zusätzlich die Tiefe der Bäume stark begrenzt wird. Eine mögliche Umsetzung dieser Herangehensweise wird im folgenden im Detail beschrieben, aber es sind natürlich viele Variationen hier denkbar.

Jedes Individuum einer Population ist in Form einer Baumstruktur zu repräsentieren. Da jeder Baum im Endeffekt *zwei* Bildoperationen repräsentieren muß, sind die Knotenfunktionen ebenfalls Bildoperationen, und die Terminale Instanzen des Eingangsbildes.

Die Knotenfunktionen werden auf Basis eines Satzes formaler *Superoperatoren* spezifiziert, die auf einer gemeinsamen Parameterstruktur ($PS$) aufbauen. Diese $PS$ besitzt die folgenden Einträge:

- Eine *Maske* $\mathcal{M} = \{(i, j)\}$ als eine Menge von Offset-Positionen. Das $k$-te Element einer Maske $\mathcal{M}$ sei durch $\mathcal{M}_k = (i_k, j_k)$ bezeichnet. Die „Anwendung“ einer Maske auf einen Pixel mit der Position $(x, y)$ liefert die Menge der Pixelpositionen $\mathcal{M} \circ (x, y) = \{(u, v) | u = x + i, v = y + j, (i, j) \in \mathcal{M}\}$, auch als *Nachbarschaft* von $(x, y)$ bezeichnet. Die in LUCIFER benutzten

Masken sind auf symmetrische $3\times3$ Masken beschränkt (also $|x_i| \leq 1, |y_i| \leq 1$ und mit $(x, y) \in \mathcal{M}$ ist auch $(-x, -y) \in \mathcal{M}$).

- Eine *gewichtete Maske* $\mathcal{M}_w$ ist ein Tupel $(\mathcal{M}, f)$ aus einer Maske $\mathcal{M}$ und einer Funktion $f : \mathcal{M} \to \mathcal{R}$, die jedem Maskenoffset $(i, j)$ eine reelle Zahl, seine *Gewichtung*, zuordnet. Eine gewichtete Maske kann durch ein Schema wie im Falle einer $3 \times 3$ Maske dargestellt werden:

$$\mathcal{M}_w = \begin{array}{|c|c|c|} \hline w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\ \hline w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\ \hline w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\ \hline \end{array}$$

  In LUCIFER sind Gewichte auf das Intervall $[-5, 5]$ beschränkt. Wenn nur $\mathcal{M}_w$ bekannt ist, kann $\mathcal{M}$ als Menge aller Offset-Positionen gefunden werden, für die $f(i, j) \neq 0$ ist.

- Ein einzelner *Offset-Vektor* $p = (\delta_x, \delta_y)$. In LUCIFER ist dabei der Betrag keiner Komponente größer als 2.

- Eine BOOLE*sche Operation* $f_{bit}$ von $\{0, 1\}\times\{0, 1\}$ in $\{0, 1\}$ mit der Nummer $N_{f_{bit}}$[2]. Die Notation $f_{bit}(n, m)$ bedeutet die bit-weise Anwendung von $f_{bit}$ auf deren Argumente, wenn diese als Dualzahlen dargestellt werden. Also: wenn $bit_i(x)$ aus der Dualzahlendarstellung von $x$ das Bit $i$ extrahiert, dann ist

$$bit_i(f_{bit}(n, m)) = f_{bit}(bit_i(n), bit_i(m)). \tag{6.12}$$

- Eine *Permutation* $\Pi$ der ersten $m$ Zahlen, wobei $\Pi_k$ die Position der Zahl $k$ in solch einer Permutation bezeichnet.

- Ein *Sortiervektor* $v = (v_1, v_2, \ldots, v_m)$, der einen Satz von $m$ sortierten Größen Gewichte zuweist (also das Gewicht $v_1$ dem größten, $v_2$ dem zweitgrößten Wert usw.). Im folgenden sollen dabei $rank_i^{max}(A)$ den $i$-tgrößten Wert von $A$ und $rank_i^{min}(A)$ den $i$-tkleinsten Wert von $A$ bezeichnen. In LUCIFER sind alle Komponenten von $v$ auf den Wertevorrat $\{-1, 0, 1\}$ eingeschränkt.

- Ein symbolischer Wert *Modus*, der den Typ des formalen Superoperators angibt, der unter Benutzung eines Teils der eben beschriebenen Parameter angewendet wird. Die möglichen Werte von *Modus* werden im folgenden beschrieben.

Folglich ist eine $PS$ ein 6-Tupel $(\mathcal{M}_w, p, N_{f_{bit}}, \Pi, v, Modus)$.

Die Werte für $Modus$ können fünf verschiedene Superoperatoren beschreiben:

[2] Es gibt genau 16 solcher Operationen, und $N$ bezieht sich dabei auf eine bestimmte Ordnung dieser Operationen, z.B. bei lexikografischer Anordnung der Operatoren auf die aus den binären Ergebniswerten konstruierbare Dualzahl.

1. Logische Verschiebung **TRANS**. Sie benutzt den Vektor $p$ und die Bitoperation mit der Nummer $N_{f_{bit}}$. Das Ergebnisbild an der Position $(x, y)$ ist durch
$$g(x, y) = f_{bit}\,(o(x - \delta_x, y - \delta_y), o(x, y)) \tag{6.13}$$
gegeben.

2. Faltung (oder Konvolution) **CONVOL**. Sie benutzt den Eintrag $\mathcal{M}_w$ der $PS$. Das Ergebnisbild an der Position $(x, y)$ ist durch
$$g(x, y) = \sum_{(i,j)\in\mathcal{M}} f(i, j)\, o(x + i, y + j) \tag{6.14}$$
gegeben. Die gewichtete Maske wird vor der Anwendung auf den Summenwert 0 verschoben, d.h. zu jedem Gewicht wird ein Wert $\delta$ addiert, so daß $\sum(f_w(\mathcal{M}_k) + \delta) = 0$ ist.

3. *Ordered Weighted Averaging* **OWA** [197]. Ursprünglich wurde diese Operation in Fuzzy-Inferenz-Systemen zur Durchführung der Defuzzifizierung eingeführt. Sie benutzt den Sortiervektor $v$. Um sie als Bildoperation anzuwenden, wird folgendes Vorgehen gewählt: die Pixelwerte (Grauwerte) der Pixel der Nachbarschaft $\mathcal{M} \circ (x, y)$ des Pixel $(x, y)$ werden in abfallender Folge sortiert, und der Wert an der Sortierposition $k$ wird mit dem Gewicht $v_k$ mulitpliziert. Alle Produkte werden aufsummiert:
$$g(x, y) = \sum_{k=1}^{|\mathcal{M}|} v_k\, rank_k^{max}(o(\mathcal{M} \circ (x, y))) \tag{6.15}$$
Einige solcher OWA repräsentieren Standardoperatoren der Bildverarbeitung. So repräsentiert $(1, 0, \ldots, 0)$ die Dilatation, $(0, \ldots, 0, 1)$ die Erosion, $(1, \ldots, -1)$ den morphologischen Gradienten, $(0, \ldots, 0, 1, 0, \ldots, 0)$ einen allgemeinen Rankoperator inkl. dem Medianoperator, wenn die Anzahl der Nullen vor und nach dem Gewicht 1 gleich ist, und $(1/n, 1/n, \ldots, 1/n)$ den Mittelwertoperator. OWA erlaubt die Formalisierung aller dieser Operation in einem geschlossenen Ausdruck mit adaptierbaren Komponenten.

4. Texturnummern **TN**. Bei den Texturnummern wird jedem Pixel ein Label zugeordnet, welches die multimodale Größenbeziehung seines Grauwertes zu den Grauwerten seiner Nachbarschaft beschreibt. Jede Einzelbeziehung eines Pixels zu einem seiner Nachbarn (der Pixel ist heller oder dunkler als der Nachbar) liefert dabei ein Bit einer Dualzahl. Zur Festlegung der Reihenfolge, in der diese Bits zu einer Dualzahl zusammengefasst werden wird die Permutation $\Pi$ aus der $PS$ benötigt. Formal bedeutet dass: an der

Position $(x, y)$ wird ein Bitstring $b$ von derselben Größe wie $\mathcal{M}$ bestimmt. Das $k$-te Element von $b$ bestimmt sich dabei gemäß

$$b_k = \begin{cases} 1 & \text{wenn} \quad (o([\mathcal{M} \circ (x,y)]_k) \geq o(x,y), \\ 0 & sonst. \end{cases} \tag{6.16}$$

Dann wird eine Zahl $n$ aus dem Bitstring $b$ konstruiert, deren $k$-tes Bit sich aus der Permutation $\Pi$ in der Form $bit_k(n) = b_{\Pi_k}$ ergibt. Nachdem $n$ auf den Grauwertbereich $\{0, \ldots, g_{max}\}$ normiert wurde, ist $n$ der Grauwert des Ergebnisbildes an der Position $(x, y)$. Der hier Texturnummer benannte Operator ist auch unter der Bezeichnung LBP für *local binary pattern* oder Census-Transformation bekannt. Diese anderen Bezeichnungen werden aber eigentlich nicht dem offensichtlichen Klassifikationscharakter dieses Operators gerecht.

5. *Ordered Weighted Minimum* **OWM**. Diese Operation nutzt die gewichtete Maske $\mathcal{M}_w$. Die Maskengewichte werden (im voraus) in abfallender Ordnung sortiert, die Pixelwerte in der Nachbarschaft von $(x, y)$ in aufsteigender Ordnung. Dann wird eine minmax Operation auf diese beiden Wertsequenzen angewendet und das Ergebnis im Ergebnisbild als Grauwert verwendet:

$$g(x,y) = \min_{k=1}^{|\mathcal{M}|} \left[\max \left[rank_k^{max}(f_w \circ \mathcal{M}), rank_k^{min}(o(\mathcal{M} \circ (x,y))\right]\right] \tag{6.17}$$

Damit ist geklärt worden, wie durch eine Parameterstruktur $PS$ eine Bildoperation spezifiziert werden kann. Im Ablauf einer genetischen Programmierung wird dann eine Baumstruktur als Individuum folgendermaßen konstruiert:

1. An der Wurzel (Ebene 0) besitzt jeder Baum genau zwei Verzweigungen (um die zwei Operationen für den 2D-Lookup in einem einzigen Baum zu spezifizieren).

2. Auf jeder Ebene $\lambda > 0$ verzweigt jeder Funktionsknoten entweder in weitere Funktionsknoten oder Terminals der nächsthöheren Ebene.

3. Jedem Funktionsknoten und Terminal ist exakt eine zufällig initialisierte Parameterstruktur zugeordnet.

Weiterhin ist jedem Funktionsknoten eine der folgenden Bildoperationen zugeordnet:

1. Pixelweise Subtraktion $-$. $g(x,y) = |g_1(x,y) - g_2(x,y)|$.

2. Pixelweises Quadrieren **sq**. $g(x,y) = g_1^2(x,y)/g_{max}$.

3. Pixelweises Minimum **min**. $g(x,y) = \min[g_1(x,y), g_2(x,y)]$.

4. Pixelweises Maximum **max**. $g(x,y) = \max[g_1(x,y), g_2(x,y)]$.
5. Ausführung der $PS$ $\lambda$. $g = PS \circ g_1$.

Schließlich beschreibt ein Baum $T$ dann die beiden benötigten Operationen in der folgenden Weise: Das Originalbild wird an alle Terminale, d.h. an alle Endpunkte der Bäume, übergeben. Jedes Terminal wendet dann die durch den *Modus*-Parameter seiner $PS$ beschriebene Operation auf das Ausgangsbild an. Die bearbeiteten Bilder werden als Operanden an die Knoten der nächsttieferen Ebene übergeben usw. Der Wurzelknoten sammelt die beiden Ergebnisbilder ein und wendet die im letzten Abschnitt beschriebene Relaxation an, um die 2D-Lookup Matrix festzulegen. Damit beschreibt jeder solcher Baum $T$ eine *vollständige* Spezifikation eines 2D-Lookup. Gleichzeitig wird so jedem Baum $T$ ein Fitnesswert zugewiesen, als Wert der Fitnessfunktion für das Ergebnis der Anwendung der gemäß der Relaxation ermittelten 2D-Lookup Matrix.

Im folgenden soll ein Beispiel für einen solchen Baum und seine Interpretation als Bildoperation gegeben werden (siehe Abbildung 6.5). Der Beispielbaum hat sechs Knoten, denen die fünf folgenden $PS = \{\mathcal{M}_w, p, N_{f_{bit}}, \Pi, v, Modus)$ zugeordnet sind:

$$PS_1 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 2 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 3 & 0 \\ \hline \end{array}, (1,1), 11, \Pi_{213}, (1,0,0), \text{ TRANS} \right\}$$

$$PS_2 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & 2 & 2 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, (1,2), 7, \Pi_{132}, (1,0,0), \text{ OWA} \right\}$$

$$PS_3 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, (-1,0), 3, \Pi_{12354}, (0,0,1), \text{ OWA} \right\}$$

$$PS_4 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 3 & 5 & 2 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, (-1,-2), 5, \Pi_{312}, (1,0,-1), \text{ TN} \right\}$$

$$PS_5 = \left\{ \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, (2,0), 5, \Pi_{51243}, (1,0,0), \text{ OWA} \right\}$$

Man beachte, daß $PS_4$ niemals auf ein Bild angewendet wird, und daß $PS_3$ und $PS_5$ dieselbe Maske besitzen. Es wird angenommen, daß Bitoperation 7 die XOR-Operation bezeichnet. Dann ergibt sich, daß die durch den Baum gegebene Operation 1 sich als die (absolute) Differenz eines sich um den Offset $(1,1)$ verschobenen und mit dem Original via XOR verknüpften Bildes ($PS_1$) und dem mit

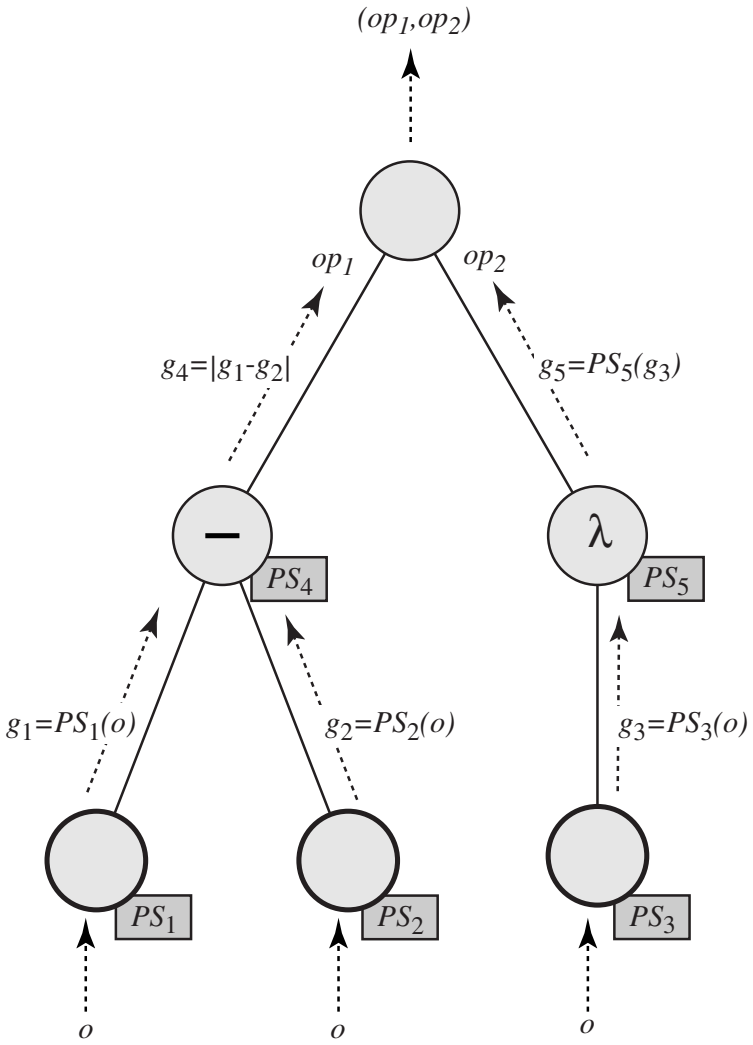



Abbildung 6.5: Beispiel für eine Baumstruktur $T$, die zwei Bildoperationen festlegt (siehe Text für weitere Erläuterungen).

einer horizontalen Maske dilatierten Originalbild ($PS_2$) darstellt. Operation 2 ist die morphologische Öffnung des Bildes mit einer Kreuzmaske ($PS_3$ und $PS_5$). In Abbildung 6.6 sind Zwischenergebnisse und Operationsbilder für dieses Beispiel dargestellt.

Die Strukturierung von Bildoperationen in der eben beschriebenen, relativ komplexen Weise wurde aus den folgenden Gründen so gewählt:

- Viele so beschriebene Operationen beschreiben bekannte Bildoperationen. Ein Baum tendiert eher dazu, Operationen wie Dilatation, Erosion, Schließung, Öffnung, Gradienten, SOBEL-Operator, GAUSS-Filter, Schattenbild usw. zu beschreiben.

- Es ist eher unwahrscheinlich, als Ergebnis der Anwendung eines Baumes un-

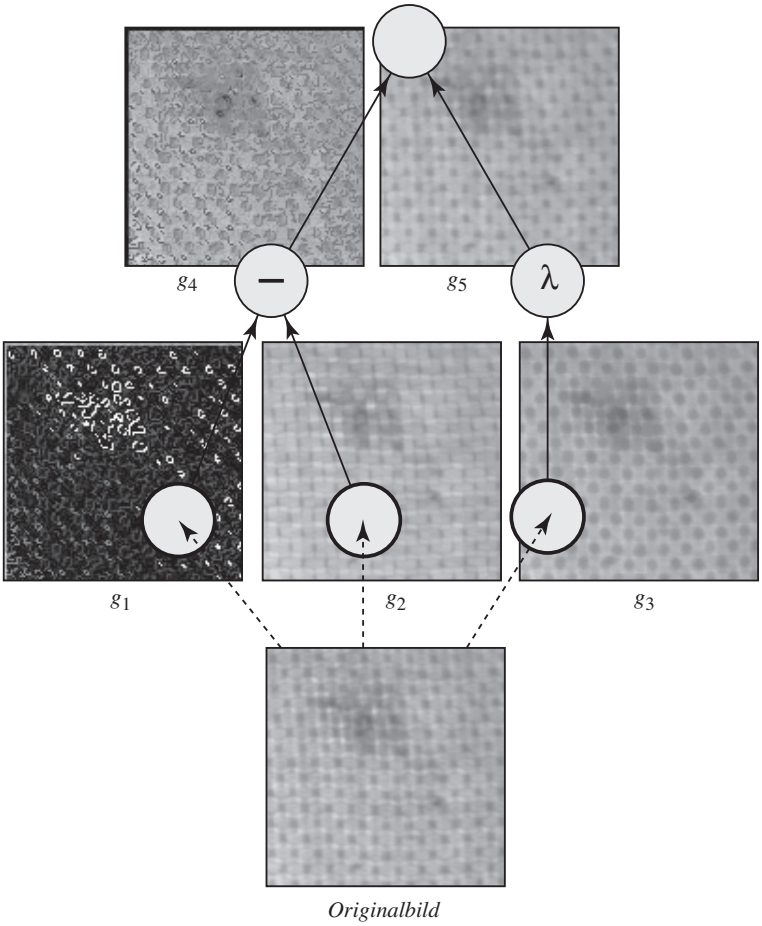



Abbildung 6.6: Zwischenbilder und Operationsbilder der durch den Baum in Abbildung 6.5 spezifizierten Operationen.

erwünschte Operationsbilder, wie z.B. Bilder, die vollständig schwarz oder weiß sind, zu erhalten.

- Alle Operationen bewahren Bildpositionen (eine in der mathematischen Morphologie *local knowledge* genannte Option).

- Die maximale Arität eines Knotens (also die Anzahl abgehender Verzweigungen) ist zwei. Auch wurde die maximale Tiefe eines Baumes auf fünf reduziert. Damit können die Bäume auch gehandhabt werden, also ihre Struktur ist nachvollziehbar, Redundanzen können erkannt werden und die Bäume überhaupt manuell nachbearbeitet werden.

Abbildung 6.7 stellt Beispiele für verschiedene, zufällig initialisierte Baumstrukturen dar. Dies demonstriert die Variabilität der eingesetzten Operationen, von denen jede verschiedene Bildstrukturen verstärkt oder abschwächt, und von denen keine ein triviales Ergebnis liefert.

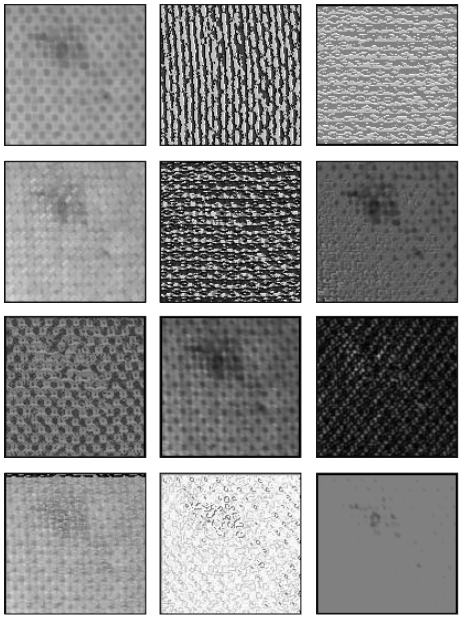


Abbildung 6.7: Einige Beispiele für Operationsbilder, die durch zufällig initialisierte Baumstrukturen aus dem linken oberen Bild gewonnen wurden.

## 6.6 Anwendungen

### 6.6.1 Detektion von Texturfehlern

Zur Demonstration der Leistungsfähigkeit des LUCIFER Systems werden Beispiele aus dem „Textilfehler-Katalog“ des IPK Berlin (TFK) betrachtet. Zum Vergleich wurde eine „konventionelle“ Merkmalklassifikation angewendet. Merkmale der Co-occurrence-Matrix [61] [132] [16] von jeweils zehn im Bild plazierten $8 \times 8$ Fenstern für jede Klasse (Texturhintergrund und Texturfehler) wurden ermittelt. Danach wurde ein MBPN (siehe Abschnitt 2.2.1) mit 14 Eingabeneuronen, 16 versteckten Neuronen und 2 Ausgabeneuronen (für die beiden Klassen) mit diesen Beispielen 1000 Zyklen lang trainiert. Die trainierten MBPN wurden dann über allen Fenstern einer Parkettierung des gesamten Bildes abgerufen. Die Ergebnisse für vier Beispiele aus dem TFK sind in Abbildung 6.8 dargestellt.

Generell kann gesagt werden, daß dieses Vorgehen keine hohe Auflösung besitzt, da diese auf die Größe der Texturfenster von $8 \times 8$ eingeschränkt ist. Die Ergebnisse für TFK 1-fr8 und TFK 1-st11 sind gut. Das Ergebnis für TFK 3-st23b enthält viele Fehler. Dies ist auf die starken Schwankungen der Grauwerte innerhalb des Fehlerbereiches zurückzuführen, die durch die zehn Trainingsbeispiele nicht ausreichend repräsentiert werden konnten. Das MBPN kann hier nicht ausreichend generalisieren. Das Ergebnis für TFK 3-fr1a ist nicht akzeptabel. Der Fehler ist sehr klein und in geringem Kontrast zu seiner Umgebung. Es treten

zu viele in Bezug auf eine Klassifikation widersprüchliche Grauwertkonstellationen innerhalb des Bildes auf. Diese können durch ein MBPN nicht klassifiziert werden.

Natürlich soll damit nicht bewiesen werden, daß eine Merkmalklassifikation nicht prinzipiell besser funktionieren kann. Zu jedem Beispiel ließe sich auch eine entsprechende Anpassung vorstellen. Es muß jedoch vermerkt werden, daß sich das Ergebnis nur dadurch verbessern läßt, daß entweder mehr Trainingsdaten verwendet werden (mit der Gefahr der Überspezialisierung auf diese kleinen Bildproben), daß das MBPN länger trainiert wird oder daß eine andere Art der Merkmalberechnung gewählt wird. Auch muß bemerkt werden, daß die daraus resultierende Datenbehandlung gerade für unerfahrene Nutzer solch eines Systems nicht gerade einfach ist.

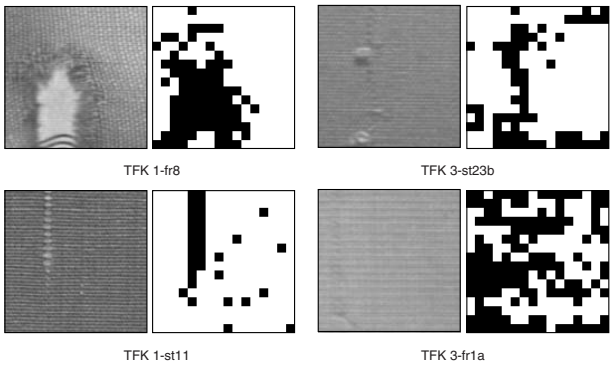



Abbildung 6.8: Ergebnisse der Anwendung der konventionellen Merkmalklassifikation auf vier Beispiele aus dem TFK (siehe Erläuterungen im Text).

Um Lucifer auf diese Beispiele anzuwenden, ist wesentlich weniger notwendig. Die Zielbilder wurden mit einem handelsüblichen Photoretuschierprogramm erstellt. Dann wird das System durch Vorgabe von Ausgangsbild und Zielbild konfiguriert und die genetische Programmierung angewendet. Die Ergebnisse für diese vier Beispiele, inklusive der beiden Operationsbilder und der relaxierten 2D-Lookup Matrix, sind in den Abbildungen 6.9 bis 6.12 dargestellt. In jedem Beispiel wurden als Konfiguration der genetischen Programmierung 30 Elternbäume und 80 Kinder benutzt. Der Algorithmus lief solange, bis die genetische Diversität der Population 0 war, also alle Individuen dieselbe Fitness aufwiesen (üblicherweise nach etwa zehn Generationen).

Wie aus den Abbildungen ersehen werden kann, erfüllt das System seine Aufgabe und generiert geeignete Texturfilter. Die Leistung dieser Filter ist gut bis sehr gut. Es muß bemerkt werden, daß eine *exakte* Übereinstimmung von Ergebnisbild und Zielbild natürlich in der Regel nicht möglich ist, und auch keinen besonderen Wert gegenüber dem Fall kleinerer Abweichungen hätte. Die visuelle Erscheinung eines Fehlers ist stets subjektiv, und die übergebenen Zielbilder

können leicht unterschiedlich gestaltet sein, wenn sie z.B. von verschiedenen Nutzern erzeugt werden. Die Begrenzungen der Fehlerbereiche in den Zielbildern sind *virtuell* und korrespondieren nicht notwendigerweise mit tatsächlich in den Grauwertkonstellationen ausmachbaren Strukturen.

Auch muß bemerkt werden, daß nicht der in einem globalen Sinne *optimale* Filter so gefunden wird. Der Suchraum des Problems ist viel zu groß, um von der genetischen Suche auch nur annähernd vollständig untersucht zu werden. Es ist eher so, daß es innerhalb des qualitativen Rahmens, den von LUCIFER generierte Filter abstecken, *sehr viele* gute, praktisch einsetzbare und durch LUCIFER auffindbare Lösungen für ein Texturproblem existieren. Pessimistisch formuliert ließe sich auch sagen, daß zufällig initialisierte Lösungen gegen etwas bessere evolvieren.

Um über die Generalisierungsfähigkeit der erzeugten Filter eine Aussage zu treffen, wurde ein größerer Satz von 24 Beispielen aus dem TFK untersucht. Zu jedem Beispiel wurden kleine Referenzfenster aus dem Gesamtbild ausgeschnitten, die einen Fehlerbereich enthielten. Das System wurde auf jeden Fehler genau einmal angewendet (im Unterschied zum praktischen Fall, in dem aus mehreren Läufen von LUCIFER der beste ausgewählt werden sollte). Die evolvierten Filter wurden dann auf das gesamte Bild angewendet. Die Anzahl der schwarzen Pixel außerhalb des Referenzbereiches wurde gezählt, genauso wie die Anzahl der grauen Pixel. Letztere treten im Gesamtbild deshalb auf, weil graue Einträge in der 2D-Lookup Matrix ja aus Grauwertpaaren resultierten, die im Referenzbereich nicht auftraten. Dennoch können sie außerhalb der Referenz natürlich auftreten, z.B. als eine Folge einer inhomogenen Hintergrundbeleuchtung.

Die Anzahl der schwarzen Pixel außerhalb der Referenz liefert ein Maß für die Fehlerrate des Filters (*„false alarms“*), die Anzahl der grauen Pixel ein Maß für die Generalisierungsfähigkeit und für die Kompatibilität von Referenz und Gesamtbild. Die Ergebnisse sind in Tabelle 6.1 angegeben. Die mittlere Fehlerrate liegt demnach bei 1.9%, die mittlere Inkompatibilität bei 2.2%. Dies sind auf jeden Fall sehr gute Ergebnisse, wenn man noch beachtet, daß sie keiner weiteren manuellen Anpassung unterlagen. Die Schwankungen jedoch gerade der Fehlerrate sind relativ hoch (zwischen 2 und 57 774 von 262 144 Pixeln).

Eine genaue Betrachtung der entsprechenden 2D-Lookup Matrizen gibt jedoch eine Erklärung für diese Schwankungen:

- Einige 2D-Lookup Matrizen sind separabel, d.h. der texturierte Hintergrund ist durch eine Gruppierung kompakter weißer Regionen im Bild der Matrix repräsentiert. Die Fehlerbereiche (also die schwarzen Punkte) umranden solche kompakten weißen Regionen. In diesem Fall wird vom Filter eine sehr gute Generalisierungsfähigkeit erwartet.

- Es können jedoch auch Regionen auftreten, in denen schwarze und weiße Punkte hoffnungslos ineinander vermischt sind. Je mehr solche Regionen in

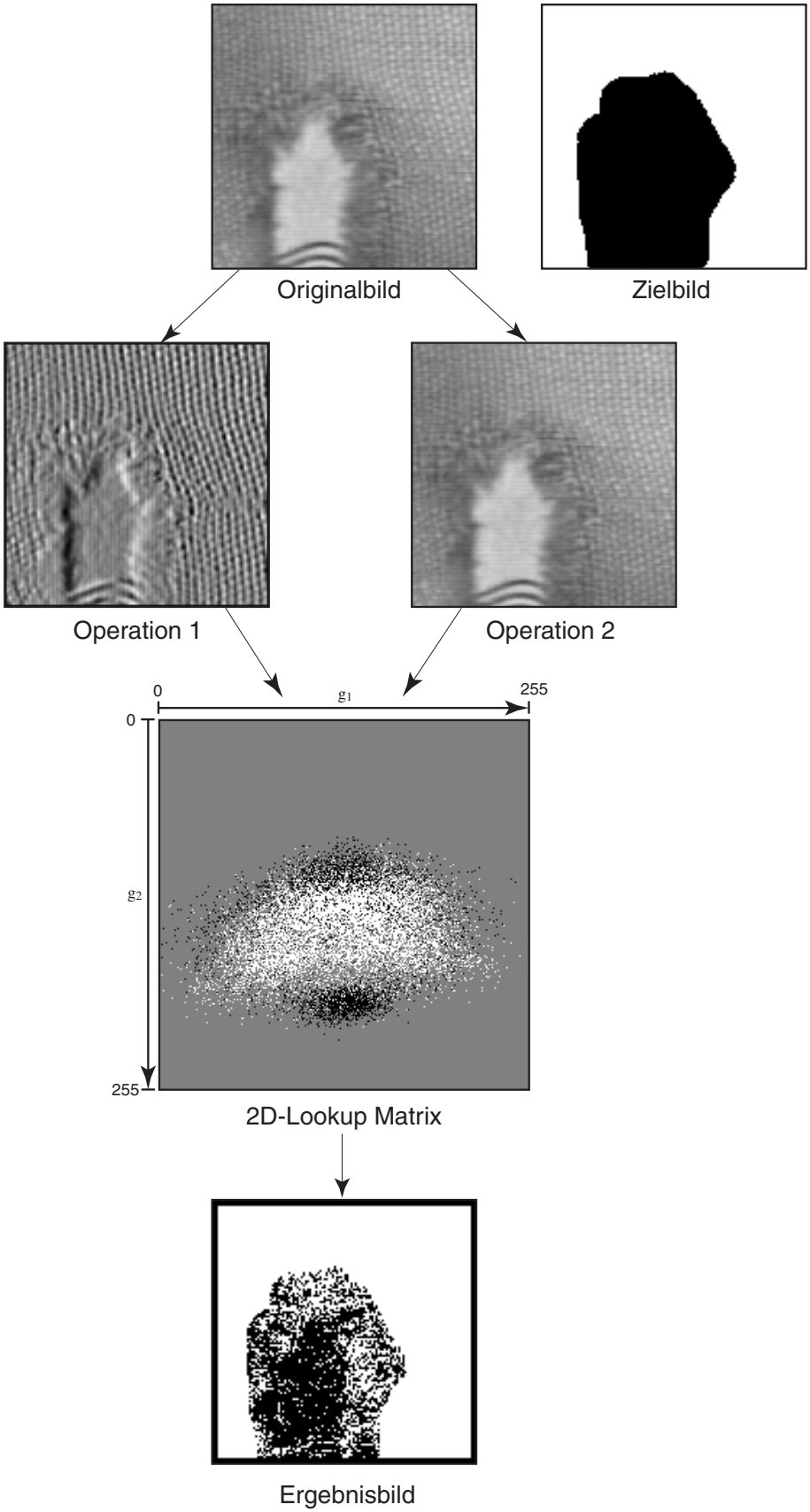



Abbildung 6.9: Ergebnis für Beispiel 1 (TFK 1-fr8).

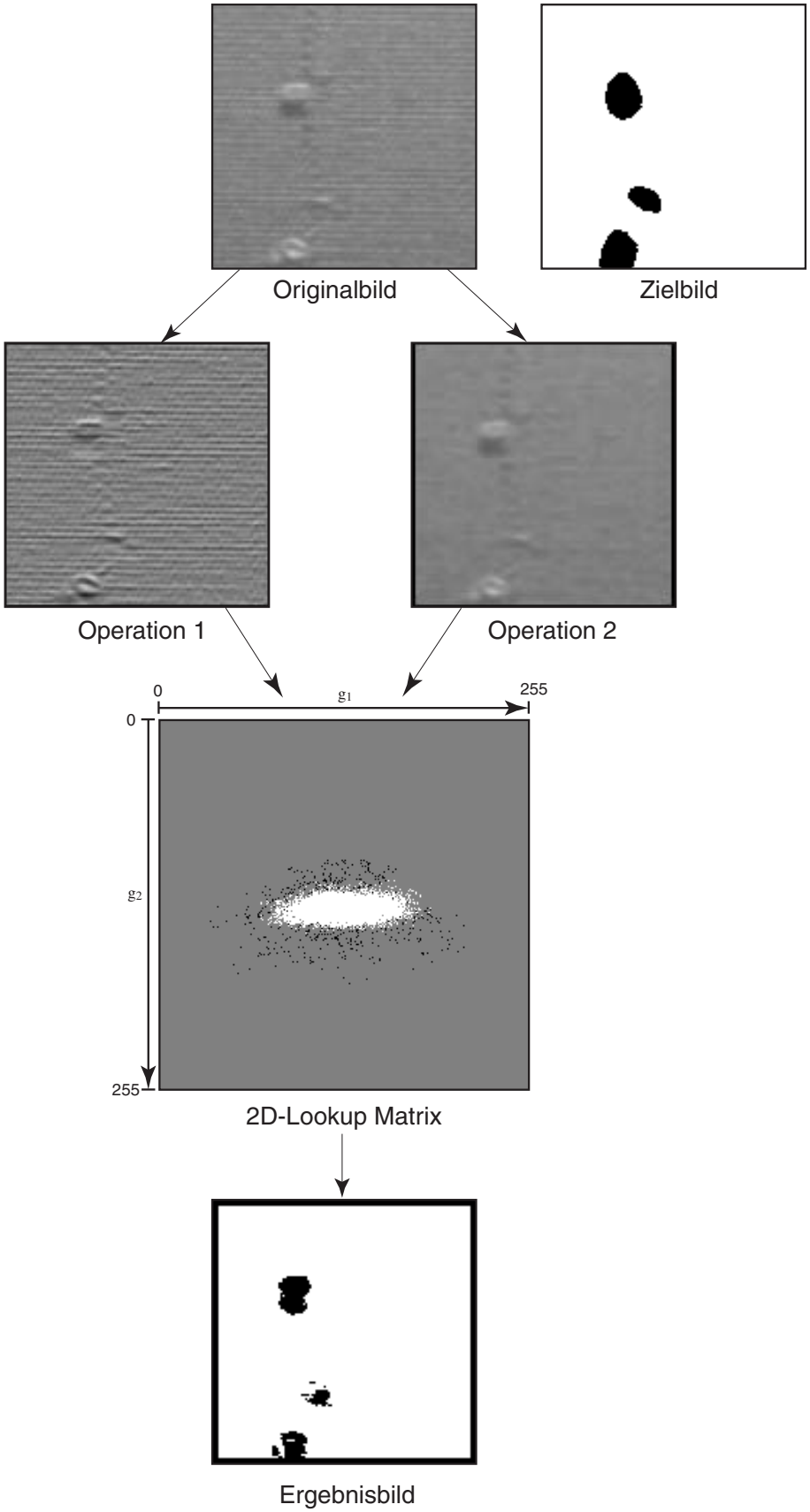



Abbildung 6.10: Ergebnis für Beispiel 2 (TFK 3-st23b).

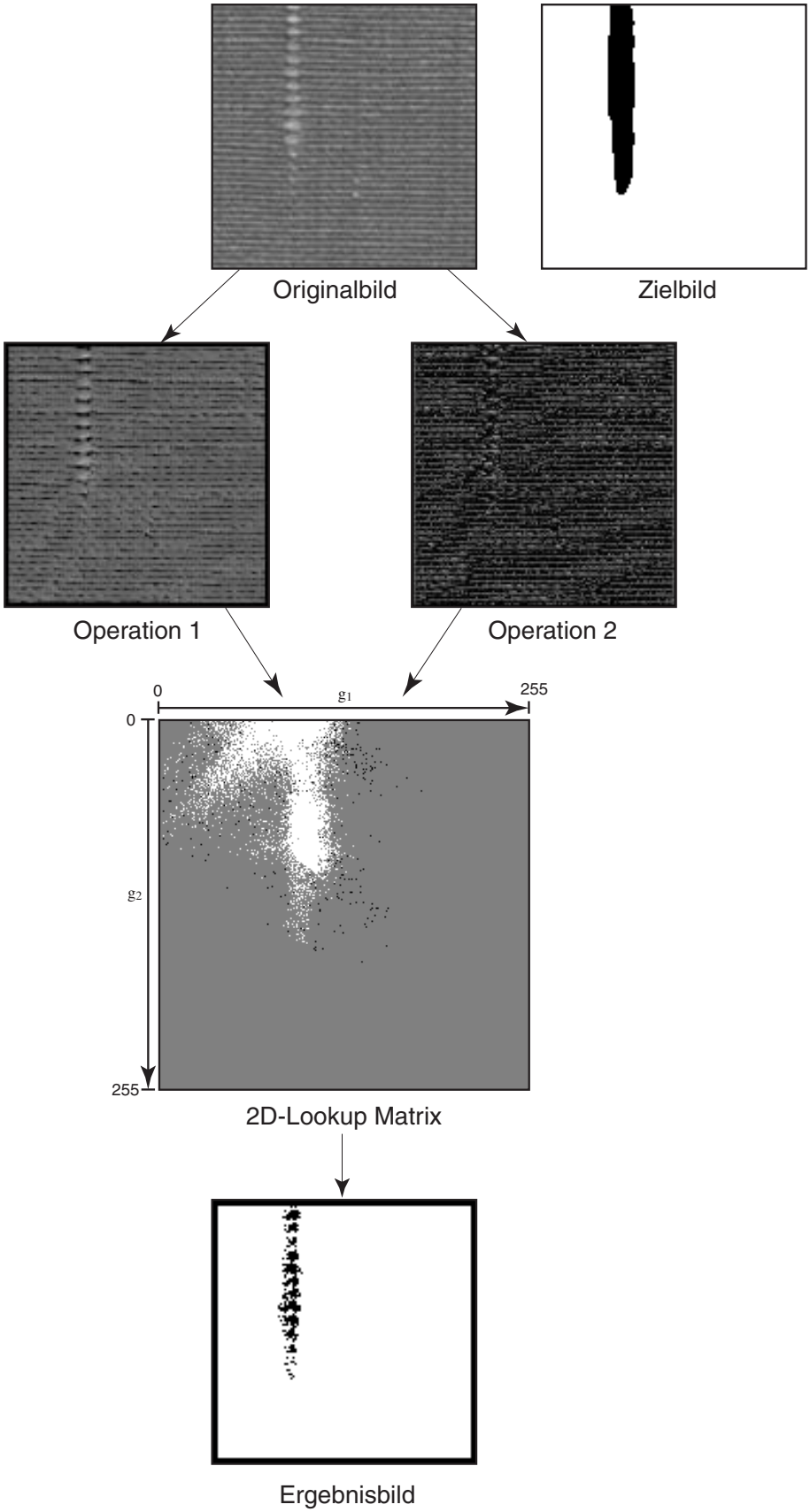



Abbildung 6.11: Ergebnis für Beispiel 3 (TFK 1-st11).

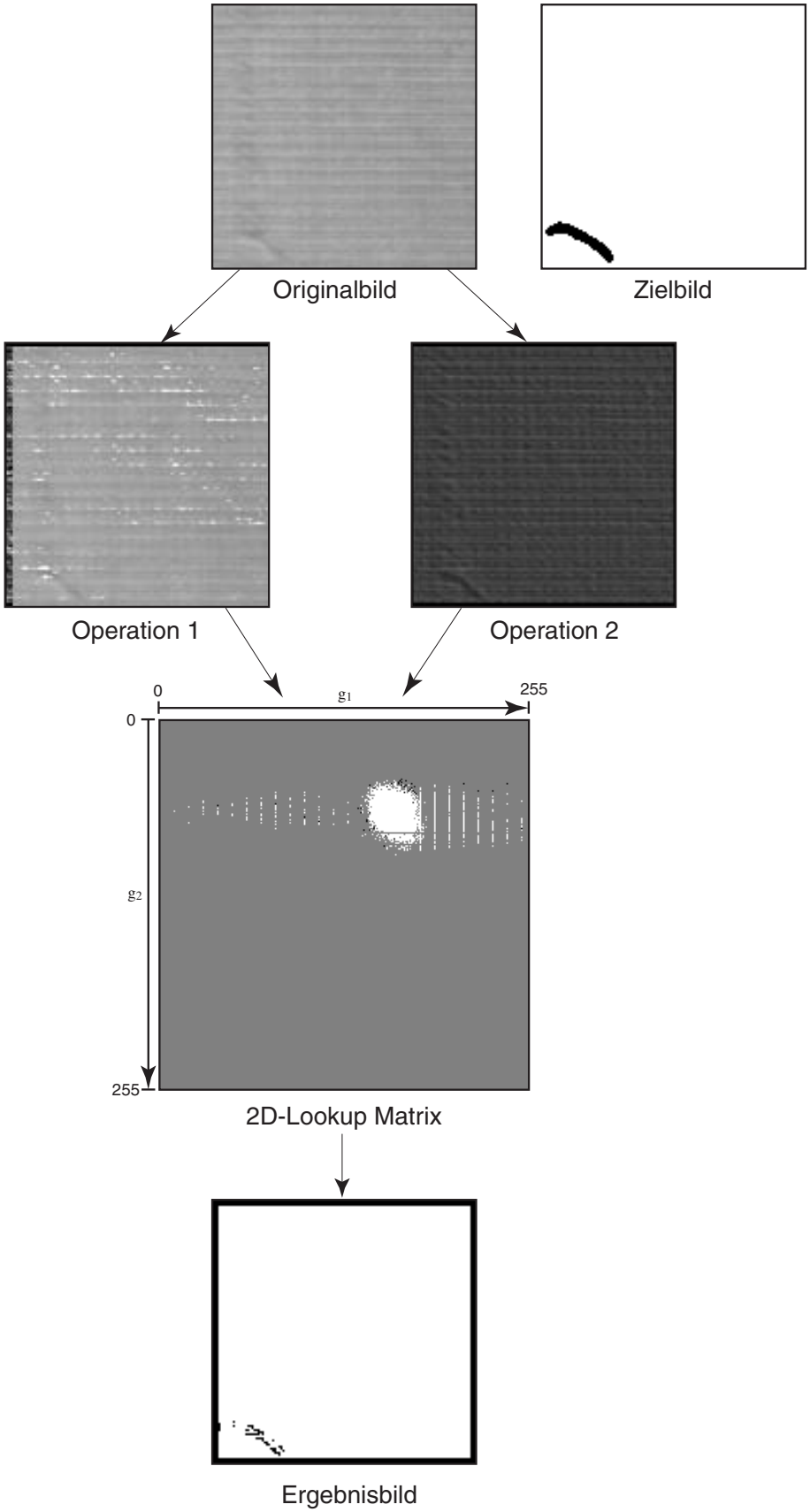



Abbildung 6.12: Ergebnis für Beispiel 4 (TFK 3-fr1a).

| TFK No. | No. of Wrong Blacks | No. of Grays |
|---|---|---|
| 1-fr13a | 407 | 12832 |
| 1-fr13b | 11 | 33030 |
| 1-fr13c | 2 | 2908 |
| 1-fr13d | 18 | 1343 |
| 1-fr19 | 15 | 1322 |
| 1-st47a | 1017 | 2276 |
| 2-i53 | 75 | 6 |
| 2-lau11b | 70 | 2098 |
| 2-lau19 | 280 | 862 |
| 2-lau2 | 524 | 3804 |
| 2-lau6 | 498 | 495 |
| 2-lau24 | 614 | 283 |
| 3-fr4c | 2325 | 16301 |
| 3-fr8 | 57774 | 5552 |
| 3-st18b | 5717 | 10352 |
| 3-st20 | 2142 | 3367 |
| 3-fr19 | 15044 | 6216 |
| 3-st47a | 17278 | 17680 |
| 4-fr1a | 88 | 2004 |
| 4-st14 | 2355 | 3882 |
| 4-fr19 | 67 | 1854 |
| 4-st22a | 536 | 2280 |
| 4-st27a | 5 | 2555 |
| 4-st29 | 12135 | 6450 |

Tabelle 6.1: Generalisierungsfähigkeit des LUCIFER Systems für 24 Beispiele aus dem TFK. Alle Bilder enthalten 262 144 Pixel. Im Mittel sind 1.9% der Pixel außerhalb der Referenz schwarz und 2.2% grau.

2D-Lookup Matrizen auftreten, je stärker hat sich die evolutionäre Anpassung auf zufällige Grauwertkonstellationen in den beiden Operationsbildern fokussiert. In diesem Fall werden die Filter eine geringe Leistung in Bezug auf andere Bilder aufweisen, die denselben Fehler oder denselben Texturhintergrund aufweisen.

Deshalb wird Kompaktheit innerhalb der 2D-Lookup Matrizen als wesentliches Kriterium zur Gewährleistung einer hohen Generalisierungsfähigkeit der generierten Texturfilter angesehen. Wenn diese Matrizen so manipuliert werden, daß sich diese Kompaktheit erhöht, werden die Filter auf anderen Bildern derselben Texturklasse eine höhere Erkennungsrate aufweisen (möglicherweise für den Preis einer geringeren Erkennungsrate für die Bildern, von denen sie generiert wurden). Im nächsten Abschnitt wird auf diese Frage weiter eingegangen.

### 6.6.2 Texturdetektion

In diesem Abschnitt soll am Beispiel der Texturdetektion dargestellt werden, wie LUCIFER auch zur Lösung anderer Aufgabenstellungen genutzt werden kann. Dies illustriert, daß die Vorgabe eines Zielbildes in einem gewissen Sinne auch als "Programmierung" des Systems verstanden werden kann und nicht nur als reine Nutzeroption. Texturdetektion ist für eine Reihe von Anwendungen insbesondere aus dem Gebiet der Bildinhaltsanalyse interessant. Als Beispiel sei hier nur genannt, daß z.B. Verfahren des digitalen Wasserzeichens Modifikationen eines Bildes besser in Bereichen mit starker Textur vornehmen können, die dem Betrachter dann weniger auffällig sind.

#### Ansatz

Der Ansatz zur Texturdetektion, d.h. zur Separation von texturierten und nichttexturierten Bereichen eines Bildes basiert auf einer einfachen Beobachtung: Abbildung 6.13 stellt zwei verschiedene Verteilungen von 20000 Zufallspositionen als schwarze Pixel in einem Bild der Größe $256 \times 256$ Pixel dar. Der Unterschied beider Verteilungen (a) und (b) liegt darin, daß die Verteilung in (b) einer zusätzlichen Einschränkung unterlag. In Teilbild (a) sind die Koordinaten aller Positionen gleichverteilt zufällig gewählt worden. Für Teilbild (b) wurden von diesen gleichverteilten Koordinaten nur Positionen akzeptiert, bei denen keine direkte Nachbarposition bereits vorher gewählt worden war. In Teilbild (b) ist also jeder schwarze Pixel von allen anderen separiert.

Der Eindruck eines "reinen Zufallsmusters" wird jedoch eher durch Teilbild (b) als (a) repräsentiert werden. Teilbild (a) scheint dem Betrachter mehr "Texturiertheit" zu bieten als Teilbild (b). Statistisch gesehen ist jedoch (a) "zufälliger". Dies ist ein subjektiver Eindruck, der schwer quantifiziert werden kann. Er zeigt jedoch, daß die Wahrnehmung von Textur an einer reinen Zufallsverteilung verifiziert werden kann, also Textur keine "*ground truth*" besitzen muss.

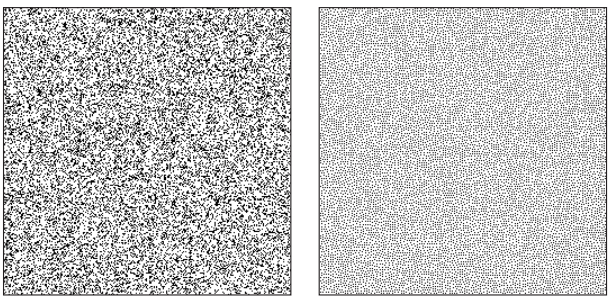



Abbildung 6.13: Zufällige Punktmusterverteilungen, die durch gleichverteilt zufällige Koordinaten generiert wurden. Für Teilbild (b) gab es die zusätzliche Einschränkung, daß ausgewählte Punkte nicht direkte Nachbarn sein dürfen. Teilbild (b) scheint trotzdem eher dem Eindruck einer positionellen Unterschiedslosigkeit zu entsprechen als (a).

Daraus läßt sich ein Ansatz ableiten, um texturierte Bereiche in Bildern zu extrahieren und zu dessen Umsetzung das LUCIFER Verfahren ebenso genutzt werden kann wie zur gezielten Gestaltung von Texturfiltern. So, wie das menschliche Auge dazu gebracht werden kann, Feinmuster in Punktverteilungen wahrzunehmen, kann der 2D-Lookup dazu gebracht werden, solche Feinmuster zu finden. Der direkte Ansatz ist es, solche Zufallsverteilungen in das Zielbild zu integrieren.

Ein kleines Experiment soll dies unterstreichen. Für ein aus gleichverteilt zufälligen Grauwerten bestehendes Grauwertbild wurde ein Zielbild gestaltet, bei dem ein willkürlich gewählter Bereich auf Schwarz gesetzt wurde (Abb. 6.14, mittlere Spalte). Natürlich kann es keinen Texturfilter geben, der das Zufallsgrauwertbild in solch ein Zielbild transformiert. Das LUCIFER Verfahren kann dies jedoch zu einem gewissen Grad schon erreichen, wie das Ergebnis rechts oben in Abb. 6.14 zeigt. Die häufigste Wiederholung eines lokalen Feinmusters (und dessen Exklusion in den als Weiß vorgegebenen Bereichen) kann zu einer Filterung nur dieser Bereiche mit einer höheren Fitness führen. Der Vergleich desselben Vorgehens mit eher homogenen Grauwertbildern, oder Gradientenverläufen zeigt, daß es in solchen Fällen nahezu unmöglich ist, den widersprüchlichen Vorgaben des Zielbilds (dieselben Grauwertkonstellationen wurde sowohl dem Vorder- als auch Hintergrund zugeordnet) gerecht zu werden. Nur einige wenige Punkte, im wesentlichen Randeffekte, können extrahiert werden.

Daraus kann der wesentliche Ansatz abgeleitet werden: Gegeben sei ein Bild $I$ unbekannten Inhalts. Diesem Bild wird ein Grauwertbild $I_g$ in eindeutiger Weise zugeordnet ($I$ kann auch ein Farbbild sein). Dann werden zufällige Binärbilder erzeugt, z.B. zwei zueinander inverse Schachbrettmuster wie in Abb. 6.15 dargestellt (die Regelmäßigkeit des Musters spielt keine Rolle für den Ansatz, es ist nur technisch besser zu handhaben). Nun wird mit jedem dieser Binärbilder als

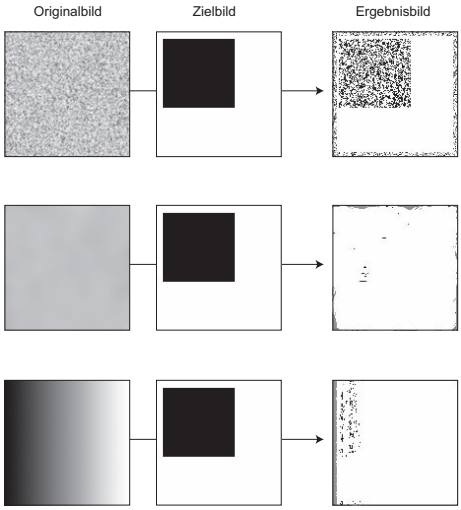



Abbildung 6.14: Anwendung des LUCIFER Verfahrens bei Vorgabe eines beliebigen Zielbildes mit einem Rauschbild (obere Reihe), einem homogenen Bild (mittlere Reihe) und einem Gradientenbild (untere Reihe) als Eingangsbild.

Zielbild LUCIFER ausgeführt, und die erhaltenen besten Ergebnisbilder werden zusammengefaßt. In diesem Fall besteht kein weiteres Interesse an den Operationsbildern oder der 2D-Lookup Matrix selbst.

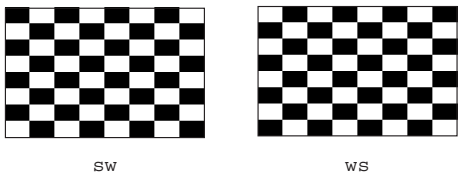



Abbildung 6.15: Zwei zueinander inverse Schachbrettmuster, die als Zielbilder zur Texturdetektion genutzt wurden.

Als Ergebnis erhält man mehr schwarze Punkte in Bereichen des Bildes $I_g$ mit starker Texturierung. Durch die Nutzung zueinander inverser Schachbrettmuster war in jedem Durchlauf jeder Pixel genau einmal dem Vordergrund zugeordnet worden. Natürlich können auch andere Mengen von Binärbildern genutzt werden. Wesentlich ist, daß es keine oder nur sehr geringe Korrelation zwischen der flächenhaften Verteilung von Grauwerten in $I_g$ und der Verteilung schwarzer Punkte in den Zufallsbildern gibt. Für allgemeine fotografische Aufnahmen ist dies auch sehr unwahrscheinlich.

### Ergebnisse

Das Verfahren wurde am Testbild "Neukaledonien" (Abb. 6.16) getestet. Die Ergebnisse mehrerer unabhängiger Anwendungen von LUCIFER (vier in diesem Fall)

wurden per Minimum-Operation zusammengefaßt, d.h. in dem in Abb. 6.16 dargestellten Ablauf ist im vorletzten Bild jeder Pixel als Schwarz gesetzt worden, der in mindestens einem der acht Ergebnisbilder auch schwarz war. Zur Nachbearbeitung dieses Bildes (unterstes Bild in Abb. 6.16) wurden die lokalen Dichten der Verteilung schwarzer Pixel dann durch Grauwerte repräsentiert.

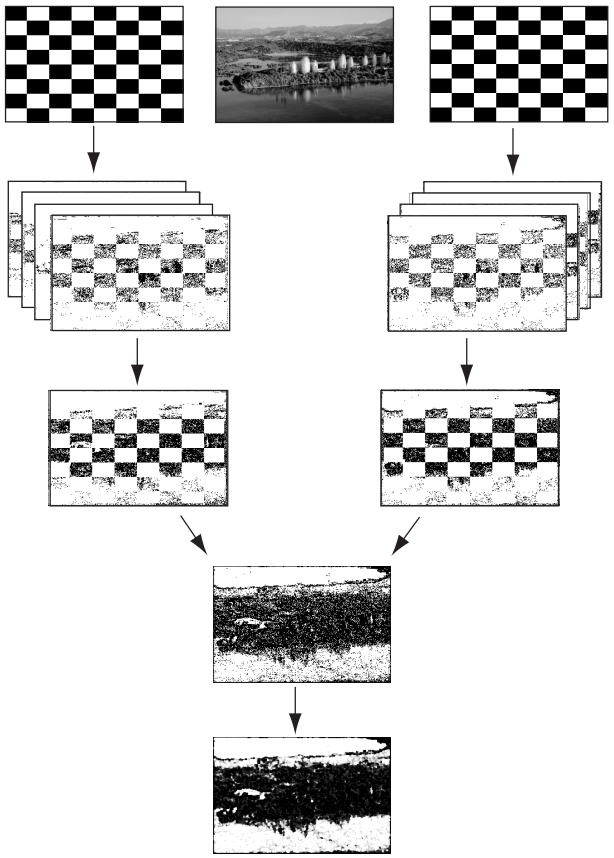


Abbildung 6.16: Texturdetektion auf dem Testbild: dargestellt sind das Eingangsbild und die zwei komplementären Zielbilder (obere Reihe), die Ergebnisse von jeweils vier Anwendungen von LUCIFER für jedes Zielbild (zweite Reihe), die Zusammenführung der Ergebnisse (dritte Reihe), das punktweise Minimumbild beider Teilergebnisse (vierte Reihe) und das nachbearbeitet Pixeldichtenbild (untere Reihe).

Im Ergebnisbild sind die stark texturierten Bereiche des Museums und der dahinterliegenden Waldlandschaft (inklusive der Reflexion des Museums im Wasser) klar von den wenig texturierten Bereichen des Wassers, des Berges am Horizont und des Himmels getrennt. Der ebenfalls kleine dunkle Bereich in der rechten oberen Ecke deutet auf einen Artefakt in der Aufnahme selbst hin, der erst bei genauer Betrachtung des Bildes wahrgenommen werden kann.

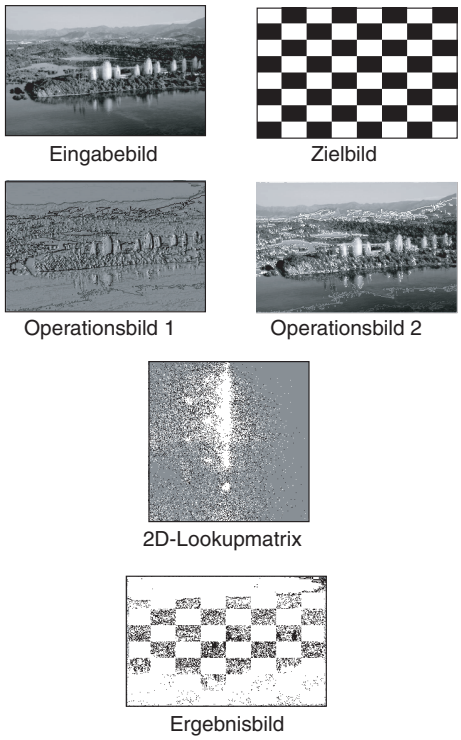



Abbildung 6.17: Zwischenergebnisse zu einem der LUCIFER Anwendungen aus Abb. 6.16.

Abbildung 6.17 stellt die internen Bilder des LUCIFER Verfahrens für einen Ablauf dar. Trotz der Zufallsvorgabe weist die adaptierte 2D-Lookup Matrix noch eine klare Struktur auf. Zum Vergleich zeigt Abb. 6.18 das entsprechende Varianzbild. Hierbei wurde die Varianz der Grauwerte in der $5 \times 5$-Nachbarschaft eines jeden Pixels als Grauwert dargestellt. Wie man sieht, verstärkt diese Operation lediglich lokale Varianzen und erkennt keine Ähnlichkeiten über größere Abstände, wie es der vorgestellte LUCIFER Ansatz leistet.

Abbildung 6.18: Varianzbild des Testbildes.

### Zusammenfassung

In diesem Abschnitt wurde vorgestellt, wie das LUCIFER Verfahren auch genutzt werden kann, um texturierte Bereiche in Bildern an sich zu detektieren,

unabhängig von einer genauen Vorgabe durch ein Zielbild. Dies kann erreicht werden, in dem Zufallsbilder als Zielbilder genutzt werden und die Ergebnisse geeignet zusammengefaßt werden.

## 6.7 Verbesserung der Systemleistung

### 6.7.1 Verbesserung der 2D-Lookup Matrizen

Die Bemerkungen am Ende des letzten Abschnitts führen in Bezug auf die per Relaxation gewonnenen 2D-Lookup Matrizen auf eine neue Fragestellung: das Finden und Verstärken kompakter Regionen in der Klasse von Bildern, zu denen die 2D-Lookup Matrizen gehören (siehe Abbildung 6.19 für weitere Beispiele).

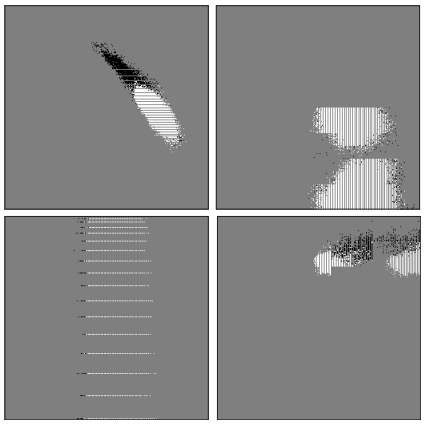


Abbildung 6.19: Beispiele für durch LUCIFER generierte 2D-Lookup Matrizen.

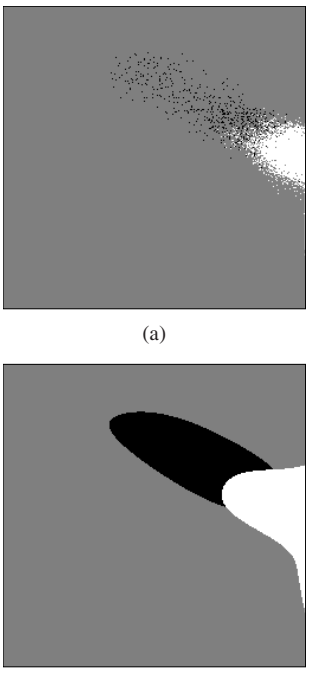
Auf den ersten Blick scheint dies ein typisches Problem für eine Clusteranalyse zu sein. Dies ist jedoch aus folgenden Gründen nicht der Fall: die Regionen mögen visuell wahrnehmbare Cluster ergeben, aber in einigen Fällen sind diese Cluster nur durch einige Pixel gegeben, die über der prospektiven Clusterregion verstreut sind. In anderen Fällen bilden sie zwar kompakte Regionen, jedoch mit unscharfen Begrenzungen. In solchen Fällen sind also Positionen trotz vorhandener Markierung (schwarz, weiß, grau) eindeutig im Rahmen der Regionenbildung anderen Gruppen zuzuordnen als im ursprünglich gegebenen Fall. Eine solche Clusterprozedur müßte also von sich aus auch generalisieren können.

Neuronale Netze stellen solche generalisierungsfähige Prozeduren dar. Im folgenden werden zwei Zugänge vorgestellt. Der erste Zugang benutzt ein MBPN, der zweite ein vereinfachtes Radiale Basisfunktionen Netzwerk (RBF), das sogenannte *Unit RBF* [4].

### Der MBPN Zugang

Ein MBPN (siehe Abschnitt 2.2.1) läßt sich mittels einer 2D-Lookup Matrix folgendermaßen trainieren: aus der Matrix werden zwei Bilder erzeugt. Im Bild $I_1$ sind alle Positionen weiß (Grauwert 255), die in der Matrix weiß sind, alle anderen sind schwarz (Grauwert 0). Im Bild $I_2$ sind alle Positionen schwarz (Grauwert 0), die in der Matrix schwarz sind, alle anderen Positionen sind weiß (Grauwert 255). Mit anderen Worten: beide Bilder stellen jeweils Grenzfälle der mit den grauen Positionen der 2D-Lookup Matrix gegebenen Ambiguität dar.

Die Bilder $I_1$ und $I_2$ geben nun positionelle Trainingsdaten für ein MBPN für ein Zweiklassenproblem (schwarze und weisse Pixel). Das MBPN wurde mit vier Eingabeneuronen, zehn versteckten Neuronen und zwei Ausgabeneuronen konfiguriert. Ein Trainingsdatensatz entsteht aus einer zufällig ausgewählten $(x, y)$ Position der Bilder und dem Pixelwert an dieser Position im jeweiligen Bild (1 für schwarz, 0 für weiß). Die vier Eingabeneuronen empfangen die Werte $x$, $y$, $x^2$ und $y^2$. Die Quadrate werden ebenfalls übergeben, damit das neuronale Netz auch Kurven zweiter Ordnung zur Separation der Eingabedaten in seinem Merkmalraum (der gleich der Koordinatenebene eines Bildes ist) nutzen kann.



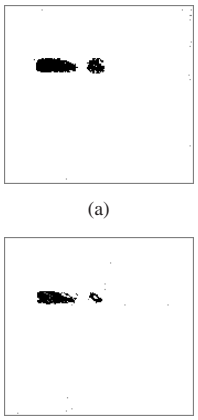
(a)

(b)

Abbildung 6.20: 2D-Lookup Matrix (a) und ihre Approximation durch ein MBPN (b) (einfacher Fall).

Zu jedem der Bilder $I_1$ und $I_2$ wurde ein eigenes MBPN trainiert. Es wurden jeweils 500 Positionen zufällig ausgewählt, und ein MBPN über jeweils 500 Zyklen trainiert (danach trat kaum noch eine Verringerung des Fehlers auf). Dann

wurde jedes Netz über alle Bildpositionen abgerufen, was ein Bild ergibt, welches entsprechend des Separationsverhaltens eines MBPN nur noch kompakte Regionen enthalten kann. Auf diese Art ergibt sich aus Bild $I_1$ das Bild $rec_1$ und aus Bild $I_2$ das Bild $rec_2$. Beide Bilder werden mittels der in Tabelle 6.2 angegebenen Regeln zu einem Ergebnisbild $rec$ fusioniert.



(a)

(b)

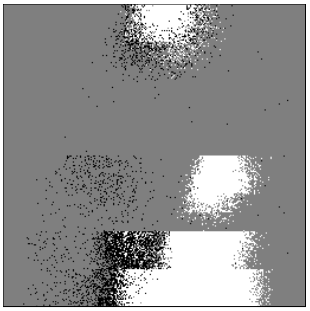
Abbildung 6.21: Ergebnisbilder unter Verwendung der ursprünglichen 2D-Lookup Matrix (a) und der approximierten (b) aus Abbildung 6.20.

| $rec_1$ | $rec_2$ | $rec$ |
|---|---|---|
| weiß | weiß | weiß |
| weiß | schwarz | weiß |
| schwarz | weiß | grau |
| schwarz | schwarz | schwarz |

Tabelle 6.2: Regeln zur Fusion der beiden Abrufbilder an einer Position $(x, y)$.

Die zweite Regel, die dabei den Konflikt beschreibt, daß $rec_1$ an derselben Position weiß ist, an der $rec_2$ schwarz ist, wurde zugunsten von weiß (also Texturhintergrund) entschieden, da in der Regel schwarze Punkte weiße kompakte Regionen umgeben. In Abbildung 6.20 ist das Ergebnis einer solchen Approximation einer einfachen 2D-Lookup Matrix dargestellt. Abbildung 6.21 zeigt den Vergleich der Anwendung beider Matrizen, der originalen und der approximierten, in Bezug auf das Ergebnisbild.

Für komplizierter geformte 2D-Lookup Matrizen ist das Ergebnis solch eines Vorgehens jedoch nicht zufriedenstellend. Zu viele isolierte schwarze Pixel, die über einen großen Teil des Bildes verstreut sind, können nicht ausreichend durch die Trainingsdaten beschrieben werden. In diesem Fall ist es sinnvoll, das Bild



(a)

(b)

Abbildung 6.22: 2D-Lookup Matrix (a) und ihre Approximation durch ein MBPN (b) (komplizierter Fall).

$I_2$ mittels morphologischer Operationen vorzuverarbeiten. Um das in Abbildung 6.22 dargestellte Ergebnis zu erzielen, wurde $I_2$ zuerst morphologisch geöffnet mit einer $3 \times 3$ Vollmaske, danach morphologisch geschlossen mit einer $3 \times 3$ Kreuzmaske. Dadurch wurden alle vollständig isolierten schwarzen Pixel entfernt, und in Gebieten mit mittelstarker Streuung der schwarzen Pixel „Minicluster“ aufgebaut.

Folgendes kann über den MBPN Zugang gesagt werden:

- Dieser Zugang ist in der Lage, 2D-Lookup Matrizen mit brauchbaren Ergebnissen zu approximieren.
- Die approximierten Regionen weisen eine glatte Form auf, jedoch ist die Approximation insgesamt sehr grob. Einige (wichtige) Details zu einer geringeren Skala können verlorengehen.
- Der Aufwand für die Approximation ist nicht sehr hoch, das Training der beiden MBPN jedoch relativ zeitaufwendig.
- Aus den Gewichten der versteckten Neuronen läßt sich jede Region als ein System von Ungleichungen beschreiben. Auf diese Art kann auch ein Texturmodell für die Hintergrundtextur erstellt werden, das jedoch nicht fuzzybasiert ist.

### Der Unit RBF Zugang

Das *Unit RBF* wurde durch Anderson vorgestellt [4]. Seine grafische Darstellung ist ein dreischichtiges *Radiale Basisfunktionennetzwerk* (RBF), dessen Gewichte einzig aus der Menge $\{-1, 0, 1\}$ stammen. Die internen Berechnungen eines *Unit RBF* können in der Formel

$$o(x) = \sum_{k=1}^{N} d_k e^{-k\|x-c_k\|^2} \tag{6.18}$$

zusammengefaßt werden, wobei $x = (x_1, x_2, \ldots, x_n)$ der Eingabevektor ist, $d_k$ eine ganze Zahl aus $\{-1, 0, 1\}$ und $c_k$ eine geeignete Sequenz von Zentren von Gauss-Verteilungen.

Das Ziel des Trainings eines *Unit RBF* ist die Approximation einer Menge durch den positiven Support von $o(x)$, d.h. wenn $\mathcal{D}$ als Menge durch den positiven Support einer Funktion $D(x)$ gegeben ist, soll $o(x) > 0$ gdw. $x \in \mathcal{D}$ für so viele $x$ wie möglich gelten.

Wenn die Menge $c_k$ gegeben ist (siehe unten), verläuft das Training eines *Unit RBF* folgendermaßen:
Initialization: $d_1$ ist das Vorzeichen von $D(c_1)$.
Schritt: Wenn $(k-1)$ Schritte bereits ausgeführt wurden und es ist

$$D(c_k) \cdot \sum_{j=1}^{k-1} d_j e^{-j\|x-c_j\|^2} \leq 0 \tag{6.19}$$

dann setze man

$$d_k = signD(c_k) \tag{6.20}$$

ansonsten.

$$d_k = 0. \tag{6.21}$$

Das heißt, daß $d_k$ 0 ist, wenn die aus $(k-1)$ Termen bestehende Funktion bereits das richtige Vorzeichen an der Stelle $c_k$ ergibt, ansonsten wird $d_k$ so verschieden von 0 gewählt, daß die Funktion an dieser Stelle korrigiert wird.

Um die benötigte Sequenz von $c_k$ Positionen zu erhalten, wird das sogenannte *Linear Pixel Shuffling* (LPS) verwendet. Wählt man $\alpha$ als postive reelle Wurzel der Gleichung $\alpha^3 = \alpha^2 + 1$ ($\alpha \simeq 1.4655712$), und $\beta = \alpha(\alpha - 1)$, so sind die $c_k$ im zweidimensionalen Fall durch $(\{\alpha k\}, \{\beta k\})$ gegeben, wobei $\{z\}$ den fraktionalen Teil von $z$ bezeichnet (also den Teil einer reellen Zahl nach dem Dezimalkomma). LPS sampelt das Einheitsquadrat auf eine hochgradig einheitliche Art, wie in Abbildung 6.23 dargestellt ist. In [4] und den dort gegebenen Referenzen finden sich viele Beispiele für Anwendungen der LPS Sequenz.

Das *Unit RBF* kann direkt auf die Aufgabenstellung der Approximation von 2D-Lookup Matrizen angewendet werden. Die beiden Bilder $I_1$ und $I_2$ werden

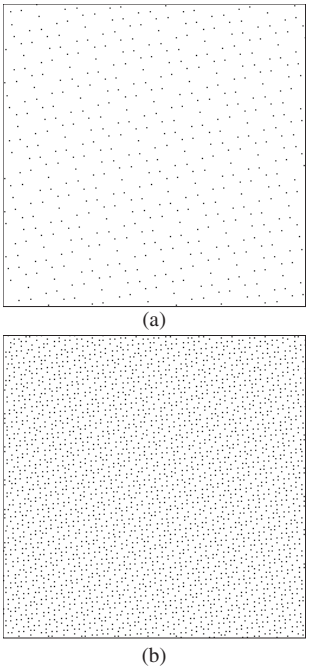


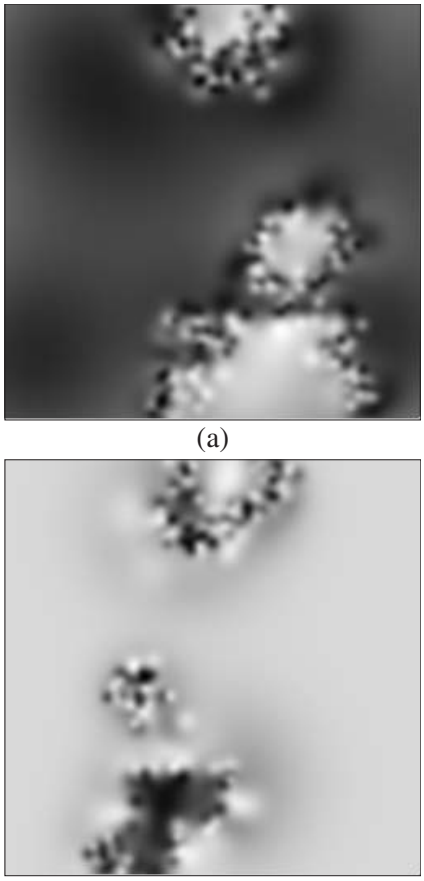

Abbildung 6.23: Die ersten 500 (a) und 3000 (b) Punkte der LPS Sequenz.

genauso wie beim MBPN Zugang bestimmt. Danach wird mit jedem Bild ein *Unit RBF* trainiert. Die Profilplots der sich dabei ergebenden Funktionen für das Beispiel aus Abbildung 6.22 (a) sind in Abbildung 6.24 dargestellt. Wenn beide „Abrufbilder“ punktweise gemäß den Regeln aus Tabelle 6.3 fusioniert werden, ergibt sich das in Abbildung 6.25 dargestellte Bild. Diese Approximation ist deutlich besser als die mit dem MBPN erhaltene.

| $rec_1$ | $rec_2$ | $rec$ |
|---|---|---|
| $> 127$ | $> 127$ | $max(rec_1, rec_2)$ |
| $> 127$ | $\leq 127$ | $rec_1$ |
| $\leq 127$ | $> 127$ | 127 |
| $\leq 127$ | $\leq 127$ | $min(rec_1, rec_2)$ |

Tabelle 6.3: Regeln zur Fusion der beiden *Unit RBF* Bilder an der Position $(x, y)$.

Aus der Approximation mittels *Unit RBF* läßt sich ein Fuzzy-Modell der Hintergrundtextur ableiten. Dazu betrachte man die ersten neun Terme des Aus-



(a)

(b)

Abbildung 6.24: Die Bilder $I_1$ (a) und $I_2$ (b) durch ein *Unit RBF* approximiert.

drucks, der zur Abbildung 6.24 (a) führte:

$$\begin{aligned} o(x,y) = &-e^{-4\|x-(220,186)\|^2} + e^{-5\|x-(83,105)\|^2} + \\ &+e^{-6\|x-(203,24)\|^2} - e^{-11\|x-(31,129)\|^2} + \\ &+e^{-13\|x-(13,222)\|^2} - e^{-23\|x-(181,177)\|^2} + \\ &+e^{-24\|x-(44,96)\|^2} + e^{-27\|x-(146,108)\|^2} - \\ &-e^{-42\|x-(141,168)\|^2} + \ldots \end{aligned} \tag{6.22}$$

Alle Gauss-Terme mit $d_k = 1$ stellen Fuzzy-Patches über der Menge weißer Pixel in der 2D-Lookup Matrix dar. Daraus können zweidimensionale Zugehörigkeitsfunktionen definiert werden, die an der Stelle $c_k$ ihr Zentrum haben und für die der Grad an Zugehörigkeit an der Stelle $x$ durch $max(o(x), 0)$ gegeben ist. Nennt man den entsprechenden linguistischen Term etwa „$c_k$-artig“, dann ergeben sich Regeln der Form

$$Wenn\,(op_1(x,y), op_2(x,y))\,c_k - artig\,ist,\ dann\,1 \tag{6.23}$$

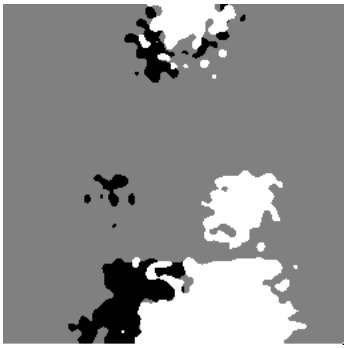


Abbildung 6.25: Approximation der 2D-Lookup Matrix aus Abbildung 6.22 (a) durch ein *Unit RBF*.

Ähnliche Regeln ergeben sich aus $I_2$ für die Konklusion 0. Abbildung 6.26 zeigt die Kontrollfläche aus den ersten 116 von 0 verschiedenen Termen der Approximation aus Abbildung 6.24 (a).

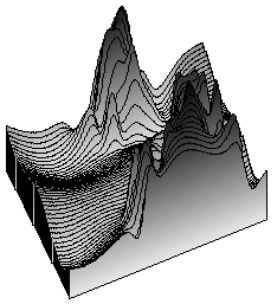


Abbildung 6.26: Kontrollfläche der Approximation mit 116 Termen.

Die so erhaltenen Regeln können in vielfältiger Weise verwendet werden:

- Eine Möglichkeit besteht in der Anpassung der Regelparameter an verschiedene globale Bedingungen des ursprünglichen Texturproblems. Wenn das Texturbild z.B. dunkler wird, würde der adaptierte Filter nicht mehr funktionieren, da sich die Region innerhalb der 2D-Lookup Matrix, die den Texturhintergrund repräsentiert, sich in Bereiche verschieben würde, die dunkleren Grauwerten entsprechen. Hätte man die 2D-Lookup Matrix nur pixelweise gegeben, wäre es nicht klar, wie diese Verschiebung zu modellieren ist. Ist die Matrix in Form von solchen Regeln gegeben, kann diese Verschiebung durch eine Modifikation der Regelparameter nachvollzogen werden.

- Jeder erneute Lauf des LUCIFER Systems würde einen neuen Texturfilter kreieren. Dabei können die Ergebnisse verschiedener Läufe nicht zusammengefaßt werden. Durch die Regeln wird dies jedoch möglich, da sie den

geeigneten Kompatibilitätsmodus für die Matrizen darstellen. Zwei oder mehr Texturfilter können einfach durch Vereinigung ihrer Regelmengen zusammengefaßt werden.

- Schritte der Bildverarbeitung, die auf das ursprüngliche Bild angewendet wurden, können auf Verarbeitungsschritte der Regelbasis abgebildet werden. Textursynthese durch Relaxation wird so möglich.

- Die Approximation, wenn ihre Ergebnisse zuverlässig sind, kann in den evolutionären Prozeß mit einbezogen werden, um die Fitnessfunktion nicht aus den Ergebnisbildern der Relaxation, sondern deren approximierten Versionen zu ermitteln.

### 6.7.2 Adaptives Vorverarbeitungsmodul

Das LUCIFER Verfahren liefert, wie in den vorhergehenden Abschnitten mehrfach gezeigt wurde, bei einer großen Zahl von Einsatzfällen sehr gute Ergebnisse. Es finden sich jedoch auch Situationen, in denen die Ergebnisse nicht ausreichend sind, wie in Abb. 6.27 illustriert. Die linienhafte Struktur des dort dargestellten Texturfehlers ist in der Ausschnittsvergrößerung kaum noch wahrnehmbar.

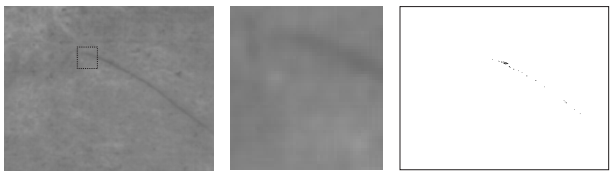



Abbildung 6.27: Bearbeitung eines Texturfehlers mit geringem Kontrast: (a) zeigt einen Kratzer auf einem Kollagentuch, (b) ist eine Ausschnittsvergrößerung. Teilbild (c) zeigt die Ausgabe einer Anwendung von LUCIFER auf dieses Problem, ein unzureichendes Ergebnis.

Eine genaue Analyse des Problems gibt, daß in diesem Fall die Verwendung von lokalen Bildoperatoren nicht ausreichend ist. Sämtliche innerhalb der Operationsbäume von LUCIFER zum Einsatz kommenden generischen Operatoren wirken nur innerhalb der unmittelbaren Nachbarschaft eines jeden Pixels, d.h. innerhalb eines begrenzten Definitionsbereiches. Strukturen wie die in Abb. 6.27 dargestellten Texturfehler weisen jedoch auch globale Merkmale auf, die von LUCIFER nicht direkt benutzt werden können.

Daher wird in diesem Abschnitt der Ansatz untersucht, auch globale Bildoperatoren innerhalb des LUCIFER Verfahrens einzusetzen. Speziell wird dazu ein Vorverarbeitungsmodul eingeführt, daß einen 2D-Lookup mittels des Histogramm des Bildes selbst durchführt.

### Überblick

In Abbildung 6.28 ist das LUCIFER Verfahren mit optionalem 2D-Histogramm Lookup (2DHL-Modul) dargestellt. Die einzelnen Komponenten werden in den nachfolgenden Abschnitten beschrieben. Wie in der Abbildung ersichtlich, ähnelt die Erweiterung im wesentlichen dem LUCIFER Verfahren ohne Erweiterung, nur die Festlegung der 2D-Lookup Matrix erfolgt anders.

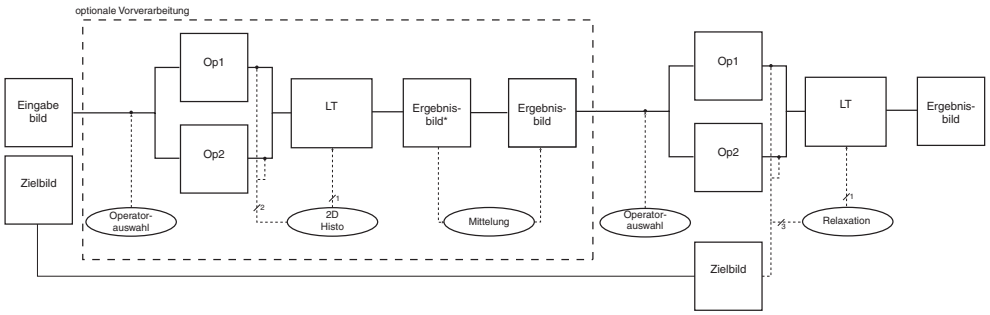



Abbildung 6.28: Das erweiterte LUCIFER Verfahren, mit dem optionalen 2D-Histogramm Lookup-Modul zur Vorverarbeitung.

Zur Wiederholung: der 2D-Lookup Algorithmus wird durch drei Komponenten beschrieben: die Bildoperatoren 1 und 2, die die beiden Operationsbilder 1 und 2 aus dem Eingangsbild generieren, und die 2D-Lookup Matrix. Beide Bildoperatoren können beliebig innerhalb der Spezifikationen des LUCIFER Verfahrens gewählt werden, daher das adaptive Potential des Algorithmus. Die Operatoren werden vom Filtergenerator (sieh Abb. 6.1) bereitgestellt.

Um eine geeignete 2D-Lookup Matrix auszuwählen, werden im LUCIFER Verfahren ohne Erweiterung und in der Erweiterung selbst verschiedene Strategien benutzt. Im LUCIFER Verfahren wird die 2D-Lookup Matrix in Relation zum Zielbild eingestellt, um so die maximal erreichbare Ähnlichkeit zwischen 2D-Lookup Ergebnis und Zielbild zu erreichen (gemessen gemäß der Fitnessfunktion, siehe Abschnitt 6.3, und erzielt mittels Relaxation, siehe Abschnitt 6.4). Im 2DHL-Modul dagegen wird die 2D-Lookup Matrix aus dem 2D-Histogramm der beiden Operationsbilder gebildet. Das aus dem 2D-Histogramm erzeugte und als 2D-Lookup Matrix genutzte Bild ist in Abbildung 6.28 mit *Ergebnisbild** bezeichnet. Die beiden Operatoren des 2DHL-Moduls werden unabhängig von den beiden Operatoren des LUCIFER Verfahrens selbst festgelegt.

### 2D-Histogramm basierter 2D-Lookup

Ein 2D-Histogramm basierter 2D-Lookup ist ein 2D-Lookup Algorithmus, bei dem das normalisierte 2D-Histogramm als 2D-Lookup Matrix benutzt wird, und der im 2DHL-Modul zum Einsatz kommt. Dabei ist, folgend der im Abschnitt

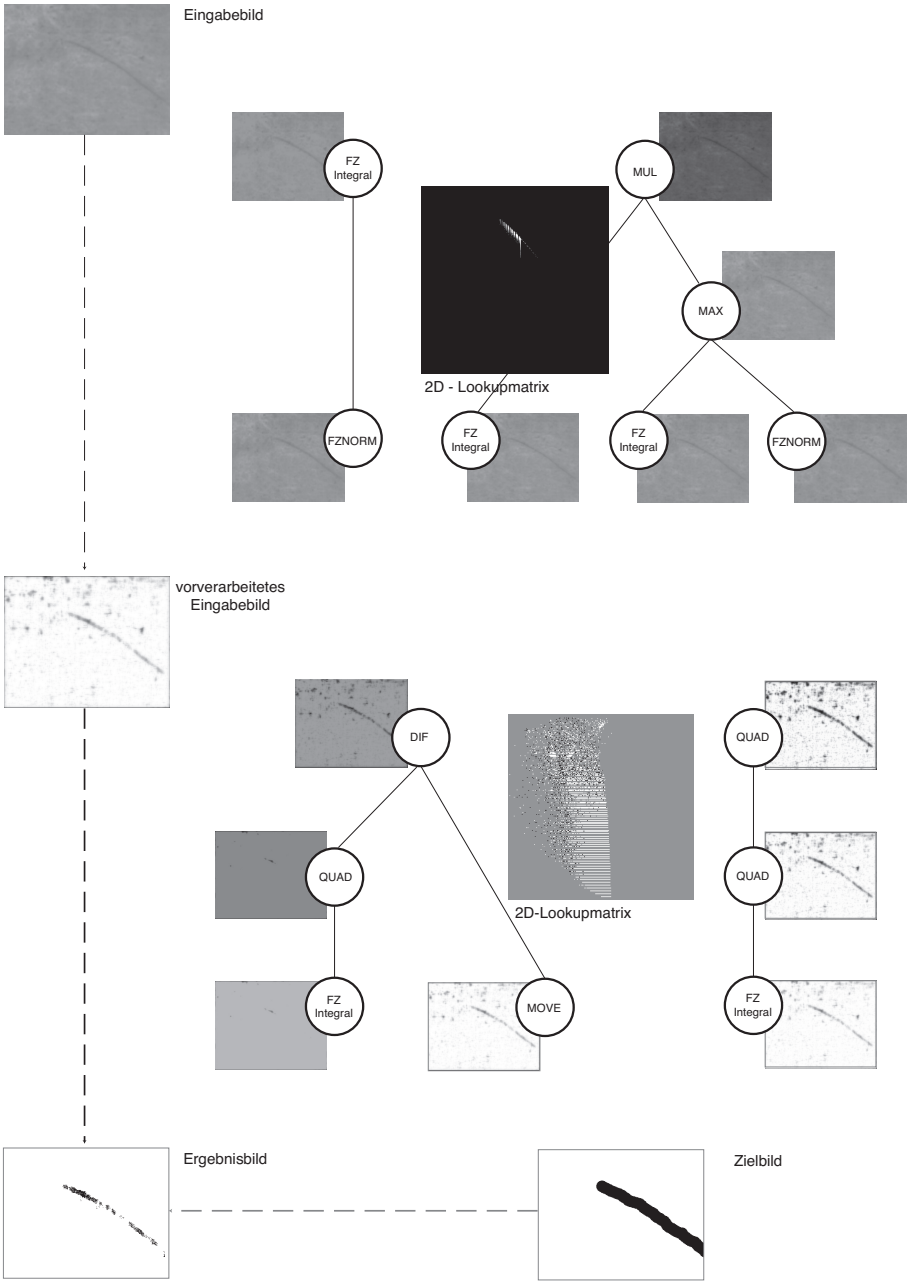



Abbildung 6.29: Ausführliche Darstellung der Anwendung des erweiterten Verfahrens mit allen Zwischenbildern auf das Beispiel von Abb. 6.27.

6.2 benutzten Notation zur Definition des 2D-Lookup Algorithmus, das 2D-Histogramm definiert als die Matrix $H(g_1, g_2)$, die aus zwei Operationsbildern $I_1$ und $I_2$ gleicher Größe und Auflösung ermittelt wird. Ein Eintrag in $H$ an der Stelle $(g_1, g_2)$ steht für die Anzahl von Pixeln in beiden Bildern, die an der gleichen Position den Grauwert $g_1$ in Bild $I_1$ und den Grauwert $g_2$ im Bild $I_2$ haben:

$$H(g_1, g_2) = \sum_{(x,y) \in I_1 \cap I_2} \delta(I_1(x, y), g_1)\ \delta(I_2(x, y), g_2) \tag{6.24}$$

wobei $\delta(a, b)$ den Wert 1 hat für $a = b$ und 0 sonst.

Um solch eine Matrix als Operand in einem 2D-Lookup Algorithmus zu benutzen, müssen die Einträge in den Wertebereich von Grauwerten abgebildet werden, da die Anzahl von solchen Grauwertpaaren theoretisch größer werden kann als der maximale Grauwert. In der Praxis sind jedoch diese Werte eher klein, und es genügt oft, alle Einträge größer als $g_{max}$ auf $g_{max}$ zu setzen.

Die so aus dem 2D-Histogramm erzeugte 2D-Lookup Matrix kann weitergehend als Bild bearbeitet werden. Interessant ist hier z.B. eine Linearisierung. Linearisierung ist dabei eine 1D-Lookup Operation, bei der das Summenhistogramm des Bildes als Lookup-Tabelle benutzt wird. Das Summenhistogramm des transformierten Bildes wird dann selbst linear, daher die Bezeichnung der Operation. Es ist dabei im Falle des 2D-Histogramms zweckmäßig, erst ab dem Grauwert 1 zu zählen, um so die hohe Anzahl von Einträgen 0 zu vernachlässigen.

Die Folge einer Linearisierung ist eine starke Anhebung des Bildkontrastes, jedoch reduziert sich die Anzahl der verschiedenen Grauwert im Bild, und die Folge kann eine Art "krümelige" Erscheinung des Ergebnisbildes des 2D-Lookup sein. Durch eine der Linearisierung nachgeschaltete GAUSS-Glättung kann dieser Effekt unterdrückt werden.

Der Spezialfall der Benutzung der Co-occurrence Matrix als 2D-Histogramm (wobei die beiden Operatoren die identische Operation und die Verschiebung des Bildes in Offset-Richtung sind) wird auch als Autolookup bezeichnet und konnte bereits erfolgreich zur Detektion globaler Fehlererscheinungen eingesetzt werden [86].

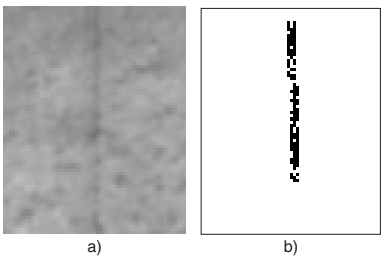



Abbildung 6.30: Ergebnis für einen schwer detektierbaren Messerfehler auf einem Kollagentuch.

Solch eine Vorverarbeitung wirkt auch als eine einfache Form eines sog. Neuheitsfilters: im Ergebnis des 2D-Histogramm basierten 2D-Lookup sind genau alle die Strukturen dunkel, die *selten* in den Operatorbildern auftreten. Dies ist ein simples Texturfehlermodell, das im LUCIFER Verfahren effizient umgesetzt werden kann.

### Anwendungsbeispiele

Das erweiterte LUCIFER Verfahren wurde angewendet, um oben benannte Probleme mit schwachkontrastigen Texturfehlern (siehe S. 129) zu bearbeiten. In Abbildung 6.29 sind alle Operationsschritte für solch einen schwachkontrastigen Texturfehler des besten mit dem LUCIFER Verfahren plus 2DHL-Modul gefundenen Texturfilters dargestellt. Folgende Aussagen können dazu getroffen werden:

- Das so erhaltene Ergebnisbild (Abb. 6.29 links unten) hat eine höhere Ähnlichkeit zum Zielbild als das in Abbildung 6.27(c) dargestellte Ergebnis.

- Die Anwendung des 2DHL-Moduls führt zu einer Verstärkung des Kontrastes des Texturfehlers gegenüber dem Hintergrund (Abb. 6.29 Mitte links). Dadurch kann das nachfolgende LUCIFER Verfahren den Fehler besser segmentieren.

- Die generierten Operatorenbäume zeigen eine hohe Redundanz. Viele Operationen haben keinen direkten Einfluß auf das Ergebnis. So erzeugt als Beispiel der linke Teilbaum von Operator 1 im 2D-Lookup (der Baum links unten in 6.29) im wesentlichen ein graues Bild, von dem eine verschobene Version des 2DHL-Ergebnisbildes pixelweise subtrahiert wird. Solche Teile können bei einem Redesign des Texturfilters entfernt werden, um die Leistung hinsichtlich speziell der Rechenzeit zu verbessern.

- In diesem Beispiel erweist sich das Ergebnis im wesentlichen als ein Autolookup des 2DHL-Ergebnisbildes.

- Der entscheidende Schritt bei diesem Ergebnis ist der Operator 2 des 2DHL-Moduls (Baum oben rechts in Abb. 6.29): in den höheren Grauwerten ist im 2D-Histogramm eine Verzweigung sichtbar, die letztlich zu einer Separation heller Bildteile in Vorder- und Hintergrund führt.

Abbildung 6.30 stellt ein weiteres Beispiel aus der Praxis dar, ohne die Zwischenergebnisse. Dabei handelt es sich um die Fehlerklasse "Messerfehler", die im Schneideprozeß von Kollagentüchern auftreten kann. Auch hier zeigt der erweiterte Ansatz eine gute Leistung.

### Zusammenfassung

In diesem Abschnitt wurde eine Erweiterung des LUCIFER Verfahrens vorgestellt, die insbesondere auf die Anwendung für schwachkontrastige Texturfehler zugeschnitten ist. Dabei wird das LUCIFER Verfahren um ein Vorverarbeitungsmodul erweitert, das auf dem 2D-Histogramm basierten 2D-Lookup (2DHL-Modul) beruht. Das LUCIFER Verfahren selbst nutzt dann das Ergebnisbild des 2DHL-Moduls als Ausgangsbild. Dadurch gehen auch globale Bildeigenschaften in das so erweiterte LUCIFER Verfahren mit ein, und es ist möglich, den Kontrast des Fehlers gegenüber dem Texturhintergrund zu erhöhen. Die Verbesserung der Ergebnisse im Falle solcher Vorverarbeitung konnte am Beispiel gezeigt werden. Der wesentliche Nachteil besteht in einem höheren Rechenaufwand bei der Erzeugung und Benutzung solcher Filter, so daß dieser Ansatz nur genutzt werden sollte, wenn die LUCIFER Ergebnisse nicht akzeptabel sind.

## 6.8 Zusammenfassung

Das hier vorgestellte LUCIFER System ist ein 2D System zur Erzeugung von Texturfiltern, daß lediglich mittels eines Eingangsbildes und eines binären Zielbildes adaptiert wird, also mit einem fast minimalen Nutzerinterface. Das System basiert auf der internen Dekomposition eines Texturfilters durch die Komponenten eines 2D-Lookup Algorithmus. Die Fusion der beiden internen Operationsstränge erfolgt dabei mittels einer Relaxation, die von den Operationen selbst unabhängig erfolgen kann. Die einzelnen Komponenten werden genetisch programmiert.

Das System wurde an Beispiele aus dem TFK getestet. Die „ohne menschliches Zutun“ erzeugten Texturfilter sind praktisch einsatzfähig. Eine Verbesserung der Filter, z.B. durch Manipulation der bei der Relaxation entstehenden 2D-Lookup Matrizen, ist jedoch jederzeit möglich und führt zu allgemeiner einsetzbaren Filtern.

Folgende Erfahrungen wurden dabei gemacht:

- Das System erzeugt Filter mit guter und sehr guter Leistungsfähigkeit.
- Das Zielbild sollte die Fehlerregion so gut wie möglich umschreiben, da Abweichungen hier zu Vermengungen von schwarzen und weißen Positionen in der 2D-Lookup Matrix führen.
- Randbereiche werden zur Bestimmung der Fitnessfunktion vernachlässigt.
- Das Sytem kann sowohl Filter für nicht-kompakte Fehlerregionen als auch Fehlerregionen mit variierender Erscheinungsform generieren.
- Die generierten Filter können manuell nachverbessert werden.

# 7 Zusammenfassung

In der vorliegenden Arbeiten wurden die Möglichkeiten des Einsatzes von *Soft Computing* in der Mustererkennung untersucht. Diese Aufgabe ist ein wichtiges Element für die weitere zukünftige Gestaltung digitaler Technologien und Infrastrukturen, die nur auf Basis neuer Wege der Repräsentation der "atomaren" Umwelt und der sie prozessierenden Verfahren geschehen kann.

Basierend auf methodischen Entwicklungen der frühen 60er Jahre, die jedoch mangels ausreichender Rechnerkapazität aus dem Hauptblickfeld der aktiven Forschungen gerieten, wurde *Soft Computing* als Dachbegriff für Verfahren wie neuronale Netze, Fuzzy-Inferenzen oder evolutionäre Algorithmen eingeführt. Hauptaspekt war jedoch die gemeinsame Fähigkeit solcher Verfahren, Ungenauigkeiten der Lösungen handhabbar zu gestalten (*exploitation of the tolerance for imprecision and uncertainty*). Die Grundthese ist, daß höhere Genauigkeit auch einen höheren Aufwand erfordert. Dies ist somit keine Alles-oder-Nichts These (wie etwa bei einer mathematisch exakten Lösung, die man entweder hat oder nicht), da die benannten Verfahren genau solch eine Möglichkeit anbieten.

Im Kapitel 2 dieser Arbeit wurden einige solcher Verfahren vorgestellt, insbesondere das Perzeptron, das Multilayer Backpropagation Neural Netwerk, Fuzzy-Logik, der Genetische Algorithmus und die Genetische Programmierung. Die Einführung dieser Verfahren erfolgte unter dem Aspekt der Vorbereitung ihrer mathematischen Diskussion, und nicht, um Vollständigkeit zu erlangen. Die Möglichkeiten der Anpassung des Aufwandes zur Steuerung der Genauigkeit oder Erlangbarkeit einer Lösung wurden diskutiert.

Damit bleibt jedoch für die *Soft Computing* Verfahren die wichtige Fragestellung offen, wie *universell* sie einsetzbar sind. Hier wurden von den Forschern in den letzten Jahren eine Reihe von Theoremen zusammengetragen, die helfen, diese Fragestellung besser zu beleuchten. Diese wurden, in Anlehnung an die in Kapitel 3 benannte Aufgabenstellung, in Kapitel 4 vorgestellt und diskutiert. Das "Ugly Duckling" Theorem wurde in diesem Zusammenhang kurz wiederholt, zum Teil auch dadurch motiviert, daß sich heutzutage relativ wenige Verweise auf dieses historisch dennoch wesentliche Theorem finden lassen. Im wesentlichen besagt daß "Ugly Duckling" Theorem, daß Objektunterscheidung alleine auf Basis von logischen Attributen nicht durchführbar ist. Objekte und ihre Beziehungen untereinander sind reichhaltiger als die bloße logische Konsternierung ihrer Existenz. Dieses Theorem liefert einen wichtigen Hinweis zur Klärung der Frage nach der Repräsentation von sensorischen Daten für die weitere Verarbeitung. Man muß davon ausgehen, daß dies auf Basis logischer Prädikate nicht möglich ist, oder

die Analyse auf Basis logischer Prädikate auf Grenzen stoßen muß. Daten für das *Soft Computing* sind also möglichst elementar zu repräsentieren.

Ebenso schränkt das "No Free Lunch" Theorem den Wirkungsradius von *Soft Computing* Verfahren ein. Es stellt heraus, daß kein Verfahren einem anderen als überlegen angesehen werden kann, insofern es um die mittlere Performance eines Verfahrens, gemessen an allen möglichen Aufgabenstellungen geht. Hier wurde erstmalig ein einfacher Beweis des Theorems präsentiert, der die dem Theorem zugrundeliegende Sichtweise auf Algorithmen besser herausstellt.

Eine wichtige Rolle kommt ebenso dem KOLMOGOROV-Theorem zu. Es garantiert die prinzipielle Möglichkeit, jede stetige Funktion (und damit auch jede unbekannte, nur aus Systemdaten referenzierbare Systemfunktion) durch neuronale Netze, Fuzzy-Patches, Fuzzy-Normalformen oder Baumstrukturen der Genetischen Programmierung zu repräsentieren. Dies ist eine reine Existenzaussage, und der Frage nach ihrer praktischen Umsetzbarkeit wurde in dieser Arbeit im Abschnitt 4.4 nachgegangen. Erstmalig wurde dabei auch die konkrete Konfiguration eines KOLMOGOROV-Netzwerkes am Beispiel der Maximum-Funktion und eines digitalen Bildes präsentiert.

Als letztes wesentliches Theorem des *Soft Computing* wurde auf das Schemata-Theorem eingegangen, welches insbesondere bei genetischen Algorithmen eine wichtige Rolle spielt. Aus dem Schemata-Theorem heraus läßt sich verstehen, daß die Gestaltung der Fitnessfunktion nicht unbedingt mit dem Ziel erfolgen muß, die Fitnesswerte optimal zu erhalten. Es können auch Anwendungen genetischer Algorithmen gestaltet werden, deren Erkennungsziel die Schemata selbst sind.

Wesentliche Konsequenz für die weitere Gestaltung von auf *Soft Computing* basierenden Bildverarbeitungssystemen ist die korrekte Behandlung der Bildoperatoren selbst. Nur durch eine ausreichende Repräsentation der Operatoren kann letztlich Lernfähigkeit gewährleistet werden. Dies ist das wesentliche Ergebnis der Ausführungen in Kapitel 5, in dem durch die mehrdimensionalen Gerüste auch der entscheidende Ansatz für solch eine Realisierung gegeben wird.

In LUCIFER kommt dann neben dem Design als mehrdimensionales Gerüst der 2D-Lookup Algorithmus der mathematischen Morphologie zum Einsatz. Dieser benötigt zur Durchführung die Festlegung dreier Komponenten: zwei Bildoperationen und die binäre 2D-Lookup Matrix. Die Bildoperationen, die aus dem Ausgangsbild ein dem Zielbild möglichst ähnliches Ergebnisbild generieren, werden mittels genetischer Programmierung gesucht. Dies beinhaltet auch eine Konzeption der allgemeinen Beschreibung von Bildoperatoren durch die Baumstrukturen des genetischen Programmes.

Die 2D-Lookup Matrix läßt sich, als ein wesentlicher Vorzug des Systems, bei gegebenen Bildoperationen aus diesen direkt fusionieren und muß nicht extra adaptiert werden. Es zeigt sich jedoch, daß diese Matrix wesentlich für die Generalisierung der Texturfilter zuständig ist. Bestimmte, die Generalisierung begünstigende Strukuren können durch eine nachgeschaltete Auswertung mittels eines speziellen neuronalen Netzes hervorgehoben werden.

Das Sytem LUCIFER erweist sich als konsistente Umsetzung der Methoden des *Soft Computing* zur Gestaltung eines lernfähigen Bildverarbeitungssystems.

# Anhang A

# Beweise zum No-Free-Lunch Theorem

## A.1 Beweis von Theorem 3

*Beweis.* Zum Beweis ermittle man in $\sum_f \delta(d_m^y, (f, m, a))$ alle die Kostenfunktionen $f_+$, für welche $\delta(d_m^y, (f_+, m, a))$ den Wert 1 annimmt. Dazu müssen folgende Bedingungen erfüllt sein:

i) $f(d_m^x(1)) = d_m^y(1)$

ii) $f(a[d_m(1)]) = d_m^y(2)$

iii) $f(a[d_m(1), d_m(2)]) = d_m^y(3)$

...

Durch diese Bedingungen wird $f_+$ für $m$ Elemente aus $X$ festgelegt. Für alle weiteren $|X| - m$ Elemente aus $X$ kann der entsprechende Wert von $f_+$ frei aus $Y$ gewählt werden. Damit sind von den $|Y|^{|X|}$ Summanden genau $|Y|^{|X|-m}$ gleich 1 und alle anderen gleich 0. Es folgt für einen beliebigen Algorithmus $a$:

$$\sum_f \delta(d_m^y, (f, m, a)) = |Y|^{|X|-m}.$$

Dies gilt unabhängig vom konkreten Algorithmus $a$. □

## A.2 Beweis von Korollar 1

*Beweis.* Dies ergibt sich sofort aus

$$\sum_f \delta(d_m^y, (f, m, a)) = |Y|^{|X|-m} \geq 1$$

(da $Y$ als nicht-leer vorausgesetzt wurde) und der Tatsache, daß alle Terme der linken Seite 0 oder 1 sind. Daher muß für mindestens ein $f$ der entsprechende Term 1 sein, was genau der Aussage des Korollars entspricht. □

## A.3 Beweis von Satz 1

*Beweis.* Es können stets mehrere Testsequenzen $d_m$ zu demselben Performance-Vektor $c$ führen. Daher wird die Summe nach diesen Möglichkeiten aufgespalten:

$$\sum_f \delta(c,(f,m,a)) = \sum_{f,d_m^y \in Y^m} \delta(c,d_m^y)\delta(d_m^y,(f,m,a)) \tag{A.1}$$

Dabei muß hier betont werden, daß diese Aufspaltung eigentlich über alle $d_m^y$ erfolgt, die tatsächlich als Ergebnis der Anwendung von $a$ auf irgendeine Kostenfunktion $f$ erhalten werden können [83]. Da allen in diesem Sinne unmöglichen Testsequenzen $d_m^y$ jedoch ein zusätzlicher Term mit dem Wert 0 zugeordnet werden kann, ohne daß sich der Gesamtwert der Summen ändert, kann genausogut gesagt werden, daß die Summe über alle Teilmengen von $Y$ läuft, die aus $m$ Elementen bestehen, also der Menge aller Testsequenzen der Länge $m$ (David Wolpert, persönliche Korrespondenz). Unabhängig davon ist die Äquivalenz der Menge aller erreichbaren Testsequenzen und der Menge aller $m$-elementigen Teilmengen von $Y$ bereits durch Korrolar 1 gewährleistet.
Um den Beweis fortzusetzen, ergibt sich dann aus Gleichung A.1:

$$\begin{aligned}
\sum_f \delta(c,(f,m,a)) &= \sum_{f,d_m^y \in Y^m} \delta(c,d_m^y)\delta(d_m^y,(f,m,a)) \\
&= \sum_{d_m^y \in Y^m} \delta(c,d_m^y) \sum_f \delta(d_m^y,(f,m,a)) \\
&= \sum_{d_m^y \in Y^m} \delta(c,d_m^y)|Y|^{|X|-m} \\
&= |Y|^{|X|-m} \sum_{d_m^y \in Y^m} \delta(c,d_m^y)
\end{aligned} \tag{A.2}$$

Der letzte Ausdruck hängt nicht von $a$ ab, da die Anzahl der Möglichkeiten, einen bestimmten Performance-Vektor $c$ aus allen möglichen $d_m^y$ zu erhalten, nicht vom Algorithmus $a$ beeinflußt wird, und die Anzahl $|Y|^m$ der Summanden ebenfalls nicht von $a$ abhängt. □

# Anhang B

# Approximation von Funktionen

Die allgemeine Problemstellung der Funktionsapproximation erfolgt über einem BANACH-Raum $C$, in dem die Funktion $y = H(x)$ einen Punkt darstellt. BANACH-Räume sind vollständige, lineare und normierte Räume: Für je zwei Punkte $f$ und $g$ eines BANACH-Raumes $C$ und jede reelle Konstanten $a$ gehören auch $f + g$ und $af$ zu $C$, $C$ ist ein *linearer* Raum. Weiterhin existiert eine *Norm* $||f||$ mit folgenden Eigenschaften:

- $||f|| \geq 0$; $||f|| = 0$ gdw. $f = 0$;
- $||af|| = |a|||f||$;
- $||f + g|| \leq ||f|| + ||g||$.

Außerdem konvergiert jede CAUCHY-Folge, also jede Folge $f_n$ von Punkten aus $C$ mit $||f_n - f_m|| \to 0$ für $n, m \to \infty$, gegen ein Element $f$ aus $C$:

$$||f_n - f|| \to 0. \tag{B.1}$$

Da durch die Norm auch eine Metrik auf $C$ charakterisiert werden kann (via $d(f, g) = ||f - g||$) sind BANACH-Räume auch metrische Räume. Die Menge aller *Kugelumgebungen* eines Punktes $f$ (eine Kugelumgebung $K_\epsilon(x)$ ist die Menge aller Punkte $y$ mit $d(x, y) \leq \epsilon$) eines BANACH-Raumes bilden eine Topologie über $C$, wodurch BANACH-Räume auch HAUSDORFFschen Räume darstellen.

Der für die meisten Anwendungen interessanteste BANACH-Raum ist der aller über einer kompakten Teilmenge $A$ eines euklidischen Raumes $R_n$ definierten stetigen reellwertigen Funktionen $S[A]$. Die Definition von Addition von zwei Funktionen und Multiplikation einer Funktion mit einem Skalar erfolgen über die Abbildungen von $A$ in $R$ selbst: $f + g$ ist die Funktion aus $S[A]$, die an der Stelle $x \in A$ den Wert $f(x) + g(x)$ annimmt, und $af$ die Funktion aus $S[A]$, die an der Stelle $x \in A$ den Wert $af(x)$ annimmt. Als Norm kommt i.d.R. die *Supremum*-Norm zum Einsatz:

$$||f|| = \sup_{x \in A} |f(x)|. \tag{B.2}$$

Da $C$ vollständig ist, wird das Supremum auch in $C$ angenommen und kann durch das Maximum über alle $x \in A$ ersetzt werden.

Für die Problemstellung der Approximation in einem BANACH-Raum $C$ wird eine spezielle Teilmenge $\Phi$ von $C$ vorgegeben.

**Definition 8.** Der Punkt $f \in C$ ist *approximierbar* durch die Linearkombinationen

$$P = a_1\phi_1 + a_2\phi_2 + \ldots + a_n\phi_n \tag{B.3}$$

mit $\phi_i \in \Phi$ und $a_i$ reellen Werten, wenn es zu jedem $\epsilon > 0$ solch ein $P$ gibt, für daß $||f - P|| < \epsilon$ ist.

Beispiele für die Mengen $\Phi$ sind etwa die der Potenzfunktionen $1, x, x^2, \ldots$, die zur TAYLOR-Entwicklung von Funktionen führen, oder die der trigonometrischen Funktionen $1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots$, die zur FOURIER-Entwicklung einer Funktion $f$ führen (wobei in letzterem Fall der BANACH-Raum $K$ der additiven Gruppe $R$ modulo $2\pi$ genutzt wird).

Eine von der Approximation unterschiedene Aufgabenstellung ist die der *Interpolation* von Funktionen. Dabei ist die exakte Übereinstimmung von $P$ für eine vorgegebene Menge $c_k = f(x_k)$ von Funktionswerten, den sog. *Stützstellen*, eine zusätzliche Forderung. Ist $\Phi$ hier die Menge von Potenzfunktionen, spricht man auch von einer *Spline*-Interpolation. Deren Einsatzgebiet ist in der Regel die Computergrafik.

Die Fragestellung nach der Auswahl geeigneter Mengen $\Phi$ wird durch das STONE-WEIERSTRASS-Theorem beantwortet. Dabei wird eine Familie $G = \{g\}$ von Funktionen aus $S[A]$ betrachtet, und es werden die $\phi_i$ in Gleichung B.3 durch Ausdrücke der Form $g_1(x)^{n_1} \ldots g_m(x)^{n_m}$ mit beliebigen natürlichen Zahlen $n_i \geq 0$ repräsentiert. Die Funktionen $P$ werden dann zu *Polynomen*

$$P(x) = \sum a_i g_1(x)^{n_1} \ldots g_m(x)^{n_m}. \tag{B.4}$$

**Definition 9.** Die Funktionsfamilie $G$ *separiert* die Punkte von $A$, wenn zu jedem Punktepaar $x_1 \neq x_2$ von $A$ eine Funktion $g \in G$ existiert mit $g(x_1) \neq g(x_2)$.

Es zeigt sich nun, daß die Separierbarkeit von $A$ durch $G$ notwendige und hinreichende Bedingung für die Approximierbarkeit von $A$ durch Polynome aus $G$ ist.

**Theorem 8.** *(*STONE-WEIERSTRASS*) Jede stetige reelle Funktion aus $S[A]$ ist durch Polynome in den Funktionen $g \in G$ dann und nur dann approximierbar, wenn $G$ die Punkte von $A$ separiert.*

Durch das STONE-WEIERSTRASS-Theorem läßt sich leicht zeigen, daß sich jede Funktion auf $K$ durch trigonometrische Polynome approximieren läßt. Dazu beachte man, daß $\sin x$ und $\cos x$ jeden Punkt in $[0, 2\pi)$ separieren, und daß sich

jedes $\phi$ der Form $\sin nx$ bzw. $\cos nx$ als Polynom in $\sin x$ und $\cos x$ darstellen läßt.

Der gegebene Beweis ist jedoch ein *Existenzbeweis*. Die Durchführung einer tatsächlichen Approximation bei gegebenem $G$ erweist sich in der Praxis als recht schwierig. Man stößt hier auf ein vergleichbares Problem wie beim Beweis des KOLMOGOROV-Theorems weiter unten. Außerdem ist anzumerken, daß das STONE-WEIERSTRASS-Theorem sich nicht auf komplexwertige Funktionen übertragen läßt.

# Literaturverzeichnis


[1] A. Abutaleb and M. Kamel, *A genetic algorithm for the estimation of ridges in fingerprints*, IEEE Trans. Image Processing, 8 (1999), pp. 1134–1139.

[2] D. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer, Dordrecht, 1987.

[3] J. Albus, *A theory of cerebellar function*, Mathematical Biosciences, 10 (1971), pp. 25–61.

[4] P. G. Anderson, *The Unit RBF Network: Experiments and preliminary results*, in Proceedings of the International ICSC/IFAC Symposium on Neural Computation (NC'98), M. Heiss, ed., International Computer Science Conventions (ICSC), Canada/Switzerland, September 1998.

[5] P. Andrey and P. Tarroux, *Unsupervised image segmentation using a distributed genetic algorithm*, Pattern Recognition, 27 (1994), pp. 659–673.

[6] S. Aoki and T. Nagao, *Automatic construction of tree-structural image transformation using genetic programming*, in Proc. ICIP99, 1999, pp. I:529–533.

[7] N. Azzabou and L. Likforman-Sulem, *Neural network-based proper names extraction in fax images*, in Proc. ICPR04, 2004, pp. I: 421–424.

[8] S. Baheerathan, F. Albregtsen, and H. Danielsen, *New texture features based on the complexity curve*, Pattern Recognition, 32 (1999), pp. 605–618.

[9] J. Bala, K. DeJong, J. Huang, H. Vafaie, and H. Wechsler, *Visual routine for eye detection using hybrid genetic architectures*, in Proc. ICPR96, vol. C, 1996, pp. 606–610.

[10] G. H. Ball and D. J. Hall, *Isodata, a novel method of data analysis and pattern classification*, tech. rep., Stanford Research Institute, Menlo Park, 1965.

[11] C. L. Begovich and V. E. Kane, *Estimating the number of groups and group membership using simulation cluster analysis*, Pattern Recognition, 15 (1982), pp. 335–342.

[12] J. Bezdek, J. Keller, R. Krisnapuram, and N. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer, August 1999.

[13] S. Bhandarkar, Y. Zhang, and W. Potter, *An edge-detection technique using genetic algorithm-based optimization*, Pattern Recognition, 27 (1994), pp. 1159–1180.

[14] B. Bhanu and S. Lee, *Genetic Learning for Adaptive Image Segmentation*, Kluwer, 1994.

[15] B. Bhanu and Y. Lin, *Genetic algorithm based feature selection for target detection in sar images*, Image and Vision Computing, 21 (2003), pp. 591–608.

[16] E. Biebelmann, M. Köppen, and B. Nickolay, *Practical applications of neural networks in texture analysis*, Neurocomputing, 13 (1996), pp. 261–279.

[17] C. Biernacki, G. Celeux, and G. Govaert, *Assessing a mixture model for clustering with the integrated classification likelihood*, Rapport de recherche 3521, Theme 4, Unite de recherche INRIA Lorraine, http://www.inria.fr, 1997.

[18] G. Boole, *An Investigation of The Laws of Thought on which are founded The Mathematical Theories of Logic and Probabilities*, Dover Publications, Reprinted 1958, 1973.

[19] G. Box, *Evolutionary operation: a method of increasing industrial productivity*, Applied Statistics, 6 (1957), pp. 81–101.

[20] H. Bremermann, *Optimization through evolution and recombination*, in Self-Organizing Systems, M. Yovits, ed., Spartan Books, Washington D.C., 1962.

[21] W. Bricken, *Distinction networks*, in KI-95: Advances in Artificial Intelligence, I. Wachsmuth, C.-R. Rollinger, and W. Brauer, eds., vol. 981 of Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin u.a., 1995, pp. 35–48.

[22] D. E. Brown, C. L. Huntley, and P. J. Garvey, *Clustering of homogeneous subsets*, Pattern Recognition Letters, 12 (1991), pp. 401–408.

[23] H. Caillol, W. Pieczynski, and A. Hillion, *Estimation of fuzzy gaussian mixture and unsupervised statistical image segmentation*, IEEE Trans. Image Processing, 6 (1997), pp. 425–440.

[24] G. Carpenter and S. Grossberg, *Adaptive resonance theory: Stable selforganization of neural recognition codes in response to arbitrary lists of input patterns*, in Proc. 8th Annu. Conf. Cognitive Sci. Soc., 1986, pp. 45–62.

[25] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, and D. Rosen, *Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps*, IEEE Transactions on Neural Networks, 3 (1992), pp. 698–713.

[26] D. Casasent and L. Neiberg, *Classifier and shift-invariant automatic target recognition neural networks*, Neural Networks, 8 (1995), pp. 1117–1129.

[27] G. Celeux and G. Soromenho, *An entropy criterion for assessing the number of clusters in amixture model*, Journal of Classification, 13 (1996), pp. 195–212.

[28] D. Chaudhuri, B. B. Chaudhuri, and C. A. Murthy, *A new split-and-merge clustering technique*, Pattern Recognition Letters, 13 (1992), pp. 399–409.

[29] S. Chen, Y. Horng, and C. Lee, *Document retrieval using fuzzy-valued concept networks*, IEEE Transactions on Systems, Man and Cybernetics B, 31 (2001), pp. 111–118.

[30] R. Chouhan, *Towards a biophysical explanation of the coronal formations obtained in kirlian photography in relation to cancer*, in Proc. of the 3rd Intl. Conf. for Medical and Applied Bioelectrography, Helsinki, Finlandia, 1996, pp. 19–21.

[31] K. K. Chow and M. Liou, *Genetic motion search algorithm for video compression*, IEEE Trans. Circuits and Systems for Video Technology, 3 (1993), pp. 440–445.

[32] A. Cross and E. Hancock, *Recognizing building patterns using matched-filters and genetic search*, ISPRS Journal of Photogrammetry and Remote Sensing, 53 (1998), pp. 95–107.

[33] C. de Stefano, A. della Cioppa, and A. Marcelli, *Character preclassification based on genetic programming*, Pattern Recognition Letters, 23 (2002), pp. 1439–1448.

[34] ——, *Exploiting reliability for dynamic selection of classifiers by means of genetic algorithms*, in Proc. ICDAR03, 2003, pp. 671–675.

[35] D. V. de Ville, M. Nachtegael, D. V. der Weken, E. E. Kerre, W. Philips, and I. Lemahieu, *Noise reduction by fuzzy image filtering*, IEEE Transactions on Fuzzy Systems, 11 (2003), pp. 429 – 436.

[36] R. J. P. DeFigueiredo, *The oi, os, omni and osman networks as best approximations of nonlinear systems under training data constraints*, in Proc. IEEE Int. Symp. Circuits Syst., Seattle, WA, 1996.

[37] J. du Buf, M. Spann, and M. Kardan, *Texture feature performance for image segmentation*, Pattern Recognition, 23 (1990), pp. 291–309.

[38] M. Dubuisson and R. Dubes, *Efficacy of fractal features in segmenting images of natural textures*, Pattern Recognition Letters, 15 (1994), pp. 419–431.

[39] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wiley and Sons, 2 ed., 2001.

[40] *Duden, Etymologie: Herkunftswörterbuch der deutschen Sprache*, Dudenverlag, Mannheim u.a., 2 ed., 1997.

[41] C. Dyer and A. Rosenfeld, *Fourier texture features: Suppression of aperture effects*, IEEE Transaction on Systems, Man, and Cybernetics, 6 (1976), pp. 703–705.

[42] C. Elkan, *The paradoxical success of fuzzy logic*, in Proceedings of the 11th International Conference on Artificial Intelligence (AAAI-93), Washington, D.C., MIT Press, 1993, pp. 698–703.

[43] T. Eltoft and R. J. P. DeFigueiredo, *A new neural network for cluster-detection-and-labeling*, IEEE Transactions on Neural Networks, 9 (1998), pp. 1021–1034.

[44] T. English, *No more lunch: Analysis of sequential search*, in 2004 Congress on Evolutionary Computation, Portland, Oregon, 2004, pp. 227–234.

[45] E.P.Klement, R. Mesiar, and E. Pap, *Triangular Norms*, Kluwer Ac. Pub., Dordrecht, Holland, 2000.

[46] S. E. Fahlman and C. Lebiere, *The cascade-correlation learning architecture*, in Advances in Neural Information Processing Systems 2, D. S. Touretzky, ed., Morgan Kaufmann, 1990.

[47] K. Franke, *The Influence of Physical and Biomechanical Processes on the Ink Trace - Methodological foundations for the forensic analysis of signatures*, PhD thesis, Artifical Instelligence Institute, University of Groningen, The Netherlands, 2005.

[48] K. Franke, J. Ruiz-del-Solar, and M. Köppen, *Soft biometrics: Soft computing for biometric applications*, International Journal of Fuzzy Systems, 4 (2002), pp. 665 – 672.

[49] A. Fraser, *Simulation of genetic systems by automatic digital computers*, Australian Journal of Biological Sciences, 10 (1957), pp. 484–491.

[50] G. Friedman, *Digital simulation of an evolutionary process*, General Systems Yearbook, 4 (1959), pp. 171–184.

[51] A. Gamba, L. Gamberini, G. Palmieri, and R. Sanna, *Further experiments with PAPA*, Nuovo Cimento Suppl., 20 (1961), pp. 221–231.

[52] C. Garcia and M. Delakis, *Convolutional face finder: A neural architecture for fast and robust face detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 26 (2004), pp. 1408–1423.

[53] Z. J. Geng and W. C. Shen, *Fingerprint classification using fuzzy cerebellar model arithmetic computer neural networks*, Journal of Electronic Imaging, 6 (1997), pp. 311 – 318.

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

[55] M. Gong and Y. Yang, *Genetic-based stereo algorithm and disparity map evaluation*, International Journal of Computer Vision, 47 (2002), pp. 63–77.

[56] R. C. Gonzalez and R. E. Woods, *Digital Imaging Processing*, Addison-Wesley, Reading, MA, 1992.

[57] S. Grossberg, *Competitive learning: From interactive activation to adaptive resonance*, Cognitive Science, 11 (1987), pp. 23–63.

[58] S. Grossberg, *A solution of the figure-ground problem for biological vision*, Neural Networks, 6 (1993), pp. 463–483.

[59] M. Gudmundsson, E. Elkwae, and M. Kabuka, *Edge-detection in medical images using a genetic algorithm*, IEEE Trans. Medical Imaging, 17 (1998), pp. 469–474.

[60] M. Hanmandlu, D. Jha, and R. Sharma, *Color image enhancement by fuzzy intensification*, Pattern Recognition Letters, 24 (2003), pp. 81–87.

[61] R. Haralick, K. Shanmugam, and I. Dinstein, *Textural features for image classification*, IEEE Transaction on Systems, Man, and Cybernetics, 3 (1973), pp. 610–621.

[62] S. Hati and S. Sengupta, *Robust camera parameter estimation using genetic algorithm*, Pattern Recognition Letters, 22 (2001), pp. 289–298.

[63] R. Hecht-Nielsen, *Kolmogorov's mapping neural network existence theorem*, in Proceedings of the First International Conference on Neural Networks, vol. III, IEEE Press, New York, 1987, pp. 11–13.

[64] H. J. A. M. Heijmans and J. B. T. M. Roerdink, *Mathematical morphology and its applications to image and signal processing*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1998.

[65] D. Hilbert, *Die Hilbertschen Probleme*, Ostwalds Klassiker der exakten Wissenschaften, Band 252, Verlag Harry Deutsch, Thun und Frankfurt am Main, 1998.

[66] J. H. Holland, *Outline for a logical theory of adaptive systems*, JACM, 3 (1962), pp. 297–314.

[67] ——, *Goal-directed pattern recognition*, in Proceedings of the International Conference on Methodologies of Pattern Recognition, Honolulu, Hawaii, S. Watanabe, ed., 1969, pp. 287–296.

[68] ——, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[69] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.

[70] J. Jacq and C. Roux, *Registration of 3-d images by genetic optimization*, Pattern Recognition Letters, 16 (1995), pp. 823–841.

[71] B. Jeon, J. Jang, and K. Hong, *Road detection in spaceborne sar images using a genetic algorithm*, IEEE Trans. Geoscience and Remote Sensing, 40 (2002), pp. 22–29.

[72] Q. Ji and Y. Zhang, *Camera calibration with genetic algorithms*, IEEE Transactions on Systems, Man and Cybernetics A, 31 (2001), pp. 120–130.

[73] E. Jones, P. Runkle, N. Dasgupta, L. Couchman, and L. Carin, *Genetic algorithm wavelet design for signal classification*, IEEE Trans. Pattern Analysis and Machine Intelligence, 23 (2001), pp. 890–895.

[74] K. D. Jong, *An analysis of the behavior of a class of genetic adaptive systems*, PhD thesis, University of Michigan, Dept. of Computer and Communication Sciences, Ann Arbor, 1975.

[75] H. Juillé and J. P. Pollack, *Co-evolving intertwined spirals*, in Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, L. J. Fogel, P. J. Angeline, and T. Baeck, eds., A Bradford Book, MIT Press, Cambridge, MA, 1995.

[76] S. Kant, T. L. Rao, and P. N. Sundaram, *An automatic and stable clustering algorithm*, Pattern Recognotion Letters, 15 (1994), pp. 543–549.

[77] E. Kim, S. Park, S. Hwang, and H. Kim, *Video sequence segmentation using genetic algorithms*, Pattern Recognition Letters, 23 (2002), pp. 843–863.

[78] S. Kim, D. Kim, and H. Kim, *A recognition of vehicle license plate using a genetic algorithm based segmentation*, in Proc. ICIP96, 1996, p. 17P8.

[79] S. Kirlian and V. Kirlian, *Photography and visual observations by means of high frequency currents*, J.Sci.Appl.Photography, 6 (1964), pp. 397–403.

[80] T. Knudsen, H. Muhammed, and B. Olsen, *A comparison of neuro-fuzzy and traditional image segmentation methods for automated detection of buildings in aerial photos*, in Proc. PCV02, 2002, p. B: 116.

[81] A. N. Kolmogorov, *On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition*, Doklady Akademii Nauk SSSR, 114 (1957), pp. 679–681. (in Russisch).

[82] L. Konikiewicz, *Kirlian photography in theory and clinical applications*, J.Biol.Photogr.Assoc., 45 (1997), pp. 115–134.

[83] M. Köppen, *Some technical remarks on the proof of the no free lunch theorem*, in Proceedings of the Joint Conference on Information Sciences (JCIS 2000), Atlantic City, NJ, 2000, pp. 1020–1024.

[84] ——, *On the training of a kolmogorov network*, in Artificial Neural Networks - ICANN 2002, J. R. Dorronsoro, ed., LNCS 2415, Madrid, Spain, August 2002, 2002, Springer-Verlag Heidelberg, pp. 474–479.

[85] M. Köppen and K. Franke, *Fuzzy morphologies revisited*, in Proceedings of International Workshop on Soft Computing in Industry 99, Muroran, Hokkaido, Japan, 1999, pp. 258–263.

[86] M. Köppen, A. S. Frisch, and T. Sy, *Binary pattern processing framework for perceptual fault detection*, in Proceedings of the 12th Scandinavian Conference on Image Analysis (SCIA2001), I. Austvoll, ed., Bergen, Norway, 2001, pp. 363–370.

[87] M. Köppen and B. Nickolay, *Design of image exploring agent using genetic programming*, Fuzzy Sets and Systems, 103 (1999), pp. 303–315.

[88] M. Köppen, B. Nickolay, and H. Treugut, *Genetic algorithm based heuristic measure for pattern similarity in kirlian photographs*, in Proceedings of the 2001 Workshop on Evolutionary Image Analysis and Signal Processing (EvoIASP2001), Lake Como, Italy, 2001.

[89] M. Köppen, C. Nowack, and G. Rösel, *Pareto-morphology for color image processing*, in The 11th Scandinavian Conference on Image Analysis, B. K. Ersboll and P. Johansen, eds., vol. 1, Kangerlussuaq, Greenland, 1999, pp. 195–202.

[90] M. Köppen and J. Ruiz-del-Solar, *Fuzzy-based texture retrieval*, in Proceedings FUZZ-IEEE'97, Barcelona, Spain, 1997, pp. 471–475.

[91] M. Köppen, J. Ruiz-del-Solar, and P. Soille, *Texture segmentation by biologically-inspired use of neural networks*, in Proceedings of the International ICSC / IFAC Symposium on Neural Computation (NC'98), Vienna, Austria, M. Heiss, ed., ICSC Academic Press, 1998, pp. 267–272.

[92] M. Köppen and P. Soille, *Two-dimensional frameworks for the application of soft computing to image processing*, in Proceedings of International Workshop on Soft Computing in Industry 99, Muroran, Hokkaido, Japan, 1999, pp. 204–209.

[93] M. Köppen, D. Waldöstl, and B. Nickolay, *A system for the automated evaluation of invoices*, in Proceedings of the IAPR Workshop on Document Analysis Systems (DAS'96), Malvern, PA, 1996, pp. 1–20.

[94] ——, *A system for the automated evaluation of invoices*, in Document Analysis Systems II, J. H. Hull and S. L. Taylor, eds., World Scientific, Singapore a.o., 1997, pp. 223–241.

[95] B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[96] ——, *Fuzzy Engineering*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[97] J. R. Koza, *A genetic approach to the truck backer upper problem and the inter-twined spirals problem*, in Proceedings of the International Joint Conference on Neural Networks (IJCNN'92), Baltimore, vol. IV, IEEE Press, Juni 1992, pp. 310–318.

[98] ——, *Genetic Programming – On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, London MA, 1992.

[99] ——, *Genetic Programming II – Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, London MA, 1994.

[100] P. Kraft, N. Harvey, and S. Marshall, *Parallel genetic algorithms in the optimization of morphological filters: A general design tool*, Journal of Electronic Imaging, 6 (1997), pp. 504–516.

[101] L. Kuncheva, *An application of owa operators to the aggregation of multiple classification decisions*, in The Ordered Weighted Averaging operators. Theory and Applications., R. R. Yager and J. Kacprzyk, eds., Kluwer Academic Publishers, 1997, pp. 330–343.

[102] M. Kupinski and M. Anastasio, *Multiobjective genetic optimization of diagnostic classifiers with implications for generating receiver operating characteristic curves*, IEEE Trans. Medical Imaging, 18 (1999), pp. 675–685.

[103] K. J. Lang and M. J. Witbrock, *Learning to tell two spirals apart*, in Proceedings of the 1988 Connectionist Summer Schools, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, eds., Morgan Kaufmann, 1988, pp. 52–59.

[104] W. Langdon and R. Poli, *Fitness causes bloat*, in Proceedings of the 2nd On-line World Conference on Soft Computing in Engineering, Design and Manufacturing, 1997.

[105] B. Lazzerini and F. Marcelloni, *A linguistic fuzzy recogniser of off-line handwritten characters*, Pattern Recognition Letters, 21 (2000), pp. 319–327.

[106] Y. Lee and R. Park, *A surface-based approach to 3-d object recognition using a mean field annealing neural network*, Pattern Recognition, 35 (2002), pp. 299–316.

[107] C. Li and R. Chiao, *Multiresolution genetic clustering algorithm for texture segmentation*, Image and Vision Computing, 21 (2003), pp. 955–966.

[108] S. Li, W. Xu, H. Wang, and N. Zheng, *A novel fast motion estimation method based on genetic algorithm*, in Proc. ICIP99, 1999, pp. I:66–69.

[109] Y. Linde, A. Buzo, and R. M. Gray, *An algorithm for vector quantizer design*, IEEE Transactions Computer, COM-28 (1980), pp. 84–95.

[110] E. Littmann and H. Ritter, *Adaptive color segmentation: A comparison of neural and statistical-methods*, IEEE Trans. Neural Networks, 8 (1997), pp. 175–185.

[111] L. Lohmann, *Bildanalysesystem zur robusten Erkennung von Kennzeichen an Fahrzeugen*, PhD thesis, Technische Universität Berlin, 1999.

[112] D. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, 60 (2004), pp. 91–110.

[113] L. Luo, D. Clewer, N. Canagarajah, and D. Bull, *Genetic stereo matching using complex conjugate wavelet pyramids*, in Proc. ICIP01, 2001, pp. II: 153–156.

[114] W. Ma and B. Manjunath, *A comparison of wavelet features for texture annotation*, in Proceedings ICIP95, 1995, pp. II: 256–259.

[115] D. Maio, D. Maltoni, and S. Razzi, *Topological clustering of maps using a genetic algorithm*, Pattern Recognition Letters, 16 (1995), pp. 89–96.

[116] A. Malanda-Trigueros, F. Calleja-Garde, and A. Figueiras-Vidal, *A genetic algorithm for noisy channel color quantization design*, in Proc. ICIP01, 2001, pp. I: 898–901.

[117] A. Malaviya and L. Peters, *Fuzzy handwriting description language: Fohdel*, Pattern Recognition, 33 (2000), pp. 119–131.

[118] P. Mandel, *Energetische Terminalpunkt-Diagnose*, Synthesis-Verlag, 1983. (in German).

[119] A. Marazzi, P. Gamba, A. Mecocci, and A. Semboloni, *Automatic selection of the number of clusters in multidimensional data problems*, in Proceedings of the International Conference on Image Processing, vol. 3, NY, USA, 1996, IEEE, pp. 631–634.

[120] D. Marr, *Vision*, MIT Press, 1981.

[121] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, *"Neural-Gas" network for vector quantization and its application to time-series prediction*, IEEE Transaction on Neural Networks, 4 (1993), pp. 558–569.

[122] A. Martinez and J. Vitria, *Learning mixture models using a genetic version of the em algorithm*, Pattern Recognition Letters, 21 (2000), pp. 759–769.

[123] I. Matalas, *A new set of multiscale texture features based on b-spline image approximation*, in Proceedings ICPR'96, Vienna, Austria, 1996, p. B94.11.

[124] D. McNeill and P. Freiberger, *Fuzzy Logic: Die „unscharfe" Logik erobert die Technik*, Knaur, 1994.

[125] R. Mercer, J. Barron, A. Bruen, and D. Cheng, *Fuzzy points: algebra and application*, Pattern Recognition, 35 (2002), pp. 1153–1166.

[126] M. Minsky, *The Society of Mind*, Simon & Schuster, New York u.a., 1985.

[127] M. L. Minsky and S. A. Papert, *Perceptrons – Expanded Edition*, MIT Press, 1988.

[128] S. Mitra, C. Murthy, and M. Kundu, *Technique for fractal image compression using genetic algorithm*, IEEE Trans. Image Processing, 7 (1998), pp. 586–593.

[129] M. Nachtegael, D. V. der Weken, D. V. de Ville, and E. Kerre, *Fuzzy Filters for Image Processing*, vol. 122 of Studies in Fuzziness and Soft Computing, Physica Verlag, Heidelberg, 2003.

[130] T. Nagao and S. Masunaga, *Automatic construction of image transformation processes using genetic algorithm*, in Proc. ICIP96, 1996, p. 19P3.

[131] N. Negroponte, *Being Digital*, Alfred A. Knopf, New York, 1995.

[132] B. Nickolay, *Überwacht lernendes Bildauswertungssystem zur Erkennung von Oberflächenfehlern*, Reihe Produktionstechnik Berlin, Carl Hanser Verlag, München, Wien, 1990.

[133] I. Oh, J. Lee, and B. Moon, *Hybrid genetic algorithms for feature selection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 26 (2004), pp. 1424–1437.

[134] J. Ohya and F. Kishino, *Detecting facial expressions from face images using a genetic algorithm*, in Proc. ICPR96, vol. C, 1996, pp. 649–653.

[135] L. Oliveira, R. Sabourin, F. Bortolozzi, and C. Suen, *Feature selection using multi-objective genetic algorithms for handwritten digit recognition*, in Proc. ICPR02, 2002, pp. I: 568–571.

[136] Y. Omura, *Acupuncture, infra-red thermography and kirlian photography*, Acupunct.Electrother.Res., 2 (1977), pp. 43–86.

[137] J. Pehek, H. Kyler, and D. Faust, *Image modulation in corona discharge photography*, Science, 194 (1976), pp. 263–270.

[138] A. Pentland, *From pixels to predicates*, in Ablex, 1986.

[139] F. Pernkopf and D. Bouchaffra, *Genetic-based em algorithm for learning gaussian mixture models*, IEEE Trans. Pattern Analysis and Machine Intelligence, 27 (2005), pp. 1344–1348.

[140] L. Pessoa and P. Maragos, *Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition*, Pattern Recognition, 33 (2000), pp. 945–960.

[141] O. Pichler, A. Teuner, and B. Hosticka, *A comparison of texture feature-extraction using adaptive gabor filtering, pyramidal and tree-structured wavelet transforms*, Pattern Recognition, 29 (1996), pp. 733–742.

[142] M. Pietikainen and A. Rosenfeld, *Edge based texture measures*, IEEE Transaction on Systems, Man, and Cybernetics, 12 (1982), pp. 585–594.

[143] R. Poli and W. Langdon, *A new schemata theorem for genetic programming with one-point crossover and point mutation*, Tech. Rep. CSRP–97–3, The University of Birmingham, 1997.

[144] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 65 (1958), pp. 386–408.

[145] M. Rosenblum, Y. Yacoob, and L. Davis, *Human expression recognition from motion using a radial basis function network architecture*, IEEE Trans. Neural Networks, 7 (1996), pp. 1121–1138.

[146] A. Rosenfeld, *The fuzzy geometry of image subsets*, Pattern Recognition Letters, 2 (1984), pp. 311–317.

[147] ——, *Fuzzy plane geometry: Triangles*, Pattern Recognition Letters, 15 (1994), pp. 1261–1264.

[148] D. Rumelhart and J. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I, II*, MIT Press, Cambridge MA, 1986.

[149] F. Samadzadegan, A. Azizi, M. Hahn, and C. Lucas, *Automatic 3d object recognition and reconstruction based on neuro-fuzzy modelling*, ISPRS Journal of Photogrammetry and Remote Sensing, 59 (2005), pp. 255–277.

[150] D. Saupe and M. Ruhl, *Evolutionary fractal image compression*, in Proceedings IEEE International Conference on Image Processing (ICIP), vol. 1, 1996, pp. 129 – 132.

[151] P. Scheunders, *A genetic c-means clustering-algorithm applied to color image quantization*, Pattern Recognition, 30 (1997), pp. 859–866.

[152] T. Seijnowski and C. Rosenberg, *NETtalk: A parallel network that learns to read aloud*, Tech. Rep. JHU/EECS-86/01, Johns Hopkins University, Januar 1986.

[153] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.

[154] ——, *Image Analysis and Mathematical Morphology 2: Theoretical Advances*, Academic Press, London, 1988.

[155] C. Shieh, H. Huang, F. Wang, and J. Pan, *Genetic watermarking based on transform-domain techniques*, Pattern Recognition, 37 (2004), pp. 555–565.

[156] H. Shyu and J. Leou, *Detection and concealment of transmission errors in mpeg-2 images–a genetic algorithm approach*, IEEE Trans. Circuits and Systems for Video Technology, 9 (1999), p. 937.

[157] M. Shyu and J. Leou, *A genetic algorithm approach to color image-enhancement*, Pattern Recognition, 31 (1998), pp. 871–880.

[158] M. Singh, A. Chatterjee, and S. Chaudhury, *Matching structural shape descriptions using genetic algorithms*, Pattern Recognition, 30 (1997), pp. 1451–1462.

[159] K. Sirlantzis and M. Fairhurst, *Optimisation of multiple classifier systems using genetic algorithms*, in Proc. ICIP01, 2001, pp. I: 1094–1097.

[160] J. Sirosh, *An algorithm for unsupervised identification of nonlinear manifolds with applications to cortical self-organization*, in Proc. of the 5th Annual Symposium on Neural Computation, Mai 1998.

[161] P. Soille, *Morphologische Bildverarbeitung*, Springer Verlag, Berlin u.a., 1998.

[162] G. Spencer-Brown, *Laws of Form*, Cognizer Connection, Portland, OR, 1994.

[163] D. Sprecher, *On the structure of continuous functions of several variables*, Transcations of the American Mathematical Society, 115 (1965), pp. 340–355.

[164] ——, *A representation theorem for continuous functions of several variables*, Proceedings of the American Mathematical Society, 16 (1965), pp. 200–203.

[165] C. Stephens and H. Waelbroeck, *Schemata evolution and building blocks*, Evolutionary Computation, 7 (1999), pp. 109–124.

[166] M. Sugeno, *Fuzzy Control*, Nikkan Kogyo Shimbun-sha, 1988. (in Japanisch).

[167] G. Syswerda, *Uniform crossover in genetic algorithms*, in Proceedings of the 3th International Conference on Genetic Algorithms, San Mateo, CA, 1989, Morgan Kaufmann, pp. 2–9.

[168] M. Takezawa, H. Honda, M. Hasegawa, and H. Kitajima, *A genetic-algorithm based quantization method for fractal image coding*, in Proc. ICIP99, 1999, pp. I:458–461.

[169] W. Tao, J. Tian, and J. Liu, *Image segmentation by three-level thresholding based on maximum fuzzy entropy and genetic algorithm*, Pattern Recognition Letters, 24 (2003), pp. 3069–3078.

[170] A. Toet and W. Hajema, *Genetic contour matching*, Pattern Recognition Letters, 16 (1995), pp. 849–856.

[171] H. Treugut, *Kirlian Fotographie: Reliabilität der Energetischen Terminalpunktdiagnose (ETD) nach Mandel bei gesunden Probanden*, Forsch.Komplementärmed., 4 (1997), pp. 210–217. (in German).

[172] ——, *Kirlian Fotographie: Reliabilität der Energetischen Terminalpunktdiagnose (ETD) nach Mandel bei Kranken*, Forsch.Komplementärmed., 5 (1998), pp. 224–229. (in German).

[173] P. Tsang and Z. Yu, *Genetic algorithm for model-based matching of projected images of three-dimensional objects*, IEE Proceedings-Vision Image and Signal Processing, 150 (2003), pp. 351–359.

[174] J. Udupa and G. Grevera, *Go digital, go fuzzy*, Pattern Recognition Letters, 23 (2002), pp. 743–754.

[175] S. Ullman, *Visual routines*, Cognition, 18 (1984), pp. 97–156.

[176] F. J. Varela, *Principles of Biological Autonomy*, North–Holland, New York, Oxford, 1979.

[177] A. Verikas, K. Malmqvist, and L. Bergman, *Color image segmentation by modular neural-network*, Pattern Recognition Letters, 18 (1997), pp. 173–185.

[178] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features*, in Proc. CVPR01, 2001, pp. I:511–518.

[179] W. von der Gablentz, M. Köppen, and E. Dimitriadou, *Robust clustering by evolutionary computation*, in Proceedings of the 5th Online World Conference on Soft Computing in Industrial Applications (CD-ROM), 2000.

[180] R. S. Wallace and T. Kanade, *Finding natural clusters having minimum description length*, in Proceedings of the 10th International Conference on Pattern Recognition, vol. 1, Los Alamitos, CA, USA, 1990, IEEE Comput. S. Press, pp. 438–442.

[181] R. Wang, C. Lin, and J. Lin, *Image hiding by optimal lsb substitution and genetic algorithm*, Pattern Recognition, 34 (2001), pp. 671–683.

[182] Y. Wang and N. Funakubo, *Detection of geometric shapes by the combination of genetic algorithm and subpixel accuracy*, in Proc. ICPR96, vol. D, 1996, pp. 535–539.

[183] S. Watanabe, *Une explication mathematique du classement d'objects*, in Information and Prediction in Science, Dockx and Bernays, eds., Academic Press, New York, 1965.

[184] ——, *Pattern Recognition: Human and Mechanical*, John Wiley and Sons, New York u.a., 1985.

[185] D. Weenink, *Category ART: A variation on adaptive resonance theory neural networks*, in Proceedings of the Institute of Phonetic Sciences (IFA Proceedings) 21(1997), University of Amsterdam, 1997, pp. 117–129.

[186] J. Weszka, C. Dyer, and A. Rosenfeld, *A comparative study of texture measures for terrain classification*, IEEE Transaction on Systems, Man, and Cybernetics, 6 (1976), pp. 269–286.

[187] J. Weszka and A. Rosenfeld, *An application of texture analysis to materials inspection*, Pattern Recognition, 8 (1976), pp. 195–200.

[188] D. Whitley, *A genetic algorithm tutorial*, Statistics and Computing, 4 (1994), pp. 65–85.

[189] B. Widrow, *Generalization and information storage in networks of adaline neurons*, in Self-Organizing Systems, G. Yovitts, ed., Spartan Books, Washington D.C., 1962.

[190] C. L. Wilson, G. T. Candela, and C. I. Watson, *Neural-network fingerprint classification*, Journal Artificial Neural Networks, 1 (1994), pp. 203 – 228.

[191] D. Wolpert, *The existence of a priori distinctions between learning algorithms*, Neural Computation, 7 (1996), pp. 1391–1420.

[192] ——, *The lack of a priori distinctions between learning algorithms*, Neural Computation, 7 (1996), pp. 1341–1390.

[193] D. H. Wolpert and W. G. Macready, *No free lunch theorems for search*, Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, 6. Februar 1995.

[194] ——, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, 1 (1997), pp. 67–82.

[195] L. Xu, *Bayesian ying-yang machine, clusterong and number of clusters*, Pattern Recognition Letters, 18 (1997), pp. 1167–1178.

[196] L. Xu, A. Krzyzak, and E. Oja, *Rival penalized competitive learning for clustering analysis, rbf net, and curve detection*, IEEE Transactions on Neural Networks, 4 (1993), pp. 636–649.

[197] R. Yager, *On ordered weighted averaging aggregation operators in multicriteria decision making*, IEEE Transaction on System, Man & Cybernetics, 18 (1988), pp. 183–190.

[198] S. Yamany, M. Ahmed, and A. Farag, *A new genetic-based technique for matching 3-d curves and surfaces*, Pattern Recognition, 32 (1999), pp. 1817–1820.

[199] P. Yin, *A new circle/ellipse detector using genetic algorithms*, Pattern Recognition Letters, 20 (1999), pp. 731–740.

[200] P.-Y. Yin and L.-H. Chen, *A new non-iterative approach for clustering*, Pattern Recognition Letters, 15 (1994), pp. 125–133.

[201] I. Yoda, K. Yamamoto, and H. Yamada, *Automatic acquisition of hierarchical mathematical morphology procedures by genetic algorithms*, Image and Vision Computing, 17 (1999), pp. 749–760.

[202] L. A. Zadeh, *Fuzzy sets*, Information and Control, 8 (1965), pp. 338–353.

[203] ——, *Fuzzy logic, neural networks and soft computing*, Communications of the ACM, 37 (1994), pp. 77–84.

[204] ——, *The birth and evolution of fuzzy logic (FL), soft computing (SC) and computing with words (CW): a personal perspective*, in Fuzzy Sets, Fuzzy Logic and Fuzzy Systems - Selected papers by Lotfi A. Zadeh, G. J. Klir and B. Yuan, eds., vol. 6 of Advances in Fuzzy Systems – Applications and Theory, 1996, pp. 811–819.