DIPLOMARBEIT

Konzeption und Realisierung einer Evolutionsstrategie zum Abgleich eines anthropometrisch-kinematischen Menschmodells mit dreidimensionalen Sensordaten

Thomas Hemmer

Oktober 1997

Autor: Thomas Hemmer
Pariserstr. 281
67663 Kaiserslautern

Betreuung: Prof. Dr. rer. nat. E. von Puttkamer

Dipl. Inform. Guido Hansen

Eingereicht am: 15. Oktober 1997

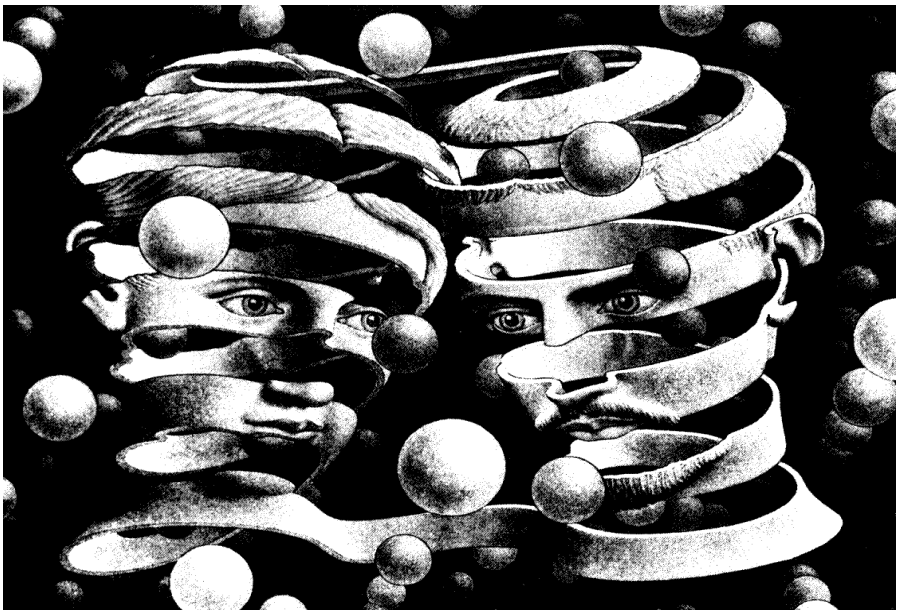
Anschrift: Universität Kaiserslautern
Fachbereich Informatik
AG Robotik und Prozeßrechentechnik
Prof. Dr. rer. nat. E. von Puttkamer
67653 Kaiserslautern

Erklärung

Hiermit erkläre ich, Thomas Hemmer, die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Kaiserslautern, 15. Oktober 1997.

Thomas Hemmer.



“Computer science differs from physics in that it is not actually a science. It does not study natural objects. Neither is it, as one might think, mathematics; although it does use mathematical reasoning pretty extensively. Rather, computer science ist like engineering - it is all about getting something to do something, rather than just dealing with abstractions.”

Richard P. Feynman

Inhalt

1 Einführung

Die vorliegende Diplomarbeit wurde im Rahmen des Projektes TOPAS (Touchless Perception and Anthropometric Adaption System) der AG Robotik und Prozeßrechentechnik an der Universität Kaiserslautern erstellt. Das Ziel dieses Forschungsprojektes besteht in der Entwicklung eines automatischen Menschvermessungssystems.

Zu diesem Zweck wurde ein 3D-Datenerfassungssystem entwickelt, das die individuelle Körpergeometrie einer Person zu erfassen vermag. Diese Daten werden durch eine Software weiterverarbeitet, die es ermöglicht, individuelle Körpermaßdaten für weitere Anwendungen zur Verfügung stellen zu können, beispielsweise zur Herstellung individuell angepaßter Produkte.

Es besteht aus einem Datenerfassungsgerät, das die individuelle Körpergeometrie[1] einer Person erfaßt und einer Software, die diese Daten aufbereitet und in eine Form bringt, um sie für sekundäre Anwendungen adäquat zur Verfügung stellen zu können. Hauptanwendung personenbezogener Individualdaten ist die individuelle Herstellung von Produkten.

Die Datenaufbereitung besteht im Abgleich der vom Meßgerät gelieferten Rohdaten an ein rechnerbasiertes anthropometrisch-kinematisches Datenmodell. Dieses Modell bildet das menschliche Erscheinungsbild und dessen Bewegungsfunktionalität (Kinematik) nach und kann zum einen realistisch Bewegungsabläufe des Menschen simulieren und zum anderen individuelle Körpergeometrien nachbilden.

Der Modellabgleich bringt durch die Daten-Transformation der Sensorrohdaten in Anthropometriedaten einen Informationsgewinn[2] und damit einen wesentlich gesteigerten Nutzeneffekt. Durch die hinzugewonnenen semantischen Inhalte wird eine sekundäre Weiterverarbeitung er-

1. Es wird die Körperoberfläche des Probanden vermessen.
2. In dem Sinne, als daß den transformierten Daten eine höhere Aussagekraft zugeschrieben werden kann, als den Rohdaten.

möglicht. Dieser Schritt stellt damit auch eine erhebliche Datenreduktion dar, die viele (aussageschwache) Daten auf wenige informationstragende Daten abbildet.

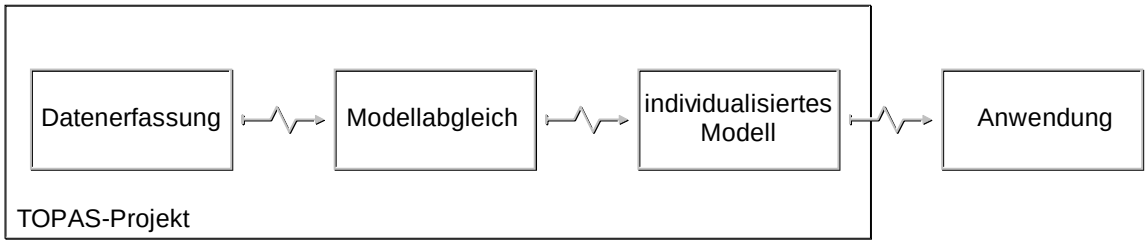


Abbildung 1.1: *Schematische Darstellung des Datenflusses*

1.1 Individualproduktion

Meyers Enzyklopädischem Lexikon definiert den Wortstamm "Individual-" als ein *Bestimmungswort von Zusammensetzungen mit der Bedeutung "das Einzelwesen betreffend, Einzel..."* [MEY74]. Dementsprechend ist die *Individualisierung* ein gesellschaftlicher Trend, der den Schwerpunkt der Sichtweise weg von Ansprüchen und Bedürfnissen der Menschen als Teil einer Gruppe (die sich im allgemeinen durch gemeinsame Eigenschaften definiert) zu den charakteristischen Einzel-Eigenschaften einer Person verschiebt.[1]

War es zur Zeit unserer Eltern noch Mode, Gruppenzugehörigkeit und Ideologie durch Uniformität kenntlich zu machen, so ist es heute "trendy", durch ein ausgefallenes Äußeres und verbale Provokation abzugrenzen und aufzufallen. Der Schwerpunkt liegt auf *Unterschied* und nicht auf *Gemeinsamkeit*.

In der Industrie bedeutet *Individualproduktion* die Herstellung eines Produkts, das individuell auf eine bestimmte Person zugeschnitten ist. Das Verfahren beruht auf einem zweischrittigen Vorgang, der aus Datenerfassung von Individualdaten und der Erzeugung eines parametrisierten Produktes besteht. Im Gegensatz zur Massenproduktion, die identische Waren millionenfach erstellt, ist ein Individualprodukt in der Regel ein Unikat.

Unsere Gesellschaft befindet sich an der Schwelle der Massenproduktion zur automatisierten Individualproduktion. Zwar wird die Herstellung von Massenartikeln nicht verschwinden, weil es für eine Reihe von Produkten prinzipiell keinen Sinn macht individuell gefertigt zu werden (Bsp: Kühlschränke). Es gibt aber einen weiten Bereich von Anwendungen, in denen maßgefertigte Güter einen deutlichen Nutzenvorteil gegenüber Massenware bringt[2].

Waren bisher Produkte, die ausschließlich in Einzelstücken hergestellt werden konnten (Maßbekleidung) für einen exklusiven Kundenkreis reserviert, so werden sie durch eine sehr weitgehende Automatisierung preiswerter und somit für größere Bevölkerungsschichten erschwinglich.

1. Der Duden definiert *individualisieren* als *"bei der Darstellung, Charakterisierung eines Gegenstandes oder Person die eigentümlichen Eigenschaften herausarbeiten"*.
2. Am Beispiel der Automobilindustrie erkennt man diesen Trend recht gut erkennen: War früher ein Ford Escort wie der andere, so kann man heute bei der Bestellung eines Neuwagens eine im Vergleich ernorme Auswahl aus einer Reihe von Wagenformen (*Coupé*, *Kombi* ...) , Attributen (*Farbe*, *Reifentyp*) und Sonderzubehör treffen.

Die Vorteile der IndividualProduktion liegen auf der Hand:

- Produkte können auf individuelle Bedürfnisse an personenbezogener Eigenschaften genauestens abgestimmt werden
- Individuellen Wünschen kann entsprochen werden

Als Folgerung kann damit das individuelle Bedürfnis nach Abgrenzung befriedigt werden.

Die automatisierte Individualproduktion wird grundsätzlich positive Konsequenzen auf die Wirtschaft haben[1]. Da die Dauer von der Bestellung bis zur Lieferung eine entscheidende Rolle an der Kundenzufriedenheit hat, ist die Nähe des Produktionsstandortes zum Kunden ein wichtiges Kriterium. Das stärkt die inländische Industrie und gibt neue Impulse für den Arbeitsmarkt.

Die Lagerhaltung der Verkaufsstellen reduziert sich auf ein Minimum. Wurde Massenware in großen Stückzahlen vom Großhändler geordert und stand u.U. bis zum Schlußverkauf in den Regalen, so führt die Individualproduktion zwangsläufig dazu, daß in Filialen nur Verkaufsmuster präsentiert werden. Durch Techniken wie der virtuellen Anprobe von Kleidungsstücken wird das Kauferlebnis des Kunden intensiviert. Nach erfolgter Bestellung erfolgt die Lieferung der Ware zum Kunden nach Hause per Paketzustellung.

Beispiel Medizin
Vermessung des Patienten mit wissensbankbasierter Diagnosesysteme können den Arzt in der Diagnostik unterstützen oder Krankheitsverläufe diagnostizieren.

Beispiel Automobilindustrie

Personenbezogene Geometriedaten des Fahrers ermöglichen die Herstellung von physiologischen ergonomischen Sitzen, die sich der Körperform optimal anpassen und die korrekte Körperhaltung unterstützen.

Automatisch generierte Anthropometriedatenbanken helfen bei der Erhebung von statistischem Material, gerade im Bereich Ergonomie im Fahrzeuginneren.

Beispiel Textilindustrie

Automatische Erfassung von Individualmaßen ermöglicht die Produktion von Maßbekleidung, die einen optimalen Tragekomfort ermöglicht.

1. Und entwickelt damit in Folge auch politische und kulturelle Auswirkungen.

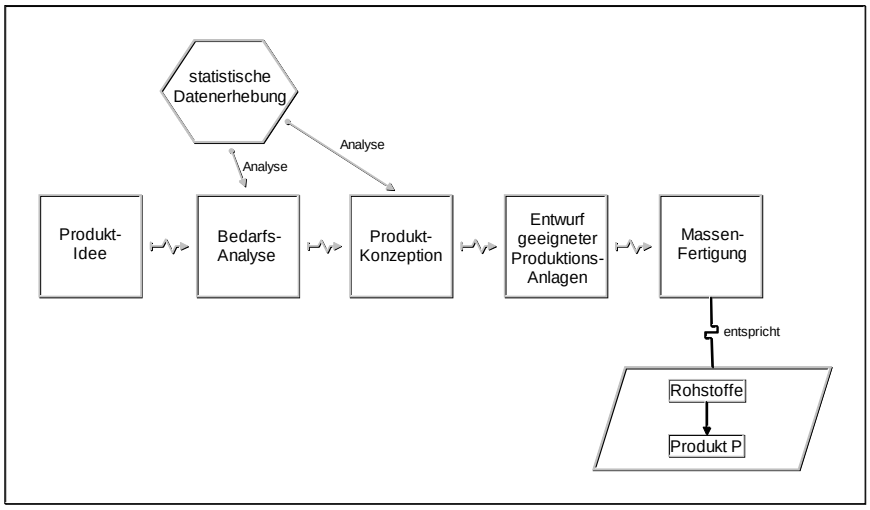


Abbildung 1.2: *Entstehungszyklus Massenprodukt*

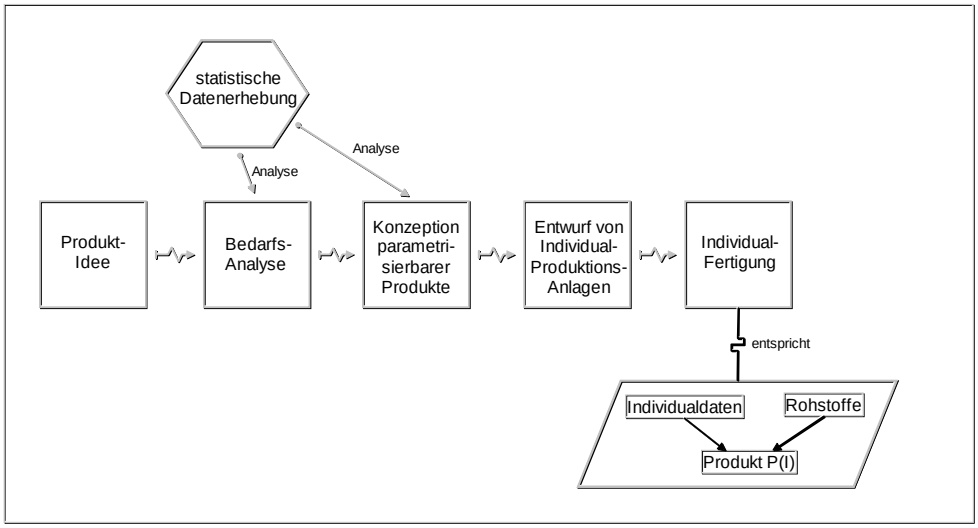


Abbildung 1.3: *Entstehungszyklus Individualprodukt*

1.2 TOPAS-Meßsystem

Der TOPAS-Scanner ist ein System zur automatischen 3D-Datenerfassung der menschlichen Geometrie zum Zwecke der Erzeugung anthropometrischer Daten und wurde im Verlauf einer Diplomarbeit am Lehrstuhl für Robotik und Prozeßrechentechnik der Universität Kaiserslautern entwickelt [HAN96]. Er liefert Rohdaten in Form von 3D-Scanpunkten, die einen wesentlichen Anteil der menschlichen Geometrie beschreiben.

Durch industrienahe Konzeption erfüllt er folgende Anforderungen:

- Aufgrund des optischen Verfahrens erfolgen keinerlei gesundheitliche Belastungen der zu vermessenden Person
- Die Auflösung liegt im Millimeterbereich. Dadurch wird kann eine hochqualitative Übereinstimmung mit dem Modell erreicht werden.
- Besonderen Wert wurde auf die extrem schnelle Erfassung eines kompletten Datensatzes innerhalb von 1,2 Sekunden gelegt. Dadurch werden Bewegungsunsicherheiten der Probanden nahezu ausgeschlossen.
- Durch die Verwendung von Standardkomponenten und einen überschaubaren Hardwareaufwand erzielt das System einen günstiges Preis/Leistungsverhältnis.
- Schließlich sind die geringen Abmessungen gut geeignet um den problemlosen Einsatz im Verkaufsbereich zu zu garantieren (Das System ist nur geringfügig größer als eine Umkleidekabine)
- Damit ist erstmals überhaupt ein System vorhanden, das den wirtschaftlichen Einsatz der 3D-Vermessungstechnik im kommerziellen Umfeld erlaubt.

Abbildung 1.4 stellt eine Fotografie des Erfassungssystems dar.

Abbildung 1.4: *Der Scanner des TOPAS- Projektes*

1.2.1 Funktionsprinzip des Scanners

Das Funktionsprinzip des Scanners basiert auf dem Lichtschnittverfahren. Bei diesem Verfahren wird eine Lichtquelle (in diesem Fall: Laser) mittels einer Optik auf eine Zeile fokussiert, die das Objekt beleuchtet. Eine mit einem Basisabstand a davon entfernte Kamera fotografiert das reflektierte Licht, aus dem sich nach dem Triangulationsverfahren 3D-Punkte berechnen lassen.

Die Kamera bildet zusammen mit der Lichtquelle einen Scankopf, der auf einer Linearführung angebracht ist. Im Verlauf der Datenerfassung wird diese Sensorik gleichmäßig von oben nach unten gefahren. Auf diese Weise kann die Komplette Ansicht des Probanden erfaßt werden.

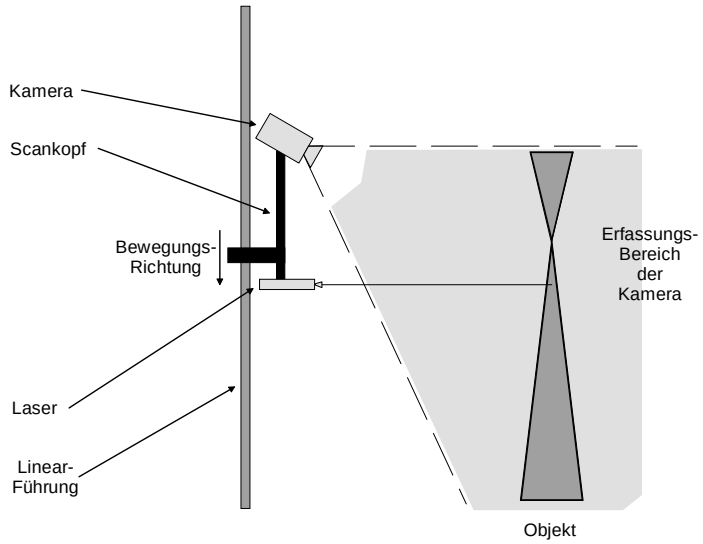


Abbildung 1.5: *Prinzipielle Funktionsweise eines Lichtschnittverfahrens*

1.3 RAMSIS-Menschmodell

Ein Modell ist eine auf die relevanten Aspekte (bzgl. einer Anwendung) reduzierte Darstellung eines Objektes. Dieses Objekt kann ein Ding der realen Welt oder eine Idee sein.

Ein anthropometrisches Menschmodell ist somit das Modell eines Menschen (im allgemeinen), welches bezüglich der Geometrie korrekt Menschen modellhaft nachbildet.

Das von der Firma TecMath entwickelte Modell RAMSIS[1] ist ein anthropo-kinematisches Menschmodell, das ursprünglich zu Ergonomiestudien in der Automobilindustrie konzipiert wurde. In Erweiterung der Modellierung der menschlichen Erscheinung bildet es zusätzlich kinematische Fähigkeiten nach. RAMSIS kann damit komplexe Bewegungsabläufe simulieren.

das zusätzlich kinematische Abläufe des menschlichen Bewegungssystems nachbildet und somit die Simulation von komplexen Bewegungsabläufen ermöglicht. Ursprünglich ist es zu Ergonomiestudien in der Automobilindustrie konzipiert.

RAMSIS unterstützt die automatische, datenbankbasierte und statistisch fundierte Generierung einer Körpermaßtypologie (ca. 90 verschiedene Typen) aufgrund von drei bestimmenden Maßen (Körpergröße, Korpulenz und Alter).

Die Modellstruktur, die intern auf ca. 1200 Parametern basiert, erlaubt es das Modell prinzipiell in feiner Granularität zu individualisieren. Aufgrund dieser Eigenschaft ist das Modell prinzipiell zu Abgleich mit dreidimensionalen Sensordaten gut geeignet.

1. Rechnergestütztes anthropologisch-mathematisches System zur Insassen-Simulation

Es wird zwischen dem *inneren Modell*, welches die für die kinematische Simulation notwendigen Knochen- und Gelenkstrukturen bildet, und dem *äußeren Modell*, welches durch Hautmodellierung eine Oberflächenstruktur und damit ein *Aussehen* schafft, unterschieden.

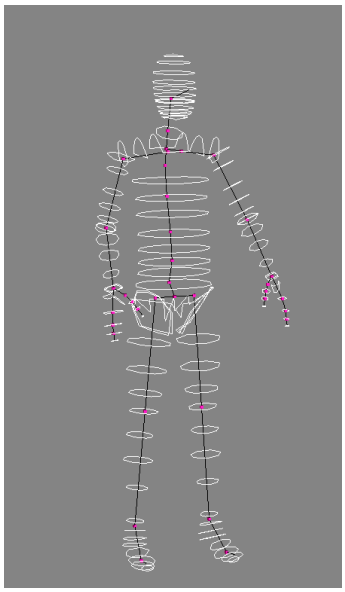


Abbildung 1.6: *Grundstruktur des RAMSIS*

Kapitel 3 wird sich eingehend mit den spezifischen Eigenschaften dieses Menschmodells beschäftigen.

1.4 Aufgabenstellung

Im Rahmen dieser Arbeit sollte ein Algorithmus konzipiert und implementiert werden, der das RAMSIS-Modell vollautomatisch mit der vom Scanner gelieferten Datenmenge abgleicht.

Dazu waren folgende Rahmenbedingungen gegeben:

- Der Prozeß sollte mit Hilfe eines Evolutionsalgorithmus realisiert werden. Zur Berechnung eines Abstandsmaßes von Modell zu den vom Scanner erfaßten Daten sollte ein Voxelraum genutzt werden, der eigens für diesen Zweck implementiert wurde [KUB97].
- Den besonderen Anforderungen des Abgleichs des RAMSIS-Menschmodell mit den Scan-Daten sollte Rechnung getragen werden. Ist prinzipbedingt durch die Art des Meßverfahrens nur die äußere Kontur des Probanden zu erfassen, so ist auch nur ein direkter Abgleich mit der RAMSIS-Haut möglich. Damit ist die Anpassung der inneren Strukturen nur indirekt möglich.

Im Sinne dieser Rahmenbedingungen Vor diesem Hintergrund wurde von Anfang an die Entwicklung eines mehrstufigen Algorithmus geplant, der die diese Punkte berücksichtigt.

- Als Basisplattform war ein Mircrosoft-Windows-basiertes System bereitgestellt. Der Code war plattformübergreifend zu realisieren. Die Implementierung geschah auf einer Windows-NT 2-Prozessormaschine.
- Als Implementierungssprache war die Programmiersprache C++ vorgesehen, die obigen Punkt unterstützt. Eine Visual-C++-Entwicklungsumgebung war zu diesem Zweck bereitgestellt. [STR92]
- Das Programm ist ein Teil eines größeren Softwareprojekts. Deshalb waren die Programmschnittstellen zu konzipieren um damit ein problemloses Zusammenarbeiten mit bereits bestehender Software zu ermöglichen (RAMSIS-Editor *mEdit*)

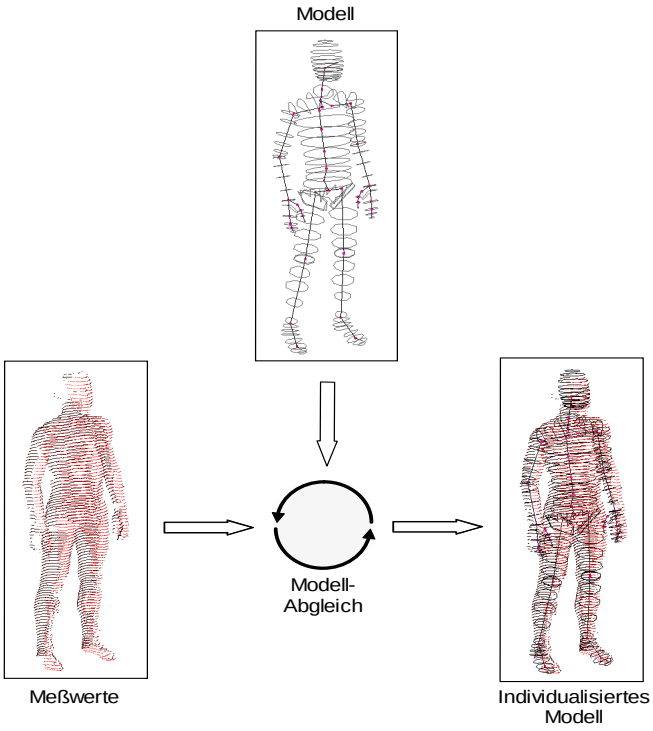


Abbildung 1.7: *Prinzipskizze des Abgleichsverfahrens*

1.5 Gliederung dieser Arbeit

Kapitel 2 beschäftigt sich mit einer Einführung in Optimierungsprobleme und den Grundlagen von Evolutionsalgorithmen.

In Kapitel 3 wird das RAMSIS-Menschmodell genauer vorgestellt und detailliert besprochen.

Kapitel 4 stellt schließlich die Konzeption und Implementierung des Adaptionsalgorithmus vor und in Kapitel 5 erfolgt die Diskussion der Abgleichsergebnisse und ein Ausblick auf Verbesserungen des Verfahrens.

Kapitel 6 stellt eine Zusammenfassung der Ergebnisse dieser Arbeit dar.

2 Evolutionsalgorithmen (EA)

Motiviert durch die erstaunliche Leistungsfähigkeit natürlicher Lebensformen bzgl. bestimmter Aufgabenstellungen versuchen viele verschiedene Zweige der Wissenschaft die grundlegenden Zusammenhänge zu analysieren und diese Erkenntnisse durch Nachbildung in künstlichen Systemen für eigene Zwecke nutzbar zu machen.

Besonders interessant erscheinen in einigen Bereichen der Informatik die Methoden des sog. *Softcomputing* zu sein (insbesondere neuronale Netze und EA). Beiden Disziplinen ist gemein, daß sie intensiv mit dem Begriff *Lernen*[1] zusammenhängen.

Vergleicht man mit dem natürlichen Vorbild, so findet hier in natürlichen neuronalen Netzen kurzzeitiges Lernen statt. Die Evolution stellt hingegen einen Lernvorgang dar, der nur über sehr lange Zeiträume ("Generationen") stattfindet. Beiden Prinzipien ist gemein, daß im Sinne einer Optimierung eine Adaption eines Systems auf Umwelteinflüsse (externe Einflüsse) stattfindet. Weitere Gemeinsamkeiten bestehen darin auch bis zu einem gewissen Grade fehlerhafte Eingaben sinnvoll verarbeiten zu können.

Ein entscheidender Vorteil der Softcomputing-Verfahren gegenüber klassischen Konzepten zur Problemlösung liegt in ihrer Eigenschaft als intelligente Lösungsmethoden begründet. Ist bei letzteren explizit ein Lösungsweg des speziellen Problems zu definieren, so basieren erstere Techniken darauf, nach und nach *Erfahrungen* zu sammeln, also externe und interne Systeminformationen über die Zeit hinweg zu nutzen, und sich damit als rückgekoppeltes System (*Kybernetik*[2]) sukzessiv an die Lösung heranzuarbeiten.

Da viele schwierige Probleme die Eigenschaft besitzen, daß kein expliziter Lösungsweg bekannt ist, aber für vom System vorgeschlagene Lösungen eine qualitative *Bewertung* möglich ist, bieten sich als Ansatzpunkt Softcomputing-Techniken an.

Dieses Kapitel gibt einen Überblick über die Familie der Evolutionsalgorithmen, die in dieser Form in der einschlägigen Literatur vermißt wird.

---

1. *Lernen* ist "*der Erwerb, die Aneignung von Kenntnissen, die Änderung von Denken, Einstellungen und Verhaltensweisen"* [BRO88].
2. *"Cybernetics is the study of systems which can be mapped using loops in the network defining the flow of information. Systems of automatic control will of necessity use at least one loop of information flow providing feedback"* [SCR90].

Zunächst wird allgemein der Begriff *Optimierungsproblem* eingeführt und ein Vergleich verschiedener gängiger Lösungsverfahren gegeben. Im Anschluß daran wird detailliert auf Evolutionsalgorithmen eingegangen und versucht, die Vorzüge von EA gegenüber anderen Verfahren herauszuarbeiten.

2.1 Einführung in Optimierungsprobleme

Nach [BRO88] ist ein *Problem "eine schwierige Aufgabe, komplizierte Fragestellung"*. Das *Problemlösen* wird als *"das Auffinden eines vorher nicht bekannten Weges von einem gegebenen Anfangszustand zu einem gewünschten und mehr oder weniger bekannten Endzustand"* definiert.

Viele Probleme lassen sich in einer Form fassen, die versuchen, ein Element einer wie auch immer beschaffenen Menge zu finden, welches durch ein besonderes Merkmal ausgezeichnet ist oder in einer gewissen Hinsicht *besser* ist als die anderen Elemente.

Äquivalent ist es möglich, dies durch dies durch eine Funktion *f* beschreiben, die jedem Element der Menge einen (ggf. reellen) Wert zuordnet. Anhand dieses Funktionswertes *f(x)* kann dann entschieden werden, inwieweit das Mengenelement *x* einem bestimmten *Gütekriterium* entspricht. Diese Funktion stellt also eine Bewertung eines Elementes dar.

Formell ist das Finden des Elements, welches den größten (oder alternativ: kleinsten) Funktionswert der Menge darstellt, ein *Optimierungsproblem*.

Wird nach einem Minimum der Funktion *f* gesucht, so wird von einem *Minimierungsproblem* gesprochen; *f* heißt dann *Kostenfunktion* oder *Energiefunktion*. Die Maximumssuche ist ein *Maximierungsproblem* und in diesem Falle heißt *f Fitneßfunktion*. Im folgenden wird oft allgemein von einem *Optimierungsproblem* gesprochen, von *f* auch als *Objektfunktion*. Im Sprachgebrauch der Evolutionsalgorithmen wird meistens für Minimierungs- und Maximierungsprobleme die Bezeichnung Fitneßfunktion verwendet und somit dem Begriff Objektfunktion gleichgesetzt. Da die Maximierung von *f* der Minimierung von *-f* entspricht besteht prinzipiell keine wesentlicher Unterschied.

Prinzipiell wird zwischen verschiedenen Verfahren zum Auffinden dieses ausgezeichneten Elements unterschieden. Exakte mathematische Methoden berechnen es in einem oder mehreren, aber endlich vielen Schritten (sog. *Ein-* bzw. *Mehrschrittverfahren*). Werden für diese Aufgabe unendlich viele Schritte benötigt, d.h. wird das Optimum während einer (endlichen) Berechnung nur bis auf einen bestimmten Fehler angenähert, so spricht man von einem *approximativen Verfahren*.

Sind die Eigenschaften der Funktion *f* nicht in ausreichendem Maße bekannt, oder sind sie nicht geeignet ein streng mathematisches Verfahren anzuwenden, so bleibt nur die Möglichkeit, den Grundbereich großflächig abzusuchen, wobei die Topologie der *Landschaft*[1] genutzt werden kann, um Hinweise, die Aufschluß auf die Lage des Optimums geben, zu sammeln. Solche Verfahren werden dann *Suchverfahren* genannt.

Gut geeignet zur Lösung solcher Aufgaben sind Verfahren, deren Rechenweg nicht deterministisch nur von den Eingabegrößen abhängig ist, sondern zusätzlich von (berechneten) Zufalls-

1. Bei Funktionen, die über einem kontinuierlichen Grundbereich definiert sind, spricht man oft von *Funktionslandschaft*, *Fitneßlandschaft* oder nur *Landschaft*.

zahlen abhängt. Diese Verfahren heißen *stochastische Algorithmen (SA)*. Ein Charakteristikum dieser Klasse ist die Eigenschaft, daß verschiedene Programmläufe mit gleichen Eingabewerten zu unterschiedlichen Ergebnissen führen können. (Die Zufallswerte werden nicht den Eingaben zugerechnet). Stochastische Algorithmen eignen sich sowohl zur Lösung von Aufgaben (Problemstellungen), deren expliziter Lösungsweg nicht vorgegeben bzw. bekannt ist und deshalb zufällig generiert werden soll, als auch für statistische Lösungsansätze rechenintensiver Anwendungen (bspw. Simulation von Vielteilchensystemen), deren Berechnung im Detail mangels Rechenleistung nicht mehr durchführbar ist.

Folgende Abbildung zeigt eine einfache Funktion, deren Maximum gesucht ist. Zur Lösung dieser einfachen Aufgabenstellung können sicherlich alle genannten Verfahren angewendet werden.

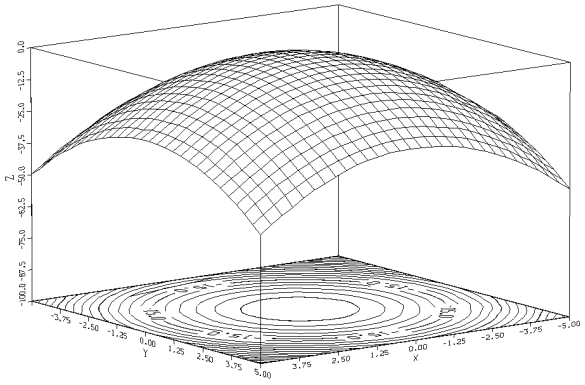


Abbildung 2.1: *Einfache Fitneßfunktion*

Im allgemeinen werden zum Lösen “einfacher” Optimierungsprobleme keine Suchverfahren verwendet. Meist sind diese mathematisch gut faßbar und somit können deren Lösungen durch mathematische Rechenverfahren wesentlich schneller und effizienter berechnet werden. Suchverfahren werden bei Problemen eines hohen Komplexitätsgrades sinnvoll eingesetzt, wenn sie mit streng mathematischen Mitteln nicht mehr handhabbar sind.

Es gibt eine Fülle verschiedener Suchstrategien, die alle mehr oder weniger an zusätzlichem Wissen über das jeweilige Problem erfordern und somit die Klasse der mit diesem Verfahren lösbaren Probleme einschränken. Am unteren Ende der Skala gibt es extrem spezielle Verfahren, die sich auf ein ganz bestimmtes Problem oder gar einen Parametersatz beschränken; am anderen gibt es weitgehend universelle Lösungsverfahren, welche die Klasse der verfahrensbedingt lösbaren Probleme nicht oder nur wenig einschränken.

Ein junger Ansatz zum Suchen von Lösungen von allgemeinen Optimierungsproblemen sind *Evolutionsalgorithmen (EA)*, deren Grundlagen in den 60er Jahren von verschiedenen Wissenschaftlern unabhängig voneinander gelegt wurden. EA ist der Oberbegriff für zwei historisch unterschiedliche Ansätze:

Die Gruppe um J. H. Holland entwickelte die *genetischen Algorithmen* an der Universität von Michigan zum Nachweis der prinzipiellen Funktionsweise evolutionsalgorithmischer Prinzipien.

Die Grundlagen der *Evolutionsstrategien* wurden von den damaligen Studenten Rechenberg und Schwefel an der TU Berlin gelegt, die sie zur Optimierung von aerodynamischen Körpern im Windkanal nutzen wollten.

Wegen ihrer Universalität und Robustheit werden die EA als einer der vielversprechendsten Ansätzen zur Lösung von allgemeinen Suchproblemen angesehen.

Im folgenden wird ein kurzer Überblick über verschiedenste Einsatzgebiete gegeben, in denen heute Evolutionsalgorithmen eingesetzt werden:

Automatisches Programmieren

Durch die Evolution von Programmen, zB. in LISP-Notation, werden schnellere und einfachere Algorithmen zur Lösung spezieller Probleme erzeugt.

Maschinelles Lernen

In weiten Bereichen der KI werden EA eingesetzt um Klassifizierungssysteme zu trainieren, die automatisch Objekte in verschiedene Kategorien einteilen und bewerten können.

Bei neuronalen Netzwerken stellen Evolutionsalgorithmen die Gewichte einzelner Neuronen ein oder optimieren die Topologie dieser Netze.

Ökonomie Die Möglichkeiten von EA werden im Rahmen der Modellierung und Simulation von Produkt-Innovationsprozessen genutzt. Auch in der Entwicklung von *bidding strategies* und bei der Untersuchung der Ausbreitung von Märkten sind diese Strategien einsetzbar.

Ökologie In der Ökologie werden Jäger-Beute-Modelle simuliert; es wird versucht die Koevolution von Parasiten und ihren Trägern zu verstehen und man untersucht soziales Verhalten in Populationen primitiver Arten.

Evolutionsstrategien stellen ein universelles Verfahren zur Optimierung vielfältiger Problemstellungen in zahlreichen Anwendungen dar. Gerade im Bereich Softcomputing, in dem lange Zeit neuronale Netze dominierten, verschiebt sich mehr und mehr die Aufmerksamkeit auf evolutionsstrategische Konzepte.

Durch die breite Aufmerksamkeit, die diese Klasse von Optimierungsverfahren in den letzten Jahren erhalten hat, ist nun die Wissenschaft darauf bedacht, ihre grundlegenden Eigenschaften genauer zu erforschen und neue Verfahrensverbesserungen herauszuarbeiten.

Mit Blick auf die von Darwin entwickelte *Theorie der natürlichen Evolution der Arten* [DAR44], deren Methodik für diese Algorithmen Pate stand, werden weitere weitere Details dieser Theorie auf Programmebene transponiert. Aber auch neuere Konzepte finden bei den Informatikern Anklang. Dabei versucht man beispielsweise den Lernprozeß, den die vielen Arten der Tier- und Pflanzenwelt durch die über Jahrmillionen dauernde Evolution erfahren hat, mit dem “aktiven” Lernen einzelner Lebewesen, welche sie in nur einem Lebenszyklus erfahren, in einem Modell zu verbinden:

Many people have drawn analogies between learning and evolution as two adaptive processes, one taking place during the lifetime of an organism and the other taking place over the evolutionary history of life on earth. To what extent do these processes interact?....

...The well-known "Lamarckian hypothesis" states that traits acquired during the lifetime of an organism can be transmitted genetically to the organism's offspring. Lamarck's hypothesis is generally interpreted as referring to acquired physical traits (such as physical defects due to environmental toxins), but something learned during an organisms lifetime also can be thought of as a type of acquired trait. Thus, a Lamarckian view might hold that learned knowledge can guide evolution directly by being passed on genetically to the next generation.[1]

([MIT96],Seite 87)

Das universelle Problemlösungsverfahren im strengen Sinne existiert nicht. Es existieren nämlich durchaus Problemstellungen, die prinzipiell keine effizientere Suche erlauben als das *Random Search*[2]*-Verfahren* bzw. das sequentielle Durchsuchen des Grundbereichs.

Folgendes Beispiel einer Funktion *f* soll dies verdeutlichen. Die Fitneßlandschaft von *f* ist völlig eben und es existiert nur ein einzelner Punkt, dessen Wert von denen aller anderen abweicht.

Die Topologie der Fitneßlandschaft gibt keinerlei Aufschluß über die Lage des Optimums. Die Eigenschaften dieser Funktion sind völlig lokal: selbst in lokaler Nachbarschaft des Optimums ist dieses nicht ermittelbar.

Für diese Problemstellung existiert nachweislich kein Optimierungsverfahren, welches das Optimum *P* schneller als die oben angegebenen *trivialen* Suchverfahren zu finden vermag.

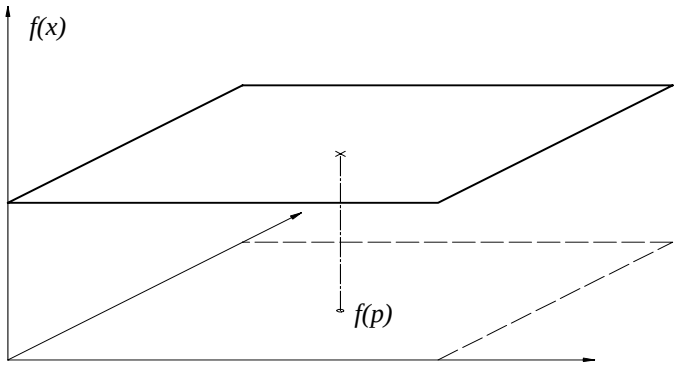


Abbildung 2.2: *Beispiel einer Funktion, deren Form keinerlei Information über die Lage des Optimums (hier: Minimum) preisgibt, das sog. "golf course problem".*

Ganz ähnliche Eigenschaften haben *fraktale Landschaften*, die sich zumindest in lokalen Bereichen chaotisch verhalten und deshalb auch hier kein Optimum zu finden ist, sofern dieses im chaotischen Bereich liegt.

1. Der Lamarkismus ist ein Erklärungsmodell für die Vererbung von Merkmalen der Arten in der Natur, welches mit den darwinistischen Theorien konkurriert.
2. *Random Search* bezeichnet ein nichtstrategisches, stochastisches Durchsuchen des Grundbereichs.

2.2 Mathematische Optimierungsprobleme

In diesem Abschnitt erfolgt eine formale Definition von Optimierungsproblemen. Im Anschluß daran werden das Transportproblem (TP) und das Travelling Salesman Problem (TSP) als Beispiele aufgeführt, anhand deren auch in einem späteren Abschnitt die verschiedene Datenrepräsentationen diskutiert werden.

Definition 2.1: Optimierungsproblem (OP)

Ein *Optimierungsproblem* ist ein Tripel $(G, f, \{g_i\}_{0 \le g \le n})$. Dabei ist G eine Menge (Grundbereich, Suchraum), $f: G \to \mathbb{R}$ die Bewertung und $g_i: G \to \mathbb{R}$ sind die Restriktionsfunktionen.

Die Lösung eines OP besteht nun in der Bestimmung des Minimums $x^* \in G$, so daß die folgenden Bedingungen erfüllt sind:

$$\text{i)} \quad \forall x \in G \quad f(x^*) \le f(x) \qquad \textbf{(2.2.1)}$$

$$\text{ii)} \quad \forall x \in G \;\; \forall i \le n \;\; g_i(x) \ge 0 \qquad \textbf{(2.2.2)}$$

Ist $G \subset \mathbb{N}^k$, so wird von einem *diskreten Optimierungsproblem* gesprochen; ist $G \subset \mathbb{R}^k$, so heißt $(G, f, \{g_i\}_{0 \le g \le n})$ reellwertiges Optimierungsproblem.

Die Funktionenmenge $\{g_i\}_{0 \le g \le n}$ sind die *Restriktionen* des Optimierungsproblems.

In praktischen Fällen wird gefordert, daß G kompakt ist; d.h. G ist beschränkt und abgeschlossen.

Um ein Gefühl für einige diskrete und reellwertige Optimierungsprobleme zu vermitteln seinen im folgenden einige Beispiele genannt.

2.2.1 Transportproblem

Die Problemstellung besteht im Aufstellen eines minimalen Kostenplans, um ein Produkt P von einer Auswahl von Quellen zu einer Menge von Senken unter folgenden Restriktionen zu transportieren: Jede einzelne Quelle hat eine Menge des Produkts vorrätig, die nicht unterschritten werden darf. Ebenso darf keiner Senke mehr geliefert werden, als sie bestellt hat.

Es existiert eine Funktion, die zu je einer Quelle und einer Senke angibt, wie hoch die Kosten sind, eine Menge *x* des Produkts zwischen diesen Punkten zu transportieren. Nach Möglichkeit soll gleichzeitig die Gesamtnachfrage gedeckt werden, das Gesamtangebot abtransportiert werden und die Gesamttransportkosten minimal gehalten werden.

Definition: *Transportproblem* (TP)

Im folgenden seinen S die Menge der Quellen, D die Menge der Senken; $source(s \in S)$ gebe den Materialvorrat der Quelle s an; $dest(d \in D)$ gebe den Bedarf der Senke d an.

Weiterhin bezeichne $x_{i,j}$ die von Quelle i nach Senke j transportierte Menge des Produkts; $f_{i,j}(x)$ bezeichne die Kostenfunktion, welche die Kosten für den Transport einer Produktmenge x von Quelle i nach Senke j angebe.

Damit formuliert sich das *allgemeine Transportproblem*[1] so:

$$minimize \sum_{i \in S} \sum_{j \in D} f_{i,j}(x_{i,j}) \qquad \textbf{(2.2.3)}$$

unter den beiden Nebenbedingungen:

$$\text{i)} \quad \forall i \in S : \sum_{j \in D} x_{i,j} \leq source(i) \qquad \textbf{(2.2.4)}$$

$$\text{ii)} \quad \forall j \in D : \sum_{i \in S} x_{i,j} \geq dest(j) \qquad \textbf{(2.2.5)}$$

Forderung (2.2.4) gibt an, daß keine größere Menge des Produkts von den Quellen abtransportiert wird, als diese herstellen. Äquivalent fordert (2.2.5), daß die Gesamtlieferung die Gesamtnachfrage nicht überschreitet.

Gilt weiterhin $\sum_{i \in S} source(i) = \sum_{j \in D} dest(j)$,

d.h. Gesamtnachfrage und Gesamtangebot sind ausgeglichen, so heißt dieses TP auch *ausgeglichenes Transportproblem* (*balanced TP*); die Nebenbedingungen (2.2.4) und (2.2.5) wandeln sich damit in Gleichungen.

Das TP heißt *linear*, falls für $f_{i,j}(x)$ gilt:

$$f_{i,j}(x) = x_{i,j}\, cost(i,j) \qquad \textbf{(2.2.6)}$$

Dabei gebe $cost(i,j)$ die Einheitstransportkosten von Quelle i nach Senke j an.

Gilt (2.2.6), so heißt das TP *lineares Transportproblem (LTP)*.

Sind die Wertebereiche von $source(s \in S)$ und $dest(d \in D)$ diskret, so gilt dies auch für die Lösungen $x_{i,j}$.

Während das LTP recht gut untersucht ist und es viele gute Lösungsmethoden gibt, entzieht sich die allgemeine Form des TP bisher jeglichen Lösungsansatzes.

Ein Beispiel:

1. *minimize* gibt dabei an, daß nach dem Minimum gesucht wird.

Die Kostenfunktion $cost(i, j)$ des LTP läßt sich als Matrix schreiben:

cost	d1	d2	d3	d4
s1	10	0	20	11
s2	12	7	9	20
s3	0	14	16	18

Mit $source = (5, 25, 5)$ und $dest = (5, 15, 15, 10)$ ergibt sich folgende optimale Lösung $M = [x_{i,j}]_{i \in S, j \in D}$:

x	d1	d2	d3	d4
s1	0	5	0	10
s2	0	10	15	0
s3	5	0	0	0

Damit ergeben sich die minimalen Transportkosten zu $f(x) = 10 \cdot 11 + 10 \cdot 7 + 15 \cdot 9 = 322$.

2.2.2 Traveling Salesman Problem (TSP)

Gegeben sei eine Anzahl von Städten in der reellen Ebene und der Abstand jeweils zweier Städte zueinander. Das Ziel ist, eine Rundreise zu finden, bei der jede Stadt genau einmal besucht wird. Dabei ist die Gesamtlänge der Strecke zu minimieren.

Obwohl das Problem einfach zu beschreiben ist genügt schon eine mittlere Anzahl von Städten um die Minimasuche extrem komplex zu gestalten[1]. Das Problem TSP hat sich als NP-vollständig erwiesen.

Definition 2.2: Traveling Salesman Problem

Sei $C = [c_{i,j}]_{i,j \in \{1 \ldots n\}}$ eine Kostenmatrix mit $c_{i,j} = c_{j,i}$ für alle $i, j \in \{1 \ldots n\}$.
Dann nennt sich das Problem eine Permutation π der Menge $\{1 \ldots n\}$ zu finden, so daß

$$\sum_{i=1}^{n} c_{\pi(i),\, \pi((i \bmod n) + 1)} \qquad \textbf{(2.2.7)}$$

minimal ist, das *Traveling Salesman Problem*.[2]

Anschaulich gesprochen entspricht obige Formel die Summe der Wegstecken der einzelnen Städteverbindungen einer Rundreise der Städte; die Gesamtstrecke soll dabei minimal werden.

Als einfache Repräsentation bietet sich beispielsweise eine Pfaddarstellung an, z.B `a=(ABC-DEFGHIKL)` für eine Tour von 11 Städten. Siehe auch Anhang B zu Details der Datenrepräsentation beim TSP.

1. Bei einer Aufgabe mit 150 Städten ergeben sich schon $149! = 3,81^{260}$ mögliche Rundreisen. (150 Touren sind in der Pfadrepräsentation äquivalent, deshalb “nur” 149! .
2. Genaugenommen unterscheidet man zwischen dem symmetrischen TSP, welches dadurch gekennzeichnet ist, daß die Reisekosten von einer Stadt a nach einer Stadt b denjenigen Kosten der Rückreise gleichen. In der allgemeinen Form dieses Problems wird diese Restriktion fallengelassen (Man stelle sich eine Landschaft vor: a liegt lauf einem Berg, b im Tal)

Als Beispiel seinen die folgenden Städte in der Ebene gegeben, für die eine minimale Tour gefunden werden soll. Im Bild rechts sieht man eine mögliche Tour: .

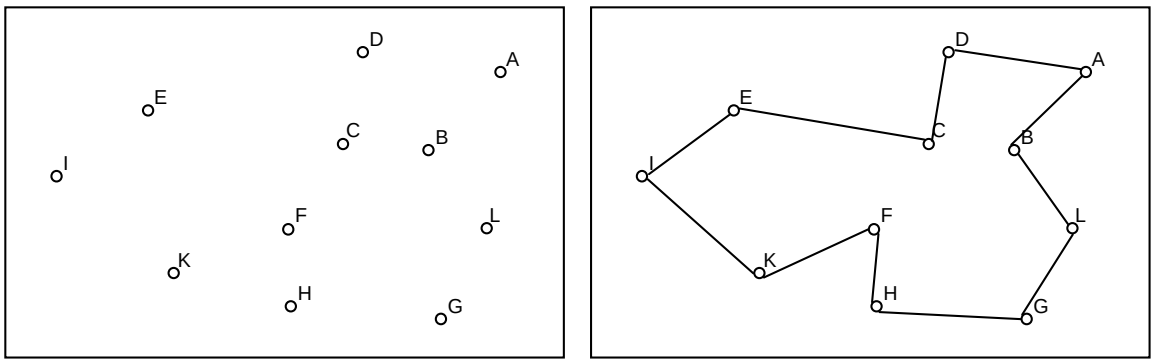


Abbildung 2.3: *Eine gegebene Menge von Städten und eine mögliche Rundreise.*

2.3 Topologien von Fitneßlandschaften

Die Komplexität der Fitneßfunktion stellt ein entscheidendes Kriterium dar, ein geeignetes Optimierungsverfahren auszuwählen. Funktionen mit einem globalen Optimum können sicherlich mittels mathematischen Ein- und Mehrschrittverfahren gelöst werden. Stark zerklüftete Fitneßlandschaften können nur mit geeigneten Suchverfahren gelöst werden. Häufig ist das Aussehen und mathematische Eigenschaften der Fitneßfunktion nicht bekannt. Abschnitt 2.4 geht auf ausgewählte klassische Optimierungsverfahren genauer ein.

Abbildung 2.4 zeigt zwei zweidimensionale Grundgebiete mit Fitneßfunktionen (siehe [BÄC93]). Während die Funktion links im Bild offensichtlich nur ein Minimum besitzt und demnach dessen Optimum beispielsweise mit dem später noch näher beschriebenen Newtonverfahren bestimmt werden kann, ist diese Vorgehensweise bei der Funktion rechts im Bild nicht durchführbar. Es wird sich zeigen, daß Evolutionsalgorithmen für diese Fälle gut geeignet sind.

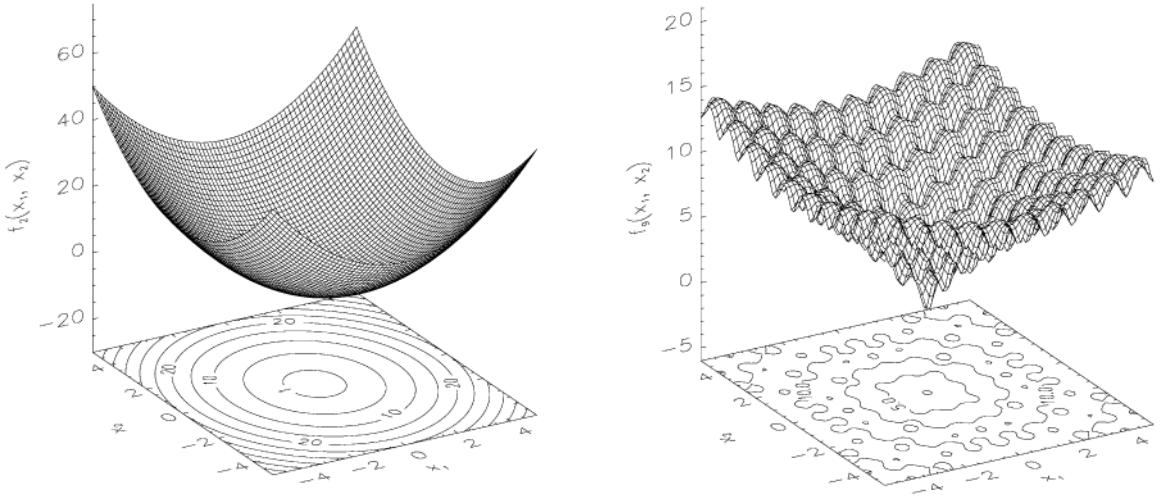


Abbildung 2.4: *Zwei Suchräume unterschiedlicher Komplexität mit zugehöriger Fitnessfunktion.*

Letztendlich gibt es Funktionen, deren globales Optimum prinzipiell nicht oder nur durch einen (unwahrscheinlichen) zufallsbedingten Sprung an den entsprechenden Punkt im Suchraum erreicht werden kann. Funktionen dieser Art haben streng lokale Eigenschaften, d.h. für einen gegebenen Punkt $x \in G$ des Grundgebiets kann zwar der Wert $f(x)$ bestimmt werden, jedoch lassen daraus sich i.a. keine Informationen über Nachbarpunkte oder ganze Umgebungen $U_\varepsilon(x)$ herleiten (siehe *Golf Course Problem* in Abschnitt 2.1, Abbildung 2.2).

Bei Funktionen, die solche Forderungen nicht erfüllen, wird der Suchalgorithmus wahllos umherirren und das Optimum allenfalls zufällig erreichen. Mag dies beim kompakten diskreten Optimierungsproblem rein theoretisch gesehen noch verkraftbar sein[1], so wird diese *Blackbox-Eigenschaft*[2] von *f* bei reellwertigen, also überabzählbar großen, kontinuierlichen Suchräumen zur Fallgrube; das Treffen des Optimums wird wahrscheinlichkeitstheoretisch zum unmöglichen Ereignis. Daraus ergibt sich, daß die Suche prinzipiell fehlschlagen muß. Siehe dazu auch Abschnitt 2.7 .

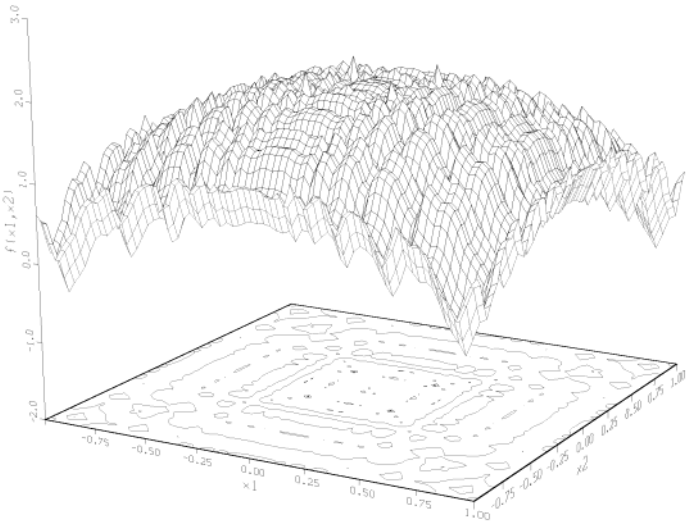


Abbildung 2.5: *Die fraktale Weierstraß-Mandelbrot-Funktion*

Als weiteres Beispiel sei nun noch eine Landschaft der schwierigsten Kategorie beschrieben. Abbildung 2.5 zeigt die fraktale Weierstraß-Mandelbrot-Funktion. Fraktale Funktionen besitzen die Eigenschaft, beliebig hochfrequente Schwingungsanteile zu besitzen, d.h. ihre Fouriertransformation belegt ein beliebiges Spektrum. Damit ist es prinzipiell nicht möglich, im nichtglatten Bereich[3] der Funktion ein Optimum zu bestimmen, denn beliebig kleine Umgebungen eines Optimums können wieder beliebig zerstückelt sein. (s. [SCH96])

1. In kompakten diskreten (also endlichen) Suchräumen kann man sich damit trösten, daß das Optimum in endlicher Zeit durch serielles Durchprobieren aller Kombinationen gefunden werden kann, was in praktischen Fällen allerdings in unakzeptabel langen Laufzeiten resultiert.
2. Als *Blackbox* seien Funktionen beschrieben, die nur durch ihr Ein-/Ausgabeverhalten charakterisiert werden (können). Anhaltspunkte über ihre *Funktionsweise* sind a priori unbekannt.
3. Im Allgemeinen haben chaotische Funktionen auch Bereiche mit gewissen Glätteeigenschaften. Liegt das Optimum dort, so ist es lokalisierbar. Siehe dazu auch die Anmerkung zum Konvergenzbeweis der (1+1)-ES (Abschnitt 2.7).

2.4 Klassische und neuere Optimierungsmethoden

In diesem Abschnitt seien einige Optimierungsverfahren besprochen und erörtert. Zunächst folgt eine grobe Klassifizierung, dann werden einige dieser Verfahren vorgestellt.

- *Exakte Verfahren*. Diese Verfahren finden (berechnen) das Optimum exakt und nach einer endlichen Anzahl von Abarbeitungsschritten. (Ein- und Mehrschrittverfahren)
- *Approximative Algorithmen*, die dem Optimum sukzessiv immer näher kommen aber letztendlich immer ein gewisser Abstand, der sogenannte Approximationsfehler, bleibt.
- *Suchverfahren*, deren Suche im Grundgebiet auf heuristischen Methoden basiert. Es ist hier keine klare Trennung zu den approximativen Verfahren zu ziehen, prinzipiell gehören die Suchverfahren auch zur letzteren Kategorie. Diese Unterscheidung ist bewußt gewählt worden, um zu verdeutlichen, daß den approximativen Verfahren strenge mathematische Grundlagen zugrunde liegen, während Suchverfahren eher ein heuristischer Ansatz zugrunde liegt.

Die stochastischen Evolutionsalgorithmen gehören damit zur letzten Kategorie.

Einen Überblick über eine Auswahl klassischer Optimierungsmethoden und neuerer stochastischer Methoden[1] gibt folgende Aufstellung:

Simplexverfahren (SV)

Das SV setzt eine lineare stetige Zielfunktion $f: G \subset \mathbb{R}^k \rightarrow \mathbb{R}$ voraus, die auf einem kompakten Suchraum *G* definiert ist. Dies garantiert die Existenz des eindeutigen globalen Optimums auf dem Rand des Suchraumes.

Gradientenverfahren (GV)

Das GV benutzt zur Suche des Optimums die Richtung das steilsten Anstiegs, bzw. des stärksten Gefälles. Notwendigerweise wird ein lokales Optimum gefunden, sofern es existiert. Als Vorraussetzung zur Anwendbarkeit des Verfahrens wird gefordert, daß die Zielfunktion stetig ist und alle partiellen Ableitungen von *f* existieren.

Newtonverfahren (NV)

Das NV arbeitet ähnlich dem Gradientenverfahren und nutzt darüber hinaus zur Bestimmung der Iterationsschrittweite die Krümmung der Optimierungsfunktion *f*, also die zweiten partiellen Ableitungen. Die Voraussetzungen sind ähnlich restriktiv wie beim GV, auch hier werden Stetigkeit und Existenz aller partiellen Ableitungen gefordert.

Quasi-Newtonverfahren* (QNV)

Das QNV verzichtet auf alle partiellen Ableitungen und generiert sich die Informationen über den Anstieg der Funktion aus der Historie der bisherigen Suche.

1. Diese wurden hier durch einen Stern * gekennzeichnet.

Simulated Annealing* (SA)

Dieses Verfahren simuliert einen thermodynamischen Abkühlprozeß zur heuristischen adaptiven Schrittweitenreduzierung, wie er z.B. bei Erstarrungsprozessen verschiedenster Materialien vorkommt. Die Grundidee liefert die Beobachtung, daß langsam abkühlende Stoffe eine sehr regelmäßige Kristallstruktur ausbilden, während sehr schnell abgekühlte Stoffe oder schockgefrorene Materialien eine unregelmäßiges Kristallgitter aufweisen.

Der SA-Algorithmus simuliert eine Brownsche Molekular-Bewegung, die anfänglich ein starkes Grundrauschen über dem Suchraum *G* nachbildet, das nach und nach verringert wird. D.h zunächst darf der Algorithmus beliebige Punkte des Suchraums zufällig[1] "anspringen", also auch solche, die eine gewisse Verschlechterung des Ergebniswertes (Funktionswert des gespeicherten Individuums) erlauben. Schritt für Schritt wird die Temperatur nach und nach bis zum Erstarrungspunkt *abgekühlt*, was sich darin äußert, daß die Schrittweite der Sprünge und die Akzeptanzschwelle für Funktionsverschlechterungen sukzessiv verringert werden. (Abbildung 2.6)

Dieses Verfahren ist den Evolutionsstrategien stark verwandt; es wird als Vorläufer der (1+1)-ES anerkannt (vergleiche Abbildung 2.14), sofern man das SA-Rauschen mit einem Mutationsoperator identifiziert.

Diskretes Hillclimbing (DHC)

DHC entspricht dem auf diskrete Räume übertragenen Gradientenverfahren. Für einen Punkt x im Suchraum *G* wird derjenige *Nachbarn* als Nachfolger genommen, welcher der größten Funktionswertverbesserung entspricht. In Räumen mit nur einem (globalen) Optimum terminiert DHC recht schnell mit der Lösung.

Reellwertiges Hillclimbing (RHC)

Durch eine kleine Erweiterung des DHC mit adaptiver Schrittweitenbestimmung kann es auch zum Einsatz in reellwertigen Räumen genutzt werden.

In einem reellwertigen *d*-dimensionalen Räum werden die Funktionswerte der *virtuellen Nachbarn*[2] eines Punktes *x* in Richtung der Koordinatenachsen mit Abstand *a* mit dem Funktionswert des Punktes *x* verglichen (das sind dann *2d* Vergleiche). Der Punkt *x'* mit dem größten Funktionswert aller betrachteten Punkte sei der Nachfolger von *x*.

1. In diesem Text taucht sehr oft der Begriff Zufall auf, ohne formell definiert zu werden. Es existieren zwar genaue mathematische Definitionen, jedoch handelt es sich hierbei um ein sehr komplexes Thema, auf das in dieser Arbeit nicht näher eingegangen werden kann. Die Brockhaus-Exzyklopädie definiert Zufall als *"das, was ohne erkennbaren Grund und ohne Absicht geschieht, das Mögliche, das eintreten kann aber nicht eintreten muß."* [BRO88]
2. In reellwertigen Räumen haben Punkte keine Nachbarn, weshalb hier durch die Einführung eines Abstands und eine willkürliche Festlegung einer Vergleichsrichtung (Koordinatenachen) künstlich welche definiert werden.

Hat *x* selbst den größten Funktionswert, so wird *a* halbiert und beginnt den Vergleich von neuem. Das Abbruchkriterium bildet ein Unterschreiten eines bestimmten Schwellwertes von a.

Der RHC-Algorithmus spannt ein *d*-dimensionales Gitter über den Suchraum, welches in der Nähe des Optimums lokal verfeinert wird. Sucht das RHC anfänglich noch in großen Schrittweiten nach Punkten mit guten Funktionswerten, so findet er mittels seiner Halbierungssuche das (globale) Optimum mit großer Genauigkeit. (Abbildung 2.7)[1]

Evolutionsalgorithmen* (EA)

EA sind eine Familie von Optimierungsstrategien, deren prinzipielles Grundmuster der von Charles Darwin beschriebenen Theorie der biologischen Evolution der Arten entstammt.

Ausgehend von einer zu optimierenden Funktion, der *Fitneßfunktion*, und einiger Menge (*Population*) von Punkten (*Individuen*) im Suchraum, wird durch eine sukzessive Generierung neuer Populationen Schritt für Schritt das Optimum approximiert. Dazu werden auf der Population *genetische Operatoren* definiert. Sie erzeugen aus vorgegebenen Individuen neue als Kandidaten für eine Nachkommenpopulationen. Als “innovativer Objektgenerator” produziert der *Mutationsoperator* zufallsgesteuert neue Punkte im Suchraum. Der *Rekombinationsoperator* setzt aus einer Menge bestehender Objekte neue Objekte zusammen. Schließlich sortiert der *Selektionsoperator*

Objekte mit weniger optimalen Funktionswert aus und bedingt somit die stochastische Konvergenz der Population zum Optimum hin.

Für eine detaillierte Beschreibung der Evolutionsalgorithmen folgt in den folgenden Kapiteln.

```
procedure simulated annealing
begin
  initialize I(0)
  t := 0
  choose initial Temperature T > 0
  while (termination_condition == FALSE) do
    I' := I(t)+ Z, Z ~ N(0,T)
    if ( f(I') > f(I(t)) )
      then I(t+1) := I'
      else if ( rand[0,1) < exp((f(I(t))-f(I'))/T) )
        I(t+1) := I'
      else I(t+1) := I(t)
    t := t+1
  T := g(T,t)
  endwhile
end
```

Abbildung 2.6: *Pseudocode des Simulated Annealing Algorithmus zur Maximasuche.*

1. E_d sei der *d*-dimensionale Einheitsvektor in positiver Richtung der Koordinatenachse der Dimension *d*.

.

```
procedure hillclimbing algorithm
begin
  init a, max
  set x randomly in search area
  while ( a > threshold )
    for i= 1..d
      xp := x+ aE_d
      xm := x- aE_d
      if ( f(xp) > max )
        x := xp
      else if ( f(xm) > max )
        x := xm
      else
        a := a/2
      endif
    endfor
  endwhile
end
```

Abbildung 2.7: *Pseudocode des RHC-Verfahrens. (Maximasuche)*

Um einen Überblick über die Fülle der verschiedenen Verfahren und deren Anwendbarkeit zu vermitteln, werden die Charakteristika im folgenden tabellarisch dargestellt.

	Suchraum diskret	Suchraum Kontinuum	partielle Ableitung *nicht* erforderlich[a]	Stetigkeit *nicht* erforderlich	Überwindet lokale Optima	stochastisches Verfahren	nichtapproximativ
Simplexverfahren	-	+	+	-	-	-	+
Gradientenverfahren	-	+	-	-	-	-	-
Newtonverfahren	-	+	--[b]	-	-	-	-
Quasi-Newtonverfahren	-	+	+	+	-	+	-
CG-Algorithmus	-	+	-	-	-	-	-
Hillclimbing	+	+	+	+	-	-	-
Simulated Annealing	+	+	+	+	+	+	-
Evolutionsalgorithmen	+	+	+	+	+	+	-

Tabelle 2.1: Vergleich verschiedener Optimierungsverfahren

a. partielle Ableitung und Stetigkeit wurden verneinend formuliert, damit die Kennzeichnung mit einem '+', welche subliminal einen Vorteil suggeriert, entsprechend mit einer vorteilhaften Eigenschaft assoziiert werden kann.
b. beim Newtonverfahren ist sogar die Kenntnis der zweiten partiellen Ableitungen vorauszusetzen.

2.5 Evolutionsalgorithmen

Unter dem Begriff Evolutionsalgorithmen versteht man eine Klasse von Optimierungsverfahren, die einem grundsätzlichen Schema folgen, welches in Abbildung 2.8 als Pseudocode dargestellt wird. Es ist zu beachten, daß das Konzept der Evolutionsalgorithmen eines der wenigen Verfahren realisieren, welches auf einer ganzen Menge von Punkten im Suchraum gleichzeitig operiert, also Populationen berechnet. Pro Rechenschritt wird also gleichzeitig eine Vielzahl von Raumpunkten berücksichtigt. Dieses Konzept realisiert eine Bereichssuche, die u.a. den Algorithmus dazu befähigt, lokale Optima zu überspringen und das globale Optimum zu finden. Durch die Verschmelzung von Punkten werden beispielsweise neue geschaffen, von denen erwartet werden kann, daß sie bezüglich ihrer Fitneß ihre Eltern überflügeln.

Dazu ein Zitat eines einschlägigen Lehrbuchs:

> *Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. They operate on a poplation of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowes from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals from who they were originally created, just as in natural adaption of the species.*
>
> ([POL96], Overview)

```
procedure evolution algorithm
begin
   initialize P(0)
   evaluate P(0)
   t := 0
   while (termination condition == FALSE) do
      select P(t) from P(t-1)
      alter P(t)
      evaluate P(t)
      t := t+1
   endwhile
end
```

Abbildung 2.8: *Allgemeines Schema eines EAs in Pseudocode. (siehe [MIC96])*

Funktionsweise des Algorithmus:

Der Algorithmus erzeugt zunächst eine Startpopulation, welche im Suchraum zufällig verteilte Punkte als Initial-Individuen erzeugt. **`evaluate`** bewertet die Individuen der Population, berechnet also ihre Fitneßwerte. Solange eine an die Population gestellte Qualitätsforderung noch nicht erfüllt ist, werden dann sukzessiv durch Anwendung genetischer Operatoren, also auf Populationen angewandte Manipulationsmethoden, neue Populationen berechnet.

Die in Abbildung 2.8 benutzen Funktionen im einzelnen:

initialize Initialisierungsfunktion, welche die Startpopulation mit zufällig erzeugten Individuen erzeugt

evaluate Berechnet die Fitneß der sich in der Population befindlichen Individuen.

select Select wählt diejenigen Individuen der Population aus, die für die Berechnung der Nachfolgepopulation in Frage kommen.

alter Die Funktion **`alter`** soll hier die Anwendung bestimmter genetischer Operatoren wie Rekombination oder Mutation andeuten. Es ist jedoch auch denkbar, noch andere problemspezifische Operatoren zu definieren.

termination_condition

Abbruchbedingung.

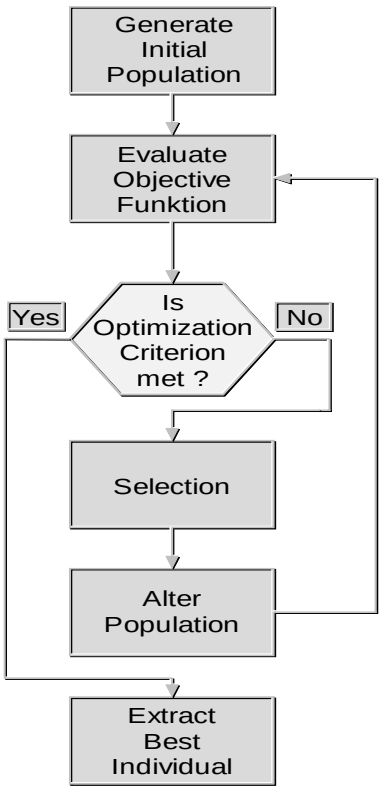


Abbildung 2.9: *Flußdiagrammdarstellung des Evolutionsalgorithmus*

Evolutionsalgorithmen stellen einen Überbegriff zu zwei unterschiedlichen Ansätzen dar, deren strikte Unterscheidung historischer Natur ist und heutzutage mehr akademisch als wissenschaftlich begründet ist. Zum einen sind dies die Genetischen Algorithmen (GA), die von J. H. Holland in den USA entwickelt wurden, zum anderen die von Rechenberg und Schwefel konzipierten Evolutionsstrategien (siehe [SCH96]).

Im folgenden werden explizite Datenstrukturen, d.h. die Struktur eines Punktes im Suchraum (z.B. eine Matrix oder ein Array), in Anlehnung an die biologische Nomenklatur als *Phänotyp* bezeichnet, die Kodierung dessen in einem Genom eines Individuums als *Genotyp*. Es gibt Algorithmen, die direkt auf dem Phänotyp operieren, und solche, die eine Umkodierung vornehmen, also einen Genotyp speichern.

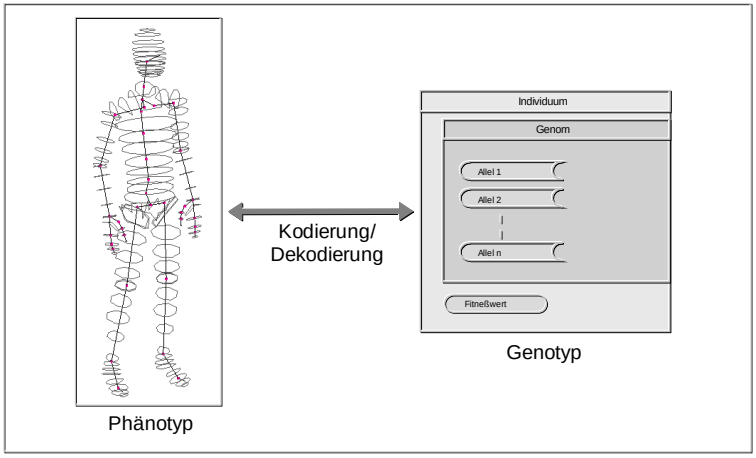


Abbildung 2.10: *Schematische Darstellung von Genotyp und Phänotyp*

In den folgenden Abschnitten wird die historische Entwicklung dieser verschiedenen Ansätze kurz nachvollzogen um dann die Konzepte genau herausarbeiten zu können und einen Einblick in die Funktionsweise zu geben.

Danach werden verschiedene hybride Konzepte erklärt und gezeigt, daß moderne Ansätze häufig auf bestimmte Probleme zugeschnittene Algorithmen realisieren, die nach dem Baukastenprinzip zusammengestellt werden können.

2.5.1 Genetische Algorithmen (GA)

GA wurden Anfang der 60er Jahre von J. H. Holland und seinen Studenten an der Universität von Michigan mit der Zielsetzung entwickelt, die natürliche Evolution in der Biologie zu simulieren bzw. das Verfahren selbst zu konkretisieren und deren Funktionieren zu beweisen.

Die Basiselemente (Individuen) der Population des Genetischen Algorithmus bestehen aus q-stelligen Binärsequenzen $\{0, 1\}^q$ von Nullen und Einsen, welche ein Individuum eindeutig als Genotyp beschreiben. Jeder solchen Folge und damit jedem Individuum entspricht ein Punkt (Phänotyp) im Suchraum[1]. GA führen also notwendigerweise eine Umkodierung der Punkte des Suchraums in die Menge der Individuen, also die Menge der möglichen Binärfolgen, statt.

1. die Verletzung dieser Forderung kann folgenschwere Konsequenzen haben (siehe Abschnitt 2.6).

Weiterhin kann, jedem Individuum ein Funktionswert zugeordnet werden, der eine Bewertung seiner Güte darstellt und im allgemeinen *Fitneßwert* oder *Fitneß eines Individuums* heißt. Die zugehörige Funktion heißt *Fitneßfunktion*.

Der Evolutionsprozeß findet folgendermaßen statt: Ausgehend von einer Startpopulation werden sukzessiv neue Populationen durch anwenden *genetischer Operatoren* erzeugt bis schließlich ein bestimmtes Gütekriterium erreicht ist und das Verfahren terminiert.Letzteres hängt i.a. von der Laufzeit des Algorithmus oder von der durchschnittlichen Fitneß der Population ab.

Die *Selektion* dient bei den genetischen Algorithmen der Auswahl von Individuen der aktuellen Population als Kandidaten zur Bildung von Nachkommenindividuen, die in die Nachfolgepopulation eingefügt werden. Der *Selektionsoperator* wählt aus der bestehenden Population, der *Elternpopulation,* Individuen als Kandidaten für die Nachfolgepopulation aus. Die Chance eines Individuums, als Kandidat ausgewählt zu werden, ist normalerweise mit seinem Fitneßwert korreliert, so daß für "fittere" Individuen eine größere Wahrscheinlichkeit besteht, für den Aufbau der Folge-P1opulation einbezogen zu werden, als für andere. Die Selektion realisiert das aus der Biologie bekannte Prinzip "*survival of the fittest"*.

Des weiteren existiert ein *Mutationsoperator*, der an zufällig gewählter Indexposition des Binärstrings eines selektierten Individuums ein Bit invertiert. Bemerkenswert ist, daß dieser Operator beliebige Sprünge im Suchraum verursachen kann, was direkt eine Eigenschaft der Umkodierung bei der Erzeugung von Binärsequenzen ist. Besteht beispielsweise der Phänotyp aus reellen Zahlen, so kann eine Bitmutation im Genotyp eine Nachkommastelle der einkodierten Zahl betreffen, was einem kleinen Sprung entspricht. Es kann aber auch der Exponent betroffen sein, was einem sehr großen Sprung im Suchraum darstellt.

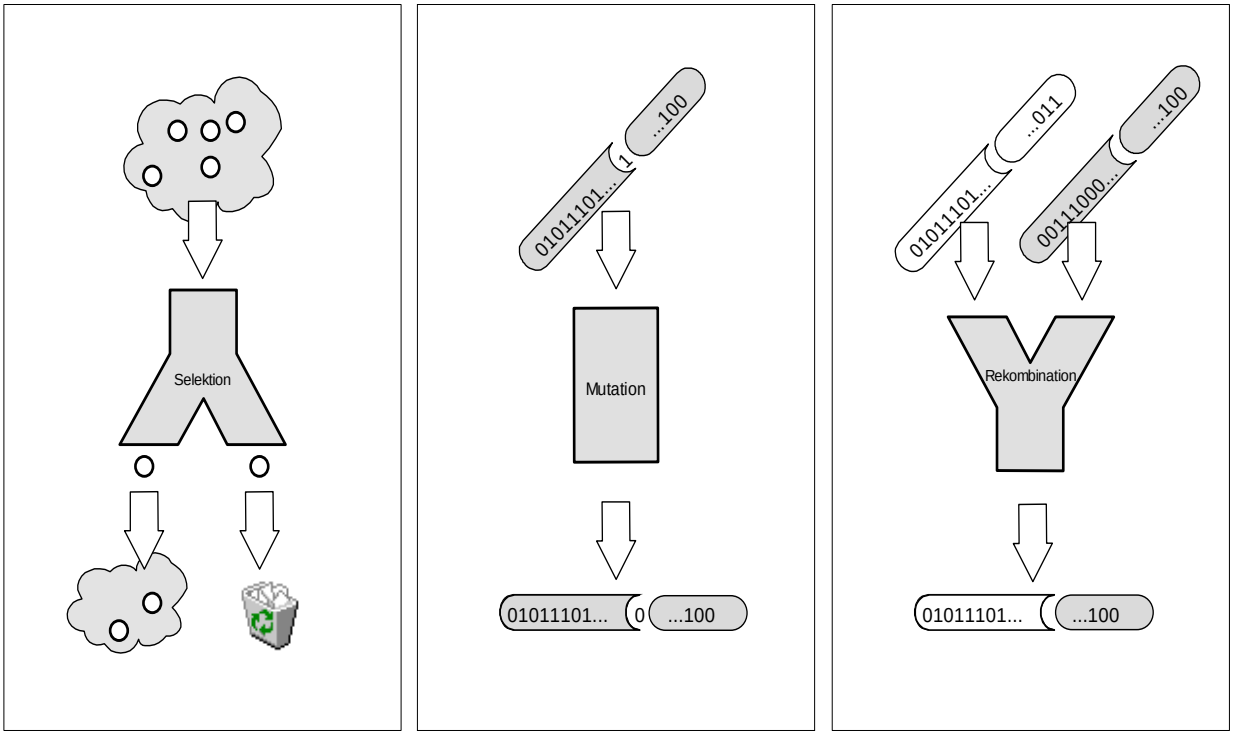


Abbildung 2.11: *Die genetischen Operatoren eines genetischen Algorithmus als Schemagraphik*

Der *Rekombinationsoperator* oder *Cross-Over-Operator* produziert aus den Binärsequenzen zweier selektierter Individuen (*Eltern-Individuen*) eine neue Sequenz, die Teilsequenzen der

Eltern-Individuen neu zusammenstellt. Daraus ergibt sich ein neues Individuum (*Nachkomme*), der in die Nachfolgepopulation eingefügt wird. Die Teilsequenzen ergeben sich aus einer zufälligen Zerlegung der Quellsequenzen in ordnungserhaltender Reihenfolge. (Abbildung 2.11). Üblicherweise werden die Sequenzen an einer oder mehreren Positionen aufgebrochen.

Da der Rekombinationsoperator je nach Wahl des Crossover-Punktes in dieser Form den Binärstring an beliebiger Stelle trennen kann, ist es bei einer Kodierung reeller Zahlen möglich, sogar wahrscheinlich, daß die Bruchstelle direkt eine Zahl zerstückelt. Der Crossover-Operator kann also beliebige Sprünge verursachen und damit wie eine Mutation wirken. Dies kann in der Endphase der Berechnung hinderlich sein, weil die Nachkommen der dem Optimum gut genäherten Individuen mit größerer Wahrscheinlichkeit weiter entfernt von diesem sind, als in seiner nächsten Umgebung. Deshalb werden die Bitstrings mit Indikatoren versehen, die den Anfang bzw. Ende zusammengehöriger Bereiche kennzeichnen. Bruchstellen sind dann ausschließlich zwischen diesen Bereichen erlaubt. (Abbildung 2.12)

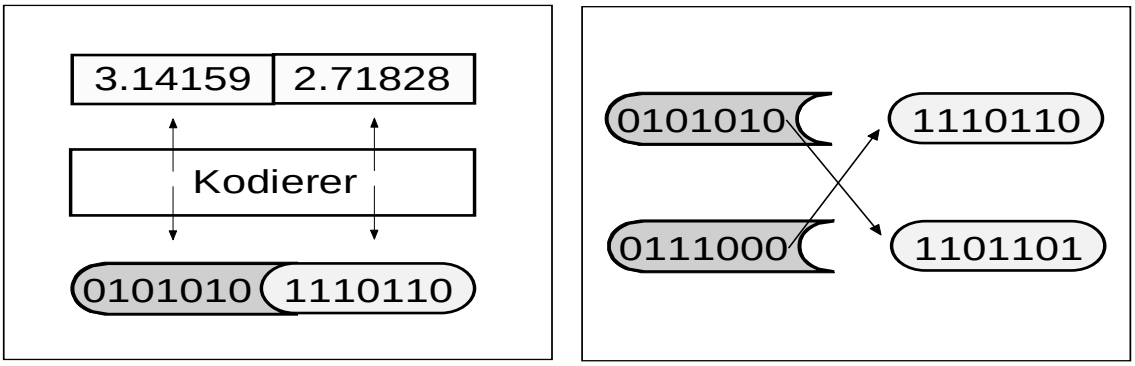


Abbildung 2.12: *Geschützte Bereiche sorgen dafür, daß logisch zusammengehörige Einheiten bei der Rekombination nicht getrennt werden.*

```
procedure genetical algorithm
begin
  initialize P(0)
  t := 0
  while (termination condition == FALSE) do
    S := select( P(t) )
    while (S ≠ ∅ ) do
      choose s1, s2 arbitrarily from S
      P(t) := P(t) ∪ mutate( recombine(s1,s2))
      S := S ∩ {s1,s2}
    endwhile
    t := t+1
  endwhile
end
```

Abbildung 2.13: *Schema eines genetischen Algorithmus. Zur Verdeutlichung der GA-spezifischen Besonderheiten sind diese durch detailliertere Darstellung hervorgehoben.*

2.5.2 Evolutionsstrategien (ES)

Evolutionsstrategien wurden als Optimierungsverfahren von Hans-Paul Schwefel und Ingo Rechenberg in den 60er Jahren an der TU Berlin ersonnen.

Diese Entwicklung entstand aus ihren Studien der Aerodynamik verschieden geformter Objekte in laminaren Strömungen eines Windkanals. Ziel der Forschung war eine Optimierung bestimmter Flügelformen (von Flugzeugen), die einen möglichst großen Auftrieb ermöglichen, bei gleichzeitiger Erzeugung minimaler Turbulenzen.

Da es keine Möglichkeiten gab solch eine gute Form mit mathematischen Werkzeugen zu berechnen, sah die damalige Vorgehensweise so aus: Man überlegte sich eine Form (von Heuristiken und dem Erfahrungsschatz des Ingenieurs abgeleitet), von der man gute aerodynamische Eigenschaften vermutete. Zu ihr wurde ein Modell gefertigt und dieses anschließend im Windkanal getestet. War die Messung schlecht, so verwarf man die Form wieder und begann die Prozedur von neuem. Dieses ineffiziente *Trial-and-Error-Prinzip* kostete sehr viel Zeit und ein Großteil der Modelle wurde vergeblich gebaut.

Die von Rechenberg und Schwefel entwickelte Innovation des Verfahrens bestand darin, ausgehend von der besten bestehenden Form eine neue durch eine zufällige kleine Änderungen[1] zu erzeugen, welche dann als neues Modell gebaut und getestet wurde. Dann wurde eine Bewertung in Form einer Messung durchgeführt. War diese Messung besser als die der alten Form, so nahm man sie als Basis für den nächsten Schritt und begann von neuem eine kleine Änderung vorzunehmen. Ergab sich keine Verbesserung, so verwarf man sie und nahm nochmals die alte Form für eine erneute Änderung.[2]

Daraus wurde die sogenannte (1+1)-ES entwickelt, die heute aufgrund der Vermischung von Konzepten mit denen der genetischen Algorithmen kaum noch in dieser Form implementiert wird.

2.5.3 (1+1)-ES

Der Algorithmus startet mit einem zufällig erzeugten Individuum **I(0)**. Solange ein Terminierungskriterium noch nicht erfüllt ist, werden sukzessiv neue Individuen **I(t)** erzeugt, die sich jeweils durch minimale Änderungen des Vaterindividuums ergeben (Mutation). Ist das neu erzeugte Individuum (**I'** = Nachkomme) bezüglich eines Fitnessmaßes besser als der Vater, so ist der Vater durch den Nachkommen zu ersetzen, andernfalls verwirft man diesen.

Es ist leicht einsehbar, daß dieser Algorithmus im *worst case* stagniert, im Mittel jedoch wird er sich auf ein Optimum zubewegen. Das theoretisch erreichbare globale Optimum kann in der Praxis nur durch sorgfältiges Einstellen der Mutationsschrittweite erreicht werden; ist diese zu klein, dann bleibt das Verfahren in der Umgebung eines lokalen Minimums stecken; ist sie zu groß, so wird der Sohn mit großer Wahrscheinlichkeit wieder aus einer einmal gefundenen Umgebung des globalen Optimums herausspringen, sofern dieses nicht genau getroffen wird und viele ungünstig beschaffene lokale Optima existieren.[3]

---

1. Diese Vorgehensweise nutzt die (notwendige) Eigenschaft der *starken Kausalität*, daß heißt, kleine Änderungen haben kleine Auswirkungen.
2. Besonders bemerkenswert ist, daß dieses Verfahren so leistungsfähig war, daß trotz des hohen physikalischen Aufwands, der es bedurfte, die Formen zu bauen, innerhalb kurzer Zeit wesentliche Verbesserungen erreichte.

Anzumerken ist noch, daß die (1+1)-ES keine Strategie ist, die auf Populationen operiert. Die Bezeichnung resultiert daher, daß man sie als einen Spezialfall der (n+m)-ES auffassen kann.

Im folgenden seien:

initialize Initialisierungsfunktion, die das zufällige Startindividuum I(0) erzeugt

mut Mutationsfunktion. Erzeugt ein neues Individuum, indem sie das alte zufällig verändert

fit Fitnessfunktion. Bewertet jedes Individuum mit einer reellen Zahl, die seiner Güte entspricht soll

termination_condition

Abbruchbedingung. Bei iterativen Verfahren wird immer eine Abbruchbedingung benötigt, die nach einigen approximativen Schritten das Verfahren beendet.

```
procedure (1+1) evolution strategy
begin
  initialize I(0)
  t := 0
  while (termination_condition == FALSE) do
    I' := mutate I(t)
    if ( fit(I') >= fit(I(t)) )
      then I(t+1) := I'
      else I(t+1) := I(t)
    t := t+1
  endwhile
end
```

Abbildung 2.14: *Pseudocode der (1+1)-ES*

2.5.4 (n,m)-ES

Die aus der (1+1)-ES hervorgegangenen (n,m)-ES ist eine Strategie, die um zusätzliche Konzepte erweitert wurde. Zum einen wird bei diesem Verfahren eine Population von Individuen verwaltet, ähnlich dem schon von GA bekannten Konzept. Statt des einen Individuums, auf welchem die (1+1)-ES operiert, hat diese Strategie eine größere Anzahl von diesen, denen die Population als Container dient. Ähnlich den GA generiert die (n,m)-ES sukzessiv neue Populationen, wobei die Erzeugung von Individuen, die in eine Nachfolgepopulation eingefügt werden, wiederum durch genetische Operatoren erfolgt.

3. Das Einstellen der Mutationsschrittweite kann durch vielfaches Herumprobieren und zum Geduldsspiel werden. Eine Verbesserung , die beispielsweise durch Selbst-Evolution der Steuervariable erreicht wird, wird in einem späteren Kapitel besprochen.

Weiterhin existiert ein Rekombinationsoperator, der Datenstrukturen mehrerer Individuen aufbrechen und neu zusammenstellten vermag.

Im Unterschied zu genetischen Algorithmen wird bei den hier besprochenen Strategien zunächst eine Nachfolgepopulation erzeugt, die *größer* als die Elternpopulation ist. D.h. ausgehend von einer Population mit n Individuen werden $m > n$ Nachfolger generiert, von denen nur die n besten für den nächsten Berechnungsschritt ausgewählt werden.

Unglücklicherweise wird diese Operation bei den Evolutionsstrategien im Vergleich zu GA ebenfalls Selektion genannt, obwohl es sich hier um einen anderen Vorgang handelt. Da allgemein bei Evolutionsalgorithmen beide Arten gleichzeitig implementiert werden können, führt dies potentiell zur Verwechslung der Begriffe. Deshalb soll in den folgenden Abschnitten und Kapiteln, um Mißverständnisse auszuschließen, zur Unterscheidung der Operatoren bei Evolutionsstrategien von *ES-Selektion* oder *Reduktion* gesprochen werden. Entsprechend lautet die Nomenklatur genetischen Algorithmen *GA-Selektion* oder einfach *Selektion*.

Das Komma in der Bezeichnung dieses Verfahrens deutet an, daß die Elternindividuen ausschließlich zur Generierung neuer Individuen benutzt werden, jedoch selbst nicht in die Nachfolgepopulation aufgenommen werden.

```
procedure (n,m) evolution strategy
begin
  initialize P(0)
  evaluate P(0)
  t := 0
  while (termination condition == FALSE) do
    P_C(t):= recombination P(t)
    mutation P_C(t)
    P(t+1) := reduce (P_C(t))
    t := t+1
  endwhile
end
```

Abbildung 2.15: *Schema der (n,m)-ES.*

2.5.5 (n+m)-ES

Die (n+m)-ES unterscheidet sich von der obigen dadurch, daß die Elternindividuen selbst wieder in die Nachfolgepopulation aufgenommen werden und somit mit ihren Nachkommen im Wettbewerb stehen. Somit geht über die Laufzeit des Algorithmus einmal erreichte gute Individuen nicht wieder verloren, es gibt keine Rückschritte im Verlauf der Evolution[1].

1. Zur Vermeidung der Überalterung der Bevölkerung und dem damit einhergehenden Verlußt an Innovation werden in der Literatur Altersbegrenzungen für Individuen vorgeschlagen.

```
procedure (n+m) evolution strategy
begin
  initialize P(0)
  evaluate P(0)
  t := 0
  while (termination condition == FALSE) do
    P_C(t):= recombination P(t)
    mutation P_C(t)
    P(t+1) := reduce (P_C(t) ∪ P(t))
    t := t+1
  endwhile
end
```

Abbildung 2.16: *Pseudocode der (n+m)-ES. Man beachte die Verwendung von Populationen und der Rekombination als genetischen Operator.*

2.5.6 Klassische Datenrepräsentation bei GA und ES

Durch die freie Wahl der algorithmeninternen Datenkodierung können ES problemspezifischer implementiert werden als GA, die traditionell auf einer Binärrepräsentation und darauf angewandten Standard-Operatoren bestehen.

Der GA ist nur einmalig zu implementieren, für eine Problemstellung ist nur die Kodierungsfunktion auszutauschen.

Im Gegensatz dazu kann eine (klassische) ES speziell auf ein Problem zugeschnitten werden, d.h. es ist eine geeignete interne Datenrepräsentation zu wählen, worauf die genetischen Operatoren zu definieren sind. Durch diesen Mehraufwand wird i.a. der Algorithmus effizienter und schneller sein.

Moderne GA lassen diese Restriktion fallen und erlauben ebenfalls komplexe interne Datenstrukturen.

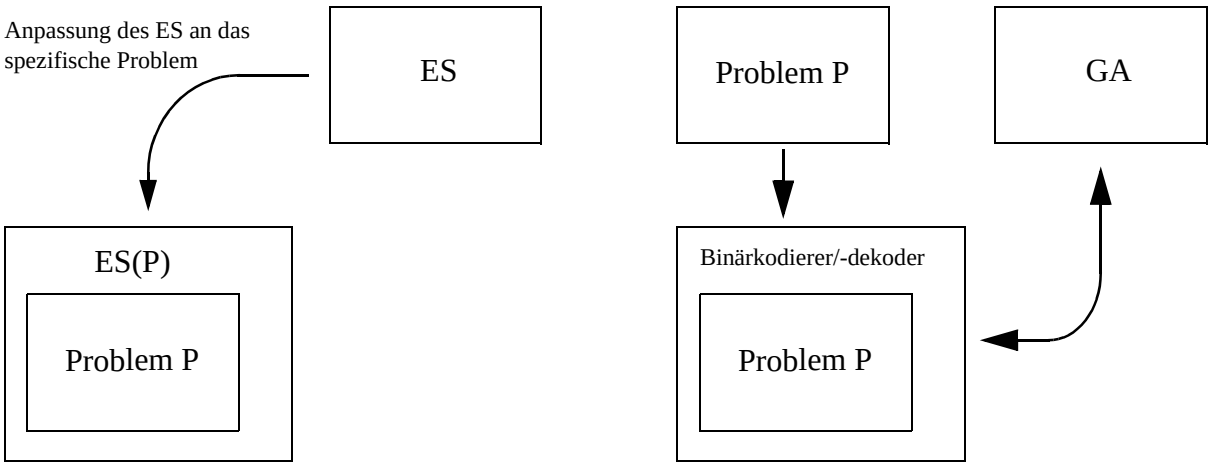


Abbildung 2.17: *Prinzipdarstellung von Evolutionsstrategie und genetischem Algorithmus.*

2.5.7 Verschiedene Strategien im Überblick

An dieser Stelle soll tabellarisch ein Überblick über die Gemeinsamkeiten bzw. Unterschiede der genetischen Algorithmen und Evolutionsstrategien gegeben werden. Wie bereits angedeutet sind die strikten Unterschiede historisch bedingt. Heute werden meistens als Hybridverfahren implementiert, deren Wahl problemabhängig ist.

Vergleich zwischen ES und GAs	Evolutionsstrategien	Genetische Algorithmen
Datenrepräsentation	Keine Festlegung auf eine spezielle Repräsentation; komplexe Repräsentationsstrukturen möglich. ((1+1)-ES urspünglich: Reelle Kodierung)	Repräsentation durch binäre Strings, auf denen Standardoperatoren definiert sind.
Mutation	Der Mutationsoperator hängt im wesentlichen von der zugrundeliegenden Datenstruktur zu Repräsentation von Punkten im Suchraum ab. Bei ES werden Kontrollvariable verwendet, welche das Verhalten der Strategie selbst bestimmen und keine Entsprechung im Suchraum haben (z.B. Mutationsschrittweite).	Der Standardmutationsoperator besteht im wesentlichen darin, an zufälliger Stelle des Binärstrings ein Bit zu kippen.
Rekombination	Der Rekombinationsoperator hängt direkt von der Repräsentationsstruktur ab. Er dient dazu, die Datenstrukturen mehrerer Individuen aufzubrechen und diese neu zu kombinieren.	Der Standardrekombinationsoperator (engl. crossover) produziert einen Nachfolgestring, indem er zwei oder mehrere Individuenstrings an ein oder mehreren Stellen überkreuzt und so neu zusammenstellt
ES-Selektion	Bei der ES-Selektion (in diesem Text auch als Reduktion bezeichnet) wird die Konvergenz der Populationen zum Optimum hin dadurch erreicht, daß ein Individuenüberschuß in der Nachfolgepopulation erzeugt wird, von denen die schlechtesten Individuen entfernt werden, indem die Population auf eine Standardgröße reduziert wird. Weitere Unterscheidung durch (n+m) und (n,m) Strategien (s.u.)	meist nicht implementiert
GA-Selektion	meist nicht implementiert	Selektion durch Auswahl von Individuen der aktuellen Population als Kandidaten für die Produktion der Nachfolgepopulation durch anwenden der Operatoren *Rekombination* und *Selektion*

Tabelle 2.2: Überblick der genetischen Operatoren bei Evolutionsstrategien und Genetischen Algorithmen

Im Anschluß an diese Übersicht soll aus oben genannten Gründen die strenge Unterscheidung zwischen den Begrifflichkeiten aufgegeben und allgemein von *Evolutionsalgorithmen* und *Evolutionsstrategien* gesprochen werden.

2.5.8 Mutationsoperatoren

Der Mutationsoperator ist von hoher Bedeutung für die Evolution, da er die einzige Möglichkeit darstellt, neue Merkmale zu erzeugen, d.h. er gibt der Evolution die Möglichkeit innovativ sein zu können. Die erste (1+1)-ES hatte als einzigen genetischen Operator die Mutation und ersten von Holland entwickelten genetischen Algorithmus war die Mutation noch durch eine einfache Bit-Veränderung an zufälliger Stelle im Binärcode des Individuums angegeben. Siehe dazu Abschnitt 2.5.1

Für die Konstruktion eines Mutationsoperators gibt es bei komplexen Datenstrukturen viele Möglichkeiten und es muß sehr genau auf die Problemstellung eingegangen werden. Es ist präzise zu überlegen, welches Resultat in welcher Weise erzielt werden soll. Bei einer Baumstruktur kann die Mutation beispielsweise darauf beschränkt werden, die Blätter auf zufällige Weise zu ändern. Es ist aber auch vorstellbar, ein Blatt durch einen beliebig erzeugten neuen Teilbaum zu ersetzen oder ganze Teilbäume zu löschen.(Abbildung 2.18)

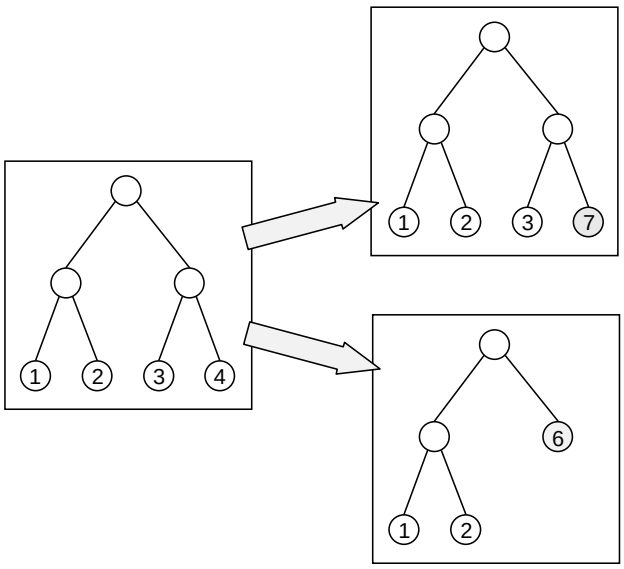


Abbildung 2.18: *Zwei mögliche Mutationen einer Baumstruktur*

Generell ist bei der Konstruktion des Mutationsoperators darauf zu achten, daß durch eine oder durch eine Folge von Mutationen jeder Punkt des Suchraums erreicht werden kann. Andernfalls entstünde u.U. das Problem, das Optimum des Suchraums nicht erreichen zu können. Damit wäre die Vollständigkeit nicht gegeben.

Außerdem ist zu beachten, daß die Menge der von der Mutation erreichbaren Punkte, die außerhalb des Definitionsbereichs liegen, möglichst klein ist. In diesem Fall gibt es zum einen die Möglichkeit durch *Korrekturoperatoren* solche Irrläufer sofort zu eliminieren. Zum anderen können diese Individuen mit einem Aufschlag auf ihren Fitneßwert bestraft werden um den Algorithmus selbst durch Selektions-Mechanismen solche Punkte bereinigen zu lassen.

Entspricht der Grundbereich einem linearen Raum, z.B. dem $\mathbb{R}^n$, so ist es sinnvoll, den Raumpunkt x mit einer normalverteilten Zufallsvariable zu mutieren. Der auf x addierte Offset ist also gaußverteilt. Damit erweist sich eine Mutation, die nicht weit von dem Ausgangsindividuum entfernt liegt, als recht wahrscheinlich, während ein sehr weiter Sprung unwahrscheinlich ist. Die Form der Verteilung kann beispielsweise mittels eines Parameters eingestellt werden (*Varianz*). Diese ist gehört zu den Kontrollvariablen der Evolutionsstrategie. (siehe dazu auch Abschnitt 2.5.11)

2.5.9 Rekombinationsoperatoren

Rekombinationsoperatoren haben die Aufgabe Speicherstrukturen zweier oder mehrerer gegebener Individuen in Teile zu zerlegen und die einzelnen Teilstücke zur Erzeugung eines Nachfolgeobjektes neu zusammenzustellen. Die Bruchstellen werden zufällig ermittelt so daß sich selten gleichartige Teile ergeben.

Die grundlegende Idee ist die, daß jedes Individuum der Population in seiner Genomstruktur gut angepaßte Erbinformationsstücke besitzt und andere, die weniger gut sind. Rekombiniert man Strukturstücke von mehreren Individuen, so ergibt sich die Chance, daß das neu entstandene Individuum eine Kombination der guten Teilstücke vereint und damit besser angepaßt ist, als seine Elternindividuen.

Genauso wie der Mutationsoperator ist auch der Rekombinationsoperator auf die gewählte Kodierung anzupassen. Bei komplexen Datenstrukturen ergibt sich oft eine Vielzahl von sinnvollen Möglichkeiten. Als Beispiel sei eine Baumstruktur genannt, bei der eine Rekombination sowohl auf Blattebene als auch auf Teilbaumebene denkbar ist. Die Wahl des Operators ergibt sich häufig aus der Aufgabenstellung des Problems.

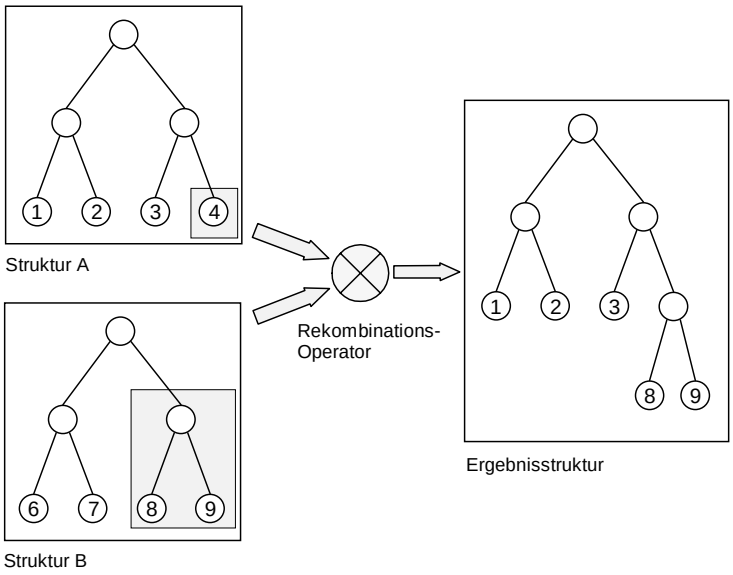


Abbildung 2.19: *Mögliche Rekombination einer Baumstruktur*

2.5.10 Der Selektionsoperator[1]

Der Selektionsoperator hat die Aufgabe aus einer gegebenen Population Individuen auszuwählen, die für weitere Anwendungen der Mutations- und Rekombinationsoperatoren bestimmt sind. Dabei wird diese Wahl bei GA vom Fitneßwert eines Individuums abhängen. Bei Evolutionsstrategien[2] kann sie mitunter beliebig sein. Das heißt, ein Individuum mit gutem Fitneßwert wird mit einer größereren Wahrscheinlichkeit als weniger “fitte” zur Konstruktion von Nachfolge-Individuen gewählt werden.

1. Damit ist die GA-Selektion gemeint.
2. Hier: *GA- Selektion*. ES besitzen als weiteren Selektionsmechanismus die *ES-Selektion* oder *Reduktion.*

Während die Selektion bei Standard-ES mit hoher Nachkommenzahl einfach aus einem zufälligen Herausgreifen von Individuen aus der alten Population bestehen kann (durch den Reduktionsoperator werden die schlechteren Nachkommen wieder gelöscht) so ist sie bei GA essentiell für den Erfolg der Evolution, da sie hier die einzige Methode ist die Besseren von den Schlechteren zu separieren.

Die Selektion hat einen entscheidenden Effekt auf die Verbesserung der durchschnittlichen Populationsfitneß über die Generationen hinweg. Sie entscheidet über den *Selektionsdruck* und gleichzeitig über die Vielfalt innerhalb der Population. Somit ist über die Effektivität dieser Strategie in ihren Eigenschaften als "intelligentes" Suchverfahren (Optimierungsverfahren). Fällt die Varianz in der Population zu stark zu schnell ab, so wird sie sich in ein lokales Optimum verfangen, dem sie in der Regel nicht mehr zu entkommen vermag. Ist der Druck zu gering, so läuft das Verfahren zu lange und wird nur langsam Verbesserungen erzielen.

Im folgenden sollen einige Selektionsverfahren diskutiert werden. Dabei wird jeweils eine Formel $select(ind, t)$ verwendet, die ein Maß für die Wahrscheinlichkeit eines Individuums *ind* darstellt, als Kandidat ausgewählt zu werden. D.h. $P_S(ind) \sim select(ind, t)$ [1]. t ist die Generation, in der sich die Population $pop(t)$ befindet. $fit(ind)$ ist die Fitneßfunkion eines Individuums und $fit\{pop\}$ soll die durchschnittliche Fitneß der Individuen einer Population darstellen.

Die Fitnessproportionale Selektion

Bei der Fitneßproportionalen Selektion steigt die Wahrscheinlichkeit für Individuen, als Kandidat gewählt zu werden proportional mit ihrer Fitness an:

$$select(ind, t) = fit(ind) / fit\{pop_t\}.$$

Die folgenden Verfahren RWS und SUS sind Verfahren, die der fitneßproportionalen Selektion zugeordnet werden können.

Roulette Wheel Sampling (RWS)

Dieses Verfahren basiert auf dem Glücksradprinzip. Man teilt die Scheibe eines Glücksrads in verschiedene Felder ein, und zwar so, daß jedem Individuum der Population ein Feld zugewiesen wird. Die Fläche der Felder sei der Fitneß der Individuen proportional. Um einen Kandidaten zu bestimmen dreht man das Glücksrad einmal, für *n* Kandidaten entsprechend *n* mal.

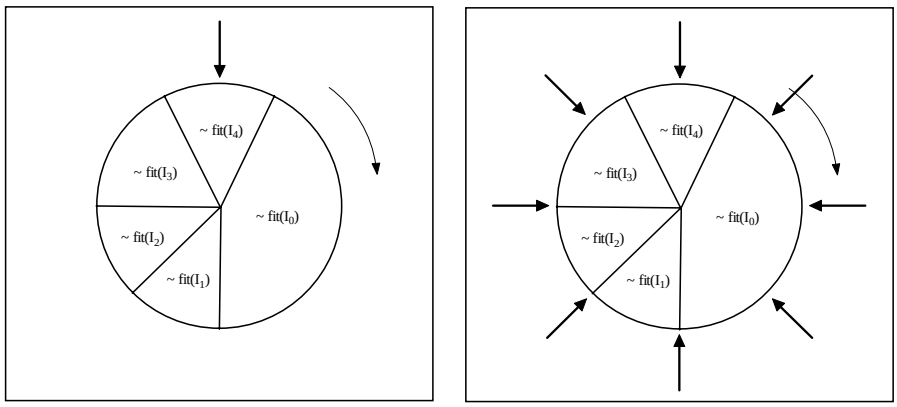
Das roulette wheel sampling hat vor allem bei kleinen Populationen den Nachteil, daß des öfteren Individuen mehrfach selektiert werden während andere überhaupt ausgewählt werden. In Extremfällen stammen alle Nachkommen vom schlechtesten Individuum der Population ab. Um dies zu korrigieren wurde das SUS-Verfahren entwickelt:

Stochastic Universal Sampling (SUS)

Beim SUS handelt es sich um ein leicht abgewandeltes RWS-Verfahren, welches das gleiche Glücksradmodell nutzt. Die Änderung besteht darin, daß dabei nicht ein einziger Zeiger zur Auswahl benutzt wird, sondern ein Zeigerstern, welcher so viele Zeiger in äquidistanten Ab-

1. Die Notation $x \sim y$ bedeutet die Proportionalität von *x* und *y*, d.h. es gibt eine Proportionalitätskonstante *c* mit $x = cy$.

ständen besitzt, wie die Zahl der zu selektierende Individuen ist. Gedreht wird beim SUS nicht mehrmals, sondern mit einem einzigen Dreh sind alle Kandidaten für die Nachfolgerpopulation bestimmt. (vgl. Abbildung 2.20)



Roulette Wheel Sampling Stochastic Universal Sampling

Abbildung 2.20: *RWS und SUS, welches nur einmaliges Drehen des Rads erfordert*

Probleme mit der fitneßproportionalen Selektion ergeben sich daraus, daß in der Anfangsphase der Evolution ein sehr starkes Gefälle in den Fitneßwerten der Populationsmitgliedern ergibt, so daß hier u.U. ganz wenige gute Individuen die ganze Nachkommenschaft erzeugen und somit die Vielfalt der Population schnell verloren geht, während in der Spätphase alle Individuen sich stark ähneln und damit sich ihre Fitneßwerte nur noch marginal unterscheiden, so daß dabei kaum noch selektiv operiert werden kann.

Den Nachteilen des SUS versucht man mit den folgenden Selektionsmethoden zu begegnen.

Sigma Scaling

Das *Sigma Scaling* versucht die Nachteile der fitneßproportionalen Selektion zu vermeiden, indem der Selektionsdruck über die Zeit hinweg konstant gehalten wird, d.h. das Wahrscheinlichkeitsgefälle in früheren und späten Generationen soll konstant gehalten werden. Dazu setzt man

$$select(ind, t) = (fit(ind) - fit\{pop_t\}) / (2\sigma(pop_t)) \text{ falls } \sigma(pop_t) \neq 0 \ ;$$

$$select(ind, t) = 1 \text{ sonst.}$$

Dabei sei $\sigma(pop_t)$ die Standardabweichung der Fitneßwerte der Individuen in der Population, d.h. obige Formel zwingt die Evolutionsstrategie auf einen moderaten Selektionsdruck bei starker Populationsvarianz und steigert diesen im Zuge der abnehmenden Varianz über die Laufzeit hinweg.

Es kann sein, daß die Struktur der Fitneßfunktion so komplex ist, daß es vorzuziehen ist, zunächst einmal ein Breitensuche im Definitionsgebiet durchzuführen und erst danach den Selektionsdruck zu erhöhen, um aus der weit verstreuten Population, die viele lokale Optima abdeckt, das globale Optimum herauszufinden. Diese Überlegung war der Basisgedanke für die Folgende Selektionsstrategie:

Bolzman-Selektion

Die *Bolzman-Selektion* selektiert auf folgende weise:

$$select(ind, t) = exp(\,|\, fit(ind) - fit\{pop_t\}\,|\, T(t))$$

$T(t)$ stellt dabei eine weitere Funktion dar, welche den Selektionsdruck, abhängig von der Generationszahl *t*, regelt. Ist $T(t) > 1$, so erhöht sich der Selektionsdruck, andernfalls erniedrigt er sich. Man sollte demnach diese Funktion so wählen, daß der Algorithmus mit niedrigem Druck beginnt und er sich nach und nach steigert. Sie stellt einen weiteren freien Parameter der Strategie dar.

Rank Selection

Die *Rank Selection* wählt die Kandidaten unabhängig von ihrer absoluten Fitneß aus, sondern stellt eine Rangfolge in der Population auf, welche die Individuen nach ihrer Fitneß sortiert. Das beste Individuum hat also den ersten Rang während das schlechteste den hintersten Rang hat. Die Wahrscheinlichkeit, ein Individuum als Kandidaten zu wählen hängt dabei nur von seinem Rang ab und nicht mehr absolut von seiner fitness.

Die Rank selection hat den Vorteil, daß absolute Fitneßunterschiede keine Rolle spielen. In einer frühen Generation wird somit das beste Individuum mit der gleichen Wahrscheinlichkeit ausgewählt wie in einer späten; der Selektionsdruck bleibt also auch bei der Rank Selektionsmethode über die Zeit hinweg konstant.

2.5.11 Parameter der Evolution und die Metamutation

Evolutionsalgorithmen besitzen eine sehr große Anzahl von externen *Steuerparametern*, die den Ablauf des Verfahrens bestimmen. Es gilt, sie dermaßen zu bestimmen, daß das Verfahren für ein spezifisches Problem optimale Ergebnisse erzielt.

Zu diesen Parametern, die auch *Kontrollparameter* oder *Kontrollvariable* genannt werden, gehören unter anderem Populationsgröße, Abbruchbedingung, Mutationsschrittweite, Rekombinationsmethode, Selektionsmethode etc.

Es ist im allgemeinen nicht möglich, diese Parameter zu berechnen oder auf eine andere Art exakt herzuleiten. Sie entstammen häufig dem Erfahrungsschatz des Programmierers und werden manuell in vielen Testläufen optimiert.

Grundsätzlich stellt die Bestimmung der Kontrollparameter selbst wieder ein Optimierungsproblem dar. Zur Lösung dieser Aufgabe erscheint gleichfalls ein Optimierungsverfahren geeignet, beispielsweise eine darauf abgestimmte Evolutionsstrategie. Zum einen gibt die Möglichkeit dies extern zu betreiben, indem jeder Programmlauf selbst wieder als ein Individuum einer größeren Evolution angesehen wird, die ein Optimum bezüglich Parametern wie Programmlaufzeit, Ergebnisgenauigkeit etc. ermittelt.

Alternativ kann die Einstellung dieser Parameter der Algorithmus selbst durchführen, indem sie rückbezüglich durch die genetischen Operatoren verändert werden. Dazu erhält jedes Individuum zusätzlich zu den Objektparametern auch noch veränderliche Kontrollparameter. Erhält es beispielsweise einen Parameter Mutationsschrittweite, der angibt, wie groß die Varianz des ei-

genen Mutationsoperators ist (einen reellwertigen Suchraum mal vorausgesetzt), der analog zu den Objektparametern behandelt werden kann, so wird dieser gleichfalls optimiert. (Abbildung 2.21)

Diese Vorgehensweise kann man an beliebigen Kontrollparametern mit folgenden Einschränkungen durchführen:

- Notwendigerweise müssen es solche Parameter sein, die einzelnen Individuen zugeteilt werden können, wie z.B. Parameter, welche Mutation, Rekombination etc. bestimmen. Populationsparameter, wie beispielsweise die der Selektion, sind von dieser Vorgehensweise ausgeschlossen, da diese prinzipiell nicht am Wettbewerb teilnehmen können und sich damit den Optimierungsprinzipien entziehen. Bei erweiterten Modellen, wie beispielsweise dem sog. *Populationenmodell* (vgl. Abschnitt 2.8), ist dies ausdrücklich auf einer höheren Ebene möglich, da dabei ganze Populationen im Wettbewerb stehen und damit auch ein Vergleich zwischen ihnen stattfindet.
- Die Kontrollparameter, manchmal auch *Metaparameter* genannt, müssen vor allen anderen mutieren (*metaparameter mutate first*). Sie bestimmen direkt das Verhalten der genetischen Operatoren, die in weiteren Schritt die Objektparameter verändern. Dies ist die einzige Möglichkeit, die Güte dieser Parameter zu bestimmen und zu vergleichen, da es ihre einzige Wirkung darstellt. D.h. die Qualität der Kontrollparametereinstellung wird indirekt über ihre Auswirkungen auf die Objektparameter mittels der individuenbewertenden Fitneßfunktion gemessen.

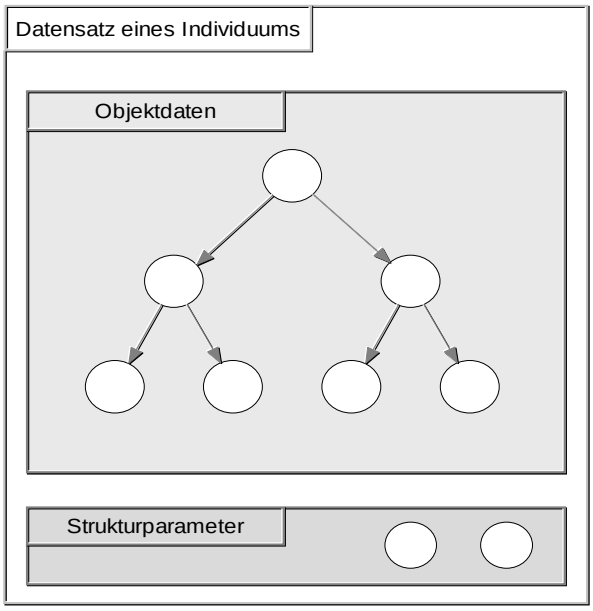


Abbildung 2.21: *Die Datenstruktur eines Individuums.*

Es ist denkbar, jedem Individuum eine eigene Datenstruktur (zur Repräsentation) und zugehörige Interpretationsoperatoren zuzuordnen. Damit ließe sich die interne Repräsentation der Objekte des Suchraums selbst wieder der Evolution unterwerfen und damit optimieren, womit das Problem der Suche der geeigneten Datenrepräsentation wegfallen würde[1].

Allerdings träten an dieser Stelle neue Probleme auf, man müßte z.B. dafür sorgen, daß eine Unterrepräsentation der Objekte des Suchraumes vermieden wird. Würden Punkte des Suchraumes nicht mehr kodiert werden können, so könnte das (eigentliche) Optimum nicht mehr gefunden

werden. Zu einer genaueren Analyse der Schwierigkeit der Kodierung betrachte man das folgende Kapitel.

2.6 Datenrepräsentation

Ein wichtiger Punkt bei der Konstruktion eines geeigneten Datenmodells zur Lösung eines Problems mit Evolutionsalgorithmen ist die Wahl einer geeigneten Kodierung. Eine “gute” Kodierung sollte folgende Eigenschaften erfüllen:

- **Korrektheit**. Die Datenstrukturen sind möglichst so zu wählen, daß jeder Code ein gültiges Element des Suchraumes darstellt.
- **Vollständigkeit**. Jeder Punkt des Suchraums, der die Restriktionen erfüllt, muß in der Kodierung darstellbar sein.
- **Minimalität**. Die Zahl der Genotypen, die den gleichen Phänotyp darstellen, ist zu minimieren.
- **Effizienz**. Die Kodierung sollte nach Möglichkeit so gewählt werden, daß die genetischen Operatoren effizient zu implementieren sind (man betrachte dazu die nachfolgenden Beispiele).
- **Geeignete Wahl der Kodierung**. Nach Möglichkeit sollte die Kodierung das Entstehen lokaler Optima vermeiden.

Leider ist es nicht für jedes beliebige Problem möglich, eine Kodierung zu finden, die alle obige Punkte erfüllt. Mitunter ist eine Beschränkung auf die wichtigen Punkte Korrektheit und Vollständigkeit bereits sehr schwer.

Ist die Vollständigkeit nicht gegeben, so kann im Extremfall das Optimum nicht gefunden werden, wenn es selbst außerhalb des in der Kodierung darstellbaren Bereichs liegt.

Eine Verletzung der Korrektheit ist dagegen gegebenenfalls mit entsprechenden Gegenmaßnahmen auszugleichen:

Korrektur illegaler Codes

Diese Methode überführt alle “illegalen” Genotypen mittels entsprechender Korrekturoperatoren in gültige, die eine externe Entsprechung im Suchraum besitzen. Diese Korrektur kann mit erheblichem Berechnungsaufwand verbunden sein und kann eine Komplexität erreichen, die der Lösung des Gesamtproblems entspricht.

1. Diese Vorgehensweise unbedingt auf der Hand. Schließlich stammt die grundlegende Methodik der Evolutionsalgorithmen der von Darwin entwickelten Theorie der natürlichen Evolution der Arten [DAR44]. Die Evolution der Repräsentation entspräche immerhin einer Selbst-Modifikation des genetischen Kodierung der Tiere und Pflanzen (Veränderung der Chromosomenstruktur), also einer Vorgehensweise, welche die Natur nach heutigem Wissensstand nicht betreibt; die genetische Sprache aller Lebewesen ist artübergreifend und universell. Zur *Datenstrukturevolution* siehe auch [MIT96], Kapitel 33, Seite 158.

Verschlechterung des Fitneßwertes

Der Fitneßwert illegaler Codes wird mit einem Strafaufschlag versehen, der beispielsweise seinem Abstand vom gültigen Terrain des Suchraumes entspricht. Das Evolutionsverfahren wird dadurch motiviert, solche Kodierungen im Sinne der Optimierung zu vermeiden.

Dies zweite Methodik hat entscheidende Nachteile, wenn bezüglich aller möglichen Code-Kombinationen einer großen Menge von ungültigen nur eine verschwindende Anzahl gültiger Kodierungen gegenüberstehen. Die Population wird dann mit illegalen Inidviduen überschwemmt. Damit wird sich die Population irgendwo im ungültigen Bereich des Kodierungsraumes bewegen.

The major strategy for genetic algorithms to handle constraints is to impose penalties on individuals that violate them. The reason is that for heavily constrained problems we just cannot ignore illegal offspring - otherwise the algorithm would stay in one place most of the time. At the same time, very often various decoders or repair algorithms are too costly to be considered, since the effort to construct a good repair algorithm is similar to the effort to solve the problem.

([MIC96], Seite 165)

2.6.1 Erzeugung lokaler Minima durch Fehlkodierung

Wie wichtig die sorgfältige Überprüfung der Eigenschaften einer Kodierung ist, soll das folgende Beispiel illustrieren.

Als bemerkenswertes Resultat dieses Beispiels wird sich zeigen, daß für dieses Problem je nach Wahl der Kodierung sich Fitneßlandschaften unterschiedlicher Komplexität ergeben, und zwar in Abhängigkeit ihrer Kodierung bzw. der Ordnung der Elemente des Kodierungsraumes (siehe [MOS95], Seiten 1-6)

Gegeben sei eine einfache diskrete Funktion $f(n) = (n-8)^2$, die auf der folgender Teilmenge der natürlichen Zahlen $D = \{0 \ldots 15\} \subset \mathbb{N}$ definiert, $f: \{0 \ldots 15\} \rightarrow \mathbb{N}$. Der Wertebereich von *f* ergibt sich damit zu $f \in W = \{0, 1, 4, 9, 16, 25, 36, 49, 64\}$.

1. Kodierung durch natürliche Zahlen:

Bewegt man sich auf dem Definitionsbereich linear in der natürlichen Ordnung von $\mathbb{N}$, so stellt sich dieser Suchraum als einfaches Optimierungsproblem mit einem einzigen globalen Optimum dar. Offensichtlich ist das globale Minimum f^* bei $n^* = 8$:

Für alle $i \in \mathbb{N}$ ist $f(n^* + i) > f^*$ und $f(n^* - i) > f^*$.

Mit diesen Voraussetzungen wird man beispielsweise mit einem Hillclimbing-Algorithmus das Optimum zu bestimmen können. Für alle Punkte $n' \in D$ mit $f(n'+1) > f(n')$ und $f(n'-1) > f(n')$, also für alle weiteren Optima in *D*, gilt $n' = n^*$.

Das heißt, es existiert kein lokales Optimum in diesem Raum.

2. Binärkodierung[1]:

Bei dieser Repräsentation sollen nun alle im Definitionsbereich vorkommenden Zahlen als vierstellige Binärzahlen aufgefaßt werden, wobei jede Zahl als ihr Binärzahläquivalent dargestellt wird. Der Null entspricht also 0000_2, der Sieben 0111_2 und der Fünfzehn 1111_2. Der Definitionsbereich präsentiert sich nun als $\{0000_2 \ldots 1111_2\} = \mathbb{B}_4$, das Optimum der Funktion *f* liegt natürlich auch hier bei $f(1000_2) = 0$.

Man beachte, daß sich weder die Größe des Definitionsbereichs noch deren (Dezimal-)Werte geändert haben, sondern ausschließlich die Repräsentation des Definitionsbereichs bzw. die Anordnung der Zahlen darin. Die Funktion *f* bleibt in ihrer Form bestehen.

Die Werte des Definitionsbereichs werden nun folgendermaßen angeordnet: Nachbarn eines Punkts *x* sollen alle diejenigen Punkte *x'* sein, die durch Umkippen eines Bits aus *x* hervorgehen. Die Menge der Nachbarn *N* des Punktes $p = 0101_2$ ist beispielsweise $N(p) = \{1101_2, 0001_2, 0111_2, 0100_2\}$.

Um die Suchraumstruktur zu veranschaulichen, kann man sich die Punkte als Eckpunkte eines vierdimensionalen Hyperkubus vorstellen, dessen Kanten jeweilige Nachbarn verbinden. Abbildung 2.22 stellt diesen Sachverhalt anschaulich dar. Wie man sieht, ist die lineare Nachbarschaftsstruktur verlorengegangen; die Ordnung ist hochkompliziert.

Man betrachte nun die Struktur der Fitneßfunktion. Neben dem globale Optimum existiert nun noch ein zweites, lokales Minimum an der Stelle $n_l = 0111_2$. Der Funktionswert dieser Stelle beträgt $f(n_l) = 1$.

Die Nachbarn von n_l sind $N(p) = \{1111_2, 0011_2, 0101_2, 0110_2\}$, für Ihre Werte gilt:

$f(1111_2) = 64$, $f(0011_2) = 25$, $f(0101_2) = 9$ und $f(0110_2) = 4$.

Man beachte, daß der Hillclimbing-Algorithmus mit Startpunkt n_l sofort terminiert.

.

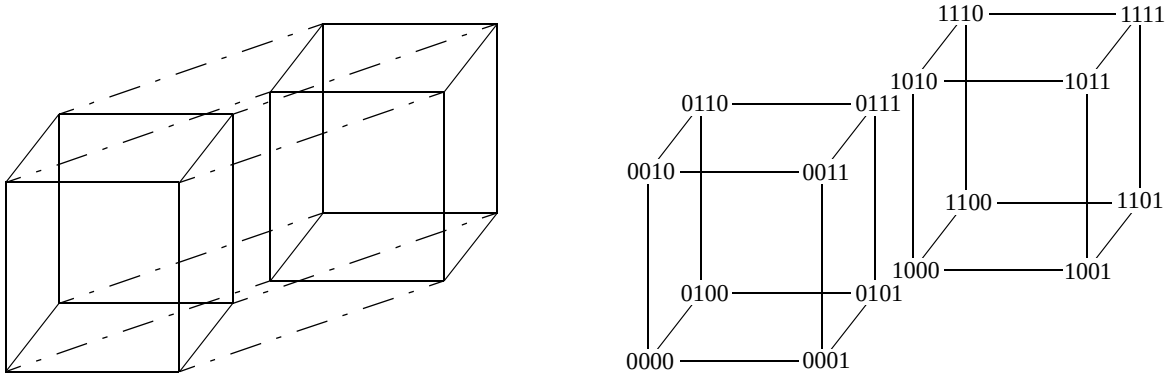


Abbildung 2.22: *Vierdimensionaler Hyperkubus[2] zur Darstellung der Nachbarschaftsbeziehungen.*

1. Im folgenden sei $\mathbb{B}_4$ die Menge der vierstelligen Binärkodes.
2. Der vierdimensionale Hyperkubus entsteht durch das Verbinden der Eckpunkte zweier dreidimensionaler Würfel, so daß jeder Eckpunkt schließlich mit vier Nachbarn verbunden ist. Ihn zeichnet die Möglichkeit einer binären Numerierung seiner Eckpunkte aus. Diese erfolgt hier derart, daß jede Ecke direkt mit allen denjenigen Ecken über eine Kante verbunden ist, die sich durch Änderung eines einzelnen Bits seiner Kodierung ergeben. Alle 1-Bit-Sprünge sind also Nachbarn.

2.6.2 Kodierung beim Linearen Transportproblem (LTP)

Das LTP wurde bereits als Beispiel für Optimierungsprobleme in Abschnitt 2.2.1 vorgestellt. Nun soll die Konstruktion einer geeigneten Kodierung zur Implementierung eines effizienten EA kurz skizziert werden. Für eine ausführliche Diskussion sei auf Anhang A verwiesen.

Ein klassischer GA würde eine Umkodierung der Matrix $M \rightarrow \{0, 1\}^q$ vornehmen und darauf mit Standard Operatoren (Crossover, Mutation) arbeiten. Da aber das LTP einen sehr eingeschränkten Suchraum besitzt, entstünden dabei viele ungültige Individuen. Insbesondere würde jede 1-Bit-Mutation eines Gen-Strings eine ungültige Lösung liefern.

Beim LTP empfiehlt es sich, die Matrixrepräsentation als Datenstruktur beizubehalten, und genetische Operatoren zu entwerfen, die den Lösungsraum nicht verlassen können.

Die grundlegende Idee besteht im Entwurf einer Funktion, die zu gegebenen Randbedingungen (hier: Materialvorräte der Quellen bzw. Materialbedürfnisse der Senken) Lösungsmatrizen generiert. (Abbildung 2.23)

Diese Funktion ist damit als Initialisierungsoperator der Ausgangspopulation geeignet. Um einen Mutationsoperator zu entwerfen, wird eine geeignete Form für Teilmatrizen definiert. Diese werden zur Laufzeit zufällig generiert und die Veränderung besteht dann darin, diese Teilmatrizen mittels des Initialisierungsoperators neu zu berechnen. Damit erreicht man, daß kleine Änderungen auf den Individuenmatrizen ebenfalls wieder gültige Lösungen des Problems darstellen.

Für eine ausführliche Darstellung eines Rekombinationsoperators sei auf den Anhang verwiesen.

```
procedure initialize
begin
  set all x_{i,j} as 'unvisited'
  while ( i,j exist with x_{i,j}= 'unvisited')
  begin
    choose i,j arbitrarily with x_{i,j} 'unvisited'
    set val= min( source'(i), dest'(j) )
    x_{i,j}= val
    source'(i)= source'(i) - val
    dest'(j)= dest'(j) - val
  endwhile
end
```

Abbildung 2.23: *Der Initialisierungsoperator*

Initialize wird mit Kopien *source*' und *dest*', den call-by-value-Namen von *source* und *dest*, aufgerufen, da diese von der Funktion geändert werden. Offensichtlich terminiert diese Funktion mit einem gültigen Zustand für M. Allerdings können nicht alle gültigen Lösungen erzeugt werden, sondern nur solche, die höchstens *k+n-1* Elemente enthalten, die verschieden von Null sind, wobei k die Anzahl der Quellen und n die Zahl der Sourcen darstellt.

Ein Aufruf von initialize mit folgender Initialmatrix *M*,

x	d1	d2	d3	d4
s1	0	0	0	0
s2	0	0	0	0
s3	0	0	0	0

source = (5, 25, 5), *dest* = (5, 15, 15, 10), könnte beispielsweise die Resultatsmatrix *M'*

x'	d1	d2	d3	d4
s1	0	0	15	0
s2	5	10	0	10
s3	0	0	5	0

liefern, *fit(M')*=700, *source'* = (0, 0, 0), *dest'* = (0, 0, 0, 0), unter Bezugnahme auf das bereits bekannte Beispiel aus Abschnitt 2.2.1 .

Eine ähnliche Vorgehensweise wird zur für Praktische Probleme interessanten Lösung des Nichtlinearen Transportproblems bestritten, da bisher keine Vorwärtsstrategien für dessen Lösung bekannt sind. Eine detailliertere Beschreibung dessen entzieht sich allerdings dem Rahmen dieses Textes. Für Interessenten sei an dieser Stelle auf [MIC96], Seiten 196ff verwiesen.

2.6.3 Kodierung beim Travelling Salesman Problem (TSP)

Auch beim TSP hat man das Bestreben, die Tourrepräsentation möglichst so zu wählen, daß die Punkte Korrektheit, Vollständigkeit und Minimalität erfüllt werden.

Hier soll eine Auswahl an Repräsentationen und Operatoren vorgestellt werden um die Problematik einer geeigneten Wahl der Datenstrukturen zu veranschaulichen. ([MIC96], Kapitel 10)

Die Pfadrepräsentation

Hierbei wird eine Tour von *n* Städten durch eine geordnete Liste ihrer Nummern dargestellt (Permutation). Somit ist jede Permutationsliste eine gültige Lösung des TSPs. Also sollten genetische Operatoren derart implementiert werden, daß ausschließlich Permutationen der Ursprungsliste generiert werden.

Beispielsweise sind zwei erlaubte Touren von 9 Städten in dieser Repräsentation so darzustellen:

`a=(123456789), b=(452187693)`

Die Pfadrepräsentation ist eine recht einfache Darstellung, allerdings sind die Mutations- und Rekombinationsoperatoren teilweise recht aufwendig. Deshalb wurde nach anderen Datenstrukturen zur Lösung dieses Problems gesucht. Eine weitere günstige ist die Darstellung mittels einer Matrix:

Die Matrixrepräsentation

Bei der MR wird eine Tour in einer (n x n)-Matrix gespeichert, die sich wie folgt ergeben:

Das Element $x_{i,j}$ der Matrix enthält genau dann eine ’1’ falls die Stadt i vor der Stadt j in der Tour vorkommt. Ansonsten enthält die Matrix Null.

Für M gelten folgende Eigenschaften:

i) $|M|_{\#1} = n(n-1)/2$ [1] **(2.6.1)**

ii) Für die Diagonalelemente $x_{i,i}$ von M gilt:
$x_{i,i} = 0$ für alle $i \leq n$ **(2.6.2)**

iii) Es gilt die Transitivität: $x_{i,j} = 1$ und $x_{j,k} = 1$
impliziert $x_{i,k} = 1$ für alle $i, j, k \leq n$ **(2.6.3)**

Beispiel: Die Matrixdarstellung[2] der Tour `c=(312874695)`:

x	1	2	3	4	5	6	7	8	9
1		1		1	1	1	1	1	1
2				1	1	1	1	1	1
3	1	1		1	1	1	1	1	1
4					1	1			1
5									
6					1				1
7				1	1	1			1
8				1	1	1	1		1
9					1				

Trotz der aufwendigeren Datenstrukturmodellierung in der Matrixdarstellung hat dies eindeutige Vorzüge:

Die Rekombinationsoperatoren sind vereinfacht gegenüber denen der Pfaddarstellung und erreichen damit eine wesentlich schnellere Abarbeitung des Algorithmus. Die Matrix ermöglicht nicht nur den Zugriff auf die Position einer Stadt in der Folge (Spaltensumme), sondern liefert auch noch alle Vorgängerstädte bzw. Nachfolgerstädte. Damit enthält dieses Modell einen höheren Informationsgehalt über die Folge als das Pfadmodell.

Zur genaueren Beschreibung der Operatoren für die alternativen Repräsentationen beim TSP sei auf Anhang B verwiesen.

Wie aus den letzten Abschnitten ersichtlich wurde, ist die Entwicklung einer problemspezifischen Datenstruktur ein wichtiger Punkt bei der Entwicklung effizienter Optimierungsstrategien. Entscheidend ist Operatoren zu definieren, die den Lösungsraum nicht verlassen können und somit die Implementierung von Korrekturoperatoren vermeiden.

Wird die externe Strukturen als interne Datenstruktur übernommen, gleicht also der Genotyp dem Phänotyp, so kann auf die Implementierung von *Umkodierungsoperatoren* verzichtet werden. Häufig macht dies aber eine Konstruktion komplexerer genetischer Operatoren notwendig.

1. wobei $|M|_{\#1}$ die Gesamtzahl der Einsen der Matrix darstellt.
2. Die Nullen sind der Übersichtlichkeit halber in der Matrix ausgelassen.

Meist ist es empfehlenswert, den erhöhten Kodierungsaufwand und die Umkodierung in Kauf zu nehmen um einfache und effiziente Operatoren definieren zu können.

2.7 Konvergenzverhalten der (1+1)-ES

Theoretische Aussagen über Evolutionsalgorithmen allgemein sind in der Literatur kaum verfügbar. Jedoch läßt sich zeigen, daß die einfachste Version dieser Algorithmen, die (1+1)-ES, schon ein für Optimierungsprobleme konvergiert, die nur sehr leicht eingeschränkt sind. Aus der Tatsache, daß komplexere Varianten auf dieser basieren und durch zusätzliche Konzepte erweitert sind, läßt sich das Verhalten der (1+1)-ES extrapolieren.

Die eingeschränkten Problemstellungen, die hier untersucht werden, sind *reguläre Optimierungsprobleme*, welche die Eigenschaft besitzen, daß eine ganze Epsilonumgebung des Optimum-Funktionswerts vorhanden ist. Diese leichte Einschränkung sorgt dafür, daß die Wahrscheinlichkeit, diese Menge durch Mutation zu erreichen, größer als Null ist. Bei langer Programmlaufzeit wird dies erwartungsgemäß auch geschehen. Da dies für beliebig große Umgebungen gilt, muß das Optimum schließlich erreicht werden.

Definition Definition 2.3 stellt die formelle Beschreibung der (1+1)-ES dar und in Definition 2.4 werden reguläre Optimierungsproblem mathematisch definiert.

Im Anschluß an diesen Abschnitt wird noch ein Beispiel dafür gegeben, daß nichtreguläre Optimierungsprobleme existieren, die aufgrund ihrer prinzipiellen Struktur mit keinem Optimierungsverfahren lösbar sind, gemäß der Einführung in diese Problemstellungen in Abschnitt 2.1.

Im folgenden seien definiert:

$U_\varepsilon(x) \stackrel{def}{=} \{y \mid \|y - x\| < \varepsilon\}$ und $support(M) \stackrel{def}{=} \{x \in M \mid f(x) \neq 0\}$.

Definition 2.3: (1+1)-ES

Seien $M \subset \mathbb{R}^n$, $fit : M \to \mathbb{R}$, $fit(x^*) = min\{fit(x) \mid x \in M\}$ existent,
$mut(x) \stackrel{def}{=} x + Z$ Mutationsoperator, $Z \sim N(0, \sigma^2 I_n)$ normalverteilte Zufallsvariable.
$\hat{x}^{(t)}$ sei ein Wert, der durch Mutation aus $x^{(t)}$ entsteht, also $\hat{x}^{(t)} \stackrel{def}{=} mut(x^{(t)})$.

Dann ist die Folge $x^{(t)}$ der Werte der (1+1)-ES nach folgender Berechnungsvorschrift in Form einer Rekurrenzrelation bestimmt:

$x^{(0)} \in M$ sei beliebiger Startwert.

$$x^{(t+1)} \stackrel{def}{=} \begin{cases} \hat{x}^{(t)}, & \text{falls } fit(\hat{x}^{(t)}) > fit(x^{(t)}) \\ x^{(t)}, & \text{sonst} \end{cases} \qquad \textbf{(2.7.1)}$$

Definition 2.4: reguläres Optimierungsproblem

Sei im folgenden μ das Lebesguesche Maß.

Ein Optimierungsproblem $(M, fit, \{g_i\}_{0 \le i \le n})$, $fit(x^*) = min\{fit(x) \mid x \in M\}$ heißt *regulär* genau dann, wenn folgende Bedingungen gelten:

i) $fit(x^*) > -\infty$ **(2.7.2)**

ii) $x^* \in int(M)$, mit
$int(M) \stackrel{def}{=} \{x \in M \mid \exists \varepsilon > 0 : U_\varepsilon(x) \subset M\}$ **(2.7.3)**

iii) $\forall \varepsilon > 0 : \mu(M_\varepsilon) > 0$, mit
$M_\varepsilon \stackrel{def}{=} \{x \in M \mid fit(x) \in U_\varepsilon(fit(x^*))\}$ **(2.7.4)**

Satz (Konvergenz der (1+1)-ES)

Sei $(M, fit, \{g_i\}_{0 \le i \le n})$ ein reguläres Optimierungsproblem und gelte o. B. d. A. $x^* \in M$ für das Optimum x^*, mit $fit(x^*) = min\{fit(x) \mid x \in M\}$. Weiterhin sei M beschränkt.

Dann gilt:

$$P[lim_{t \to \infty}(fit(x^{(t)}) - fit(x^*)) = 0] = 1, \quad \textbf{(2.7.5)}$$

also $lim_{t \to \infty}(fit(x^{(t)}) - fit(x^*)) = 0$ sicheres Ereignis.

Beweis:

Die $x^{(t)}$ bilden eine Markov-Kette im Zustandsraum M. Für $x \in M$ und $A \subseteq M$ sei $P_x(A)$ die Wahrscheinlichkeit, daß der Prozeß in einem Schritt aus dem Punkt x in die Menge A springt, d.h. $P_{x^{(t)}}(A) = P[x^{(t+1)} \in A]$.

Sei $\varepsilon > 0$ beliebig. Dann ist $x^* \in M_\varepsilon \subseteq M$.

ν_x sei die Kurzschreibweise für $N(x, \sigma^2 I_n)$, also der Gaußverteilung mit Erwartungswert x.

Sei $x^{(t)} \in M$ der Wert eines beliebigen Schritts des Verfahrens.

Für $x^{(t)}$ können folgende Fälle unterschieden werden:

1. $x^{(t)} \notin M_\varepsilon$

   Dann[1] gilt für alle $m \in M_\varepsilon : fit(m) < fit(x^{(t)})$, also $P_{x^{(t)}}(M_\varepsilon) = \nu_x(M_\varepsilon)$.

   Nach Eigenschaft (iii) eines regulären Optimierungsproblems gilt: $\mu(M_\varepsilon) > 0$.

   Da M beschränkt ist gibt es für alle $x \in M$ ein $\alpha > 0$ mit $\nu_x(M_\varepsilon) \ge \alpha$.

---

1. $x \notin M_\varepsilon \Rightarrow x \notin \{x \in M \mid fit(x) \in U_\varepsilon(fit(x^*))\} \Rightarrow fit(x) \notin U_\varepsilon(fit(x^*))$
   $\Rightarrow \forall m \in M_\varepsilon : fit(m) < fit(x)$

Also gilt $P[x^{(t+1)} \in M_\varepsilon \mid x^{(t)} \notin M_\varepsilon] = \nu_{x^{(t)}}(M_\varepsilon) \geq \alpha$ und damit

$$P[x^{(t+1)} \notin M_\varepsilon \mid x^{(t)} \notin M_\varepsilon] < 1 - \alpha \qquad \textbf{(2.7.6)}$$

2. $x^{(t)} \in M_\varepsilon$

Aufgrund der Konstruktionsvorschrift der Folge $\{ x^{(t)}\}_{t \in \mathbb{N}}$ (2.7.1) liegt notwendigerweise $x^{(t+1)}$ auch wieder in M_ε. Das heißt

$$P[x^{(t+1)} \in M_\varepsilon \mid x^{(t)} \in M_\varepsilon] = 1 \qquad \textbf{(2.7.7)}$$

Hier ist nichts zu zeigen.

Für $P[\forall t : x^{(t)} \notin M_\varepsilon]$, also der Wahrscheinlichkeit, daß die Folge $\{ x^{(t)}\}_{t \in \mathbb{N}}$ M_ε niemals erreicht, gilt:

$$\begin{aligned} & P[\forall t : x^{(t)} \notin M_\varepsilon] \\ &= \prod_{t \in \mathbb{N}} P[x^{(t+1)} \notin M_\varepsilon \mid x^{(t)} \notin M_\varepsilon] \\ &\leq \prod_{t \in \mathbb{N}} (1 - \alpha) \quad \text{(wg.(2.7.6))} \\ &= 0, \quad \text{da } (1 - \alpha) < 1 . \end{aligned}$$

Damit ist auch $P[\forall t : x^{(t)} \notin M_\varepsilon] = 0$.
Für das Gegenereignis gilt $P[\exists t : x^{(t)} \in M_\varepsilon] = 1 - P[\forall t : x^{(t)} \notin M_\varepsilon] = 1$.

Damit ergibt sich für die Behauptung des Satzes

$$\begin{aligned} & P[lim_{t \to \infty}(fit(x^{(t)}) - fit(x^*)) = 0] \\ &= P[\forall \varepsilon > 0 : \exists t : \forall \hat{t} > t : x^{(\hat{t})} \in M_\varepsilon] \\ &= P\left[\forall n \in \mathbb{N} : \exists t : \forall \hat{t} > t : x^{(\hat{t})} \in M_{n^{-1}}\right] \\ &= P\left[\bigcap_{n \in \mathbb{N}} \left\{\exists t : \forall \hat{t} > t : x^{(\hat{t})} \in M_{n^{-1}}\right\}\right] \\ &= lim_{n \to \infty} P\left[\exists t : \forall \hat{t} > t : x^{(\hat{t})} \in M_{n^{-1}}\right] \qquad \textbf{(2.7.8)} \end{aligned}$$

Wegen (2.7.7) ist $P\left[\exists t:\ \forall \hat{t} > t:\ x^{(\hat{t})} \in M_{n^{-1}}\right] \quad = \quad P[\exists t:\ x^{(t)} \in M_{n^{-1}}]$ und somit

$$lim_{n \to \infty} P\left[\exists t:\ \forall \hat{t} > t:\ x^{(\hat{t})} \in M_{n^{-1}}\right] \quad = \quad lim_{n \to \infty} P[\exists t:\ x^{(t)} \in M_{n^{-1}}] \quad = \quad 1$$

Und damit folgt die Behauptung □

Definition 2.5: polynomiale und exponentielle Konvergenz

$\delta_t = E[fit(x^{(t)}) - fit(x^*)]$ heißt erwarteter Fehler in Schritt *t*.

Ein Algorithmus konvergiert *polynomial*, falls $\delta_t = O(t^{-\alpha})$ für ein $\alpha > 0$.

Ein Algorithmus konvergiert *exponentiell*, falls $\delta_t = O(\beta^t)$ für $0 < \beta < 1$.

Theorem

Ist $fit: M \to \mathbb{R}$ strikt konvex und die Mutationsschrittweite sphärisch verteilt, also $Z = RU$, U gleichverteilte Zufallsvariable auf der Einheitskugel und R gleichverteilte Zufallsvariable auf $(0, a)$, $support(0, a) \neq \varnothing$, $x^{(0)} \in M$ beliebig.

Dann ist der erwartete Fehler δ_t im Schritt *t* bei konstanter Mutationsschrittweite von Ordnung $\delta_t = O(t^{-\alpha})$ bei adaptiver Schrittweite $\delta_t = O(\beta^t)$, $0 < \beta < 1$, $R^{(t+1)} = \left\|\nabla fit(x^{(t)})\right\| R^{(t)}$.

Zum Beweis und zu weiteren Ausführungen zur Konvergenzgeschwindigkeit siehe auch [SCH95] und [RUD95].

Bemerkung

Speziell soll hier nochmals auf Punkt (2.7.4) der Definition 2.4 (reguläres Optimierungsproblem) eingegangen werden. Dieser stellt die Forderung einer gewissen "Gutartigkeit" des Problems, nämlich die Existenz ganzer μ-meßbaren Epsilonumgebungen $U_\varepsilon(fit(x^*))$ um das Optimum, sogenannter Epsilon-Niveaumengen. Deren Existenz garantiert, daß die Wahrscheinlichkeit solch eine Niveaumenge zu erreichen immer größer als Null ist. Damit existiert eine Art Pfad für die Evolution, der sie Schritt für Schritt zum Optimum führt.

Um ein Beispiel für einen einfachen Fall zu geben, für den die ES kapituliert, nehme man die Fitnessfunktion

$$fit: M \to \mathbb{R},\ fit(x) = 1_{P \in M},\ M = [0, 1]^k \subset \mathbb{R}^k,$$

welche Eins gleicht für einen bestimmten Punkt $P \in M$ und ansonsten den Wert Null annimmt. Dies entspricht dem *golf course problem*, das schon im ersten Abschnitt erwähnt wurde. Wie man sieht, erfüllt diese Funktion Forderung (2.7.4) nicht.

Für die Wahrscheinlichkeit *p*, dieses Optimum zu erreichen, gilt

$$p = P[x \in M_\varepsilon] = P[x \in \{P\}] = \int_{\{P\}} f_Z(\xi)\, d\xi = 0$$

D.h. auch eine beliebig lange Laufzeit des Algorithmus wird das Erreichen des Optimums nicht garantieren können. Anschaulich gesehen hat die Evolution bei einer völlig flachen Landschaft

keine Möglichkeit aus der Topologie der Fitneßlandschaft auf die Lokation des Optimums zu schließen; sie kann es nur erreichen, indem sie zufällig die gesuchte Stelle x^* trifft. Die Wahrscheinlichkeit dieses Ereignisses ist allerdings gleich Null.

In der Praxis wird man häufig “gutartige” Fitneßfunktionen benutzen, für die uneingeschränkt alle (wenig restriktiven) Forderungen eines regulären Optimierungsproblems gelten, so daß obige Probleme nicht auftauchen.

2.8 Erweiterungen und Modifikationen der EA

In diesem Abschnitt werden mögliche Erweiterungen des Evolutionsalgorithmenberiffs besprochen. Zum einen wird aufgezeigt, wie die Berechnungen parallelisiert werden können um damit das Feld leistungsstarker Parallelrechner bzw. Rechnernetzwerke für die Evolution zu erschließen. Zum anderen wird durch Verbesserungen des Evolutionsmodells bzw. der genetischen Operatoren eine Steigerung der Effizienz erreicht.

2.8.1 Migrationsmodell

Beim *Populationenmodell* oder *Migrationsmodell* werden mehrere konkurrierende Populationen parallel, auf denen unabhängige Evolutionsprozesse laufen, berechnet.

Einzelne Populationen ausschließlich durch einen gelegentlichen Austausch von einzelnen Individuen, die Lösungsinformationen von einer Population zur nächsten transportieren. Dabei ist keinerlei Synchronisation der einzelnen Prozesse bzw. Computer notwendig; Individuen könenn zu beliebigen Zeitpunkten ausgetauscht werden.

Effektiv wird durch diese Form der Populationenevolution eine breitere Basis des Suchbereichs geschaffen, die i.e. der Gesamtzahl der Individuen aller Populationen entspricht. Allerdings ist der Ansatz keinesfalls äquivalent zu einer monopopularistischer Evolutionsstrategie mit gleicher Anzahl von Individuen. Durch die große Isolation der Populationen bilden sich unabhängige Familien mit einem gewissen *Verwandtheitsgrad* heran, die also untereinander weniger unterschiedlich voneinander sind , als von Individuen anderer Populationen.

Migriert ein Individuum in eine wesentlich besser bewertete Population, so wird es mit großer Sicherheit nicht zur Bildung von Nachkommen beitragen. Trotzdem hat es die Möglichkeit mit Hilfe des genetischen Rekombinations-Operators einen Teil seiner Parameter an Nachkommen zu weiterzuvererben, so daß ggf. (gut angepaßte) Teile seines Erbguts erhalten bleiben.

Kommt umgekehrt ein besseres Individuum in eine schlechtere Population, so überlebt es selbst (Kommastrategie) und kann einen Großteil seiner hochqualitativen Parameter an seine Nachkommen weiter. Nach wenigen Generationen ist die Population je nach Fitneßunterschied zum

migrierten Individuum durch dessen Gene geprägt, so daß sich eine entscheidende Verbesserung und dadurch ein Abgleich der Qualität der Populationen einstellt.

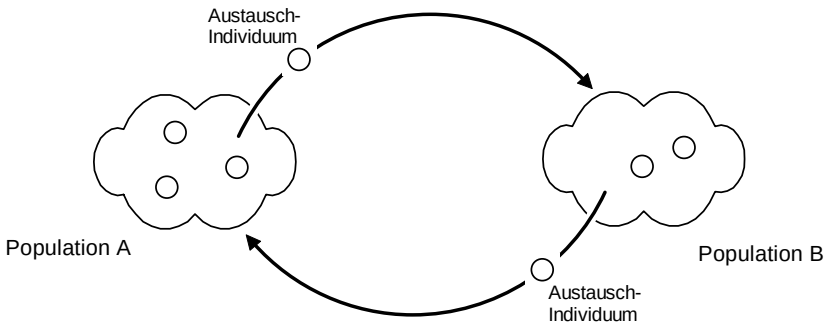


Abbildung 2.24: *Das Populationsmodell erlaubt die parallele Evolution mehrerer Populationen, die durch den Austausch von Inidviduen in Kontakt stehen.*

Das Migrationsmodell[1] eignet sich hervorragend zum Einsatz von Parallelrechnerarchitekturen bzw. Rechnernetzwerken; jede Population wird dann in einem eigenen Prozeß bzw. auf einem separaten Rechner mittels einem Standard-Evolutionsalgorithmus berechnet und gelegentlich werden Individuen ausgetauscht. Da aufgrund der Stabilität der Verfahren auf die Synchronisation der Einheiten verzichtet werden kann bremsen Wartezyklen die Prozessoren nicht aus. Eine optimale Auslastung der einzelnen Recheneinheiten ist damit gewährleistet. Selbst die fehlerhafte Übertragung von Daten wird den Algorithmus nicht weiter stören. Ein verfälscht empfangener Datenstrom, der als Individuum interpretiert wird, kann man als eine andere Art von Mutation auffassen, die evtl. sogar als Element des Verfahrens selbst interpretiert werden kann.[2]

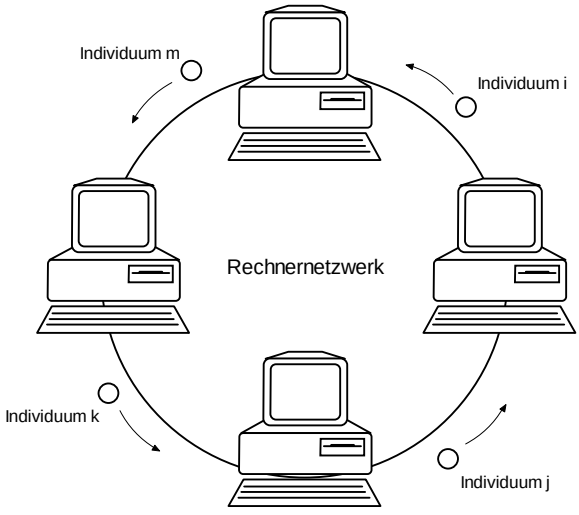


Abbildung 2.25: *Populationenmodell in einem Rechnernetzwerk*

1. Im Rahmen dieser Diplomarbeit wurde die Software auch auf die Bedürfnisse des Migrationsmodells hin ausgerichtet. Erste Testläufe wurden erfolgreich auf einer 2-Prozessor-Maschine durchgeführt.
2. Interessant wäre die Konzeption eines parallelen *Evolutionsrechner*s, der aufgrund von mangelhafter Datenübertragung Evolution betreibt. Gestörte Leitungen wären dabei die Hardwarerealisierung eines Mutationsoperators.

2.8.2 Modifikationen des Standardverfahrens

An dieser Stelle sei ein Überblick über mögliche Modifikationen der Standardverfahren gegeben.

Alterung von Individuen

Bei dieser Erweiterung der (n+m)-Evolutionsstrategie erhält jedes Individuum einen Zähler, der mit dem Einbringen desselben in die Nachfolgepopulation um eins erhöht wird, also seinem *Alter* entspricht. Überschreitet dieser Zähler einen bestimmten Wert darf es nicht mehr an der Evolution teilnehmen, es hat demnach eine beschränkte Lebenserwartung.

Diese Variante stellt eine Art Mittelweg zwischen Komma- und Plusstrategie dar. Liegt die Schwelle bei eins, so entspricht es der ersteren, ist sie unendlich der letzteren.

Variable Populationsgröße

Durch eine variable Populationsgröße, die beispielsweise abhängig von der Varianz innerhalb der Population ist, optimiert sich der Algorithmus bezüglich seiner Laufzeit. Ist sie anfänglich sehr groß und erfolgt dadurch eine Parallelsuche im Suchraum, schrumpft sie im Laufe der Generationen immer mehr im Zuge des Varianzverlußtes auf ein minimales Ausmaß. In der Spätphase der Optimierung ist die Varianz in der Population auf einen kleinen Wert geschrumpft, bei dem die Rekombination kaum noch einen zusätzlichen Gewinn bringt.

Paarungsverbot ähnlicher Individuen

Bei dieser Variante dürfen zur Selektion keine Individuen gewählt werden, die sich zu ähnlich sind, also im Suchraum zu dicht beieinander liegen. Damit erhält man die Innovativität über einen längeren Zeitraum, als beim Standardverfahren und ermöglicht auch eine Varianzerhaltung in der Population.

Paarungsverbot sehr verschiedener Individuen

Geht man den umgekehrten Weg und verbietet die Paarung von Individuen, welche im Suchraum zu weit auseinanderliegen, dann kommt es zu einer Ausbildung verschiedener Arten innerhalb einer Population. Diese tauschen sich nur noch in wenigen Schnittstellen aus und durchsuchen relativ unabhängig verschiedene Bereiche des Suchraumes.

Mehrgeschlechtlichkeit

Die Mehrgeschlechtlichkeit ist eine weitere Variante, die dem natürlichen Vorbild entstammt. Statt uniformer Individuen existieren hierbei mehrere Individuentypen, deren Rekombinationsmöglichkeiten auf ein bestimmtes Schema eingeschränkt sind.

2.8.3 Memetische Algorithmen

Das Konzept der *Memen* stellt eine Verallgemeinerung des Evolutionsprinzips dar. In dem Sinne, in dem ein *Gen* die Kodierung eines Erbmerkmals des Menschen ist, so entspricht ein *Mem*[1] einer Übertragung (Vererbung) eines kulturellen Wertes bzw. eines Ideengebildes in einem allgemeinen Sinne dar.

Die Konzeption der Weitergabe dieser Werte oder Verhaltensweisen erfolgt nicht in einem biologischen Kontext, sondern durch Überlieferung von Generation zu Generation durch einen *Lehrmeister*, der seine Schüler zielgerichtet anleitet. Diese intelligente Zielfokusierung fehlt dem klassischen Evolutionskonzept, welches eine Optimierung durch zufällige Änderungen der Entitäten erreicht.

Überträgt man diese Konzeption auf algorithmische Verfahren, so sollten anstelle ungerichteter zufälliger Manipulationsoperatoren zielgerichtete, intelligente konzeptioniert werden, die ein direktes Voranschreiten in Richtung des Optimums ermöglichen. An dieser Stelle bleibt offen, woher der Operator sein Wissen um das Optimum bezieht. Denkbar ist eine Erweiterung genetischer Standardverfahren um heuristische Operatoren, die *lernfähig* sind. So könnte trotz verschiedener Fitneßfunktionen doch das Prinzip über viele Programmläufe erlernt werden.

Um die Idee des Konzepts der Memetischen Algorithmen zu unterstreichen soll folgendes Zitat Hilfe leisten:

> *...but the analogies (to the concept of memes) with the genetic coding and natural selection can not include the mutations. In GA they are considered to be the operator that includes the necessary amount of noise to do hill-climbing, while it is very improbable to find a good improvement in matial arts*[2] *as a consequence of the introduction of some random movements into a form. The process seems to be different. Only the masters have the sufficient knowledge that permits them create a new movement and to incorporate it to the form. And this happens with a very low frequency. So, there is much problem specific knowledge that is applied to each modification. Almost all modifications give improvements rather than create a disorder. This fast-feedback flow of information from high order phenotype knowledge to genotype level, seems to have differences with the processses of biological evolution, but we must consider the latter is constrained with the physical structure of the DNA and their processes are direct consequence of the primitive replicating macromolecules that give its origin.*
>
> *...this is a point where the breaking of the chains of copying-fidelity, combined with the freedom of blending concepts gave the new meme the necessary degree of refinement to even improve the previous one. The raw combination of good ideas is not always a good idea. Ascientist does not pass on an idea after blending it with his own without checking the logic of what he is saying or his reputation would be in trouble.*
>
> ([MOS95], Seite 18-19)

Für eine theoretische Vertiefung der Materie sei auf [RAD94] verwiesen, für nähere Betrachtungen des philosophischen Konzepts von Memen auf [DAW76].

1. meme= *unit of imitation in cultural transmission.*
2. Der Autor erklärt anhand von *Kampfsportarten* die Grenzen der Genetischen Algorithmen und das seiner Ansicht nach übergeordnete Konzept der Memetischen Algorithmen.

3 Menschmodell RAMSIS

RAMSIS ist ein computergestütztes, anthropometrisch-kinematisches Menschmodell. Das heißt, es ist ein 3D-Datenmodell, welches die menschliche Haut und Bewegungsfähigkeit nachbildet. Ursprünglich zu Ergonomiestudien in der Automobilindustrie entwickelt umfaßt die Anwendbarkeit dieses Modells ein breites Spektrum von Anwendungsmöglichkeiten:

- Bequemlichkeits- bzw. Reichweitenanalysen
- Sichtbereichsanalysen
- Kollisionsüberprüfungen/-Vermeidungen im Fahrzeugbereich
- Realitätsnahe Bewegungssimulation
- Anthropometriedatenbank zur statistischen Körperformdimensionierung
- Automatische, statistisch fundierte Generierung von Körpertypen, die auf einer Angabe von drei bestimmenden Maßen basiert.

Die datenbankbasierte Generierung von unterschiedlichen Körpertypen läßt sich durch Angabe dreier Leitparameter erzeugen. Dabei werden aufgrund dieser Daten statistisch korrekte Modelle erzeugt.

Die interne Modellstruktur des RAMSIS basiert auf insgesamt über 1200 Parametern, die eine individuelle Modellformung ermöglichen. Dabei wird zwischen einer äußeren Modellkomponente (*äußeres Modell*) und einer inneren unterschieden. Die äußere bildet die Modell-Haut und prägt das Erscheinungsbild des Modells. Um die kinematischen Fähigkeiten des Menschen nachzubilden wurde ein vollständiges Skelett bestehend aus Knochen und Gelenken entworfen, das Bewegungssimulationen des RAMSIS ermöglicht und als *inneres Modell* bezeichnet wird. Um nahe an das menschliche Vorbild heranzukommen, sind Winkelbeschränkungen vorhanden, die individuell jedes Gelenk seiner Funktionalität entsprechend in der Beweglichkeit einschränken.

3.1 RAMSIS-Körpertypen

Das Modell unterstützt die Generierung von Körpertypen, die sich durch drei Parameter bestimmen lassen. Es sind dies Größe, Alter und Korpulenz. Mittels einer Datenbank werden auf diese Art aus einer Menge von ca. 90 Körpertypen ein Modell generiert, welches einer Person mit diesen Daten am nächsten kommt. Die Datenbasis wurde an der Humboldt Universität in Berlin unter der Leitung von Dr. Greil erstellt. Das Verfahren ist statistisch fundiert, es stützt sich auf die Datenerhebung aus mehrerern Millionen Einzelmessungen.

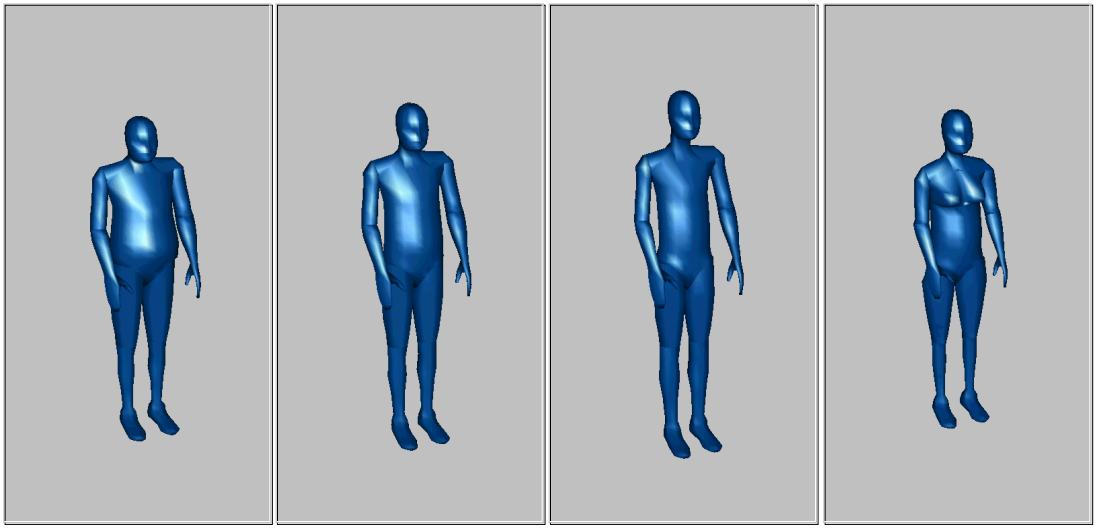


Abbildung 3.1: Drei der RAMSIS-Typen und das Modell einer Frau

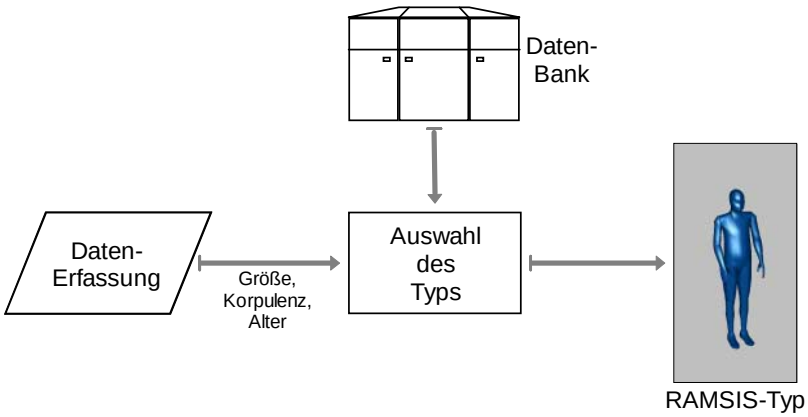


Abbildung 3.2: Auswahl eines der RAMSIS-Typen

3.2 Interner Aufbau

Der Modell-Aufbau stützt sich auf die Basis-Komponenten *Knochen*, *Gelenke* und *Hautscheiben* (RAMSIS-*Elemente*).

Wie angedeutet, wird zwischen einem *äußeren Modell*, welches durch die Hautobjekte definiert ist und eine äußere Erscheinung des Modells realisiert, und einem inneren Modell, welches das kinematische (und nach außen nicht sichtbare) Modellgerüst ausbildet, unterschieden.

Knochen und Gelenke geben als Stützskelett die Motorik des Modells vor. Dabei ist der Spielraum eines jeden Gelenks je nach Funktion eingeschränkt, so daß unrealistische Bewegungen ausgeschlossen werden. Dadurch ist beispielsweise das Kniegelenk wie beim Vorbild Mensch derart in seiner Beweglichkeit eingeschränkt, daß sich das Bein von einem gestreckten Zustand in einen angewinkelten überführen läßt, bei dem sich der Fuß in der Nähe des Gesäßes befindet

Basis des *Hautmodells*, welches für jedes Körperelement definierbar ist, bilden Hautpunkte, deren Anzahl und Lage spezifisch festgelegt ist. Basis der Visualisierung ist eine stückweise lineare Vernetzung dieser Punkte. Diese Modellierungseigenschaft bildet die Grundlage für mögliche Individualisierungen des Modells.

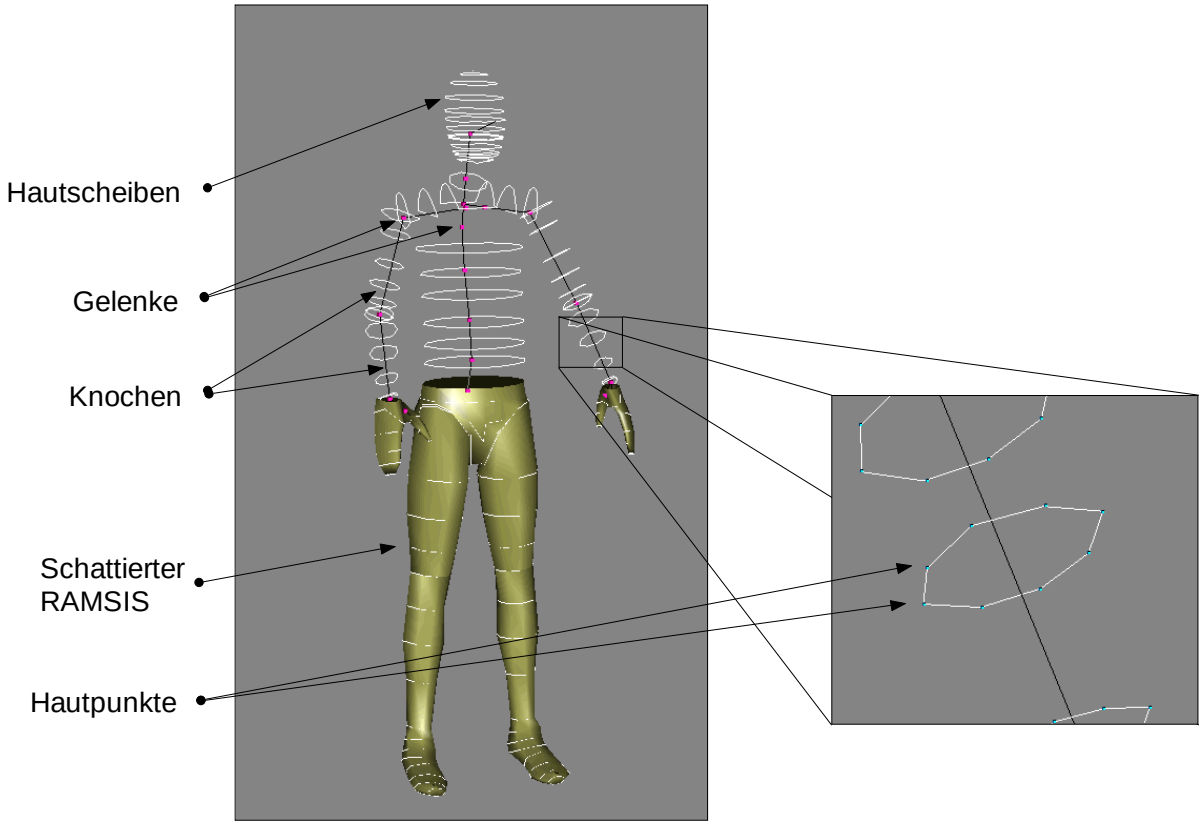


Abbildung 3.3: Aufbau des RAMSIS

Der interne Aufbau des RAMSIS definiert die Modellelemete als Teil einer Baumstruktur, in der hierarchisch geordnet die einzelnen Einheiten miteinander verbunden sind. Mit dieser Hierarchie wird eine Abhängigkeitsbeziehung zwischen den RAMSIS-Objekten definiert.

Jedes Element besitzt ein eigenes lokales Koordinatensystem. Innerhalb dieses Systems gibt es freie Parameter für Längen-, Breiten- und Tiefenabmessungen von Skelett und Haut.

Wurzel dieses Baumes ist das sog. *Startelement (STE)*, das sich im Beckenbereich befindet und die Lage bzw. Verdrehung des Modells im Raum festlegt. Im Anschluß daran befinden sich *Knochenelemente*, die durch *Gelenke* verbunden sind und in den Extremitäten Zehen, Finger und im Kopf enden.

An diesen Knochen befinden sich Hautpunkte, welche die Haut des Modells (also dessen Oberfläche) darstellen. Diese sind zu *Hautscheiben* zusammenfaßbar.

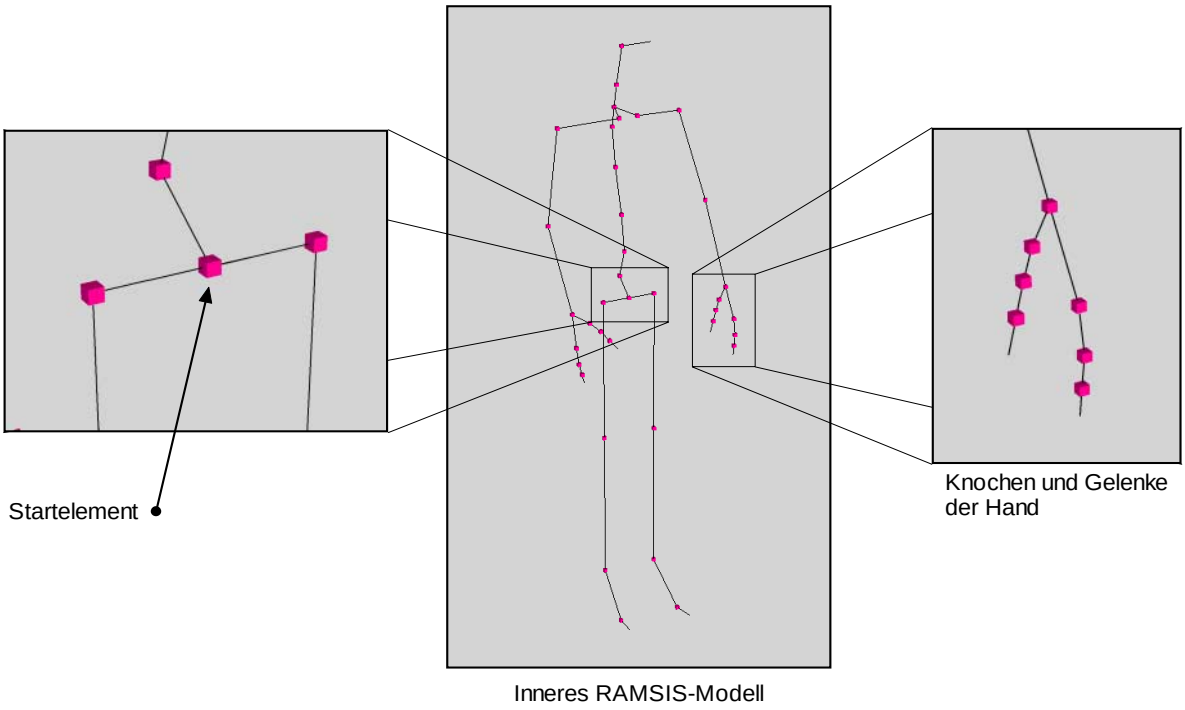


Abbildung 3.4: Knochen und Gelenke beschreiben das innere RAMSIS-Modell.

Die Elemente besitzen ihrer Funktion entsprechend unterschiedliche Parameter. Ein Knochen hat beispielsweise einen freien Parameter (Länge). Eine sich an einem Knochen befindliche Hautscheibe bildet ein räumliches Objekt, welches durch sechs Parameter in seiner Form verändert werden kann (Abbildung 3.5).

Auf der Hautscheibenoberfläche befinden sind einzelne Hautpunkte, die nur in ihrem Abstand vom Knochen verändert werden können. Ein Gelenk kann prinzipiell um drei Freiheitsgrade verdreht werden und besitzt somit drei Parameter für Winkel, die seine Verdrehung in der jeweiligen Dimension bestimmen. Zusätzlich ist diese Verdrehung jeweils durch Anschläge eingeschränkt. Das sind also pro Dimension zusätzlich zwei Parameter, um Winkelbeschränkungen der menschlichen Gelenke zu modellieren. Knie- und Ellenbogengelenke sind gemäß des menschlichen Vorbilds nur in einer Dimension verdrehbar; Schultergelenke sind (in Grenzen) in allen drei Dimensionen veränderbar.

Das Startelement als künstlicher Bezugspunkt hat natürlich keine Entsprechung. Es beschreibt die rotatorische und translatorische Lage des gesamten Maniken im Raum.

Die Lage der Modellelemente ergibt sich immer relativ zu dem in der Baumstruktur übergeordneten Element. Eine Verdrehung der Gelenke der Wirbelsäule zieht also eine absolute Ände-

rung der Arm- und Kopfknochenpositionen nach sich. Eine Verlängerung eines Oberschenkelknochens ändert die Lage der zu diesem Bein gehörigen Objekte usw.

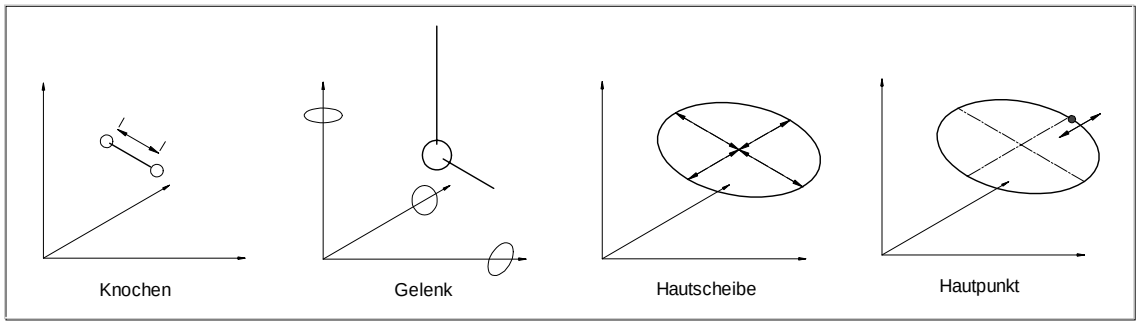


Abbildung 3.5: Die Freiheitsgrade der RAMSIS-Komponenten

Die mannigfaltigen Manipulationsmöglichkeiten des Modells erlauben eine Individualisierung des Modells in einer feinen Granularität. Damit ist RAMSIS prinzipiell gut für den Anwendungszweck des Abgleichs mit einem Scan geeignet.

4 Konzeption und Implementierung des Adaptionsalgorithmus

4.1 Aufgabenstellung

Ziel des TOPAS-Projekts der Universität Kaiserslautern ist die automatische Erfassung der Oberflächengeometrie von Menschen und der anschließende Abgleich mit einem Menschmodell. Ergebnis dieser Datenverarbeitung ist ein Modell, welches sowohl die Oberflächengeometrie der vermessenen Person besitzt als auch die modellinhärenten semantischen Informationen der inneren RAMSIS-Strukturen in sich trägt.

Weitere Verarbeitungsschritte können dann am Individualmodell durchgeführt werden. Die Rohdaten des Erfassungsgeräts sind damit nicht mehr notwendig. Folgende Abbildung veranschaulicht die Verarbeitungsschritte des Systems:

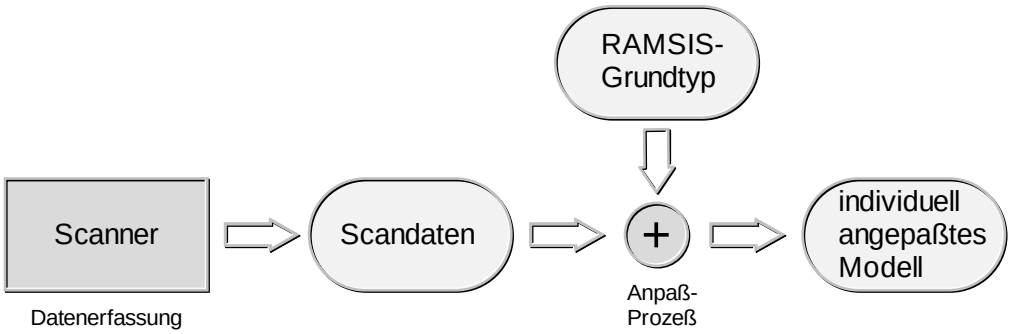


Abbildung 4.1: *Datenverlaufsdiagramm des Menschvermessungssystems*

Ziel dieser Diplomarbeit ist die Konzeption und Implementierung eines adäquaten Anpaßalgorithmus, der als Eingaben die Daten des Erfassungsgeräts und eines RAMSIS-Grundtyps erhält und als Ausgabe ein angepaßtes Modell liefert.

Dabei soll der Algorithmus tolerant gegenüber folgenden Eigenschaften des Scans sein:

- Aufgrund des Aufbaus des Erfassungsgeräts sind einige Bereiche des Vermessungsobjekts abgeschattet, so daß der 3D-Datensatz teilweise unvollständig ist (vergleiche Abbildung 4.13)
- Durch unzureichende Abdunklung der Scankabine kann die Meßwerterfassung gestört werden. Dies kann sich als streifenförmige Artefakte im 3D-Datensatz äußern.

4.2 Dateninterpretation

Allgemein enthalten von einem Eingabegerät stammende Daten keine Semantik. Eine Zahl im Rechner ist zunächst nur eine Folge von Bitwerten, deren Bedeutung durch eine Interpretation mit einer geeigneten Interpretationsfunktion *I* festgelegt wird. Verschiedene Interpretationen ergeben dabei verschiedene Inhalte dieser Daten. Beispielsweise kann ein bestimmtes Datum als Zahl oder als Farbwert interpretiert werden.

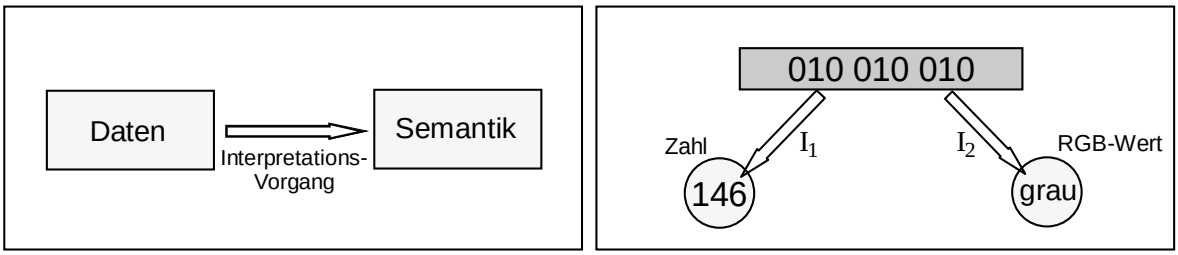


Abbildung 4.2: *Interpretation von Daten*

4.2.1 Modellbasierte Dateninterpretation

Stützt sich die Dateninterpretation auf das Wissen einer modellhaften Beschreibung, so spricht man von *modellbasierter Dateninterpretation*. Dabei können beispielsweise die Parameter des Modells so bestimmt werden, daß sie mit den Meßwerten oder Daten möglichst gut übereinstimmen. Existiert dabei eine feste Zuordungsvorschrift so können diese durch einen einfachen Rechenvorgang erzeugt werden. Ist diese Zuordnung nicht gegeben, existiert jedoch ein Fehlermaß, welches einen Abstand der Modellform zu den Daten angibt, so kann die Parametereinstellung

stellung durch ein Interationsverfahren bestimmt werden, welches die Einstellungen durch Fehlerminimierung bestimmt.

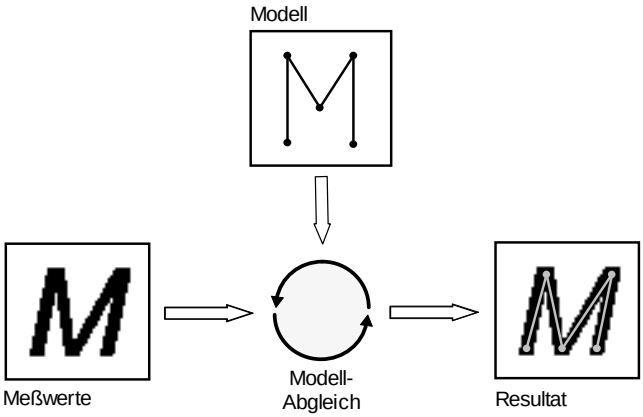


Abbildung 4.3: *Abgleich eines Modells mit Daten*

Enthält der Datensatz kleinere Meßfehler bzw. Unvollständigkeiten, so ändert sich das Fehlermaß i.a. nur wenig und die Anpassung kann trotzdem durchgeführt werden. Das heißt, der iterative Prozeß ist in weiten Grenzen fehlertolerant. .

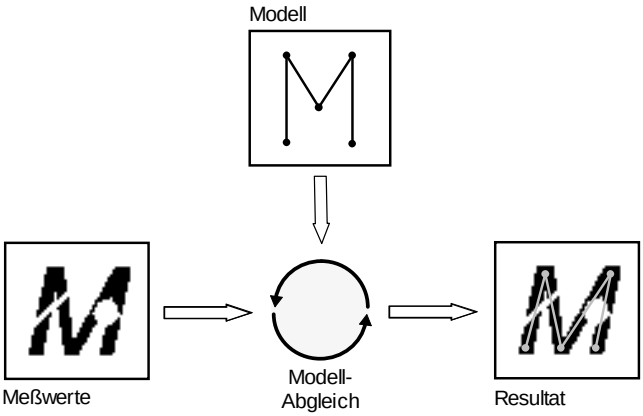


Abbildung 4.4: *Abgleich mit unvollständigen Daten*

4.2.2 Handhabung von Mehrdeutigkeiten

Nun gibt es Modelle, deren Parametersatz überbestimmt ist, d.h. aus den Meßdaten läßt sich keine eindeutige Einstellung der Parameter herleiten; es gibt viele verschiedene Lösungen, die einen äquivalente Abgleichsqualität bezüglich des Fehlermaßes erreichen. Es werden zusätzliche

Bedingungen benötigt, die einen eindeutigen Parametersatz bestimmen, sollen nicht zufällige Ergebnisse das Resultat sein (siehe Abbildung 4.5) .

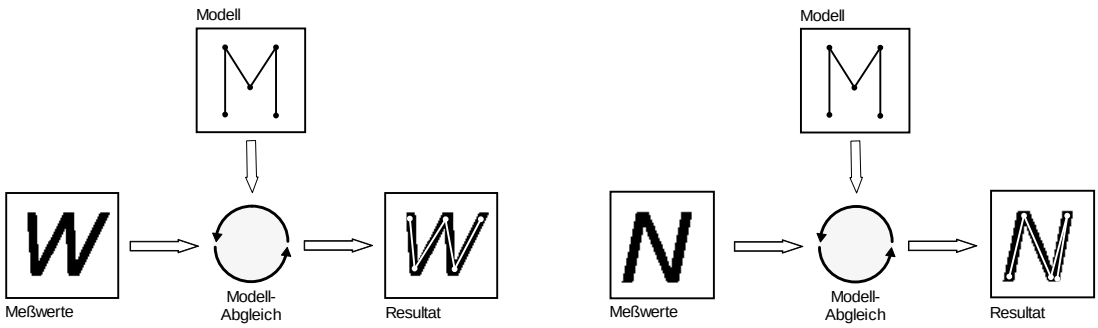


Abbildung 4.5: *Fehlinterpretationen beim Abgleich*

Beim RAMSIS z.B. ist die Wirbelsäule ein Bereich, der aus einer Vielzahl von Knochen besteht, die alle einen eigenen Längenparameter besitzen. Verlängert sich ein Knochen und ein anderer verkürzt sich gleichzeitig, wird das auf die äußere Gestalt des Modells kaum Auswirkungen haben, Größe und Form bleiben nahezu konstant. Somit ist es zwar für das Erscheinungsbild eine marginale Änderung, die bzgl. des Abstandsmaßes nicht messen läßt , jedoch ändern sich wichtige semantische Beziehungen im Modell. Bei vielen Änderungen über die Laufzeit hinweg kann die interne Modellstruktur zerstört werden.

Folgende Abbildung macht die Problematik an der Lage der Kniegelenke deutlich. Links sind deutlich zu hoch angesetzte Kniegelenke auszumachen, die rechte Darstellung setzt die Lage der Knie zu tief an. Selbst eine manuelle Bestimmung der korrekten Lage ist hier schwierig. Ein wesentlicher Hinweis gibt hier die anatomisch korrekte Verhältnismäßigkeit des Modells Es empfiehlt sich also, die Modellproportionen im Abgleichprozeß möglichst lange nicht zu verändern oder mittels einer Anatomiedatenbank diese Relationen ständig zu überwachen.

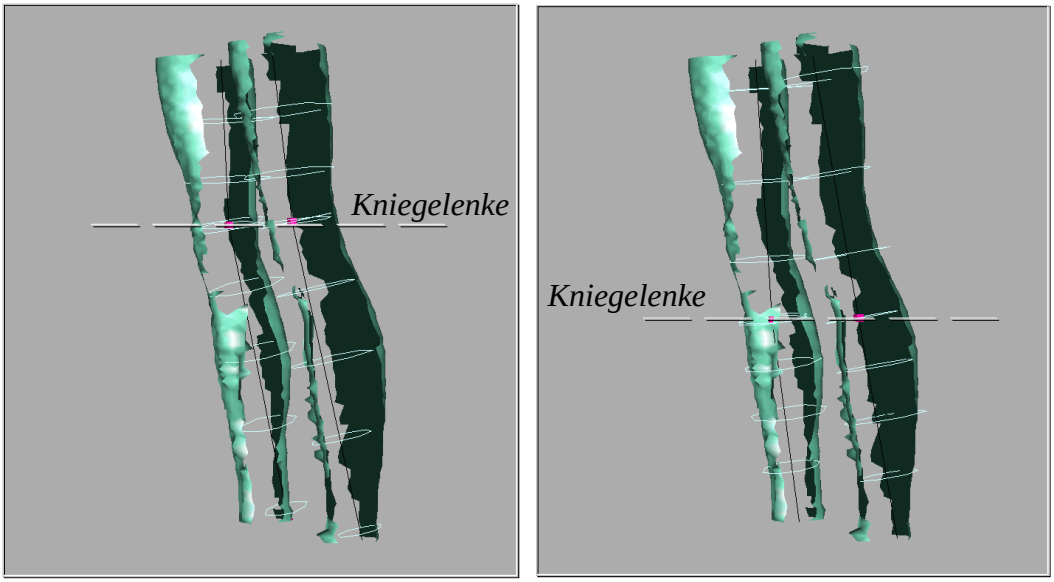


Abbildung 4.6: *Lage der Kniegelenke bei gestreckten Beinen*

4.3 Modellabgleich mittels Evolutionsstrategien

Evolutionsstrategien sind zum Abgleich von Meßdaten mit einem Modell sehr gut geeignet. Aus gutem Grund fiel deshalb in der Konzeptionsphase des Anpaßverfahrens die Wahl auf ein solches Optimierungsverfahren.

In diesem Abschnitt wird die Anpassungsproblematik weiter vertieft und die Gründe für Wahl dieser Strategien diskutiert bzw. Nachteile anderer Verfahren verdeutlicht.

4.3.1 Vorteile von Evolutionsstrategien

Die Nutzung von Evolutionsstrategien für den Modellabgleich hat einige Vorteile gegenüber anderen Verfahren. So bedarf es keiner expliziten Definition einer Strategie für die Zielsetzung, was die damit verbundenen negativen Konsequenzen (siehe folgender Abschnitt) vermeidet.

Die ES sucht sich ihren Weg zur Lösung selbständig, es bedarf also prinzipiell keiner expliziten Angabe einer Abfolge fest vorgeschriebener Teilschritte der Anpassung. Es findet vielmehr ein ganzheitlicher Prozeß statt, der lediglich durch eine geeignete Definition eines geeigneten Abstandsmaßes bestimmt wird, welcher richtungsweisend für die Evolution wirkt. Es ist im Sinne des Abschnitt 4.2.1 ein iteratives Verfahren und damit gut geeignet für den modellbasierten Abgleich von Meßdaten.

4.3.2 Nachteile direkter Vorwärtsstrategien

Eine Vorwärtsstrategie definiert sich dadurch, daß verschiedene Abgleichsschritte durchlaufen werden, die jeder für sich nur einen Teil des Gesamt-Abgleichs durchführen, d.h. die Gesamtaufgabe wird nach dem *Divide and Conquer* Prinzip in verschiedene kleinere Teilaufgaben zerlegt, die nacheinander gelöst werden. Abhängig von bestimmten Bedingungen ändert sich die Abfolge der Rechenschritte.

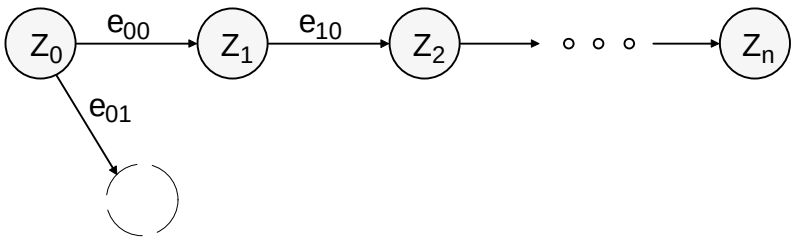


Abbildung 4.7: *Zerlegung eines Problems in kleine Lösungsschritte*

Ein wesentlicher Aspekt einer Vorwärtsstrategie bildet ein sogenanntes Ursache-Wirkungs-Paar. Dabei werden i.a. spätere Programmabschnitte auf Ergebnisse früherer Schritte zurückgreifen und also auch mit deren Ergebnissen weiterarbeiten. Sind dabei die Ergebnisse der vorangegangenen Schritte schon falsch da die Ursachenerkennung mißlang, scheitert also das ganze Konzept. Dies kann zum einen durch Datenfehler bzw. Datenverlußte verursacht sein. Zum anderen kann es sein, daß gegenseitige Abhängigkeiten verschiedener Schritte deren sequentielle Abfolge gar nicht zulassen, also ein konzeptioneller Fehler vorliegt. In diesem

Falle ist eine Aufteilung des Gesamtproblems nicht zulässig, es muß ein Einschrittverfahren zur Lösung konzipiert werden.

Die folgende Abbildung symbolisiert die Datenabhängigkeit eines Abfolgeschritts von seinem Vorgänger. Sind zwei Schritte gegenseitig von ihren Ergebnissen abhängig, so kann keine Abfolge hergestellt werde, da jeder Schritt auf die Ergebnisse des anderen angewiesen ist. Deshalb muß die Berechnung in einem Schritt erfolgen, sofern dies möglich ist.

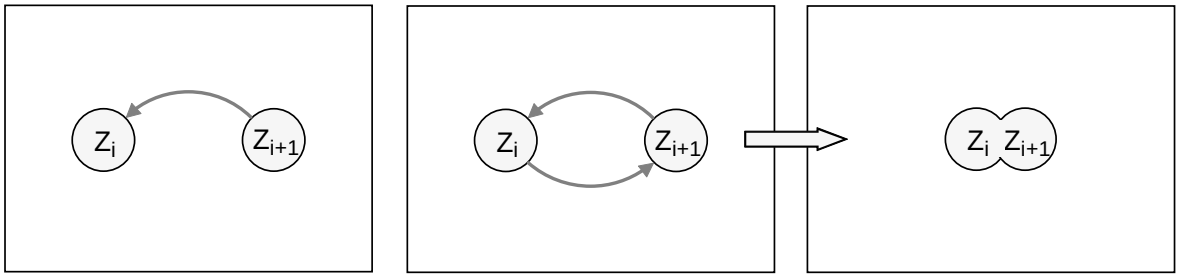


Abbildung 4.8: *Abhängigkeiten der Lösungsschritte*

Als Beispiel soll in einem Gedankenexperiment die Anpassung des Startelements (STE) beim RAMSIS in einer Vorwärtsstrategie untersucht werden:

Der Algorithmus soll systematisch in einem *Innen-nach-außen*-Ansatz Startelement, Knochen, Gelenke und Hautscheiben nacheinander an einen Scan anpassen.

Die erste Frage, die sich stellt ist, wie bestimmt sich die Lage des STE im Raum? Da es sich hierbei um einen Knochen handelt, der also zur Innenstruktur des Modells gehört, ist seine Lage nicht direkt mit dem Scan abgleichbar. Vielmehr muß mittels Hilfselementen (wie daran angrenzende Hautscheiben) seine Güte abgeschätzt werden.

Werden die Hautscheiben des unteren Rumpfes dazu genutzt um die Fitneß des Startelements abzuschätzen, so zeigt sich, daß diese die genaue Raumlage nicht zu bestimmen vermögen. Dazu liefern sie zu wenige topologische Kontextinformationen über die äußere Beschaffenheit des Modells. Man könnte sie genausogut im oberen Brustbereich vermuten als auch, in kleinerem Maßstab, im unteren Teil des Beines.

Also reichen diese wenigen Hautscheiben nicht aus, um eindeutige Ergebnisse zu bekommen. Aus Ausweg bietet sich an um Mehrdeutigkeiten einzugrenzen, die Hautscheiben größerer Extremitäten, wie beispielsweise die Hautscheiben der Beine hinzuzunehmen. Allerdings ergibt sich dann das Problem, daß die Beine eigentlich schon angepaßt sein müßten, um die Modellform gut mit dem Scan in Übereinstimmung zu bringen. Somit hängt das eine vom anderen ab. Es ist nicht möglich, Prioritäten der Abarbeitung festzulegen. Es ergibt sich die Konfiguration aus Abbildung 4.8. Sequentiell abfolgende Schritte bedingen sich gegenseitig.

Die Anpassung ist also ein Prozeß, der i. A. nicht auf kleinere Teiloptimierungen reduziert werden kann, sondern ungeteilt ablaufen muß.

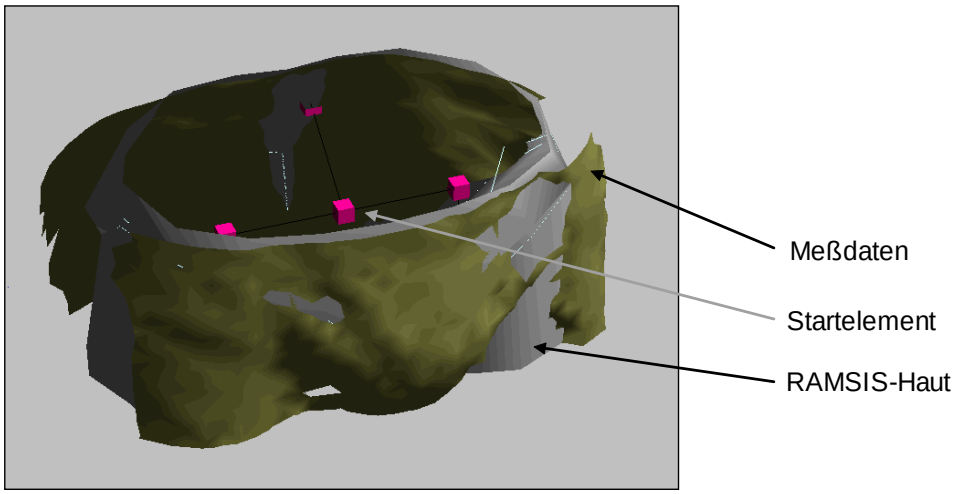


Abbildung 4.9: *Startelement des RAMSIS-Modells*

4.4 Entwicklung eines geeigneten Gütekriteriums

Die Entwicklung einer geeigneten Qualitätsfunktion ist Vorraussetzung für jeden Anpaßprozeß. In diesem Abschnitt soll Schritt für Schritt eine Funktion hergeleitet werden, die dem Algorithmus zugrunde liegt.

In diesem Falle geht es darum, den Abstand der Meßdaten zur RAMSIS-Hautoberfläche mathematisch zu fassen. Stimmen beide gut überein, so soll diese Qualitätsfunktion möglichst kleine Werte liefern (Minimierungsproblem), sind sie sehr unterschiedlich, dann muß sie große Werte liefern. Weitere Anforderungen an die Qualitätsfunktion sind Effizienz bei gleichzeitiger Vermeidung lokaler Optima.

Zur Definition des Abstandsmaßes von einander entsprechender Punkte (Modell-Hautpunkt zu Scanpunkt) entsteht das Problem, daß die Sensordaten a priori keine Semantik tragen. Es ist also nicht klar, zu welchen 3D-Daten der Abstand des betrachteten Hautpunktes ermittelt werden soll. Erst nach einer Grobanpassung besteht eine inhaltliche Beziehung zwischen diesen. Damit ist es also nicht möglich, das Abstandsmaß auf den Abstand einander entsprechender Punkte zurückzuführen. Diese Entsprechungsbeziehung kann prinzipiell nicht angegeben werden.

Der Abstandsbegriff kann aber für die Lage eines Punktes zu einer Menge der entsprechenden Punktemenge definiert werden. Dafür gibt es grundsätzlich zwei Möglichkeiten:

1. Abstand der Scanpunkte zum Modell
2. Abstand der Modellhautpunkte zum Scan

Jeder der Definitionen hat Vor- und Nachteile, die im folgenden erörtert werden.

4.4.1 Abstandsmessung vom Scan zur Modell-Haut

Bei diesem Ansatz ist für jeden Scanpunkt der Abstand zur RAMSIS-Hautoberfläche zu berechnen und deren Werte zu Summieren. Dabei ist von Vorteil, daß dieser Wert direkt berechnen werden kann, indem die Strecke vom Scanpunkt zum nächstgelegenen Modell-Hautoberflächenpunkt ermittelt wird.

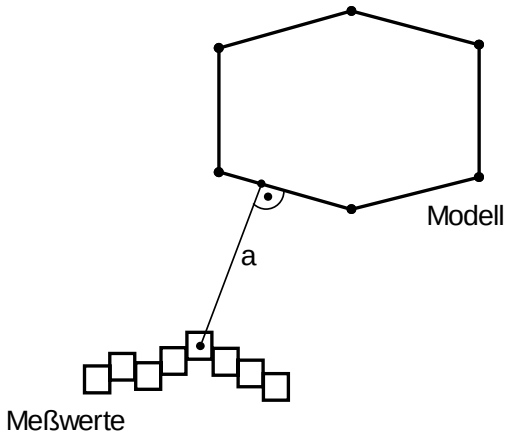


Abbildung 4.10: *Abstand eines Scanpunktes zum Modell*

Dieses Verfahren kam für unsere Zwecke allerdings nicht in Betracht, da diese Ermittlung für eine sehr große Anzahl von Scanpunkte (ca. 30000) und einer komplexen Modell-Oberfläche durchgeführt werden muß. Da das Modell während der Anpassung ständigen Änderungen unterliegt kann keine Hash-Struktur o.ä. zur schnelleren Berechnung eingesetzt werden. Für jedes Individuum müssen die Abstände neu ermittelt werden, so daß eine derartige Berechnung zu aufwendig ist.

4.4.2 Abstandmessung von der Modell-Haut zum Scan

Diese Variante berechnet den Abstand in umgekehrter Richtung. Ausgehend von einem Modell-Hautpunkt ist der kleinste Abstand zum Scan zu ermitteln. Dies hat gegenüber dem obigen Ansatz den Nachteil, daß die Abschattungen in der Scanpunktewolke zum tragen kommen.

Aufgrund der Tatsache, daß der Scan über die Programmlaufzeit hinweg nicht verändert wird, kann eine geeignete Datenstruktur entworfen werden, die sehr effizient zur Abstandsberechnung genutzt werden kann.

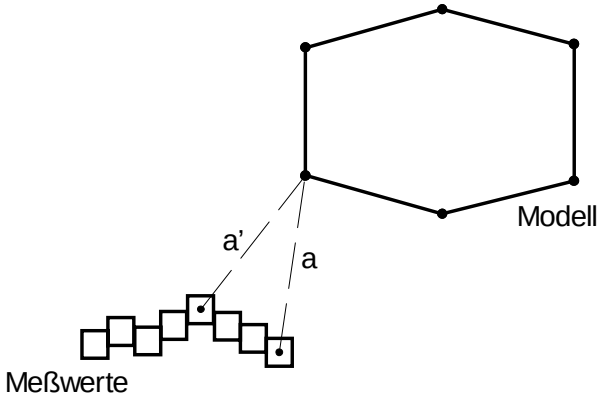


Abbildung 4.11: *Abstand vom Modellpunkt zum Scan*

Um diese Funktion effizient zu implementieren ist eine Methode entworfen worden, welche direkt den Abstand eines Modellhautpunktes zum nächstgelegenen Scanpunkt bestimmt. Sie vermeidet dabei insbesondere die zeitraubende sequentielle Suche über alle Meßwerte (siehe Abbildung 4.11).

Eine formalere Definition dieser Funktion im nächsten Abschnitt. Die Vermeidung komplexer Berechnungen durch einen Voxelraum wird in Abschnitt 4.4.5 besprochen.

4.4.3 Herleitung des Abstandsmaßes

Zur Definition des Abstandsmaßes von einem RAMSIS-Hautpunkt zum Scan bereitet die Nichtzuordenbarkeit von RAMSIS-Hautpunkten zu Scanpunkten Probleme. Dies ist prinzipiell nicht umgehbar, die Zuordnung dieser Punkte soll die Anpassung ja erst ermitteln. Somit ist es nicht möglich, den Abstand des Modells auf den Abstand einzelner, sich entsprechender Punkte zurückzuführen.

Definiert wird als Abstandswert eines Hautpunktes zur Scanpunktewolke der Abstand zum *nächstgelegenen* Punkt des Scans.

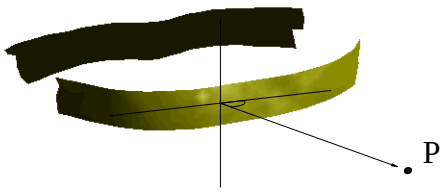


Abbildung 4.12: *Abstand eines Punktes P zum Scan, der hier als schattiertes Objekt dargestellt ist.*

Der Abstand eines Modell-Hautpunktes *P* ist als das Minimum der Abstände zu allen Scanpunkten definiert. Entsprechend wird dieses Abstandsmaß auf eine Menge von Hautpunkten

erweitert, indem es als die Summe der Abstände der Punkte definiert wird. In der Abbildung sei P ein RAMSIS-Hautpunkt, H_{Ramsis} die Menge dieser Punkte, $dist(x, y)$ der Abstand zweier Punkte x und y im Raum, $dist(x, \{y_i\})$ der Abstand eines Punktes x zur Punktemenge $\{y_i\}$ (unten definiert) und *Scan* die Menge aller Punkte des Scans:

$$fit_{Scan}(P) = dist(P, \{s \mid s \in Scan\}) \stackrel{def}{=} min\{dist(P, s) \mid s \in Scan\}$$

Damit läßt sich der Abstand $fit_{Scan}(H_{Ramsis}) = fit_{Scan}\{P \mid P \in H_{Ramsis}\}$ eines Modells(H_{Ramsis}) vom Scan definieren

$$fit_{Scan}\{P \mid P \in H_{Ramsis}\} \stackrel{def}{=} \sum_{P \in H_{Ramsis}} fit_{Scan}(P)$$

4.4.4 Unvollständigkeit der Meßdaten

Bedingt durch den Aufbau des Scanners ist die produzierte Meßdatenmenge, welche die Hautoberfläche der zu vermessenden Person repräsentiert, nicht geschlossen, sondern enthält Lükken. Diese entstammen zu dunklen Objektteilen (wie beispielsweise die Haare) oder Stellen, die von der Lichtquelle des Scanners nicht beleuchtet oder von der Kamera nicht eingesehen werden. Solche Datenlücken sole das Anpassungsverfahren ausgleichen können.

Die folgende Abbildung zeigt (links im Bild) die vom Scanner nicht erfaßten Fußsohlen der Person und den Ausfall von Meßdaten im Bereich des Kopfes (Haare, rechts im Bild).

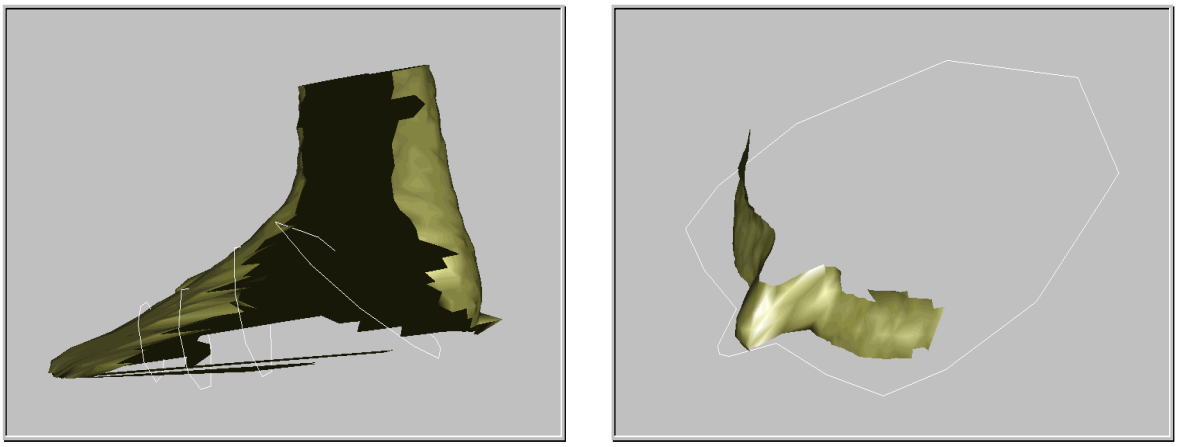


Abbildung 4.13: *Scans mit RAMSIS-Hautscheiben (weiß) eines Fußes und eines Geschichtsausschnitts.*

4.4.5 Der Voxelraum

Zur effizienten Implementierung der Fitneßfunktion wurde ein Algorithmus implementiert, der einen speziellen Voxelraum[1] generiert [KUB97]. Ein Voxelraum ist ein dreidimensionales Array, in das numerische Werte eingetragen werden können. Für die Nutzung als Abstandsmaß der Abgleichsverfahrens wird dieser Raum derart gefüllt, daß eine Anfrage an Position *(x,y,z)*

1. Das Kunstwort *Voxel* steht als Abkürzung für Volumen-Pixel.

des Raumes einen zur Scan-Entfernung proportionalen Wert ergibt. Eine Anfrage ergibt also einen der beiden Werte:

- Eine Anfrage an der Position *(x,y,z)* im Raum ergibt den Wert 0, falls an dieser Stelle ein Punkt des Scans lokalisiert ist.
- Eine Anfrage an der Position *(x,y,z)*, an der sich kein Scanpunkt befindet. ergibt einen Wert, der dem Abstand zum nächsten Scanpunkt proportional ist.

Damit ist eine Datenstruktur definiert, die eine sehr effiziente Anfrage der Abstandswerte zuläßt. Eine Anfrage hat typischerweise eine Laufzeit, die einem Hauptspeicherzugriff entspricht.

Die Erzeugung des Voxelraumes erfolgt in folgenden Schritten:

1. An die Raumposition der Scan-Werte wird der Wert Null eingetragen.
2. Sukzessiv werden die noch nicht berechneten Stellen mittels eines linearen Filters berechnet.

Das hier verwendete Filter hat folgendes Schema.

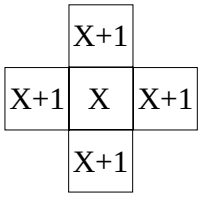


Abbildung 4.14: *Das für das Auffüllen des Voxelraums verwendete Filter*

Abbildung 4.15 bis Abbildung 4.17 veranschaulichen die Konstruktion an einem 2-dimensionalen Beispiel:t

						0	0						
					0			0					
					0				0				
					0				0				
						0			0				
							0	0					

Abbildung 4.15: *Eintragen der Scanpunkte in den leeren Voxelraum.*

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

Abbildung 4.16: *Erster Schritt des Füllalgorithmus.*

8	7	6	5	4	3	2	2	3	4	5	6	6	7	8
7	6	5	4	3	2	1	1	2	3	4	4	5	6	7
6	5	4	3	2	1	0	0	1	2	3	3	4	5	6
5	4	3	2	1	0	1	1	0	1	2	2	3	4	5
5	4	3	2	1	0	1	2	1	0	1	2	3	4	5
5	4	3	2	1	0	1	2	1	0	1	2	3	4	5
6	5	4	3	2	1	0	1	1	0	1	2	3	4	5
7	6	5	4	3	2	1	0	0	1	2	3	4	5	6
8	7	6	5	4	3	2	1	1	2	3	4	5	6	7
9	8	7	6	5	4	3	2	2	3	4	5	6	7	8

Abbildung 4.17: *Vollständig gefüllter Voxelraum*

4.4.6 Trilineare Interpolation im Voxelraum

Um die qualitative Lücke zwischen den diskreten Scanwerten im Voxelraum und den reellwertigen Positionen der Hautpunkte des RAMSIS zu schließen und die Daten im Array zu glätten, wird bei Anfragen von Punktdaten zwischen den Voxelwerten linear interpoliert. Dies geschieht durch die Verrechnung der Werte der acht unmittelbar benachbarten Voxel des Anfragepunktes unter Beachtung seiner relativen Position zu diesen.

Diese Berechnung sei an einem zweidimensionalen Beispiel verdeutlicht: Zugrundegelegt wird ein diskreter 2D-Voxelraum, der wie beschrieben jedem Voxel eine Ganzzahl zuordnet. Für die Bestimmung von Funktionswerten von Anfragepunkten der reellen Ebene sei der Voxelraum in diese eingebettet. Trifft die Anfrage exakt die Mitte eines Voxels, so wird der Wert an dieser Stelle zurückgeliefert. Für abweichende Lokationen wird eine Ausgabe entsprechend den Werten der vier Nachbarvoxel in den beiden Koordinatenachsenrichtungen interpoliert.

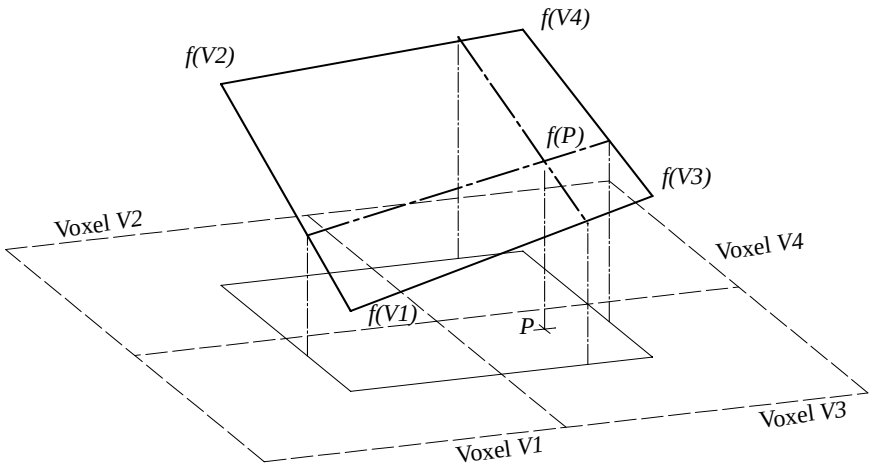


Abbildung 4.18: *Die bilineare Interpolation auf Basis des Voxelraumes.*

Die folgende Abbildung visualisiert einen horizontalen Schnitt durch den Voxelraum. Dazu sind in den drei Bildern Niveauebenen des Raumes abgebildet, also nur Raumpunkte des Voxelraums, die einen bestimmten Wert und damit diesen Abstand vom Scan besitzen. Dies sind die Werte 0 (Scan, links im Bild), 3 (Abstand 3 vom Scan, Bildmitte) und 10 (Abstand 10,

rechts im Bild). Die Voxel sind weiß dargestellt, während die Meßdaten als schattierte dunkle Flächen zu erkennen sind. In den Bildern sind deutlich die aufgrund des Filters typischen rautenartigen Strukturen zu erkennen, die mit größem Abstand vom Scan immer deutlicher werden.

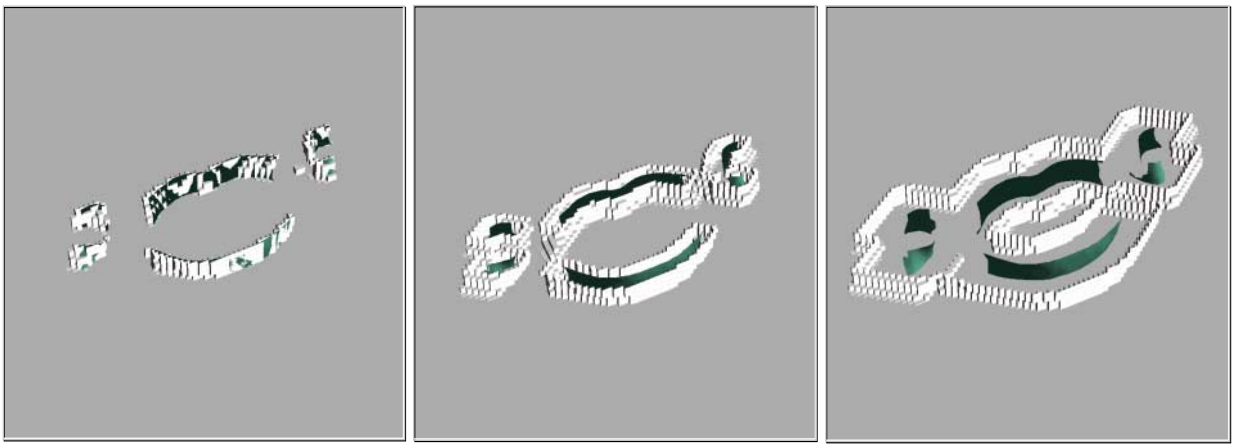


Abbildung 4.19: *Visualisierung eines Schnitts durch den Voxelraum mit zugehörigem Scan-Ausschnitt.*

4.5 Konzeption des Algorithmus

Im Abschnitt 4.4 wurde schon angesprochen, daß es von großer Wichtigkeit ist, das modellinhärente semantische Wissen weitgehend zu erhalten und möglichst nur die äußere Form des Modells zu verändern, also eine ’sanfte’ Anpassung durchzuführen. Trotzdem müssen aber Gelenkwinkel, Knochenlängen und Hautscheiben richtig eingestellt werden. Die Kunst besteht also darin, die Modellparameter in einer Weise so zu verändern, daß die semantischen Inhalte weitgehend erhalten bleiben, ohne die Individualisierung zu behindern.

Aus diesem Grunde liegen der Konzeption des Algorithmus mehrere Stufen zugrunde, die anfänglich größere funktionale Einheiten gemeinsam anpassen und nach und nach immer mehr Freiheitsgrade freischalten, so daß in der Endphase des Algorithmus fast alle Parameter veränderbar sind. Auf diese Weise bleiben die Verhältnismäßigkeiten der inneren Struktur relativ lange erhalten.

Ein weiterer Vorteil der Stufenkonzeption ist die Komplexitätsreduktion des Optimierungsverfahrens und damit auch die Beschleunigung des Anpaßvorgangs, da der Suchraum in den ersten Phasen deutlich eingeschränkt ist.

Durch die starke Kopplung in der Anfangsphase wird bei diesem Verfahren eine grobe Anpassung in Haltung und äußerer Form an die Meßdaten erreicht. Dabei bleiben semantische Inhalten wie Proportionen und Details des Modells fixiert. In den weitern Phasen und der Endphase werden diese Zusammenhänge dann sukzessiv aufgebrochen und das Modell wird im Detail angepaßt. Da in diesen Schritten die Grobanpassung bereits vollzogen ist, wird das Modell nur noch in kleinen Maßstäben verändert, die groben Zusammenhänge werden nicht mehr aufgebrochen:

In der Anfangsphase wird zunächst die Position des Modells im Raum, die Rotation um die drei Raumachsen, die Körpergröße und die Korpulenz ermittelt. Diese wenigen Parameter reichen

schon aus um eine grobe Übereinstimmung mit dem Scan zu erreichen. In den folgenden Phasen wird eine immer besser werdende Näherung sukzessiv durch Hinzunahme von immer kleiner werdenden Granularitäten erreicht. Eine Gesamtkorpulenz teilt sich beispielsweise in Oberkörperkorpulenz, Arm- und Beinkorpulenz auf. In späteren Schritten ist eine noch feinere Unterteilung realisiert (Oberschenkelkorpulenz, Unterschenkelkorpulenz) bis schließlich die letzte Phase die Anpaßten einzelner Hautscheiben durchführt.

Eine ähnliche Hierarchie existiert für die Körpergröße, die letztlich auf einzelnen Knochenlängen basiert. Gelenkwinkel stellen hier eine Ausnahme dar, da die Gelenke alle einzeln angepaßt werden; lediglich für die Anpassung der Wirbelsäule macht eine Zusammenfassung von Gelenken zur *Wirbelsäulenkrümmung* Sinn.

Im folgenden sind zum einen die Korpulenzhierarchie abgebildet, welche Hautscheibenformen zu komplexen Einheiten zusammenfaßt und zum anderen die Größenhierarchie, welche eben dies für Knochenlängen erreicht. Auf die Abbildung der oben erwähnten Modellierung der Wirbelsäule wurde an dieser Stelle verzichtet.

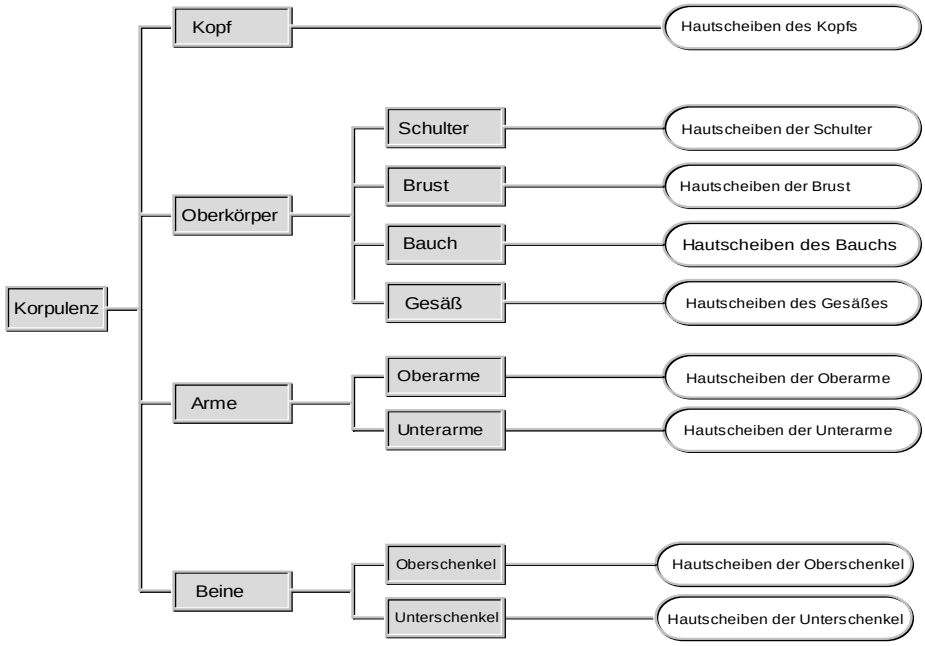


Abbildung 4.20: *Korpulenzhierarchie des Anpaßalgorithmus*

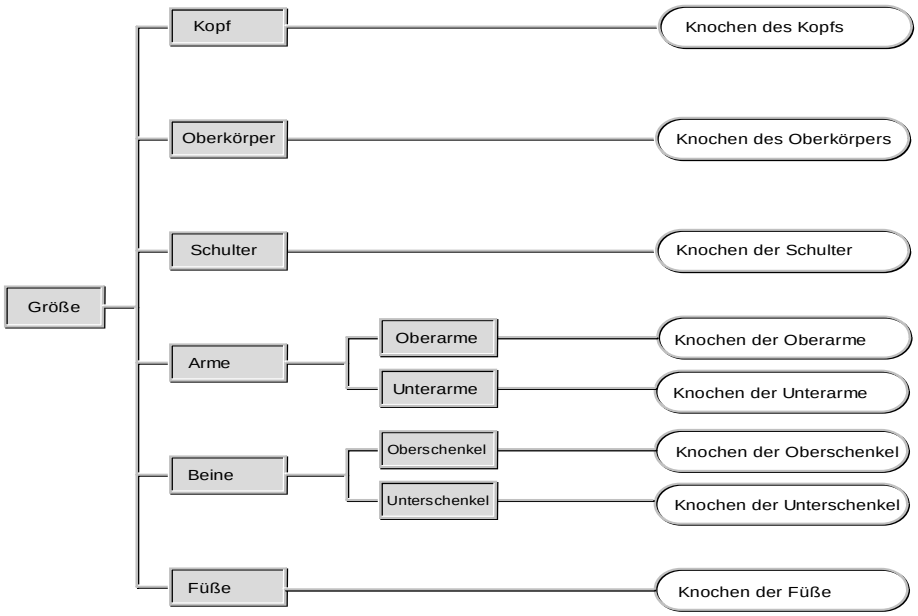


Abbildung 4.21: *Knochenlängenhierarchie*

Durch die weitestgehende Gestaltbarkeit der Anpaßstufen in einer Initialisierungsdatei (siehe Abschnitt 4.9) ist es möglich funktional unabhängige Bereiche des Modells separat anzupassen. Ein Beispiel hierfür sind die Arme, deren Aussehen und Lokation für die Anpassung des restlichen Modells irrelevant sind. Die Längenbestimmung bzw. Korpulenzbestimmung kann unabhängig von der des restlichen Oberkörpers in einer separaten Anpaßstufe vorgenommen werden.

Generell sollten Einheiten, die unabhängig voneinander sind, auf mehrere Stufen aufgeteilt und getrennt angepaßt werden. Mit jeder Verkleinerung des Parametersatzes pro Stufe wird der Suchraum wesentlich kleiner, das Fitneßgebirge u.U. einfacher und die Anpassung schneller.

Der gleiche Effekt wird auch durch die starke Granularisierung durch Merkmalshierarchien erreicht, welche größere funktionale Einheiten mittels weniger Parameter anpassen.

Die Konstruktion von Hierarchien ist eine semantische Eigenschaft, die dem RAMSIS-Modell fehlt und die deshalb für die Evolution nachgebildet wurde. Das Modell besteht aus den Modellgrundelementen Hautscheiben, Knochen und Gelenke, die über ihre Namen bzw. Identifizierer (Objekt-IDs) adressiert werden. Höhere semantische Begriffe wie *Oberkörper* sind ihm daher fremd.

4.6 Schematisches Ablaufdiagramm des Algorithmus

Im folgenden wird der prinzipielle Ablauf des Anpaßalgorithmus LIFE[1] erläutert. Er gliedert sich in einen äußeren Teil, der die Realisierung der im vorigen Abschnitt erläuterten Stufen des

1. LIFE ist zusammen mit einem Editor mEdit lauffähig, der der Visualisierung und Modifikation von RAMSIS-Modellen dient.

Verfahrens umsetzt und einen inneren Teil, in welchem die eigentliche Evolutionsstrategie durchgeführt wird.

Abbildung 4.22: *Das Konzept von LIFE*

Die entscheidenden Programmteile des Verfahrens werden in den nächsten Abschnitten im Detail beschrieben.

4.6.1 Die äußere Schleife

```
01 procedure adaption algorithm
02   begin
03   t := 0
04   VoxelSpace.initialize( Scan )
05   while (IniFile_termination_condition == FALSE) do
06     create DistanceMeasure
07     create TemplateIndividual( RAMSIS )
08     Pop[0].initialize( TemplateIndividual )
09     while (termination_condition == FALSE) do
10       Pop[t+1] := evolution( Pop[t] )
11       t := t+1
12     endwhile
13   endwhile
14 end
```

Abbildung 4.23: *Grundstruktur des Anpaßalgorithmus*

In Abbildung 4.23 ist schematisch der äußere Anpaßalgorithmus dargestellt. Im folgenden werden die wichtigen Abarbeitungsschritte genauer erläutert:

Zeile 04 **Initialisierung des Voxelraumes**. Der Voxelraum wird mit den Scandaten initialisiert und mit Werten aufgefüllt (vgl. Abschnitt 4.4.5). Der Voxelraum wird nur einmal pro Programmlauf berechnet, da er über die einzelnen Anpaßstufen hinweg eine Konstante darstellt.

Zeile 05 **Hauptschleife mit Abbruchbedingung**. Die Hauptschleife durchläuft nacheinander die Stufen des Algorithmus, deren Definitionen aus einer Initialisierungsdatei gelesen werden. Ist die letzte Population erreicht, so wird die Schleife verlassen und das Programm terminiert.

Zeile 06 **Erzeugung des Abstandsmaßes**. Aus dem IniFile werden die RAMSIS-Namen der Hautpunkte, die die Basis für das Abstandsmaß darstellen, eingelesen und in einem Array gespeichert. Da dieser Vorgang für jede einzelne Population erfolgt, kann das Abstandsmaß genau auf die in diesem Schritt anzupassenden RAMSIS-Objekte zugeschnitten werden.

Zeile 07 **Erzeugung eines Beispiel-Individuums**. Dieser Bearbeitungsschritt erzeugt aus den aus dem IniFile gelesenen Strukturdefinitionen, die als *Erbinformation* die anzupassenden RAMSIS-Elemente beschreibt, ein Beispiel-Individuum, das auch die Ur-Einstellungen des RAMSIS speichert. Dieses ausgezeichnete Individuum dient ausschließlich als Template, das zum Zwecke der Populationsinitialisierung mehrfach kopiert wird, und nimmt selbst nicht an der Evolution teil.

Zeile 08 **Initialisierung der Population**. Als Parameter wird das Beispielindividuum übergeben. Die Population wird mit mutierten Kopien dieses Individuums gefüllt, um eine gewisse Varianz in die Initialpopulation zu bringen (siehe Abbildung 4.24)

Zeile 09 **While-Schleife der Evolutionsstrategie**. Solange die Abbruchbedingung der aktuellen Evolutionsstufe nicht erfüllt ist, werden sukzessiv neue Populationen erzeugt, die sich jeweils aus der Anwendung der genetischen Operatoren auf die vorhergehende Population ergeben. Eine detaillierte Aufschlüsselung hiervon zeigt Abbildung 4.25.

4.6.2 Initialisierung der Population

Da es in diesem Falle der Anpassung des RAMSIS-Menschmodells nicht sinnvoll erscheint, die Anfangspopulation mit völlig zufällig im Suchraum verteilten Punkten (also beliebigen Parametern des Modells) zu generieren, wird diese mit Individuen gefüllt, welche alle aus einer speziellen Mutation[1] eines Initialisierungs-Individuums entstehen, das die genotypische Kodierung des Initial-RAMSIS ist.

```
procedure initialize population
begin
  for ( i:= 0; i< Pop.parents; i++)
    Pop[0] := Pop[0] ∪ Template.mutate()
  endfor
end
```

Abbildung 4.24: *Die Initialisierungsfunktion des Algorithmus.*

1. Für unsere Modellabgleichszwecke würde eine rein zufälliges Individuum keinen Sinn machen. Die Intention bei der Vorgestellten Vorgehensweise ist die semantischen Verhältnismäßigkeiten der Proportion des Initial-Individuums zu erhalten.

4.6.3 Evolutionsstrategie des Algorithmus (innere Schleife)

Im folgenden wird der Kern des Anpaßalgorithmus beschrieben, wobei es sich hier um eine für unsere Zwecke modifizierte aber ansonsten klassische Evolutionsstrategie handelt. Viele Parameter sind durch die Initialisierungsdatei definierbar, beispielsweise, ob der Algorithmus die Plus- oder Kommastrategie anwendet.

```
01 procedure evolution( Pop )
02   begin
03   DescPop := ∅
04   for ( i:= 0; i< Pop.parents; i++)
05     Pop.select(I1, I2)
06     I := I1.recombine( I2 )
07     I.mutate()
08     I.calcfitness := VoxelSpace.calcfitness( I )
09     DescPop := DescPop  ∪ {I}
10   endfor
11   if ( strategy == 'plus' )
12     DescPop := DescPop  ∪ Pop
13   endif
14   DescPop := DescPop.truncate()
15 end
```

Abbildung 4.25: *Der Ablauf eines Evolutionsschrittes in Pseudocode.*

Zeile 05 **Selektion**. Aus der aktuellen Population werden zwei Individuen I1 und I2 ausgewählt. Dies kann durch den Zufallsgenerator bestimmt sein oder durch die Fitneßproportionale Selektion geschehen.

Zeile 06 **Rekombination**. Die Datenstruktur des Individuums I1 wird mit der des Individuums I2 rekombiniert. Dabei entsteht ein neues Individuum I. Für jedes Allel kann gewählt werden, ob binäre oder intermediäre Rekombination stattfindet. Diese Parameter werden ebenfalls in der Initialisierungsdatei festgelegt. Zusammengesetzte Allele werden als Einheit rekombiniert und bleiben in ihrer Struktur fixiert.

Zeile 07 **Mutation**. Die Allele des Individuums I werden zufällig geändert, wobei die modifizierende Zufallsvariable um den aktuellen Wert gaußverteilt ist. Die Varianz dieser Gaußverteilung durch einen Metaparameter bestimmt, der durch die Evolution mitoptimiert wird.

Zeile 08 **Berechnung der Fitneß**. Durch Aufruf der entsprechenden Methode des Voxelraumes wird die Fitneß des Individuums I berechnet. Der Wert wird innerhalb der Datenstruktur von I gespeichert. Das ist deshalb durchführbar, da Individuen über die Laufzeit einer Generation nicht mehr verändert werden.

Zeile 09 **Einfügen des Individuums in die Nachfolgepopulation.** Individuum I wird in die Nachfolgepopulation DescPop eingefügt.

Zeile 12 **Einfügen der Individuen der Vaterpopulation**. Ist die plus-Strategie gewählt, so nehmen die Individuen der Ausgangspopulation am Wettbewerb teil und müssen demnach in die Nachfolgepopulation eingefügt werden.

Zeile 14 **Sondieren der Nachfolgepopulation**. Da bei der hier implementierten Evolutionsstrategie die Nachfolgepopulation zunächst mehr Individuen enthält als die Vaterpopulation, werden in diesem Schritt Individuen abhängig von ihrem Fitneßwert entfernt.

Als Eingaben erhält der Evolutionsalgorithmus ein RAMSIS-Menschmodell sowie die Meßdaten des Scanners. In mehreren Stufen erfolgt dann die Anpassung des Modells an den Scan. Die Anpassung selbst wird durch die aus der Initialisierungsdatei gelesenen Parametern gesteuert. Als Anpassungsergebnis resultiert ein Menschmodell, welches sowohl die Inidvidualinformationen der Körperoberfläche der gescannten Person als auch eine plausible Schätzung der inneren Struktur (Knochenlängen und Gelenkwinkel) enthält.

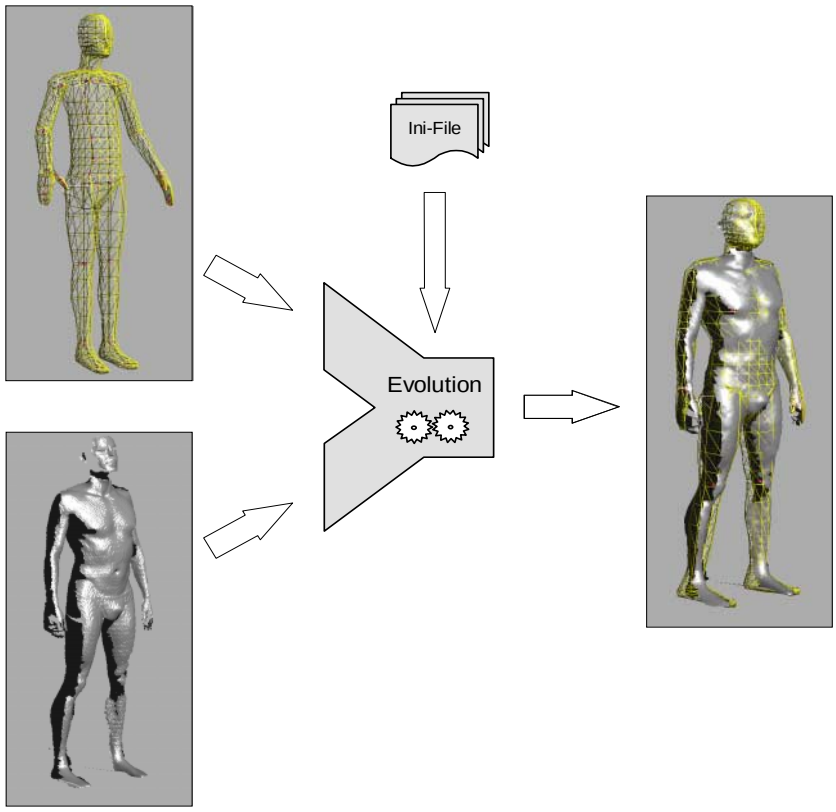


Abbildung 4.26: *Schemagraphik der Anpassung.*

4.7 Abbruchbedingung der ES-Schleife

Da Evolutionsalgorithmen prinzipiell ein iteratives Verfahren sind, stellt sich bei der Implementierung die Frage, wann diese Iteration abgebrochen werden soll. Dieser Punkt ist sicherlich

vom Problem abhängig, d.h. es stehen entweder laufzeitspezifische Aspekte im Vordergrund oder es wird eine gewisse Qualität des Ergebnisses gefordert.

Hier eine kleine Aufstellung von möglichen Abbruchbedingungen:

- Abbruch der Schleife bei Erreichen eines absoluten Fitneßwertes der Population
- Abbruch der Schleife bei Erreichen eines relativen Fitneßwertes (Fitneßverbesserung)
- Abbruch der Schleife bei zu geringen Fitneßverbesserungen (Ableitung klein)
- Abbruch der Schleife durch ein zeitliches Limit

Diese Aufzählung ist sicherlich nicht vollständig und kann natürlich je nach Anforderung ergänzt werden.

Die Abbruchbedingung des Anpaßalgorithmus sollte einen gewissen qualitativen Aspekt beinhalten, zum anderen auch flexibel genug sein um robust auch schwierige Fälle zu behandeln.

Im folgenden werden zwei Terminierungsbedingungen vorgestellt, die den obigen Anforderungen entsprechen:

Abbruch bei Unterschreiten eines Schwellwertes für die Populationsvarianz:

Die EA-Schleife wird dabei abgebrochen, wenn eine gewisse *Varianz* der Fitneßwerte der Individuen in einer Population eine bestimmte Schwelle unterschreitet, d.h. wenn beispielsweise folgende Relation erfüllt ist:

$$\left| max\{fit(I) \mid I \in Pop\} - min\{fit(I) \mid I \in Pop\} \right| < s$$

Dabei sei der unterschrittene Schwellwert mit s bezeichnet. In diesem Falle kann mit hoher Wahrscheinlichkeit davon ausgegangen werden, eine bestimmte Mindestgröße der Population vorausgesetzt, daß die Evolution “auf der Stelle” tritt und keine nennenswerten Fortschritte mehr machen wird.

Abbruch bei Einhalten der Beschränkungen eines Fitneßfensters:

In diesem Falle wird die Entwicklung der Fitneß der Gesamtpopulation oder des besten Individuum der Population über eine gewisse Laufzeit hinweg betrachtet. Bleiben die Maxima und Minima dieser Wertemenge unter einer im Initialisierungsdatei definierten Schwelle, so bricht die Anpaßstufe ab, ansonsten wird die Nachfolgepopulation berechnet.

Die folgende Abbildung illustriert anschaulich diese Bedingung:

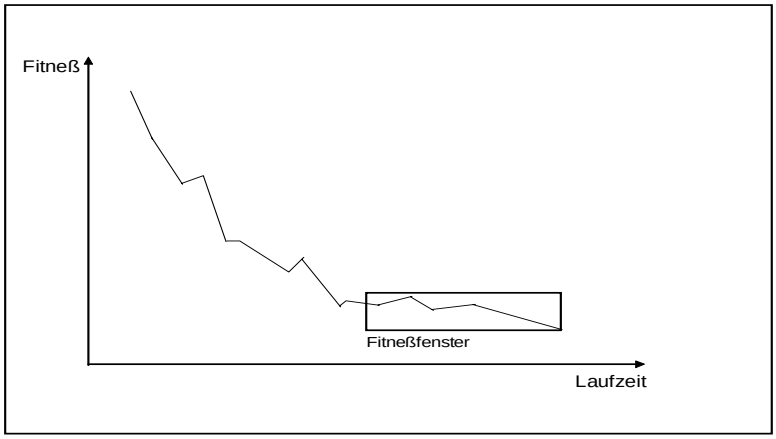


Abbildung 4.27: *Fitneßfenster als Abbruchkriterium*

Implementiert wurden beide Abbruchbedingungen, wobei sich die letztere als effektiver herausstellte. Dies liegt im wesentlichen daran, daß sie die einen zeitlichen Abschnitt der Folge der Populationen berücksichtigt und somit einen historischen Aspekt einbezieht.

Ein weiterer Vorteil erwächst aus der effizienteren Implementierung. Es kann auf Rechenschritte zu Berechnung der Gesamtpopulationsfitneß verzichtet werden.

4.8 Datenstruktur des Algorithmus

Das Hauptprogramm LIFE des Algorithmus operiert auf Populationen, d.h. einer Menge von Individuen, auf denen mit den genetischen Operatoren sukzessiv Nachfolgepopulationen berechnet werden, bis das Terminationskriterium erfüllt ist.

Dabei besteht ein Individuum aus einem *Genom*[1], das die relevanten RAMSIS-Daten in Form einer eigenen Datenstruktur beinhaltet, und dem Fitneßwert des Individuums. Weil das Individuum nach seiner Erzeugung während der Laufzeit nicht mehr geändert wird, ist es sinnvoll,

1. Ein Genom enthält die Gesamtinformation eines Modells in einer eigenen kodierten Form.

diese Bewertung nur einmal vorzunehmen und bei Bedarf ausschließlich auf diese Speicherstelle zuzugreifen.

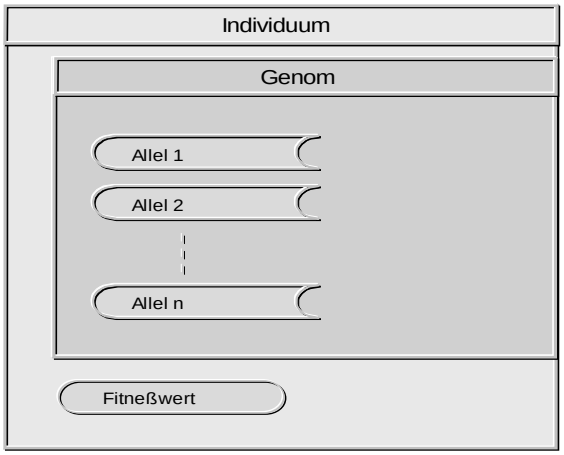


Abbildung 4.28: *Strukturschema eines Individuums*

Das Genom setzt sich aus einzelnen Allelen zusammen, die jeweils ein bestimmtes Modell-Element[1] speichern.

Ein Allel besteht aus einer Reihe von individuellen Objektparametern und einer Anzahl *Metaparametern* oder *Kontrollvariable*, die den Ablauf der Evolutionsstrategie selbst steuern und keine Entsprechung im Suchraum besitzen.

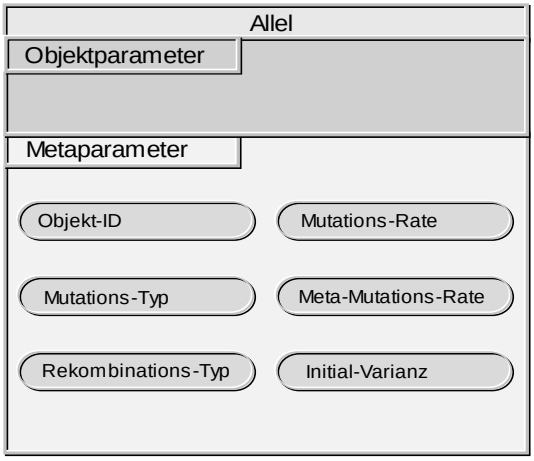


Abbildung 4.29: *Grundaufbau eines Allels*

Es existieren 5 Grundtypen von Allelen. Vier davon speichern Grundelemente des RAMSIS, nämlich Hautscheiben, Hautpunkte, Knochen und Gelenke. Das fünfte kann hierarchische Strukturen aufbauen um komplexe Merkmale einheitlich anpassen zu können.

1. Also Hautscheibe, Hautpunkt, Knochen oder Gelenk.

Ein Hautscheibenallel besitzt insgesamt 6 Objektparameter, welche die Form der 3-dimensionalen Hautscheiben bestimmen. Das Hautpunktallel kann ausschließlich den Abstand zum zugehörigen Knochen verändern und besitzt deshalb nur einen Objektparameter.

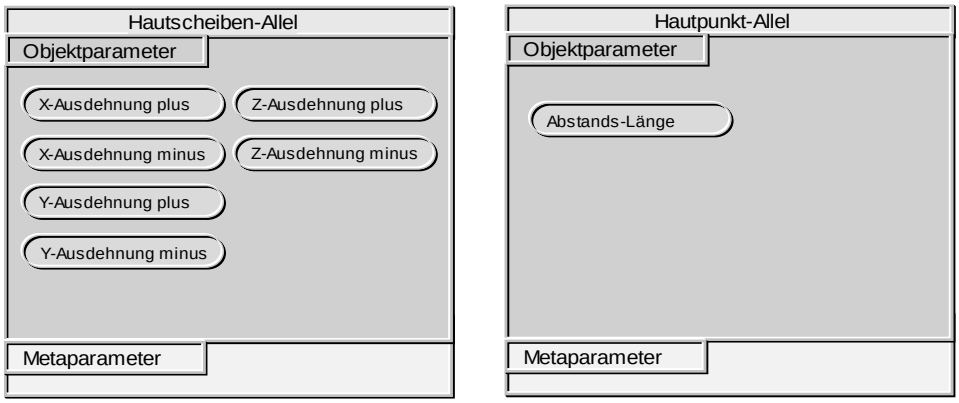


Abbildung 4.30: *Hautscheiben- und Hautpunktallel.*

Gelenke besitzen 3 Winkelfreiheitsgrade, die durch ein Gelenkallel beschrieben sind. Ein Knochen hat als einzigen Parameter seine Länge. Weiterhin besitzt es Begrenzungsparameter[1], die Wertebereichüberschreitungen verhindern. Diese sind während der Anpassung konstant und deshalb nicht der Abbildung nicht dargestellt.

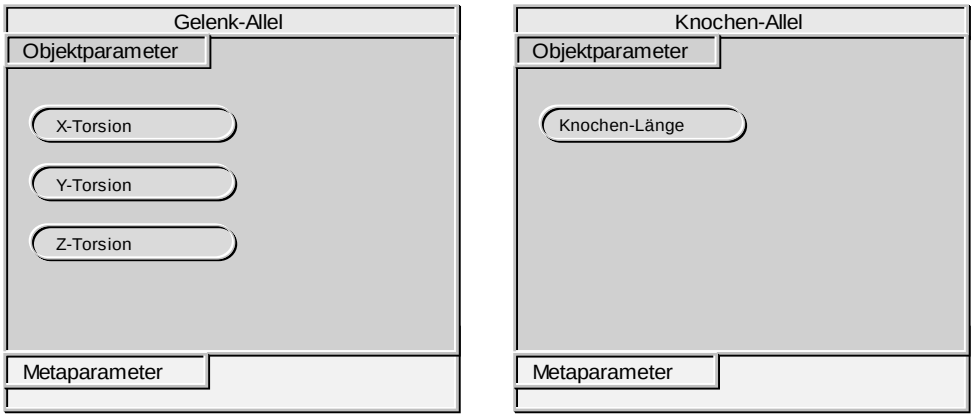


Abbildung 4.31: *Gelenk- und Knochenallel.*

1. Das Gelenkallel ist das einzige, das Parametereinschränkungen unterliegt.

4.8.1 Strukturierungswerkzeug Metaallel

Das Metaallel enthält keine eigenen Objektparameter in dem Sinne, daß diese Daten direkt im Modell repräsentierten, sondern dient ausschließlich der Zusammenfassung von anderen Allelen zu komplexen Einheiten, nämlich die in Abschnitt 4.5 vorgestellten Semantik-Hierarchien. .

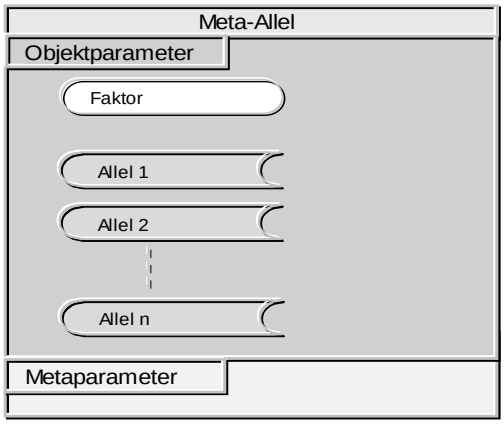


Abbildung 4.32: *Das Metaallel*

Zu diesem Zweck enthält das Allel Speicherstellen, die andere Allele aufnehmen können, wobei diese einer der vier Grundtype sein können oder selbst wieder ein Metaallel sind. Hierdurch ergibt sich eine Baumstruktur, die als Knoten Metaallele und als Blätter eine der vier Basistypen enthält.

Weiterhin hält ein Metaallel einen Faktor als Objektparameter, der ein Multiplikator für die Ausgangsparamter darstellt, die in der Baumstruktur gespeichert sind. Diese werden während der Anpassung nicht verändert, sondern dienen ausschließlich als Speicher der relativen Verhältnisse der Werte untereinander. Deshalb wird durch alle genetischen Operatoren ausschließlich der Faktor im Knoten-Metaallel des Baums verändert. Bei der Initialisierung wird dieser Wert mit 1.0 belegt.

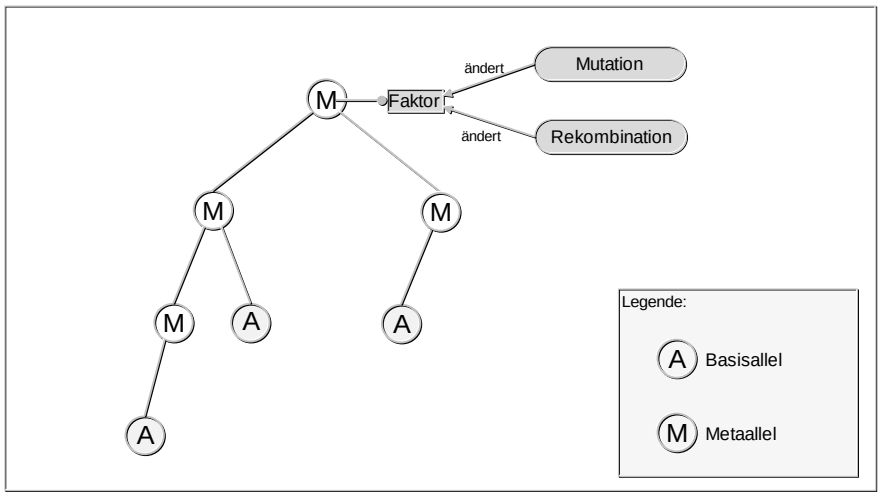


Abbildung 4.33: *Wirkungsweise genetischer Operationen beim Metaallel*

4.8.2 Zurückschreiben der Parameter in das Modell (Realize)

Zur Bestimmung der Fitneß eines Individuums muß zunächst ein Abgleich mit dem Modell stattfinden. Mittels des Voxelraums kann ein Abstand zum Scan berechnet werden und es resultiert daraus ein Fitneßwert, der im Individuum gespeichert wird.

Dieses Zurückschreiben der Parameter erfolgt dadurch, daß die Realize-Funktion der einzelnen Allele des Genoms der Reihe nach aufgerufen werden. Die Allele sind in der Lage ihre Parameter selbst ins Modell zu übertragen. Zu beachten ist die Realize-Funktion des Metaalls. Diese ruft nacheinander die entsprechenden Funktionen der eingebundenen Allele mit dem seinem Faktor-Parameter auf. Handelt es sich dabei um Basisallele, so schreiben diese ihren Wert multipliziert mit diesem Faktor ins Modell zurück. Ist das eingebundene Allel wieder ein Metaallel, so kaskadiert der Vorgang von neuem. Abbildung 4.34 stellt diesen Vorgang anschaulich dar.

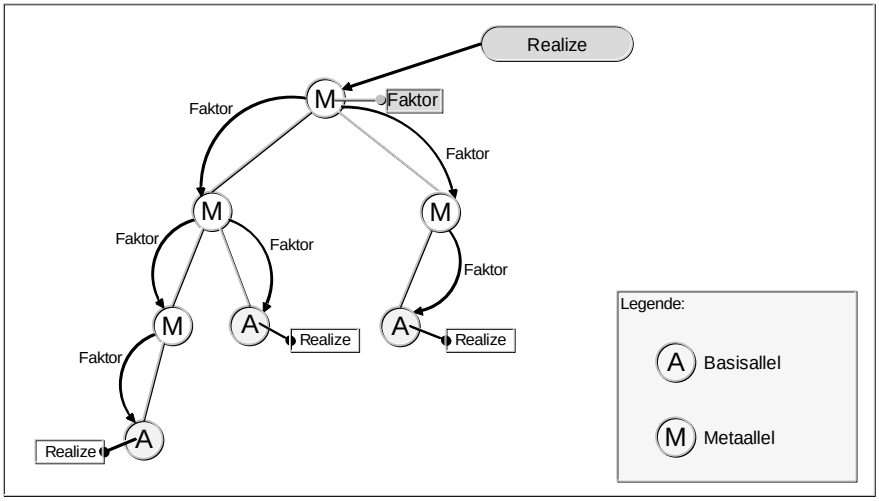


Abbildung 4.34: *Realize beim Metaallel*

4.9 Initialisierungsdatei des Algorithmus

Um eine maximale Flexibilität zu Erreichen und den Algorithmus für die Entwicklungsphase leicht konfigurierbar zu gestalten wird ein Großteil der Parameter und Strukturdefinitionen in eine Initialisierungsdatei ausgelagert. Diese wird zum Initialisierungszeitpunkt durch einen Parser eingelesen, ausgewertet und in einer internen Datenstruktur manifestiert. Die Datei ist in einem ASCII-Format gespeichert um sie leicht mit einem Texteditor bearbeiten zu können.

Die folgende Abbildung zeigt exemplarisch eine Populationsdefinition als Ausschnitt des Ini-Files. Diese sind sequentiell beginnend mit der Zahl 0 durchnumeriert. Der Eintrag **`LastPopulation= cTRUE`** zeigt an, daß es sich um die letzte Population handelt.

```
[Population   3]

OutputString= "*******     Grobanpassung"
ParentSize=        15
OffspringSize=    100
Accuracy=       0.007
Strategy Type=  komma

Genome size=        6

Allel   0= 0.005 0.40 0.00 unfixedprotected binary XXXX    SETranslationAllel
Allel   1= 0.002 0.40 0.00 unfixedprotected binary XXXX    SERotationAllel
Allel   2= 0.050 0.50 0.00 unfixedprotected binary OHW     ArbitraryBoneLengthAllel
Allel   3= 0.050 0.50 0.00 unfixedprotected binary QS10101 ArbitrarySkinSliceAllel
Allel   4= 0.050 0.50 0.00 unfixedprotected binary UAL0201 ArbitrarySkinPointAllel
Allel   5= 0.010 0.50 0.00 unfixedprotected binary GHUR    ArbitraryJointAllel
Allel   6= 0.050 0.50 0.00 unfixedprotected binary XXXX    UpperBodyLengthUDA
```

Abbildung 4.35: *Definition der Genomstruktur einer Algorithmusstufe*

Folgende Schlüsselworte sind Parameter einer Populationsdefinition:

OutputString optionale Angabe einer Zeichenkette zu Debuggingzwecken. Erlaubter Datentyp: Zeichenkette.

ParentSize Anzahl der (Eltern-)Individuen in der Population. Erlaubter Datentyp: Integerwert.

OffspringSize Anzahl der Nachkommenschaft der aktuellen Population. Diese wird auf die Populationsgröße reduziert (Reduction-Operator). Erlaubter Datentyp: Integerwert.

Accuracy Definiert die Breite des Fensters für das Abbruchkriterium. Unterschreitet die Varianz in der Population diese Bandbreite, wird die aktuelle Stufe beendet und die nächste Stufe initialisiert. Erlaubter Datentyp: Fließkommazahl.

Window Length Breite des Fitneßfensters für die Schleifenabbruchbedingung. Erlaubter Datentyp: Integerwert. (Bemerkung: noch nicht implementiert.)

Strategy Type Bestimmt den Strategietyp der aktuellen Stufe. Erlaubte Werte: (**`plus`**, **`komma)`.**

Genome size Bestimmt die Zahl der im Anschluß folgenden Strukturdefinition des Genoms. Erlaubter Datentyp: Integerwert.

Allel i Definition der i-ten Komponente des Genoms. i liegt im Bereich von Null bis *Genome size*. Erlaubter Datentyp: Folge von 7 Typen, welche die Parameter des jeweiligen Allels bestimmen. Diese Typen sind in

der Reihenfolge von links nach rechts folgende Parameter (mit Datentypen): Mutationsrate (Fließkommazahl), Metamutationsrate (Fließkommazahl), Initialmutationsrate (Fließkommazahl), Mutationsart (**`fixed`**, **`unfixed`**, **`unfixedprotected`**)[1] , Rekombinationsart (**`binary`**, **`intermediary`**, **`none`**)[2] , RAMSIS-Objektname (gültiger Objektname des RAMSIS oder beliebige Zeichenkette bei komplexen Objekten) und Alleltyp (vordefinierter Grundtypus oder neudeklarierter Typ).

Es ist darauf zu achten, daß der Eintrag unter *Genome Size* die korrekte Anzahl von Alleldefinitionen angibt. Weiterhin müssen die Allelnummern fortlaufend belegt sein.

Im Zuge der Beschleunigung der Anpassung in der Endphase wurde noch eine besondere Populationsdefinition implementiert, welche die im Abstandsmaß definierten Hautscheiben und Hautpunkte mittels einem Hillclimbing-Algorithmus einzeln anpaßt. Im strengen Sinne handelt es sichhierbei nicht um eine *Population* im Sinne der Evolutionsalgorithmen, vielmehr um ein Einindividuenverfahren, das mittels Hillclimbing-Algorithmus optimiert.

Indikator für diese Definition ist der Eintrag **`PerformHillclimbing= cTRUE`** , der das Programm auffordert in dieser Stufe die Hillclimbing-Routine aufzurufen. Weiterhin ist dann die Angabe der Schlüssel **`Accuracy`** und **`Acceptance Threshold`** erforderlich:

Accuracy Dieser Key definiert nun die Abbruchgenauigkeit der Hillclimb-Routine, d.h. den Abstand der Hautscheibe zum Scan. Erlaubter Datentyp: Fließkommazahl.

Acceptance Threshold

Dieser Wert stellt einen Schwellwertbedingung an die absolute Fitneß eines Hautpunkts dar. Ist diese Überschritten, ist der Punkt zu weit von der Scanpunktewolke entfernt und es wird keine Anpassung durchgeführt.

Alle weiteren Definitionen möglicher Populationsattribute in dieser Population sind irrelevant und werden ignoriert.

```
[Population  17]
OutputString= "performing HillClimbing"
PerformHillclimbing= cTRUE
Accuracy=0.00001 ; 1/100 mm
Acceptance Threshold=20
```

Abbildung 4.36: *Spezialpopulation, die nur Hillclimbing durchführt.*

In jeder Populationsdefinition müssen weiterhin Objekte aufgeführt werden, die das Fitneßmaß definieren. Im wesentlichen ist dies eine Aufzählung der Hautpunkte und Hautscheiben, deren Abstand im Voxelraum gemessen und aufsummiert wird. Zu jedem *Hautobjekt*, also Hautschei-

1. **`fixed`** erlaubt keine Mutation der Kontrollvariable. **`unfixed`** läßt diese in beliebigen Grenzen zu und **`unfixedprotected`** erlaubt ebenfalls die Mutation der Kontrollvariable mit der Einschränkung, daß eine Schwelle für die Mutationsrate nicht unterschritten werden darf. (Verhindert das Absinken der Mutationsrate auf 0)
2. **`binary`** wählt die binäre Rekombination, **`intermediary`** die intermediäre und **`none`** verbietet die Rekombination.

be oder Hautpunkt des RAMSIS, kann ein Wert angegeben werden, der sein relatives Gewicht gegenüber den anderen bestimmt. Des weiteren können mit einem zusätzlichen Attribut bei Hautscheiben die Auswahl der zugehörigen Hautpunkte eingeschränkt werden.

```
[Population   3]

SkinObjects= 9

; KOPF
SkinObject   0=   QS082902   front  0.1
SkinObject   1=   QS082905   front  0.1
SkinObject   2=   QS082907   front  0.2
SkinObject   3=   QS082909   front  0.1
SkinObject   4=   QS082910   front  0.1
; Hals
SkinObject   5=   QS020601     all  0.1
SkinObject   6=   QS020602     all  0.1
; SCHULTER
SkinObject   7=   QS040901     all  0.1
SkinObject   8=   QS040902     all  0.1
```

Abbildung 4.37: *Fitneßmaß-Definition einer Population.*

Folgende Schlüsselworte sind bei dieser Definition gegeben:

SkinObjects Gibt die Anzahl der definierten Hautobjekte des Fitneßmaßes an. Erlaubter Datentyp: Integerwert.

SkinObject i Definiert ein Hautobjekt, das Teil der Fitneßfunktion ist. An dieser Stelle ist eine Angabe von drei Datentypen notwendig. Es sind in dieser Reihenfolge *Hautobjektnamen* bzw. *Namen des komplexen Hautobjektes* (Zeichenkette), *Auswahlattribut* (**`all`**, **`front`**, **`back`**, **`withoutSides`**, **`sidesOnly`**, **`singlePoint`**) und *Gewichtung* (Fließkommazahl) sind.

Hierbei ist zu beachten, daß der Eintrag unter **`SkinObjects`** die richtige Anzahl von Definitionen angibt bzw. daß die Objektnummern fortlaufend belegt sein müssen.

Komplexe Hautobjekte sind in einem eigenen Abschnitt des IniFiles definiert, welches durch **`[CompoundSkinObject Section]`** eingeleitet wird. Im wesentlichen werden hier neue Namen für eine Menge von Hautobjekten definiert, wobei auch bereits definierte komplexe Hautobjekte wieder zu größeren Gruppen zusammengefaßt werden dürfen. Es entsteht eine Objekthierarchie von semantisch zusammengehörigen Objekten.

```
[CompoundSkinObject Section]

NumberOfCompoundSkinObjects= 1

NewSkinObject 0= headSO 3

CompoundSkinObject 0 0= QS040404
CompoundSkinObject 0 1= QS050505
CompoundSkinObject 0 2= QS060606
```

Abbildung 4.38: *Definition komplexer Fitneßmaßeinheiten*

NumberOfCompoundSkinObjects

Gibt die *Gesamtzahl* der neu definierten Objekte an. Erlaubter Datentyp: Integerwerte.

NewSkinObject *i* Definiert den *Namen* (Zeichenkette) und die *Zahl* (Integerwert) der zu diesem Objekt zusammenzufassenenden Komponenten.

CompoundSkinObject *i j*

Dieser Eintrag gibt den Namen der Komponente *j* des komplexen Objekts *i* an, wobei der Name ein RAMSIS-Hautobjekt oder ein bereits definiertes komplexes Objekt sein kann.

Ein weiterer Abschnitt des IniFiles **`[CompoundAllelType Section]`** definiert komplexe Alleltypen. Wie bei den Hautobjekten kann ein komplexer Typ entweder aus den Basisalleltypen **`SETranslationAllel`**, **`SERotationAllel`**, **`SkinSliceAllel`**, **`ArbitraryBoneLengthAllel`**, **`ArbitraryJointAllel`** bestehen oder selbst wieder ein bereits definierter komplexer Typ sein.

```
[CompoundAllelType Section]
NumberOfCompoundAlleles=   26;

NewAllelType   0= UpperBodyLengthUDA 0.0 0.0 0.0 unfixedprotected intermediary 5

CompoundObject 0 0= ArbitraryBoneLengthAllel UHW
CompoundObject 0 1= ArbitraryBoneLengthAllel OBW
CompoundObject 0 2= ArbitraryBoneLengthAllel UBW
CompoundObject 0 3= ArbitraryBoneLengthAllel OLW
CompoundObject 0 4= ArbitraryBoneLengthAllel ULW
```

Abbildung 4.39: *Definition komplexer Allele*

NumberOfCompoundAlleles

Gibt die Gesamtzahl der neu definierten Allele an. Erlaubter Datentyp: Integerwerte.

NewAllelType *i* Definiert den neuen Typ mit Nummer *i*. Datentyp: 7 Typen, die der Reihenfolge nach den eigenen *Namen* (Zeichenkette), *Mutationsrate* (Fließkommazahl), *Metamutationsrate* (Fließkommazahl), *Initialmutationsrate* (Fließkommazahl), *Mutationsart* (**`fixed`**, **`unfixed`**, **`unfixedprotected`**), *Rekombinationsart* (**`binary`**, **`intermediary`**, **`none`**) und die *Zahl der Komponenten* (Integerwert) angeben.

CompoundObject *i j*

Definiert eine Komponente *j* des neuen Alleltyps *i*. Datentyp: 2 Typen: der *Name des Typs* (Zeichenkette), der entweder ein Elementaralleltyp oder ein komplexer Typ sein kann, und der *Name der Komponente* (Zeichenkette).

Um Gelenkwinkelbeschränkungen des intern gespeicherten RAMSIS-Modells vor der Anpassung ändern zu können existiert ein besonderer Abschnitt **`[Joint Angle Settings]`** in der Initialisierungsdatei.

```
[Joint Angle Settings]

NumberOfSettings=   5

SetDegreeOfFreedom   0= GHZ NormalDoF       -2   2
SetDegreeOfFreedom   1= GHZ BinormalDoF     -5   5
SetDegreeOfFreedom   2= GHZ TangentialDoF -100 100

SetDegreeOfFreedom   3= GHUL NormalDoF      -3 XXX
SetDegreeOfFreedom   4= GHUR NormalDoF     XXX   3
```

Abbildung 4.40: *Gelenkwinkelbeschränkungsteil des IniFiles.*

NumberOfSettings

Benennt die Zahl der folgenden Beschränkungsänderungen.

SetDegreeOfFreedom *i*

Beschränkungsänderung Nummer *i*. Es folgen 4 Parameter, die den Namen des Gelenks (Zeichenkette), den Freiheitsgrad (**`NormalDoF`**, **`BinormalDoF`**, **`TangentialDoF`**) und den neuen Wertebereich mit der Einheit Grad, der durch 2 Fließkommazahlen dargestellt wird, die das Maximum bzw. Minimum angeben. Wird eine der Zahlen durch eine nichtnumerische Zeichenkette ersetzt, so wird die betreffende Einstellung nicht verändert.

4.10 Hillclimbing-Algorithmus zur Feinanpassung

Ist die Länge der Knochen bzw. die Winkel der Gelenke, also die innere Struktur bereits gut gefunden und damit die semantischen Informationen von Modell und Scan abgeglichen, so kann die Feinanpassung der Hautscheiben an den Scan von einem einfachen Hillclimbalgorithmus vorgenommen werden. In der Abbildung sind Scan (farbig) und eine Hautscheibe (weiß) mit

zugehörigem Knochen dargestellt, der Abstand der Hautscheibe zum Scan ist zu Darstellungszwecken stark vergrößert.

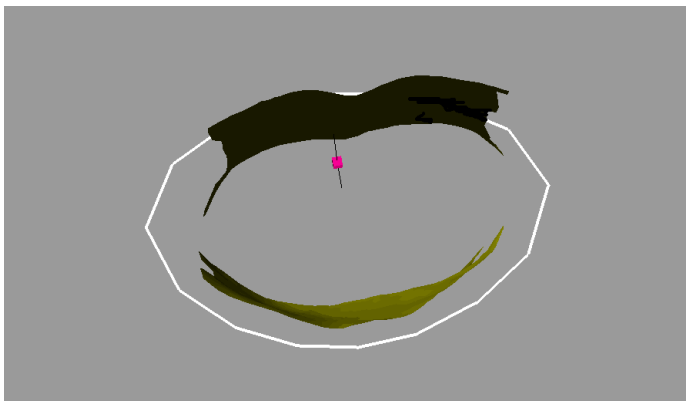


Abbildung 4.41: *Hautscheibe mit zugehörigem Scanausschnitt.*

4.11 Individuenaustausch mehrerer LIFE-Threads

Um den Algorithmus effizient auf Multiprozessorsystemen und in Rechnernetzwerken einsetzen zu können, wurde die Schnittstelle um Funktionen erweitert, die den Austausch von Individuen der Populationen unterstützen.

Gemäß dem Populationenmodell können unabhängig von der aktuellen Berechnung eines LIFE-Objekts Individuen angefragt werden, die in andere Populationen migrieren können. Diese Übergabeindividuen werden direkt in die aktuell bearbeitete Population integriert und nehmen dann am genetischen Prozeß sofort teil.

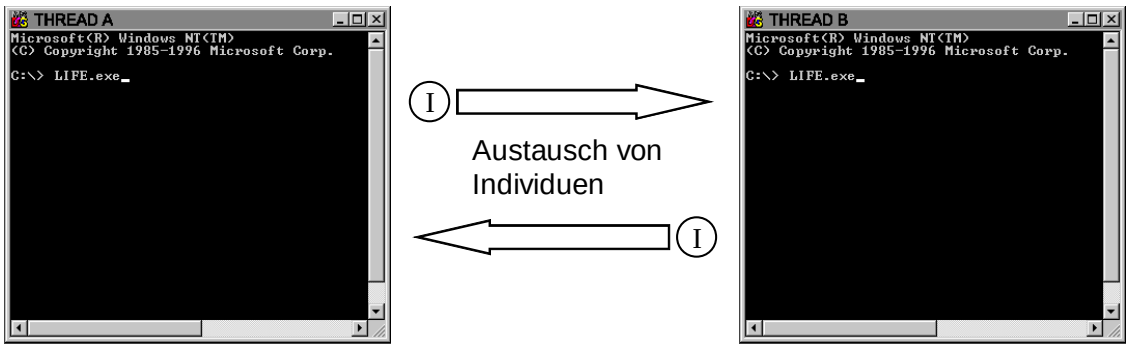


Abbildung 4.42: *Mehrere LIFE-Threads tauschen Individuen aus*

5 Diskussion und Ausblick

5.1 Diskussion

Die Qualität eines Abgleichsvorgangs kann unter verschiedenen Gesichtspunkten bewertet werden:

- Übereinstimmung der Modellhaut mit den Scandaten (Deckung der externen Struktur mit dem Scan).
- Plausibilität der Anpassung inneren Modellstruktur.
- Vergleich abgeleiteter Körperdaten mit manuell erfaßten Daten.[1]

Dabei läßt sich die Bewertung der Modelle auf quantitativer und auf qualitativer Ebene durchführen.

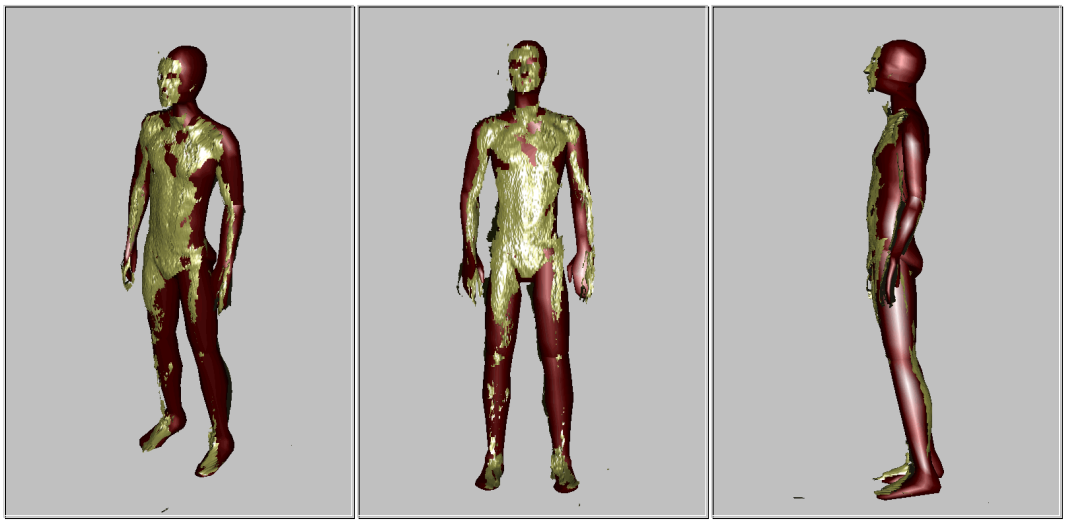
Als qualitative Bewertung bietet sich beispielsweise die voxelraumbasierte Fitneßfunktion an, welche die Grundlage des Evolutionsalgorithmus darstellt. Allerdings bewertet sie ausschließlich gemäß Punkt eins obiger Aufzählung, also auf Basis des äußeren Modells. Ein Vorteil dieser Vorgehensweise besteht in der Möglichkeit ihrer automatischen Durchführbarkeit.

Eine quantitative Analyse kann die manuelle optische Bewertung der Ergebnisse sein. Sie nutzt den Vorteil, daß das menschliche Gehirn optimal dafür geeignet ist solche Vergleiche durchzuführen. Der Mensch hat ein ausgeprägtes Feingefühl für die Proportionierung der Gliedmaßen sowie für die Abschätzung der Korrektheit der Lage der inneren Modellstruktur. Voraussetzung dafür ist die optische Aufbereitung der Daten und des Modells in einer Art, daß die visuellen Fähigkeiten des Menschen gut genutzt werden können.

Das heißt, das Modell bzw. der Scan sollten in einer schnellen 3D-Darstellung begutachtet werden können, welche zusätzlich verschiedene Optionen wie Punktedarstellung, Schattierung, Hidden Surface etc. bietet, welche die menschliche Wahrnehmung entsprechend unterstützen.

1. Gerade dieser Punkt ist für die Anwendung von entscheidender Wichtigkeit, da letztlich nur auf Basis dieser Daten gearbeitet wird.

Folgende Abbildung zeigt solch eine Visualisierung eines Abgleichsergebnisses aus verschiedenen Betrachtungswinkeln.



Ansicht schräg Frontansicht Seitenansicht

Abbildung 5.1: *Verschiedene Ansichten einer Anpassung mit zugehörigem Scan-Datensatz*

5.1.1 Anpassungsergebnisse

Zur Evaluierung des Abgleichsprozesses standen insgesamt circa 30 Datensätze zur Verfügung. Dazu ist anzumerken, daß diese allerdings nicht von höchster Qualität waren. Sie enthalten abgesehen von Konturen des Gesichtes keine Punkte des Kopfes[1]. Weiterhin gibt es seitliche Fehlstellen, die durch Abschattungen bedingt sind.

Zum Vergleich der Ergebnisse vermag das Programm innerhalb eines Batch-Betriebs[2] nach jeder Anpassung einen “Screen Shot” als Bitmap zu Kontrollzwecken zu speichern. Diese Bilder erlauben einen schnellen Vergleich vieler Ergebnisse in einer Übersichtsdarstellung.

Eine Auswahl von gelungenen Anpassungen zeigt Abbildung 5.2. Bei diesen Scans wurden die Gliedmaßen korrekt gefunden, die Körperhöhe sollte korrekt gefunden sein und die Umfänge stimmen ebenfalls recht gut überein. Bemerkenswert ist, daß der Algorithmus sehr stabil auf

1. Stark diffuse Reflexionen im Bereich von Haaren führt zu Ausfällen in der Messung. Daher ist der Kopf und damit auch die Körperhöhe meist nur anhand des Gesichtes rekonstruierbar. I.a. stellt dies zur Ableitung von Individualdaten im Bekleidungsbereich kein großes Problem dar.
2. Der Batch-Modus erlaubt die Abarbeitung vieler Anpassungen im Stand-Alone-Betrieb.

Störungen reagiert (siehe Bild 2 und 3). Weiterhin werden fehlende Meßwerte bis zu einem gewissen Grad ausgeglichen.

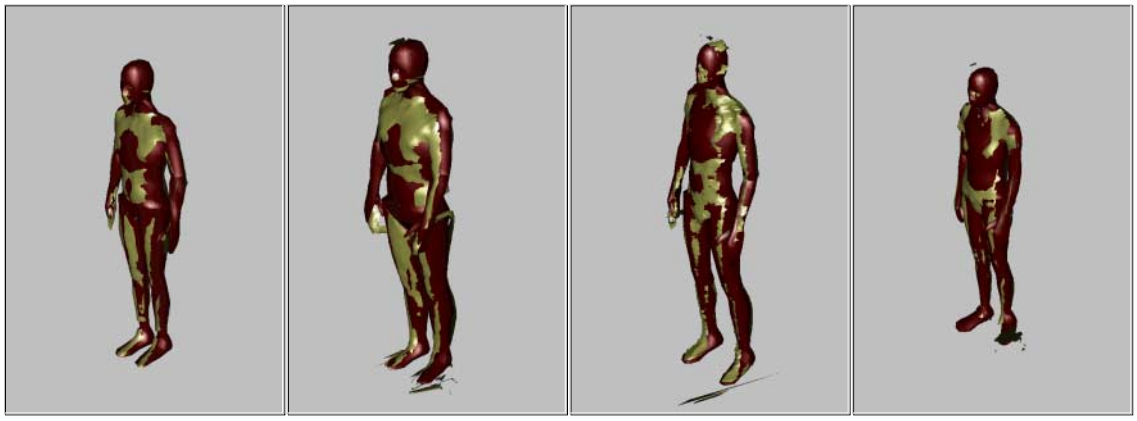


Abbildung 5.2: *Verschiedene gute Anpassungsergebnisse*

Sinkt die Qualität der Scans unter ein gewisses Niveau ab sind Fehlanpassungen die Folge.

Der optische Eindruck des Scans links in Abbildung 5.3 legt den Schluß nahe, daß die Anpassung nicht gut gelungen ist. Aufgrund eines im Datensatz fast nicht vorhandenen linken Armes kommt es zu Fehlern bei der Bestimmung der Schlüsselbeinlänge.

Beim mittigen Scandatensatz sind keine Füße vorhanden, so daß die Beine des Modells zu kurz eingestellt wurden. Weiterhin waren am linken Arm so wenige Scanpunkte vorhanden, daß der Modellarm in den Körperbereich hineingezogen wurde.

Beim rechten Scan in Abbildung 5.3 sind schätzungsweise weniger als 50% der Körperfläche erfaßt, was auf Probleme bei der Datenerfassung zurückzuführen ist. Deshalb gelingt die Anpassung nicht mehr.[1]

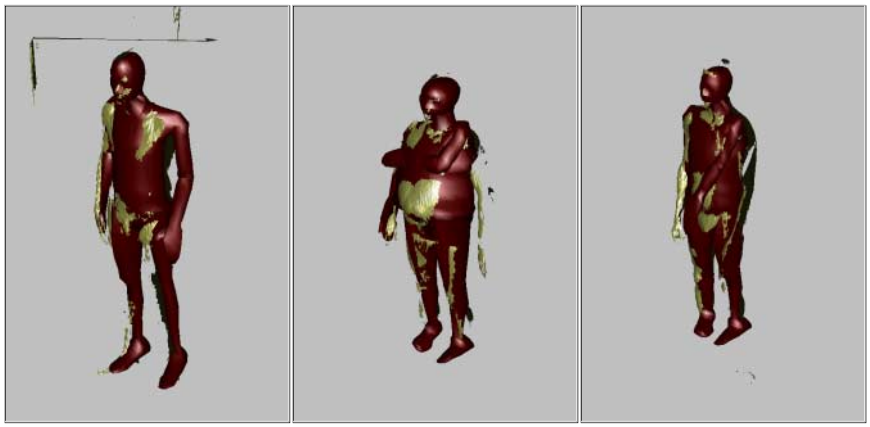


Abbildung 5.3: *Einige mißlungene Anpassungen*

Bei einer hinreichend guten Qualität der Scans approximiert der Anpaßalgorithmus die Scans recht gut. Dabei ist es keineswegs Voraussetzung, daß Scan-Datenwolke geschlossen ist. Sind kleinere Bereiche der Körperoberfläche der Objektperson nicht erfaßt, so wird die Anpassung

1. Generell ist festzustellen, daß der Algorithmus um so stabiler arbeitet je qualitativ hochwertiger die Datenerfassung war. Der aktuelle Systemaufbau des TOPAS-Scanners arbeitet dabei offensichtlich an seinem Grenzbereich.

des Körperteils trotzdem korrekt durchgeführt. Eine exakte Anpassung wird dabei allerdings nicht mehr möglich sein. Erst bei großflächigeren Fehlstellen im Meßdatensatz werden Körperteile nicht mehr korrekt angepaßt bzw. das Modell deformiert.

Die aktuelle Version des Programms erreicht eine Trefferquote von ca. 95% bei Scans, die optisch als “gelungen” bezeichnet werden. Die Steigerung der Robustheit und zugleich die Verbesserung des Datenmaterials ist Gegenstand zukünftiger Forschungsarbeit.

5.1.2 Verbesserungen durch LIFE-Threads

Exemplarisch wurde für eine 2-Prozessor-Parallelrechner das Migrationsmodell implementiert, welches Anpassungen in zwei Threads parallel durchführt. Diese Threads tauschen in unregelmäßigen Abständen Individuen zwischen den internen Populationen aus.

Verglichen wurde die Laufzeiten der Anpassungen mit den Single-Thread Applikationen, wobei die Initialisierungsdateien für alle Versionen die gleichen waren. Das heißt, es gab pro Thread u.a. die gleiche Populationsgröße, was bedeutet, daß die Double-Thread-Version gegenüber der einfachen Version insgesamt doppelt so viele Individuen verwalten mußte.

Vorläufige Ergebnisse ergaben, daß die Multiprozessor-Version ungefähr 10% schneller arbeitet als das einfache Programm. Die subjektive Beurteilung der Anpaßergebnisse ergab eine höhere Robustheit der Anpassung bei Scans minderer Qualität. Eine größere Geschwindigkeitssteigerung ist zu diesem Parallelisierungsmodell nicht zu erwarten.

5.2 Ausblick

Im folgenden werden einige Kriterien aufgezählt, bezüglich derer eine Verbesserung des Abgleichsalgorithmus möglich erscheint:

- **Steigerung der Geschwindigkeit**

  Die hohe Leistung des Anpassungssystems ermöglicht einen Abgleich des Modells mit einem Scan in nur ca. 3 bis 7 Minuten. Trotzdem ist es für eine industrielle Anwendung, die um wirtschaftlich arbeiten zu können eine hohe Durchsatzrate benötigt, sinnvoll, diese Werte noch weiter zu steigern.

- **Steigerung der Robustheit**.

  Eine Verbesserung der Robustheit bezweckt eine Maximierung der Ergebnis-Qualität bezüglich des vorhandenen Datenmaterials. Ziel ist hierbei auch bei qualitativ schlechtem Material ein optimales Quantum an Anthropometrie-Information zu erkennen, das aus dem Scan ableitbar ist[1].

1. Sind die Beine durch Fehlstellen bedingt nicht im Scan vorhanden, so sollte die Anthropometrie-Bestimmung anderer Körperteile (z.B. Brustumfang) trotzdem möglich sein.

Im folgenden werden einige Ansätze beschrieben, die eine Verbesserung obiger Punkte erlauben.

5.2.1 Optimierung der Strategieparameter

Wie in Kapitel 3 beschrieben, enthält die Initialisierungsdatei eine sehr große Anzahl von Parametern, die alle optimal bestimmt werden müssen, damit der Anpaßvorgang schnell und robust arbeitet.

Allein dieser Umfang ergibt wieder ein hochkomplexes Problem in einem vieldimensionalen Parameterraum.

Von Vorteil wäre ein Verfahren, welches diese Optimierung automatisch vornimmt. In Betracht kommt dafür wieder eine Evolutionsstrategie.

Im folgenden ist eine Möglichkeit für solch ein Verfahren beschrieben:

Ausgangspunkt für solch eine Evolution der Ini-Files sind eine kleine Anzahl Initialisierungsdateien, die sich unter realen Bedingungen bewährt haben. Das heißt, mit ihnen wurden schon erfolgreiche Anpassungen durchgeführt. Damit kommen diese Dateien als Basispopulation des Metaverfahrens in Frage.

Mit jedem Programmlauf von LIFE wird für die einzelnen Ini-Files eine qualitative Bewertung erzeugt, die ihre Fitneß darstellt. Ist jedes Individuum der Population bewertet, so kann durch geeignete genetische Operatoren eine neue Population erzeugt werden.

Damit erhält man eine sukzessive Verbesserung der Initialisierungsdateien, die aufgrund der genetischen Optimierungsprinzipien besser sind als die ursprünglichen. Abbildung 5.4 stellt dieses Konzept anschaulich dar. .

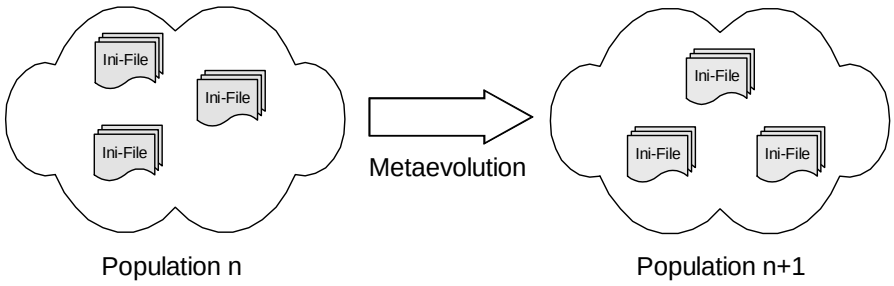


Abbildung 5.4: *Evolution von Initialisierungsdateien*

5.2.2 Beschleunigung des Durchsatzes mittels einer Rechnerpipeline

Eine Möglichkeit, den Datendurchsatz des Programms zu beschleunigen, ist die Konzeption einer Rechnerpipeline. Dahinter steckt die Idee, den Prozeß auf mehrere Rechner[1] zu verteilen. Bei einer großen Anzahl zu bearbeitender Daten hat das eine maßgebliche Verbesserung der

1. Hier bietet sich beispielsweise ein SCI gekoppeltes PC-Cluster an [BUT97], welches mit 8 PCs schon eine theoretische Leistung von über 1500 MFlops erreicht. Der Preis eines solchen Systems beläuft sich auf deutlich unter 50000 DM.

Laufzeit pro Anpassung zur Folge. Dabei ist zu beachten, daß die Durchlaufzeit nur in dem Maße gesteigert werden kann, in dem die einzelnen Stufen getaktet werden können. Gegebenenfalls müssen mehrere schnelle Stufen auf einem Rechner zusammengefaßt werden um die Prozessor-Pipeline optimal auszulasten.

Eine Verteilung in einem schon bestehenden Netzwerk kann vorhandene Rechenleistung (bspw. zur Nachtzeit) nutzen und ist daher mit geringen zusätzlichen Kosten verbunden.

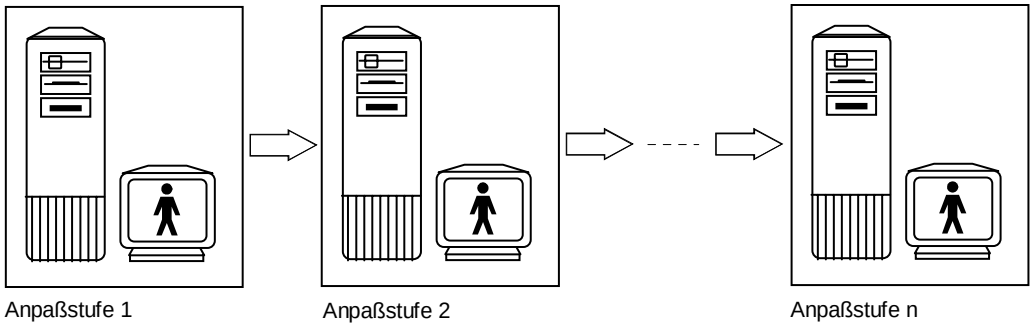


Abbildung 5.5: *Rechnerpipeline zur Verteilung der Algorithmusstufen über mehrere Rechner*

5.2.3 Memetischer Ansatz

Ein weiterer Ansatz wäre, Informationen der vorangegangenen Anpaßvorgänge zu protokollieren und späteren Anpassungen in einer Form als Wissensbasis zur Verfügung zu stellen. Damit stünde dem Anpaßalgorithmus eine gewisse *Weginformation* anderer Anpassungen zur Verfügung, die mittels einer geeigneten Heuristik zu Nutzen wäre.

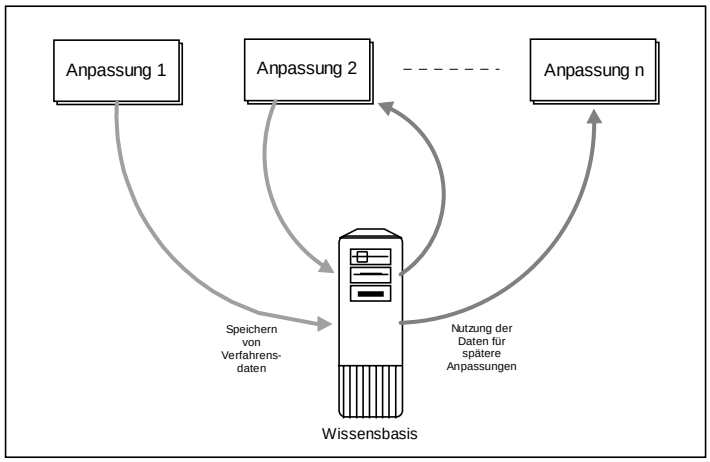


Abbildung 5.6: *Nutzung von Metawissen über die Programmläufe hinweg*

6 Zusammenfassung

Die Aufgabenstellung dieser Arbeit bestand im Entwurf und der Entwicklung einer Evolutionsstrategie zum Abgleich eines Menschmodells mit einem 3D-Körper-Scan.

Dabei waren verschiedene Randbedingungen zu beachten:

- Beim Abgleich sind besonderen Anforderungen des Modellabgleichs mit RAMSIS zu beachten. Im besonderen ist dies, eine Übereinstimmung der Modellhaut mit dem Scan unter gleichzeitiger Schätzung einer plausiblen Skelettform herbeizuführen.
- Das Verfahren sollte sowohl an fehlerbehafteten Scan-Daten Anpassungen vornehmen als auch für diesen Fehlstellen entsprechende Modellteile sinnvolle Einstellungen finden können

Durch die Entwicklung eines voxelraumbasierten Abstandsmaßes als Grundelement der Evolutionsstrategie sowie der Konzeption des Verfahrens als Stufenalgorithmus wurde versucht diesen Punkten Rechnung zu tragen.

Es hat sich gezeigt, daß die Berechnung des Abstandsmaßes mittels des Voxelraumes effizient und effektiv ist. Dadurch ist es dem Algorithmus möglich selbst Scans, die bis zu einem gewissen Grad Fehlstellen enthalten, anzupassen.

Durch die Einteilung des Verfahrens in verschiedene Stufen, die nacheinander Anpassungen mit zunehmend kleinerer Granularität durchführen, wird eine gute Übereinstimmung des äußeren Modells mit den Meßwerten erreicht. Gleichzeitig gelingt eine plausible Schätzung der Position des inneren Modells.

Die flexible Konfigurierbarkeit des Abgleichsprozesses (durch eine Initialisierungsdatei) führt zu einer umfangreichen Programmierbarkeit des Vorgangs. So ist prinzipiell ein vollständiger haltungsunabhängiger Abgleich sowie auch beispielsweise eine speziell auf Stehhaltung optimierte Strategie möglich.

Die gute Qualität aktueller Anpassungsergebnisse läßt erwarten, daß die für eine Anwendung der Meßtechnik wichtige Ableitung von produktspezifischen Daten mit hoher Präzision möglich ist.

Als Ergebnis aus verschiedenen Testläufen kann die grundsätzliche Eignung des Verfahrens für Anforderungen des automatischen Modellabgleichs unter den hier gegebenen Anforderungen bestätigt werden.

Die aktuelle Version des Programms erreicht eine Trefferquote von ca. 95% bei Scans, die optisch als “gelungen” bezeichnet werden. Eine weitere Steigerung der Robustheit und zugleich die Verbesserung des Datenmaterials ist Gegenstand zukünftiger Forschungsarbeit.

A Datenrepräsentation beim LTP

A.1 Das Transportproblem

Problemstellung: Aufstellen eines minimalen Kostenplans, um ein Produkt P von einer Auswahl von Quellen zu einer Menge von Senken zu transportieren, unter folgenden Restriktionen: Jede einzelne Quelle hat eine Menge des Produkts vorrätig, die nicht unterschritten werden darf. Genausowenig darf einer Senke mehr geliefert werden, als sie bestellt hat.

Es existiert eine Kostenfunktion, die zu je einer Quelle und einer Senke angibt, wie hoch die Kosten sind, eine Menge x des Produkts zwischen diesen zu transportieren. Nach Möglichkeit soll die Gesamtnachfrage gedeckt werden, das Gesamtangebot abtransportiert werden und die Gesamttransportkosten minimal gehalten werden.

Definition: *Transportproblem* (TP)

Im folgenden seinen S die Menge der Quellen, D die Menge der Senken; $source(s \in S)$ gebe den Materialvorrat der Quelle s an; $dest(d \in D)$ gebe die Bedarf der Senke d an.

Weiterhin bezeichne $x_{i,j}$ die von Quelle i nach Senke j transportierte Menge des Produkts; $f_{i,j}(x)$ bezeichne die Kostenfunktion, welche die Kosten für den Transport einer Produktmenge x von Quelle i nach Senke j angebe.

Damit formuliert sich das *allgemeine Transportproblem*[1] so:

$$minimize \sum_{i \in S} \sum_{j \in D} f_{i,j}(x_{i,j}) \qquad \textbf{(6.0.1)}$$

unter den beiden Nebenbedingungen:

i) $$\forall i \in S : \sum_{j \in D} x_{i,j} \leq source(i) \qquad \textbf{(6.0.2)}$$

ii) $$\forall j \in D : \sum_{i \in S} x_{i,j} \geq dest(j) \qquad \textbf{(6.0.3)}$$

1. *minimize* gibt dabei an, daß nach dem Minimum gesucht wird.

Gilt weiterhin $\sum_{i \in S} source(i) = \sum_{j \in D} dest(j)$,

d.h. die Gesamtnachfrage entspricht dem Gesamtangebot, so heißt dieses TP auch *ausgeglichenes Transportproblem* (*balanced TP*); die Nebenbedingungen (6.0.2) und (6.0.3) wandeln sich damit in Gleichungen.

Das TP heißt *linear*, falls für $f_{i,j}(x)$ gilt:

$$f_{i,j}(x) = x_{i,j}\, cost(i,j) \qquad \textbf{(6.0.4)}$$

für eine Kostenfunktion $cost(i,j)$, die die Einheitstransportkosten von Quelle *i* nach Senke *j* angibt; in diesem Falle spricht man vom *linearen Transportprobmen (LTP).*

Sind die Wertebereiche von $source(s \in S)$ und $dest(d \in D)$ diskret, so gilt dies auch für die Lösungen $x_{i,j}$.

Während das LTP recht gut untersucht ist und es viele gute Lösungsmethoden gibt, entzieht sich die allgemeine Form des TP bisher jeglichen Lösungsansatzes.

Ein Beispiel:

Die Kostenfunktion $cost(i,j)$ des LTP läßt sich gut als Matrix schreiben:

cost	d1	d2	d3	d4
s1	10	0	20	11
s2	12	7	9	20
s3	0	14	16	18

Mit $source = (5, 25, 5)$ und $dest = (5, 15, 15, 10)$ ergibt sich folgende optimale Lösung $M = [x_{i,j}]_{i \in S, j \in D}$:

x	d1	d2	d3	d4
s1	0	5	0	10
s2	0	10	15	0
s3	5	0	0	0

Damit ergeben sich die minimalen Transportkosten zu *f(x)* = 10·11+10·7+15·9 = 322.

A.2 Die Kodierung beim Linearen Transportproblem

Ein standard-GA würde eine Umkodierung der Matrix $M \rightarrow \{0, 1\}^q$ vornehmen und darauf mit Standard Operatoren (Crossover, Mutation) arbeiten. Da aber das LTP einen sehr eingeschränkten Suchraum besitzt, entstünden dabei viele Individuen, die ungültig sind, die also keine Lösung des Problems darstellen. Insbesondere würde jede 1-Bit-Mutation eines Gen-Strings eine ungültige Lösung liefern.

Beim LTP empfiehlt es sich, die Matrixrepräsentation als Datenstruktur beizubehalten, und genetische Operatoren zu entwerfen, die den Lösungsraum nicht verlassen können.

Die grundlegende Idee, die hinter der Lösung steckt, ist die, eine Funktion zu entwerfen, die zu gegebenen Randbedingungen, die hier aus den Materialvorräten der Quellen bzw. den Materialbedürfnissen der Senken Lösungsmatrizen generiert.

Diese Funktion ist damit geeignet als Initialisierungsoperator der Ausgangspopulation verwendet werden. Um einen Mutationsoperator zu entwerfen, definiert man eine geeignete Form für Teilmatrizen. Diese werden zur Laufzeit zufällig generiert und die Veränderung besteht dann darin, diese Teilmatrizen mittels des Initialisierungsoperators neu berechnen zu lassen. Damit erreicht man eine kleine Änderungen auf den Individuenmatrizen, die ebenfalls wieder gültige Lösungen des Problems darstellen.

Der folgende Operator stellt den Initialisierungsoperator dar, der neue gültige Lösungsmatrizen liefert:

```
procedure initialize
begin
  set all x_{i,j} as 'unvisited'
  while ( i,j exist with x_{i,j}= 'unvisited')
  begin
    choose i,j arbitrarily with x_{i,j} 'unvisited'
    set val= min( source'(i), dest'(j) )
    x_{i,j}= val
    souce'(i)= source'(i) - val
    dest'(j)= dest'(j) - val
  endwhile
end
```

Abbildung A.1: Der Initialisierungsoperator

Initialize wird mit Kopien *source*' und *dest*', die die call-by-value-Namen von *source* und *dest* sind, aufgerufen, da diese von der Funktion geändert werden. Offensichtlich terminiert diese Funktion mit einem gültigen Zustand für M. Allerdings können nicht alle gültigen Lösungen erzeugt werden, sondern nur solche, die höchstens $k+n-1$ Elemente enthalten, die verschieden von Null sind, wobei k die Anzahl der Quellen und n die Zahl der Sourcen darstellt.

Ein Aufruf von initialize mit foldender Initialmatrix *M*,

x	d1	d2	d3	d4
s1	0	0	0	0
s2	0	0	0	0
s3	0	0	0	0

$source = (5, 25, 5)$, $dest = (5, 15, 15, 10)$, könnte beispielsweise die Resultatsmatrix *M'*

x'	d1	d2	d3	d4
s1	0	0	15	0
s2	5	10	0	10
s3	0	0	5	0

liefern, *fit(M')*=700, *source'* $= (0, 0, 0)$, *dest'* $= (0, 0, 0, 0)$, unter Bezugnahme auf das bereits bekannte Beispiel aus Abschnitt 1.2.1

Weiterhin soll noch ein Mutationsoperator definiert werden, der ebenfalls nur gültige Lösungen produziert. Betrachtet man einen Eintrag einer bestimmten Position einer *gültigen* Matrix und verändert ihn um einen Betrag b, so leuchtet ein, daß dieser Betrag in der gleichen Zeile wieder abgezogen werden muß, damit die Nebenbedingung nicht verletzt wird. Eben das gleiche gilt auch für die Spalte. D.h eine bestimmte Eintragsänderung der Position $x_{i,j}$ zieht eine Kaskade von weiteren Änderungen nach sich, und zwar mindestens an den Stellen $x_{i+k,j}$, $x_{i,j+l}$ und $x_{i+k,j+l}$.

Praktischerweise wird man diese Operationen auf eine bereits bekannte zurückführen, nämlich auf *initialize*. In diesem Fall soll *initialize* aber nicht die ganze Matrix verändern, sondern nur einen Teil davon, den wir ihr in Form einer Submatrix übergeben:

Eine Submatrix M^S einer Matrix *M* sei definiert durch folgende Beziehung:

$$M^S = [x_{i,j}]_{i \in I \subset S, j \in J \subset D},$$

d.h M^S besteht aus ausgewählten Zeilen und Spalten der Matrix *M*.

Seien weiterhin

$$source_{M^S}(i) \stackrel{def}{=} \sum_{j \in J} x_{i,j} \text{ für alle } i \text{ und}$$

$$dest_{M^S}(j) \stackrel{def}{=} \sum_{i \in I} x_{i,j} \text{ für alle } j \text{ die}$$

abgewandelten Nebenbedingungen von M^S, so kann man die oben definierte Initialisierungsfunktion dazu verwenden, um M^S so zu verändern, daß sich beim zurückschreiben ihrer Werte $x_{i,j}$ nach M eine zufällig veränderte Lösungsmatrix ergibt, die die Nebenbedingungen (1.2.4) und (1.2.5) erfüllt.

```
procedure mutate
begin
  generate Submatrix M_S
  source(i) := sum(j; x_i,j)
  dest(j) := sum(i; x_i,j)
  initialize(M_S)
  for i=1..k and for j=1..m
    if ( M.x_i,j in M_S )
      M.x_i,j := M_S(x_i,j)^a
    endif
  endfor
end
```

Abbildung A.2: Der Mutationsoperator

a. $M_S(x_{i,j})$ deutet hier an, daß das dem Matrixelement $x_{i,j}$ entsprechende Element der Submatrix M_S in die Hauptmatrix an die ursprüngliche Stelle zurückkopiert wird.

Für die Rekombination benötigt man nun noch ein Verfahren, welches aus zwei Matrizen des Lösungsraumes eine oder mehrere neue generiert, welche aus einer neuen Zusammenstellung von Werten aller Vatermatrizen bestehen und auch wieder gültige Lösungen darstellen. Eine einfache Zerlegung der Matrizen in disjunkte Untermatrizen und deren anschließendes Vertrauschen funktioniert hier wegen der Nebenbedingungen, der Erhaltung der Spalten- und Zeilensummen, nicht, so daß man hier tiefer in die Trickkiste greifen muß.

Die Matrizen $M^{(1)}$ und $M^{(2)}$ seinen im folgenden die Vatermatrizen. Weiterhin seien M_D die Divisionsmatrix von diesen beiden mit zwei und M_R die Matrix der Restklassen *modulo* zwei, definiert durch

$$M_D \stackrel{def}{=} [(x^{(1)}_{i,j} + x^{(2)}_{i,j})/2]_{i \in S, j \in D} \text{ und}$$
$$M_R \stackrel{def}{=} [(x^{(1)}_{i,j} + x^{(2)}_{i,j}) \; mod \; 2]_{i \in S, j \in D}.$$

Damit gilt dann: $2M_D + M_R = M^{(1)} + M^{(2)}$, d.h M_D stellt eine gewisse Zerlegung der Ausgangsmatrizen mit dem Rest M_R dar.

Eine interessante Eigenschaft von M_R, die wir hier nutzen wollen, ist, daß sie sowohl in jeder Zeile als auch in jeder Spalte eine gerade Anzahl von Einsen besitzt. Das liegt daran, daß die Zeilen- und Spaltensummen der Vatermatrizen gleich sind, weil sie ja das gleiche Problem lösen, somit also deren Summe eine *gerade Zahl* ist und damit jeweils eine gerade Anzahl von geraden bzw. ungeraden Summanden hat.

Diese Eigenschaft soll nun von der Rekombinationsoperation genutzt werden, indem man die Restklassenmatrix in zwei sich ergänzende Matrizen aufteilt, die jeweils nur eine Eins pro Zeile und Spalte besitzen. Addiert man nun noch jede von diesen zur Divisionsmatrix M_D hinzu, so erhält man zwei Nachfolgermatrizen, die dank der geschickten Aufteilung beide wieder die Nebenbedingungen erfüllen und und positiv sind, also ebenso gültige Elemente des Lösungsraums sind:

Man generiere also eine Matrix $M^{(1)}_R$, die

- an den Stellen $x_{i,j}$ Nullen enthält, an denen auch M_R Nullen hat
- an zufälligen Stellen Einsen enthält
- in jeder Zeile und Spalte genau halb so viele Einsen hat, wie M_R

Weiterhin setzt man $M^{(2)}_R \stackrel{def}{=} M_R - M^{(1)}_R$; damit sind $M^{(1)}_R$ und $M^{(2)}_R$ regelmäßige Aufteilungen von M_R.

Die Nachkommen der Ausgangsmatrizen kann man nun durch $M^{(3)} \stackrel{def}{=} M_D + M^{(1)}_R$ und $M^{(4)} \stackrel{def}{=} M_D + M^{(2)}_R$ definieren.

Wie man sieht, gilt $M^{(3)} + M^{(4)} = 2M_D + M^{(1)}_R + M^{(2)}_R = 2M_D + M_R = M^{(1)} + M^{(2)}$. Wie man sich leicht überlegt, haben $M^{(3)}$ und $M^{(4)}$ definitionsbedingt die gleichen Spalten- und Zeilensummen wie der Ausgangsmatrizen. Damit stellen sie eine andere Aufteilung der Zahlen der Vatermatrizen dar, liegen im Lösungsraum in der Mitte zwischen diesen und erfüllen die

Nebenbedingungen des linearen Transportproblems. Es ist also ein funktioneller Rekombinationsoperator für dieses Problem gefunden.

```
procedure crossover
begin
  for i=1..k and for j=1..m
    MD.x_{i,j} := (M1.x_{i,j}+ M2.x_{i,j}) div 2
    MR.x_{i,j} := (M1.x_{i,j}+ M2.x_{i,j}) mod 2
  endfor
  calculate MR1
  MR2 := MR- MR1
  M3 := MD+ MR1
  M4 := MD+ MR2
end
```

Abbildung A.3: Der Rekombinationsoperator

Eine ähnliche Vorgehensweise beschreitet man zur Lösung des Nichtlinearen Transportproblems, welches ja eigentlich das interessantere ist, da es sich bisher anderen Vorwärtsstrategien entzieht. Eine detailiertere Beschreibung dessen entzieht sich allerdings dem Rahmen dieses Textes. Für Interessenten sei an dieser Stelle auf [MIC96], Seiten 196ff verwiesen.

B Datenrepräsentation beim TSP

B.1 Das Traveling Salesman Problem

Gegeben sei eine Anzahl von Städten in der reellen Ebene und der Abstand jeweils zweier Städte zueinander. Das Ziel ist, eine Rundreise zu finden, bei der jede Stadt einmal besucht wird, die letzte der ersten gleicht und deren Gesamtlänge minimal ist.

So einfach das Problem zu beschreiben ist, so schwer tun sich damit Algorithmen, die eine größere Anzahl von Städten (wenige tausend reichen da schon) lösen sollen; das Problem hat sich als NP-vollständig erwiesen.

Definition 6.1: Traveling Salesman Problem

Sei $C = [c_{i,j}]_{i,j \in \{1\ldots n\}}$ eine Kostenmatrix mit $c_{i,j} = c_{j,i}$ für alle $i, j \in \{1\ldots n\}$.
Dann nennt sich das Problem eine Permutation π der Menge $\{1\ldots n\}$ zu finden, so daß

$$\sum_{i=1}^{n} c_{\pi(i),\,\pi((i \bmod n)+1)} \qquad \textbf{(6.0.5)}$$

minimal ist, das *Traveling Salesman Problem*.[1]

Anschaulich gesprochen entspricht obige Formel die Summe der Wegstecken der einzelnen Städteverbindungen einer Rundreise der Städte; die Gesamtstrecke soll dabei minimal werden.

Als einfache Repräsentation bietet sich beispielsweise eine Pfaddarstellung an, z.B `a=(123456789)` für eine Tour von 9 Städten. Siehe auch Anhang B zu Details der Repräsentation beim TSP.

1. Genaugenommen unterscheidet man zwischen dem symmetrischen TSP, welches dadurch gekennzeichnet ist, daß die Reisekosten von einer Stadt a nach einer Stadt b denjenigen Kosten der Rückreise gleichen. In der allgemeinen Form dieses Problems wird diese Restriktion fallengelassen (Man stelle sich eine Landschaft vor: a liegt lauf einem Berg, b im Tal)

Als Beispiel seinen die folgenden Städte in der Ebene gegeben, für die eine minimale Tour, wie man so eine Rundreise auch nennt, gefunden werden soll. Im Bild rechts sieht man eine mögliche Tour:

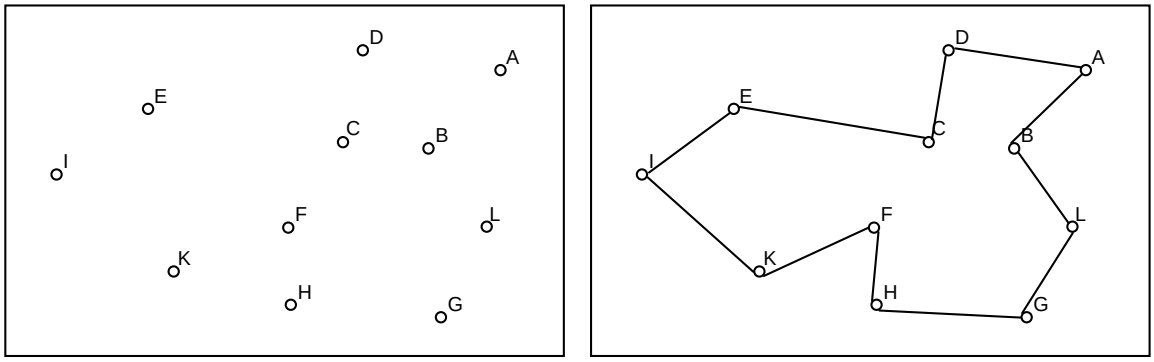


Abbildung B.1: Eine gegebene Menge von Städten und eine mögliche Rundreise.

B.2 Kodierung beim Travelling Salesman Problem (TSP)

Auch beim TSP hat man das Bestreben, die Tourrepräsentation möglichst so zu wählen, daß nur gültige Touren kodiert werden können bzw. die Operatoren so zu implementieren, daß sie einmal nicht aus der Menge der erlaubten Lösungen herausführt und zum anderen trotzdem alle gültigen Lösungen erreicht werden können.

Hier soll eine Auswahl an Repräsentationen und Operatoren vorgestellt werden um die Problematik einer geeigneten Wahl der Datenstrukturen zu veranschaulichen. ([MIC96], Kapitel 10)

Die Pfadrepräsentation

Hierbei wird eine Tour von n Städten durch eine geordnete Liste ihrer Nummern dargestellt (als Permutation). Somit ist jede Permutationsliste eine gültige Lösung des TSPs. Also sind die genetischen Operatoren derart zu implementieren, daß ausschließlich Permutationen der Ursprungsliste generiert werden.

Beispielsweise sind für diese Datenstrukturen folgende Rekombinationsoperatoren vorgeschlagen worden:

PMX (partially mapped crossover):

PMX sucht sich zufällig korrespondierende Teiltouren der Gesamtsequenzen aus, kopiert diese vertauscht zu den Nachkommen, so daß also der Nachkomme von a die Teiltour von b erhält und umgekehrt, und vervollständigt die noch freien Plätze der Nachkommen derart mit den restlichen Städten der Ursprungstouren, daß erstere wieder gültige Permutation im Sinne des TSP darstellen.

Bspw. seien `a=(123456789)` und `b=(452187693)` Touren, und seien die Subtouren mit dem Zeichen '`|`' markiert. Dann könnte PMX folgende Schritte durchführen Das `X` steht hier für den Crossover-Operator):

`a X b` = `(123|4567|89) X (452|1876|93)` erzeugt

`1 (...|1876|..) X (...|4567|..)`

`2 (4..|1876|5.) X (18.|4567|..)`

`3 (423|1876|59) X (182|4567|93)`

Im Schritt 1 werden die Teiltouren `|1876|` und `|4567|` vertauscht. Im nächsten Schritt werden die Ziffern der direkten Eltern der Reihe nach geprüft, ob sie in den Nachkommen übernommen werden können. Für den ersten Nachkommen im Schritt 2 kann die Stadt '1' nicht übernommen werden, da sie schon in der Subtour aufgetreten ist. Deshalb wird die Stadt '4' von b übernommen. Das gleiche gilt für die Städte '1' und '8' des zweiten Nachkommens. Schließlich werden konfliktfreie Städte von den jeweiligen korrespondierenden Ursprungstouren übernommen.

OX (order crossover)

Der OX sucht sich ebenfalls Teiltouren aus den beiden Lösungen a und b. Allein diese kopiert er unverändert in die Nachkommen und fügt die Tour des jeweiligen anderen Elter in ordnungserhaltender Reihenfolge nach der TEiltour beginnend ein, und zwar eben jene Städte, die noch nicht vorkommen, so daß auch jeweils wieder eine Permutation der Städte entsteht. Von den obigen a und b ausgehend hier wieder ein Beispiel:

`1 (...|4567|..)` X `(...|1876|..)`

`2 (218|4567|93)` X `(345|1876|92)`

Weiterhin sind folgende Mutationsoperatoren denkbar:

insertion nimmt eine beliebige Stadt aus der Tour heraus und fügt sie an einer zufälligen Position ein.

displacement nimmt eine beliebige Subtour aus der Tour heraus und fügt sie an einer zufälligen Position ein.

reciprocal exchange

sucht sich zufällig zwei Städte der Tour und verstauscht ihre Positionen

subtour exchange

sucht zufälligerweise zwei disjunkte Teiltouren der Tour und vertauscht ihre Positionen.

Die Matrixrepräsentation

Bei der MR wird eine Tour in einer (n x n)-Matrix gespeichert, die sich folgendermaßen aufbaut:

Das Element $x_{i,j}$ der Matrix enthält genau dann eine ’1’ falls die Stadt i vor der Stadt j in der Tour vorkommt. Ansonsten enthält die Matrix Nullen.

Für M gelten folgende Eigenschaften:

i) $|M|_{\#1} = n(n-1)/2$ [1] **(6.0.6)**

ii) Für die Diagonalelemente $x_{i,i}$ von M gilt:
$x_{i,i} = 0$ für alle $i \leq n$ **(6.0.7)**

iii) Es gilt die Transitivität: $x_{i,j} = 1$ und $x_{j,k} = 1$
impliziert $x_{i,k} = 1$ für alle $i, j, k \leq n$ **(6.0.8)**

Beispiel der Repräsentation der Tour `c=(312874695)` in der Matrixdarstellung:[2]

x	1	2	3	4	5	6	7	8	9
1		1		1	1	1	1	1	1
2				1	1	1	1	1	1
3	1	1		1	1	1	1	1	1
4					1	1			1
5									
6					1				1
7				1	1	1			1
8				1	1	1	1		1
9					1				

Eine sehr interessante Eigenschaft haben Matrizen, für die

$$|M|_{\#1} < n(n-1)/2 \quad \textbf{(6.0.9)}$$

und zusätzlich die Eigenschaften (6.0.7) und (6.0.8) gelten:

Sie sind partiell geordnet und stellen somit Teiltouren der Gesamttour dar.Damit kann man sie durch entsprechendes Auffüllen mit Einsen zu einer gültigen Tour erweitern. Auf diesem Prinzip bauen dann auch die für diese Repräsentation ersonnenen Rekombinationsoperatoren auf:

Intersection Sei M^S die Schnittmatrix zweier Matrizen M^1 und M^2, definiert durch: $x^S_{i,j} = 1$ genau dann, wenn $x^1_{i,j} = 1$ und $x^2_{i,j} = 1$ für alle $i, j \leq n$.

wie man sieht, hat M^S damit die Eigenschaften (6.0.7), (6.0.8) und (6.0.9), ist also zur vollständigen Tour erweiterbar. Im ersten Schritt wird also bei intersection die Schnittmatrix gebildet, im zweiten wird diese zur vollständigen Tour erweitert.

union Der *union* Operator basiert auf der Tatsache, daß eine Teilmenge der einen Matrix widerspruchdsfrei mit einer Teilmenge der anderen kom-

1. wobei $|M|_{\#1}$ die Gesamtzahl der Einsen der Matrix darstellt.
2. Die Nullen sind der Übersichtlichkeit halber in der Matrix ausgelassen.

biniert werden kann, vorausgesetzt diese Teilmengen haben einen leeren Schnitt. Also wird im ersten Schritt die Menge der Städte in zwei disjunkte Mengen geteilt (die dann Untermatrizen mit obigen genannten Eigenschaften darstellen) und diese zur Nachfolgermatrix kombiniert, welche dann zur gültigen Tourmatrix durch Auffullen mit Einsen erreicht wird durch Analyse der Zeilen- und Spaltensummen. Beispielsweise ist folgende Matrix aus der *union* von der Beispielmatrix (*) und der Matrix der Tour `c=(123456789)` hervorgegangen:

x	1	2	3	4	5	6	7	8	9
1		1	1	1	.	.	.	.	.
2			1	1	.	.	.	.	.
3				1	.	.	.	.	.
4					.	.	.	.	.
5	.	.	.	.					
6	.	.	.	.	1				1
7	.	.	.	.	1	1			1
8	.	.	.	.	1	1	1		1
9	.	.	.	.	1				

(die Werte der mit '.' gekennzeichneten Stellen müssen noch berechnet werden, d.h. der zweite Schritt ist noch nicht durchgeführt)

Trotz der aufwendigeren Datenstrukturmodellierung der Matrixdarstellung hat diese eindeutige Vorzüge: die Rekombinationsoperatoren sind wesentlich vereinfacht gegenüber denen der Pfaddarstellung und ermöglichen damit eine wesentlich schnellere Abarbeitung des Algorithmus. Die Matrix ermöglicht einem nicht nur Zugriff auf die Position einer Stadt in der Folge (Spaltensumme), sondern liefert auch noch alle Vorgängerstädte bzw. Nachfolgerstädte. Damit enthält dieses Modell einen höheren Informationsgehalt über die Folge als das Pfadmodell:

> *The boolean matrix representation of a sequence encapsulates all of the information about the sequnce, including both the microtopology of individual city-to-city connections and the macro-topology of predecessors and successors. The boolean matrix representation can be used to understand existing operators and to develop new operators that can be applied to sequences to produce desired effects while preserving the necessary properties of the sequence.*
>
> ([RAW91], Seiten 284ff)

Wie aus den letzten Abschnitten ersichtlich wurde, ist die problemspezifische Datenstruktur ein wichtiger Punkt bei der Entwicklung effizienter Optimierungsstrategien zu einem spezifischen Problem. Entscheidend ist, Operatoren zu finden, die den Lösungsraum nicht verlassen können und somit die Implementierung von Korrekturoperatoren vermeiden, sofern dies möglich ist. Übernimmt man die problemspezifischen Strukturen als interne Datenstruktur, so kann man auf die Implementierung von Umkodierungsoperatoren verzichten - allerdings ist es meist besser, die Repräsentation im Hinblick auf einfache genetische Operatoren hin konstruieren und den Aufwand nicht scheuen, Umkodierungsfunktionen zu implementieren. Dies ist meist die bessere Alternative im Hinblick auf eine optimal angepaßte Evolutionsstrategie.

C Mathematische Grundlagen

An dieser Stelle soll nur ein kleiner Überblick über die mathematischen Grundlagen gegeben werden, die zum Verständnis der im Hauptteil vorkommenden Beweise erforderlich sind. Für ein genaueres Verständnis der Materie sei auf [BAU74] verwiesen. Dort lassen sich auch die die Beweise der angegebenen Sätze nachlesen.

C.1 Maßtheorie

Definition: σ-Algebra[1]

Ein System von Teilmengen $\boldsymbol{M}$ einer Menge Ω heißt σ-Algebra, falls gilt:

i) $\Omega \in \boldsymbol{M}$

ii) $A \in \boldsymbol{M} \Rightarrow C(A) \in \boldsymbol{M}$

iii) Für jede Folge $(A_n)_{n \in \mathbb{N}} \in \boldsymbol{M}$ gilt $\bigcup_{n=0}^{\infty} A_n \in \boldsymbol{M}$

Definition: Maß

Sei $\boldsymbol{M}$ eine σ-Algebra auf Ω. Eine Funktion $\mu : \boldsymbol{M} \to \mathbb{R}$ heißt Maß, falls gilt:

i) $\mu(\varnothing) = 0$

ii) Für alle $A \in \boldsymbol{M}$: $\mu(A) \geq 0$

iii) $\mu\left(\bigcup_{n=0}^{\infty} A_n\right) = \sum_{n=0}^{\infty} \mu(A_n)$, $(A_n)_{n \in \mathbb{N}}$ paarweise disjunkt in $\boldsymbol{M}$

1. $C(A)$ sei das Komplement der Menge A, d.h. $C(A) \stackrel{def}{=} \Omega - A$.

Eigenschaften von Maßen:

i) $\mu(A \cup B) + \mu(A \cap B) = \mu(A) + \mu(B)$, für alle $A, B \in \mathbb{M}$

ii) $A \subset B \Rightarrow \mu(A) \leq \mu(B)$, für alle $A, B \in \mathbb{M}$

iii) $\mu\left(\bigcup_{n=0}^{\infty} A_n\right) \leq \sum_{n=0}^{\infty} \mu(A_n)$ „für alle $(A_n)_{n \in \mathbb{N}}$ in $\mathbb{M}$

Beispiel:

Sei $\mathbb{M}$ eine σ-Algebra auf Ω. Dann ist ε_ω, definiert durch

$$\varepsilon_\omega(A) \overset{def}{=} \begin{cases} 1 & \omega \in A \\ 0 & \omega \notin A \end{cases}, A \in \mathbb{M},$$

ein Maß. Man nennt es die *Einheitsmasse* von ω auf $\mathbb{M}$.

Definition: Borelmengen

I^p sei das System der halboffenen Intervalle $[a, b[$; $a, b \in \mathbb{R}^p$, im Grundraum $\Omega = \mathbb{R}^p$.

Die Elemente der vom System I^p im $\mathbb{R}^p$ erzeugten σ-Algebren heißen die *Borelschen Mengen des* $\mathbb{R}^p$. Dementsprechend heißt $\mathbb{M}(I^p)$ die σ-Algebra der Borelschen Mengen des $\mathbb{R}^p$; sie wird mit $\mathbb{B}^p$ bezeichnet.

Definition: Lebesgue-Maß

Es gibt genau ein Maß λ^p auf $\mathbb{B}^p$, welches jedem rechts halboffenen Intervall $[a, b[$ im $\mathbb{R}^p$ seinen p-dimensionalen Elementarinhalt $(b_1 - a_1)(b_2 - a_2)\ldots(b_p - a_p)$ zuordnet. Man nennt λ^p das Lebesgue-Maß auf $\mathbb{R}^p$.

Im folgenden bezeichne O^p bzw. P^p bzw. C^p das System der offenen bei. abgeschlossenen bzw. kompakten Teilmengen im $\mathbb{R}^p$.

Einige Eigenschaften des Lebesgue-Maß:

i) $\mathbb{B}^p = O^p = P^p = C^p$

ii) Jede Hyperebene H des Raumes $\mathbb{R}^p$ ist eine λ^p-Nullmenge, d.h. $\lambda^p(H) = 0$

iii) Jede abzählbare Teilmenge des $\mathbb{R}^p$ ist λ^p-Nullmenge

Definition: Maßraum

Ein Tripel $(\Omega, \mathcal{M}, \mu)$, bestehend aus Grundmenge Ω, einer σ-Algebra $\mathcal{M}$ und einem Maß μ nennt man Maßraum.

Insbesondere ist $(\mathbb{R}^p, \mathcal{B}^p, \lambda^p)$ der Borelsche Maßraum.

Definition: meßbare Abbildung

Seinen $(\Omega, \mathcal{M}, \mu)$ und $(\Omega', \mathcal{M}', \mu')$ zwei Maßräume und sei $T: \Omega \to \Omega'$ eine Abbildung. Dann heißt T $(\mathcal{M} - \mathcal{M}')$-meßbar, wenn gilt:

$T^{-1}(A') \in \mathcal{M}$ für alle $A' \in \mathcal{M}'$

Definition: Definition Bildmaß

Sei $T: \Omega \to \Omega'$ meßbare Abbildung. Dann wird für jedes Maß μ auf $\mathcal{M}$ durch $A' \to \mu(T^{-1}(A'))$ ein Maß μ' auf $\mathcal{M}'$ definiert.

$\mu' = T(\mu)$ nennt man Bildmaß bezüglich der Abbildung T.

C.2 Integrationstheorie

In diesem Abschnitt soll der Schwerpunkt auf pellen Funktionen beruhen, d.h. auf Abbildungen der Form $f: \Omega \to \mathbb{R}$. In diesem Falle soll der Begriff der reellwertigen Funktion noch erweitert werden, indem man sogenannte *numerische Funktionen* $f: \Omega \to \overline{\mathbb{R}}$ betrachtet, deren Bildraum die kompaktifizierte Menge der reellen Zahlen $\overline{\mathbb{R}} \stackrel{def}{=} \mathbb{R} \cup \{-\infty, +\infty\}$ ist.

Es gilt:

- Eine numerische Funktion $f: \Omega \to \overline{\mathbb{R}}$ ist genau dann $\mathcal{M}$-meßbar, falls gilt: $\{\omega \in \Omega \mid f(\omega) \geq \alpha\} \in \mathcal{M}$ für alle $\alpha \in \mathbb{R}$

- Für je zwei $\mathcal{M}$-meßbare Funktionen $f, g: \Omega \to \overline{\mathbb{R}}$ liegen die Mengen $\{\omega \in \Omega \mid f(\omega) \geq g(\omega)\}$, $\{\omega \in \Omega \mid f(\omega) > g(\omega)\}$, $\{\omega \in \Omega \mid f(\omega) = g(\omega)\}$ und $\{\omega \in \Omega \mid f(\omega) \neq g(\omega)\}$ selbst wieder in $\mathcal{M}$.

Definition: Elementarfunktion

Eine reelle Funktion u auf Ω heißt ($\mathcal{M}$-)Elementarfunktion, wenn sie $\mathcal{M}$-meßbar, nicht negativ ist und nur endlich viele Werte annimmt. $E = E(\mathcal{M})$ bezeiche die Menge aller Elementarfunktionen über $\mathcal{M}$.

Alle $u \in E$ haben die Darstellung $u = \sum_{i=0}^{n} \alpha_i \mathbf{1}_{A_i}$, wobei $\{\alpha_i\}$ die endliche Wertemenge darstellt.

Dabei sei $\mathbf{1}_A$ definiert durch $\mathbf{1}_A(x) \stackrel{def}{=} \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}, A \in \mathcal{M}$

Sind $u, v \in E$, so gilt $(u + v) \in E$, $uv \in E$, $sup(u + v) \in E$ und $inf(u + v) \in E$, das heißt, E ist abgeschlossen bezüglich der Operationen $\{ +, \bullet, sup, inf \}$.

Definition: Integral auf E

Für eine Elementarfunktion $u = \sum_{i=0}^{n} \alpha_i \mathbf{1}_{A_i} \in E$ ist das Integral definiert als:

$$\int u \; \mathbf{d}\mu \stackrel{def}{=} \sum_{i=0}^{n} \alpha_i \, \mu(A_i)$$

Weitere Eigenschaften des Integrals für Elementarfunktionen:

i) $\int \mathbf{1}_A \; \mathbf{d}\mu = \mu(A), \forall A \in \mathcal{M}$

ii) $\int \alpha u \; \mathbf{d}\mu = \alpha \int u \; \mathbf{d}\mu, u \in E, \alpha \in \mathbb{R}_+$

iii) $\int u + v \; \mathbf{d}\mu = \int u \; \mathbf{d}\mu + \int v \; \mathbf{d}\mu, u, v \in E$

iv) $u \le v \Rightarrow \int u \; \mathbf{d}\mu \le \int v \; \mathbf{d}\mu$

Definition: E^*

E^* sei die Menge aller numerischen Funktionen $f \ge 0$, $f: \Omega \to \overline{\mathbb{R}}$, zu welchen eine monoton steigende Folge $(u_n)_{n \in \mathbb{N}}$ von Elementarfunktionen aus E existiert mit $f = sup(u_n)$.

Ihr Integral ist dann definiert als $\int f \; \mathbf{d}\mu \stackrel{def}{=} sup_{n \in \mathbb{N}} \int u_n \; \mathbf{d}\mu$

E^* ist die Menge der $\mathcal{M}$-meßbaren, positiven ($f \ge 0$), numerischen Funktionen auf Ω.

Man kann zeigen, daß folgende Eigenschaften der Elementarfunktionen auf die Funktionsmenge E^* erweitert werden können:

i) Sind $f, g \in E^*$, so gilt $(f + g) \in E^*$, $fg \in E^*$, $sup(f + g) \in E^*$ und $inf(f + g) \in E^*$; E^* ist ebenfalls abgeschlossen bezüglich $\{ +, \bullet, sup, inf \}$

ii) $\int \alpha f \; \mathbf{d}\mu = \alpha \int f \; \mathbf{d}\mu, \; f \in E^*, \alpha \in \mathbb{R}_+$

iii) $\int f+g \ \mathbf{d}\mu = \int f \ \mathbf{d}\mu + \int g \ \mathbf{d}\mu \ , \ f, g \in E^*$

iv) $f \leq g \ \Rightarrow \ \int f \ \mathbf{d}\mu \leq \int g \ \mathbf{d}\mu \ , \ f, g \in E^*$

Weiterhin gilt der elementare Satz von Beppo-Levi, der zeigt, daß für monoton steigende oder fallende Funktionenfolgen aus E^* Supremum und Integral vertauscht werden können:

Satz von Beppo-Levi

Für alle isotonen Folgen von Funktionen $(f_n)_{n \in \mathbb{N}}$, alle $f_n \in E^*$, ist
$sup_{n \in \mathbb{N}} \ (f_n) \in E^*$ und es gilt $\int sup_{n \in \mathbb{N}} (f_n) \ \mathbf{d}\mu = sup_{n \in \mathbb{N}} \ (\int f_n \ \mathbf{d}\mu)$

Bisher wurde das Integral nur für *positive* Funktionen definiert; diese Beschränkung soll nun noch entfernt werden.

Dazu noch einige Definitionen:

Für eine beliebige numerische Funktion $f : \ \Omega \to \overline{\mathbb{R}}$ auf sei
$f^+ \stackrel{def}{=} \ sup(f, 0) ; \ f^- \stackrel{def}{=} \ (-f)^+ = - inf(f, 0) .$
f^+ nennt man den *Positivteil*, f^- den *Negativteil* von f. Es gilt : $f = f^+ - f^-$

Definition: Integral auf E^*

Sei $f : \ \Omega \to \overline{\mathbb{R}}$ numerische Funktion. f ist *integrierbar*, falls f $\mathcal{M}$-meßbar ist und es gilt
$\int f^+ \ \mathbf{d}\mu < \infty$ und $\int f^- \ \mathbf{d}\mu < \infty$.
Dann ist das Integral von f definiert als: $\int f \ \mathbf{d}\mu \stackrel{def}{=} \ \int f^+ \ \mathbf{d}\mu \ - \int f^- \ \mathbf{d}\mu$

Jede einzelne der folgenden Bedingungen ist hinreichend für die Integrierbarkeit einer numerischen Funktion $f : \ \Omega \to \overline{\mathbb{R}}$:

i) f^+ und f^- sind integrierbar

ii) Es existieren integrierbare numerische Funktionen
$g, h \in E^*$ mit $f = g - h$

iii) $|f|$ ist integrierbar

Seien f und g integrierbare numerische Funktionen auf Ω, $\alpha \in \mathbb{R}$. Dann sind

i) $\int \alpha f \ \mathbf{d}\mu = \alpha \int f \ \mathbf{d}\mu \ ,$•

ii) $\int f+g \ \mathbf{d}\mu = \int f \ \mathbf{d}\mu + \int g \ \mathbf{d}\mu \ ,$

iii) $sup(f, g)$

iv) $inf(f, g)$

ebenfalls integrierbare Funktionen auf Ω.

Ferner gilt: $f \le g \Rightarrow \int f \, \mathbf{d}\mu \le \int g \, \mathbf{d}\mu$ und $\left|\int f \, \mathbf{d}\mu\right| \le \int |f| \, \mathbf{d}\mu$.

Die Menge aller reellen, μ-integrierbaren Funktionen über Ω bezeichnet man mit $L^1(\mu)$.

Für Teilmengen $A \subset \Omega$ definiert man $\int_A f \, \mathbf{d}\mu \stackrel{def}{=} \int (f \mathbf{1}_A) \, \mathbf{d}\mu$. Insbesondere gilt:

$$\int_\Omega f \, \mathbf{d}\mu = \int (f \mathbf{1}_\Omega) \, \mathbf{d}\mu = \int f \, \mathbf{d}\mu$$

und

$$\int_{\{p \in \Omega\}} f \, \mathbf{d}\mu = \int (f \mathbf{1}_{\{p\}}) \, \mathbf{d}\mu = f(p)\mu\{p\} = 0;$$

letzteres ergibt sich aus der Eigenschaft, daß $f \mathbf{1}_{\{p\}} \in E$ ist.

Satz (Transformationssatz)

Seinen $(\Omega, \mathcal{M}, \mu)$ Maßraum, $(\Omega', \mathcal{M}')$ Maßraum, $T: \Omega \to \Omega$ $(\mathcal{M} - \mathcal{M}')$-meßbare Abbildung, $T: \Omega \to \Omega'$ und $\mu' = T(\mu)$.

Dann gilt für jede $\mathcal{M}'$-meßbare numerische Funktion $f' \ge 0$ auf Ω':

$$\int_{\Omega'} f' \, \mathrm{d}T(\mu) = \int_\Omega f'(T) \, \mathrm{d}\mu$$

Satz

Sei $(\Omega, \mathcal{M}, \mu)$ Maßraum, $T: \Omega \to \Omega$ bijektive Abbildung von Ω in sich selbst und T, T^{-1} seinen $(\mathcal{M} - \mathcal{M})$-meßbar. Dann gilt für alle $f \in E^*$:

$$T(f\mu) = f(T^{-1})T(\mu)$$

Definition: Lebesgue-Integral

Für die Integrale Borel-meßbarer numerischer Funktionen f auf $B \in \mathbb{B}^p$ definiert man das *Lebesgue-Integral* als

$$\int_B f(x)dx \stackrel{def}{=} \int f \, d\lambda_B^p = \int_{\mathbb{R}^p} \mathbf{1}_B f \, d\lambda^p$$

Für f meßbare numerische Funktion über Ω gilt: $|f|^p$ meßbar für alle $p \in \mathbb{N}^+$.

Damit definiert man

Definition: p-fache Integrierbarkeit

Sei $f: \Omega \to \overline{\mathbb{R}}$ meßbare numerische Funktion auf Ω, $p \in \mathbb{N}^+$. f heißt *p-fach integrierbar,* falls $|f|^p$ μ-integrierbar ist.

$L^p(\mu)$ ist dann der Raum der p-fach integrierbaren Funktionen.

Das Lebesgue-Integral und das Riemannsche Integral

Satz

Sei f eine auf einem kompakten Intervall $\Omega = [\alpha, \beta] \subset \mathbb{R}$ definierte, Borel-meßbare reelle Funktion. Falls f Riemann-integrierbar ist, so ist sie auch Lebesgue-integrierbar und beide Integrale sind gleich.

Satz

Sei $f \geq 0$ reelle, Borel-meßbare, auf $\mathbb{R}$ definierte Funktion, die auf jedem kompakten Intervall $I = [\alpha, \beta] \subset \mathbb{R}$ Riemann-integrierbar ist.

f ist genau dann Lebesgue-integrierbar, wenn ihr uneigentliches Riemann-Integral existiert.

1. Beispiel[1] $f \in R^1$, $f \notin L^1$

   Folgende Funktion *f(x)* ist zwar Riemann-integrierbar, aber nicht Lebesgue-integrierbar

$$f(x) = \lim_{a \to \infty} \int_{[0,\, a]} \frac{sin(x)}{x} \mathrm{d}x$$

   ([BAU74], Seite 85).

2. Beispiel 2: $f \in L^1$ $f \notin R^1$

   Sei

1. R^1 sei die Menge der einfach Riemann-integrierbaren Funktionen

$$f(x) \stackrel{def}{=} \begin{cases} 1, & \text{falls } (x \in \mathbb{Q} \cap [0,1]) \\ 0, & \text{falls } \overline{(x \in \mathbb{Q} \cap [0,1])} \end{cases}$$

das heißt, $f(x) = \mathbf{1}_{\mathbb{Q} \cap [0,1]}$.

Da $\mathbb{Q} \cap [0,1]$ abzählbar ist, ist $\mathbb{Q} \cap [0,1] \in \mathcal{B}^1$, also ist f integrierbar.

Wegen $\lambda^1_{[0,1]}(\mathbb{Q} \cap [0,1]) = 0$ ist $\int\limits_{(\mathbb{Q} \cap [0,1])} f(x) d\lambda^1 = 0$,

aber f nicht Riemann-integierbar!

C.3 Wahrscheinlichkeitstheorie

Definition: Wahrscheinlichkeitsmaß

Sei P ein Maß mit $P: \mathcal{M} \subset \Omega \to [0,1]$, $P(\Omega) = 1$.

Dann heißt P Wahrscheinlichkeitsmaß, $(\Omega, \mathcal{M}, P)$ Wahrscheinlichkeitsraum.

Definition: Zufallsvariable

Seinen $(\Omega, \mathcal{M}, P)$ Wahrscheinlichkeitsraum, $(\Omega', \mathcal{M}')$ Maßraum.

Dann heißt jede $(\mathcal{M} - \mathcal{M}')$-meßbare Abbildung $X: \Omega \to \Omega'$ Zufallsvariable auf $(\Omega', \mathcal{M}')$.

Ist $\Omega' = \mathbb{R}$ und $\mathcal{M}' = \mathcal{B}^1$ bzw. $\Omega' = \overline{\mathbb{R}}$ und $\mathcal{M}' = \overline{\mathcal{B}}^1$ so sprechen wir von reellen bzw. von numerischen Zufallsvariablen.

Für jedes $A \in \mathcal{M}$ ist die Indikatorfunktion $\mathbf{1}_A$ reelle Zufallsvariable; man nennt sie auch *Indikatorvariable* von *A*.

Bringt man zum Ausdruck, daß Ein Ereignis X in $A' \in \mathcal{M}'$ liegt, so schreibt man dies als $\{X \in A'\} \stackrel{def}{=} X^{-1}(A')$. Die Wahrscheinlichkeit dieses Ereignisses ist demnach $P\{X \in A'\} \stackrel{def}{=} P(X^{-1}(A'))$.

Definition: Verteilung

Sei X eine $(\Omega', \mathcal{M}')$-Zufallsvariable auf dem Wahrscheinlichkeitsraum $(\Omega, \mathcal{M}, P)$, $X: \Omega \to \Omega'$. Dann heißt das Bildmaß

$$P' \stackrel{def}{=} X(P)$$

die *Verteilung* von *X* bzgl. *P* und es gilt: $P_X(A') = P\{X \in A'\}$ für alle $A' \in \mathcal{M}'$

Definition: Erwartungswert

Sei X eine $(\Omega', \mathcal{M}')$-Zufallsvariable auf dem Wahrscheinlichkeitsraum $(\Omega, \mathcal{M}, P)$, $X: \Omega \to \Omega'$. Ist weiterhin $X \geq 0$ oder X P-integrierbar, so heißt

$$E(X) = E(X) \stackrel{def}{=} \int X \, dP$$

der *Erwartungswert* von *P.*

Nach dem Transformationssatz gilt für alle reelle Borel-meßbare f auf $\mathbb{R}$, $f \geq 0$ oder f P-integrierbar:

$$E(f(X)) = \int f \, dP_X, \text{ und damit folgt } E(X) = \int x P(X) \, dx.$$

Für Zufallsvariable X und Y, die wie oben definiert sind, sowie reeller Konstanten c, d gelten weiterhin folgende Eingenschaften:

i) $E(cX + dY) = cE(X) + dE(Y)$

ii) $P\{X \leq Y\} = 1 \rightarrow E(X) \leq E(Y)$

iii) $|E(X)| \leq E(|X|)$

iv) $P\{|X| \leq c\} = 1 \rightarrow |E(X)| \leq c$

Definition: Varianz und Standardabweichung

Für jede reelle Zufallsvariable X heißt

$$V(X) = \sigma^2(X) \stackrel{def}{=} E((X - E(X))^2)$$

Varianz von X, $\sigma(X)$ heißt Standardabweichung von X.

Satz

Ist eine reelle Zufallsvariable $(\Omega', \mathcal{M}')$-Zufallsvariable X auf dem Wahrscheinlichkeitsraum $(\Omega, \mathcal{M}, P)$, $X: \Omega \to \Omega'$, quadratisch integrierbar, dann gilt

$$V(X) = E(X^2) - E^2(X) = \int x^2 P_X \, dx - \left(\int x P_X \, dx\right)^2$$

Weitere Eigenschaften der Varianz:

i) $V(aX + b) = a^2 V(X)$

ii) $V(\sum X_i) = \sum V(X_i)$

für eine Folge von Zufallsvariablen $(X_i)_{i \in \mathbb{N}}$ und reelle Konstanten a und b.

Definition: Verteilungsfunktion

Eine reelle Funktion F auf $\mathbb{R}$ ist eine Verteilungsfunktion eines Wahrscheinlichkeitsmaßes P auf $\mathcal{B}^1$, falls alle folgenden Punkte erfüllt sind:

i) F ist isoton

ii) F ist linksseitig stetig

iii) $\lim_{x \to -\infty} F(x) = 0$

iv) $\lim_{x \to +\infty} F(x) = 1$

F ist eindeutig durch P bestimmt.

Wichtige Verteilungen

1. Für jede Punktfolge $(x_n)_{n \in \mathbb{N}}$ im $\mathbb{R}^k$ und jede Folge $(\alpha_n)_{n \in \mathbb{N}}$ reeller Zahlen in $\mathbb{R}$ ist

$$\mu = \sum_{n \in \mathbb{N}} \alpha_n \varepsilon_{x_n}$$

ein Wahrscheinlichkeitsmaß auf $\mathcal{B}^k$. Dies ist die allgemeine Darstellung einer *diskreten Verteilung*.

2. $\pi_\alpha = \sum_{k \in \mathbb{N}} \left(e^{-\alpha} \frac{\alpha^k}{k!} \varepsilon_k \right)$, $\alpha \in \mathbb{R}^+$, ε_k Einheitsmasse in $\mathcal{B}^1$, ist ein Wahrscheinlichkeitsmaß auf $\mathcal{B}^1$. Man nennt es auch die *Poissonverteilung* mit Parameter α.

Eigenschaften einer nach π_α verteilten, reellen Zufallsvariablen X:

i) $E(X) = \sum_{k \in \mathbb{N}} \left(e^{-\alpha} \frac{\alpha^k}{k!} k \right) = \alpha$

ii) $E(X^2) = \sum_{k \in \mathbb{N}} \left(e^{-\alpha} \frac{\alpha^k}{k!} k^2 \right) = \alpha^2 + \alpha$

iii) $V(X) = \alpha$

3. Für ein $\kappa \in \mathbb{R}^+$ ist $e_\kappa = \int_{\mathbb{R}^+} \kappa e^{-\kappa x} \mathrm{d}x$ die *Exponentialverteilung* mit Parameter κ.

4. $N_{\alpha,\sigma^2}(x) = \int_{-\infty}^{x} (2\pi\sigma)^{-\frac{1}{2}} \exp\left(-\frac{(x-\alpha)^2}{2\sigma^2}\right) dx$ heißt die *Normalverteilung* mit Erwartungswert α und Varianz σ^2, oder auch *Gaußverteilung*.

Eigenschaften einer nach N_{α,σ^2} verteilten, reellen Zufallsvariablen X:

i) $E(X) = \int x N_{\alpha,\sigma^2}(x)\, dx = \alpha$

ii) $E(X^2) = \int x^2 N_{\alpha,\sigma^2}(x)\, dx = \sigma^2 + \alpha^2$

iii) $V(X) = \sigma^2$

D Übersicht der Klassenstruktur

Folgendes Diagramm beschreibt die Programmstruktur des implementierten Verfahrens. Dabei sind für das Verfahren unnötige Basisklassen bewußt nicht aufgeführt.

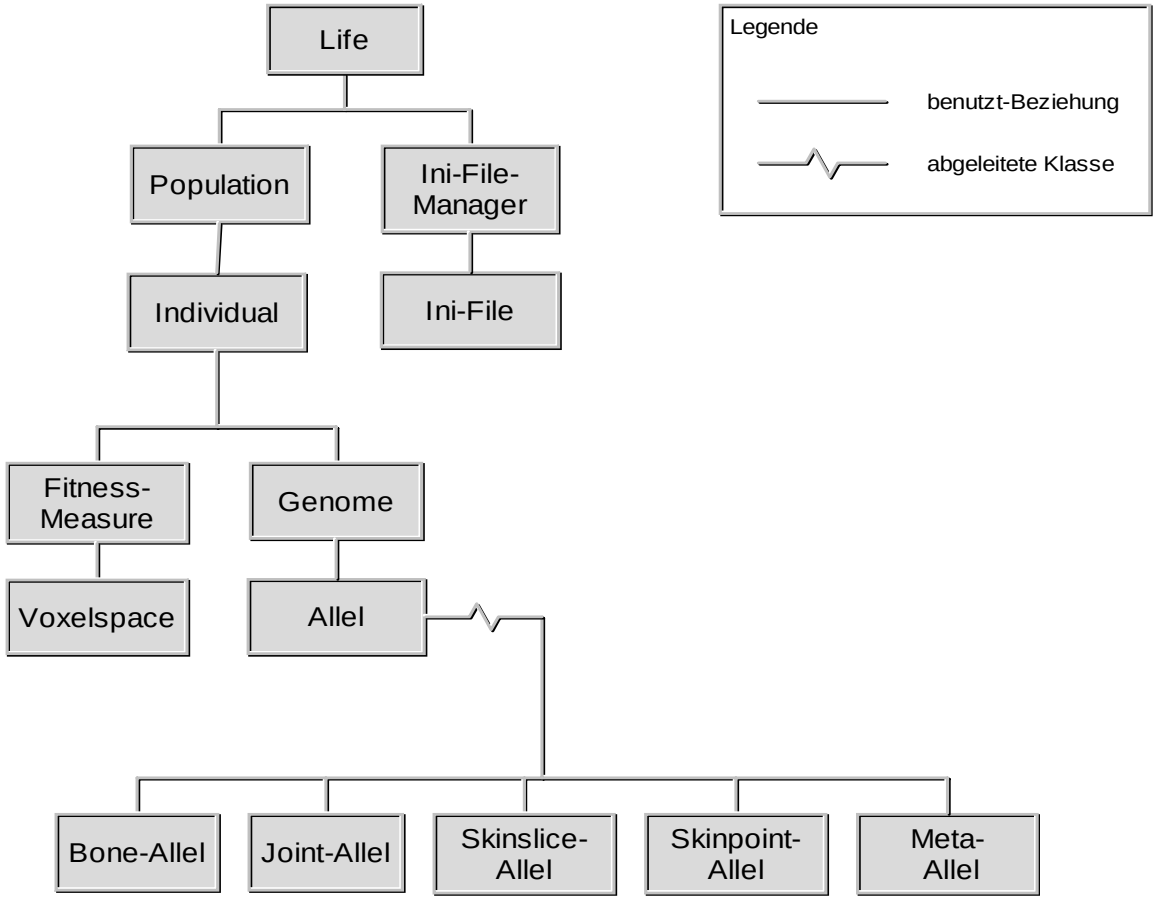


Abbildung D.1: Klassenstruktur des LIFE-Algorithmus

E Listing des LIFE-Headers

```
#ifndef __CesLife__
#define __CesLife__

#include"StandardES.h"
#include "StandardHead.h"

class CVoxelSpace;
class CRamsisMaster;

class CesIndividual;
class CesPopulation;
class CesMeasurement;
class CesGenome;
class CesAllel;
class CesProb;
class CesAllelPool;
class CIniFile;
class CesIniManager;
class CesLife
{
protected:

   CRamsisMaster* mRamsisMaster;
   CRamsisMaster* mConstraintRamsisMaster;
   CesIniManager* mIniManager;
// CIniFile*         mIniFile;
   CesProb*          mProb;
   CesAllelPool*  mAllelPool;
   // ------------------------------------------------------------------------
         CesMeasurement*mMeasurementHandler;
   // ------------------------------------------------------------------------
   CesPopulation* mRamsisPopulation[2];
   tCARDINALmParents;
   tCARDINALmOffspring;
   tBOOLEANmAlreadyInitialized;
   tBabbleMode mBabbleMode;
   tLONGCARDmStepNumber;// Nummer des aktuellen ES-Schritts
   tLONGCARDmActualIndex;
   tLONGCARDmCounter;// Zähler für Schleifenabbruch
   tFitnessmActualFitness;
   tBOOLEANmCalcFinished;// flag für: Berechnung beendet
   tREAL mAccuracy;
   tREAL mInitialFitness;// Fitnesswert der Startpopulation.
```

```
public:
   CesLife();
   virtual ~CesLife();

protected:
   tBOOLEAN InitializeStepPop();
   tBOOLEAN BuildGenome( CesGenome*** a_Genome,
                         tStrategy& strategyType, tLONGCARD a_stepNumber);
public:
   tBOOLEAN Initialize( CRamsisMaster* an_externalRamsisMaster );
            // Initialisiert die Evolution mit dem initialen Ramsis
            // an_enternalRamsisMaster

   tBOOLEAN InitializeInternalRamsis(CRamsisMaster* a_RamsisMaster);
            // Funtion, die den extern übergebenen Ramsis KOPIERT
            // und dem internen zuweist.

   tBOOLEAN SetIniFile ( char* an_IniFile );
            // Setzt den Pfad und Namen des gewünschten IniFiles.
            // Falls diese Funktion
            // nicht benutzt wird, gilt die Default Einstellung.

   tBOOLEAN SetBabbleMode ( tBabbleMode a_BabbleMode );
            // legt den Ausgabemodus fest. Ist a_BabbleMode == esSilent
            // erfolgt keine Bildschirmausgabe, sonst
            // ( a_BabbleMode == esVerbose ) werde die Protokollfenster geoeffnet

   tBOOLEAN GetRamsis( CRamsisMaster* a_RamsisMaster, tLONGCARD IndividualIndex);
            // schreibt den internen Ramsis der Klasse CesLife auf den
            // in der Parameterliste angegebenen Parameter a_RamsisMaster
            // zurück. Welches Individuum dafür ausgewählt wird, gibt der Index
            // an.

   tBOOLEAN GetFitness(  tLONGCARD IndividualIndex, tREAL& a_Fitness);
            // Ergibt den Fitnesswert eines bestimmten Individuums in
            // der Population. Welches Individuum dafür ausgewählt wird,
            // gibt der Index IndividualIndex an.

   tBOOLEAN GetVoxelSpacePtr( CVoxelSpace *& aVoxelSpacePtr );
            // Diese Funktion gibt einen Pointer auf den internen Voxelraum
            // in aVoxelSpacePtr zurück.
            // War der interne Voxelraum noch nicht initialisiert, so liefet
            // sie cFALSE zurück, der Inhalt
            // des Rückgabepointers ist nicht definiert. Bei erfolgreicher
            // Aktion wird cTRUE zurückgegeben.

   tLONGCARD IndexRange();
            // ergibt eine Zahl, die den maximalen Index für
            // IndividualIndex der Funktion GetRamsis beschreibt

   tBOOLEAN StepLife( tBOOLEAN& calcFinished);
            // Berechnet einen Evolutionsschritt. solange calcFinished
            // cFALSE ergibt, ist die Berechnung noch nicht beendet

   tBOOLEAN GetPopulation(CesPopulation*& anPopulation);
```

```
      // Methoden des Migrationsmodells ...
      tINTEGER GetStep();
               // gibt den Populationszähler des letzen ausgeführten
               // Schrittes zurueck
               // anhand PopCount kann fuer mehrere parallele Life-
               // objekte geprüft werden ob sie "strukturkompatibel" sind
               // ist der elementare Bestandteil des Migrationsmodells.

   };
   #endif // __CesLife__
```

Abbildungsverzeichnis

Kapitel 1

Kapitel 2

Kapitel 3

Kapitel 4

Kapitel 5

Kapitel 6

Anhang A

Anhang B

Anhang C

Anhang D

Anhang E

Literaturverzeichnis

[BÄC93] Thomas Bäck, Günther Rudolph, Hans-Paul Schwefel. *Evolutionary Programming and Evolution Strategies: Similarities and Differences*. Universität Dortmund, 1993. ftp://ftp.uni-dortmund.de

[BAU74] Heinz Bauer. *Wahrscheinlichkeitstheorie und Grundzüge der Maßtheorie*. Walter de Gruyter Verlag Berlin 1974. 2. Auflage.

[BER89] Stefan Berchtold. *Evolution im Computer. Optimierung mit genetischen Algorithmen*. c't-Magazin, Heft 6, 1989.

[BRO87] I. N. Bronstein, K. A. Semendjajew. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Thun, 1987.

[BRO88] *Brockhaus Enzyklopädie*, F.A. Brockhaus GmbH, Mannheim, 1988.

[BUT97] Roger Butmuth, *Superrechner im Baukasten*, c't 10/97, Seite 178.

[DAR44] Charles Darwin. *The Origin of Species*. http://www.talk.origin.org/faqs/origin.htm, 1844.

[DAW76] R. Dawkins. *The selfish Gene*, Oxford University Press, Oxford 1976.

[DEN97] Jörg Denzinger, Stephan Scholz. *Using Teamwork for the Distribution of Approximately Solving the Traveling Salesman Problem with Genetic Algorithms*. Fachbereich Informatik, Universität Kaiserslautern, 1997.

[DIV87] A. Divivier, A. Seidel, R. Trieb. *Redesign RAMSIS. Strategiepapier zur Umsetzung eines neuen RAMSIS-Programmiersystems*. TecMath (interner Konzeptentwurf), 1994.

[DOB92] Dr. Dobb's Journal. *Pseudo-Random Sequence Generator for 32-Bit CPUs*. February 1992. Pages 34 to 40.

[DUD83] *Duden "Deutsches Wörterbuch"*. Bibliographisches Institut AG, Mannheim, 1983

[HAN96] Guido Hansen. *Konzeption und Realisierung eines Sensorsystems zur dreidimensionalen Erfassung anthropometrischer Daten*. Diplomarbeit, Universität Kaiserslautern, 1996.

[HEI96] J. Heitkötter, D. Beasley. *The Hitchhikers Guido To Evolutionary Computation*. Issue 4.3, 1996. ftp://ftp.germany.eu.net/pub/research/softcomp/EC/FAQ/

[KUB97] Klaus Kubitschek. *Implementierung und Visualisierung eines Voxeldatenraumes zur effizienten Generierung der Fitneßfunktion eines evolutionären Algorithmus*. TOPAS-Projekt, Universität Kaiserslautern, Fachbereich Informatik, 1997 (noch nicht fertiggestellt)

[KUR91] Frank Kursawe, Hans-Paul Schwefel. *Künstliche Evolution als Modell für natürliche Intelligenz*. Lehrstuhl für Systemanalyse, Universität Dortmund.

[MEY74] *Meyers Enzyklopädisches Lexikon*. Bibliographisches Institut, Mannheim, 1974.

[MIC96] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Third Edition. Springer Verlag Heidelberg, 1992, 1994, 1996.

[MIT96] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Massachusetts Institute of Technology 1996.

[MOS95] Pablo Moscato. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts*. California Institute of Technology, Pasadena California, 1995

[NEL95] Randolph Nelson. *Probability, Stochastic Processes and Queueing Theory.* Springer Verlag 1995.

[POL96] Hartmut Polheim. *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB*, 1996. http://www.systemtechnik.tu-ilmenau.de/~polheim/GA_Toolbox

[RAD94] Nicholas J. Radcliffe, Patrick D. Surry. *Formal Memetic Algorithms*. Edinburgh Parallel Computing Centre, University of Edinburgh, 1994.

[RAW91] G. Rawlins. *Foundations of Genetic Algorithms*. Morgan Kauffman Publishers, San Mateo, California, 1991

[RUD95] Günther Rudolf. *Massively Parallel Simulated Annealing und its Relation to Evolutionary Algorithms*. ftp://ftp.uni-dortmund.de

[SCH95] Hans-Paul Schwefel und Thomas Bäck. *Evolution Strategies II: Theoretical Aspects*. Universität Dortmund, 1995.ftp://ftp.uni-dortmund.de

[SCH96] Hans-Paul Schwefel, Thomas Beck. *Künstliche Evolution - eine intelligente Problemlösungsstrategie?* Universität Dortmund, 1996. ftp://ftp.uni-dortmund.de

[SCR90] Alan B. Scrivener. *A Curriculum for Cybernetics and Systems Theory*, 1990. http://www.well.com/user/abs/curriculum.htm

[STR92] Stroustrup, Bjarne. *Die C++ Programmiersprache*. Addison-Wesley Germany GmbH, Second Edition, 1992.

[SRZ93] Hans Rudolf Schwarz. *Numerische Mathematik*. Teubner Verlag, 3. Auflage, Stuttgart, 1993

[TBL91] Thomas Blümecke. *Wunder der Evolution. Optimierung mit Evolutionsstrategien und genetischen Algorithmen*. c't-Magazin, Heft 12, 1991.