Optimierung von Knotenkoordinaten eingeschränkter Triangulationen mit evolutionären Algorithmen

Diplomarbeit

Jochen Hanff
Berlin 1998

Diplomarbeit

am
Fachgebiet Theoretische Methoden der Bau- und Verkehrstechnik
Institut für Bauingenieurwesen
Technische Universität Berlin

Thema:

Optimierung von Knotenkoordinaten eingeschränkter
Triangulationen mit evolutionären Algorithmen

Autor:

cand.-Ing. Jochen Hanff, Matrikelnummer 147 157

Betreut von

o.Prof.Dr. Peter Jan Pahl und Dipl.Ing. Gerrit Schutte

Erklärung

Hiermit versichere ich, die vorliegende Arbeit nur mit den angegebenen Quellen und Hilfsmitteln und ohne fremde Hilfe angefertigt zu haben.

Berlin, den 17.12.98

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einführung

Die Finite-Element-Methode (FEM) ist ein im Bauingenieurwesen und anderen Ingenieursdisziplinen häufig eingesetztes numerisches Verfahren, um das physikalische Verhalten von Festkörpern, Flüssigkeiten und Gasen zu bestimmen. Diese Methode kann z.B. verwendet werden, um die Verformung einer Stahlbetonplatte oder das dynamische Verhalten einer Bogenstaumauer zu berechnen. Der große Vorteil der FEM liegt darin, daß auch Körper berechnet werden können, die beliebige Formen aufweisen und für die keine geschlossenen analytischen Lösungen gefunden werden können.

Die grundsätzliche Idee bei der FEM ist das Zerlegen der Struktur in kleine, endliche Teile. Das physikalischen Verhalten dieser finiten Elemente wird durch geeignete Näherungen beschrieben und anschließend werden diese wieder zur Gesamtstruktur zusammengesetzt.

Bei der Zerlegung der Struktur in finite Elemente wird das zu untersuchende Kontinuum diskretisiert. So ist es möglich, das physikalisches Verhalten an diskreten Punkten, den Knoten, zu berechnen. Durch das Zusammensetzen der finiten Elemente zur Gesamtstruktur kann man das Verhalten der Gesamtstruktur approximieren. Sind die Lösungen an den Knoten bekannt, so können auch Lösungen an beliebigen Stellen durch Interpolation bestimmt werden.

Die Diskretisierung benötigt ein zulässiges Netz, das die Form der Struktur beschreibt. Die Maschen des Netzes bestimmen die Form der finiten Elemente. Die Güte der Berechnungsergebnisse hängt stark von der Güte des zugrunde liegenden Netzes ab. Degenerierte Elemente beeinträchtigen die Ergebnisse der FE-Berechnung und die numerischen Lösungen liefern keine gute Näherung an das tatsächliche physikalische Verhalten des Körpers.

Deshalb erscheint es sinnvoll, die Geometrie des Netzes, d.h. die Knotenkoordinaten, für die FE-Analyse zu optimieren. In dieser Arbeit soll versucht werden, die Optimierung von Knotenkoordinaten eingeschränkter Triangulationen zu beschreiben und umzusetzen. Dabei soll die Optimierungsaufgabe mit einem auf die Berechnung von Funktionswerten, d.h. ohne die Ableitungen der Zielfunktion, beschränkten Verfahren gelöst werden. Für diese Aufgabe wird ein evolutionärer Algorithmus zur Lösung von Optimierungsproblemen verwendet.

Im ersten Teil dieser Arbeit werden die Grundlagen der Vernetzung und verschiede Methoden zur Netzgenerierung vorgestellt und erläutert. Das für die gestellte Aufgabe am besten geeignete Verfahren wird ausführlich beschrieben.

Für die Optimierung von Netzen ist ein quantifizierbares Merkmal für die Qualität eines Netzes erforderlich. In dieser Arbeit wird dazu ein Maß für die Güte von FE-Netzen definiert.

Im zweiten Teil dieser Arbeit sind die Grundlagen der Optimierung dargelegt. Faßt man die Koordinaten eines jeden Punktes, dessen Koordinaten nicht fest vorgegeben sind und der im Innern des zu vernetzenden Gebietes liegt, als Entwurfsvariablen einer Optimierungsaufgabe auf, so erscheinen aufgrund der Vielzahl der Parameter und der Restriktionen evolutionäre Optimierungsstrategien zweckmäßig. Im zweiten Teil dieser Arbeit werden die Grundlagen und Anwendungsgebiete evolutionärer Algorithmen vorgestellt.

Es wird eine Methode zur Netzgenerierung und zur Optimierung von Knotenkoordinaten gewählt. Im dritten Teil wird die Implementierung dieser Verfahren ausführlich beschrieben. An dieser Stelle wird auf die Datentechnik und die Anwendung der entwickelten Softwaremodule eingegangen.

Die entstandene Software wurde an einfachen und komplexen Beispielen getestet. Die Ergebnisse dieser Tests werden in Kapitel 4 gezeigt.

Die Arbeit schließt mit einem Ausblick auf zu untersuchende Fragestellungen im Zusammenhang mit der Anwendung von evolutionären Optimierungsstrategien in den Ingenieurwissenschaften und Hinweise auf mögliche Erweiterungen und Verbesserungen der Optimierung von Knotenkoordinaten.

Kapitel 1

Generierung von Netzen für FE-Anwendungen

1.1 Einführung in die Netzgenerierung

Die Diskretisierung einer Struktur für eine FEM-Analyse benötigt ein Netz, das zulässig ist und das die zu untersuchende Struktur möglichst gut approximiert. In den folgenden Abschnitten werden wichtige Eigenschaften von Netzen und die für die Generierung von Netzen grundlegenden Begriffe erläutert.

Informationen in einem Netz

In Bezug auf die FEM-Analyse muß das Netz verschiedene Informationen beinhalten. Nach [9] gehören dazu

- Geometrische Informationen.

  Die geometrische Beschreibung, die das zu untersuchende Kontinuum vollständig erfaßt.

- Informationen über die Interpolation in den finiten Elementen.

  Ein Netz kann, je nach Aufgabenstellung und verwendeten finiten Elementen, aus verschiedenen geometrischen Figuren bestehen. Es können, um nur einige Beispiele zu nennen, 3-Knoten Dreiecksnetze, 6-Knoten Dreiecksnetze oder 4-Knoten Viereckснetze erstellt werden. Für manche Aufgaben werden auch im selben Netz unterschiedliche geometrische Figuren kombiniert. Die Elemente des Netzes müssen mit den gewählten Interpolationsvorschriften im Element kompatibel sein.

  Aufgrund der Aufgabenstellung dieser Arbeit liegt der Schwerpunkt im folgenden bei der Generierung von Dreiecksnetzen. Eine Vernetzung mit Dreiecken heißt Triangulation.

- Informationen über Randbedingungen.

  Die Beschreibung des Netzes muß eine geeignete Modellierung der physikalischen und geometrischen Randbedingungen zulassen. Dazu gehören z.B. die Einprägung von Lasten oder von Lagerbedingungen.

Strukturierte und unstrukturierte Netze

Man unterscheidet grundsätzlich zwei Arten von Netzen: strukturierte und unstrukturierte Netze. Jeder Knoten eines strukturierten Netzes hat die gleiche Anzahl von Nachbarknoten. Ein Knoten V_i ist dann ein Nachbarknoten eines beliebigen Knotens V, wenn V_i über eine Kante mit dem Knoten V verbunden ist. Die Anzahl der Nachbarknoten bei unstrukturierten Netzen ist unterschiedlich.

Zulässigkeit von Netzen

In einem globalen Koordinatensystem sind die Koordinaten einer Punktmenge P gegeben. Es ist ein zulässiges Netz zu bestimmen, das die Punkte P als Knoten enthält. Zulässig ist ein Netz dann, wenn das zu vernetzende Gebiet lückenlos und ohne Überschneidungen in die gewählten geometrischen Figuren unterteilt wird. Dabei sind alle gegebenen Punkte zu verwenden. Ist eine Menge von Randkanten gegeben, die das Netzgebiet beschränken, so heißt die Vernetzung eingeschränkt. Kanten des Netzes dürfen nur vollständig im Innern oder auf dem Rand des Netzgebietes liegen. Bild 1.1 zeigt ein zulässiges und ein unzulässiges Dreiecksnetz.

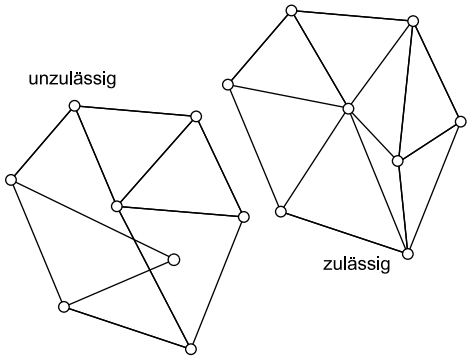


Bild 1.1: Unzulässiges und zulässiges Dreiecksnetz

Feste und freie Punkte

Im Zusammenhang mit der Optimierung von Knotenkoordinaten muß bei der Vernetzung eines Gebietes zwischen festen Punkten und freien Punkten unterschieden werden. Nur die Koordinaten freier Punkte können beim Optimierungsprozeß verändert werden.

Gegeben sei die Menge der Punkte $P = \{P_1, ..., P_n\}$ und die Menge der Randkanten $R = \{R_1, ..., R_k\}$, die das zu vernetzende Gebiet Ω beschränken. Die Punkte der Menge P sind fest. Im Innern des Netzgebietes sind neue, freie Punkte zu erzeugen. Dabei können zwei verschiedene Strategien verfolgt werden. Zum einen soll dem Nutzer die Möglichkeit gegeben werden, ein Raster mit definiertem Knotenabstand zu unterlegen, auf dessen Knoten freie Punkte erzeugt werden. Zum anderen soll die Anzahl von Innenpunkten vom Nutzer vorgegeben werden können, die nach bestimmten Gesichts-

punkten unregelmäßig im Innern des Gebietes verteilt werden. Beide Strategien werden von der in Abschnitt 3 vorgestellten Software angeboten.

Konvexität

Ein zentraler Begriff im Zusammenhang mit der Vernetzung von Punktmengen ist der Begriff der Konvexität eines Netzes. Ein Netz N ist dann konvex wenn gilt:

$$\begin{aligned} \forall \vec{x}_1 \in N \quad \text{und} \quad \forall \vec{x}_2 \in N \quad \text{und} \quad \forall \lambda \in]0,1[\\ \text{gilt} \quad \lambda \vec{x}_1 + (1-\lambda)\vec{x}_2 \in N. \end{aligned} \tag{1.1}$$

Einige Algorithmen, so z.B. die Delaunay-Triangulation, erzeugen ein konvexes Netz. Diese Eigenschaft kann zur effizienten Manipulation von Netzen genutzt werden. Ist das Netzgebiet beschränkt, so ist von vornherein vorgeben ob die Triangulation konvex oder nicht-konvex ist.

In den folgenden Abschnitten werden die wichtigsten Verfahren zur Netzgenerierung vorgestellt. Als Beispiel für das Generieren von strukturierten Netzen wird das Transport-Mapping Verfahren in Abschnitt 1.2.1 beschrieben. In den nachfolgenden Abschnitten werden verschiedene Strategien zum Generieren von unstrukturierten Netzen behandelt. Die Methode zur Delaunay-Triangulation und ihre Erweiterung zur Triangulation beschränkter Netzgebiete erscheinen für die gegebene Aufgabenstellung zweckmäßig und werden deshalb ausführlich erläutert.

Die vorliegende Arbeit beschäftigt sich hauptsächlich mit der Optimierung der Knotengeometrie gegebener Triangulationen. Für manche Aufgaben erscheint es zweckmäßig, aufgrund geometrischer Randbedingungen, a priori, d.h. vor einer FE-Analyse, die Feinheit der Netzstruktur beeinflussen zu können. Dazu wird in Abschnitt 1.3 eine Methode entwickelt, die es dem Anwender erlaubt, die Dichte der Punkte innerhalb des Netzes zu steuern. Diese vorgegebene Dichte ist beim Optimierungsprozeß mit zu berücksichtigen. Es sei ausdrücklich betont, daß dies keine adaptive Netzverdichtung darstellt, die Ergebnisse aus der FE-Analyse berücksichtigt, sondern dies nur eine manuelle Steuerung der Punktdichte durch den Anwender zuläßt.

Da die Güte der Ergebnisse einer FEM-Analyse vom zugrundeliegenden Netz beeinflußt wird, ist es notwendig, ein Maß für die Qualität eines Netzes zu definieren. Im Abschnitt 1.4.1 werden Kriterien zur Beschreibung der Güte von FE-Netzen genannt und Kenngrößen definiert, die die Qualität von Netzen quantifizieren.

1.2 Verfahren zur Netzgenerierung

Die Verfahren zur Vernetzung sind vielfältig und die Wahl einer geeigneten Methode hängt meist von der konkreten Aufgabenstellung ab. In den folgenden Abschnitten werden vier der gebräuchlichsten Methoden vorgestellt.

1.2.1 Transport-Mapping

Die Transport-Mapping-Methode beruht auf der Abbildung eines Referenznetzes Ω' auf ein zu vernetzendes Gebiet Ω. Das Referenznetz ist einfach zu bestimmen und die Koordinaten seiner Knoten werden mit einer Mapping-Funktion in das Netzgebiet transformiert. Dieses Vorgehen wird auch als 'Algebraische Methode' oder 'Transfinite Interpolation' [1] bezeichnet.

Im folgenden wird das generelle Prinzip dargestellt. Der Einfachheit halber wird bei der Beschreibung des Verfahrens ein quadratisches Referenznetz zugrunde gelegt.

- Gegeben sei ein Gebiet Ω, das ähnlich einem Viereck ist. Es wird angenommen, daß die Anzahl der Punkte auf zwei gegenüberliegenden Kanten gleich groß ist. Ω' sei ein Einheitsquadrat mit den Eckpunkten a'_n mit $n = \{1, ..., 4\}$. Die Eckpunkte von Ω werden auf die Eckpunkte a'_n von Ω' abgebildet. Die Positionen von Punkten zwischen zwei Eckpunkten werden so festgelegt, daß ihre Abstände untereinander das gleiche Verhältnis wie in Ω aufweisen. Schritt (1) in Bild 1.2 illustriert dieses Vorgehen.

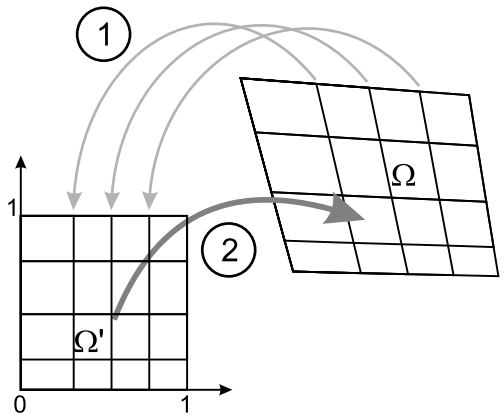


Bild 1.2: Transport-Mapping

- Das Referenzgebiet Ω' wird vernetzt indem gegenüberliegende Punkte verbunden werden. Um ein Dreiecksnetz zu erhalten, werden die Rechtecke in Dreiecke unterteilt.

- Mit einer Mapping-Funktion werden die Punkte in Ω' in das globale Koordinatensystem von Ω transformiert. Da die Vernetzung des Referenzgebietes üblicherweise strukturiert ist, ergibt sich auch eine strukturierte Vernetzung von Ω.

  Die geeignete Mapping-Funktion wird aufgrund der Form des zu vernetzenden Gebietes gewählt. Je nach Form können lineare, quadratische oder Funktionen höheren Grades benutzt werden. Auch für polygonal berandete Formen können Mapping-Funktionen aufgestellt werden. Hierbei kann es vorkommen, daß bei nicht-konvexen Gebieten Punkte, die innerhalb Ω' liegen, auf Punkte abgebildet

[1] siehe [8]

werden, die im globalen Koordinatensystem nicht in Ω liegen. In [9] und [8] ist dies näher beschrieben.

Die Bilder 1.3 und 1.4 zeigen eine Anpassung des Verfahrens für nicht-konvexe Gebiete, die das Abbilden von Punkten in Ω' auf Punkte im globalen Koordinatensystem außerhalb des Netzgebietes Ω vermeidet. Hier wird ein polygonal umrandetes Gebiet mit einem viereckigen, vereinfachten Gebiet beschrieben, welches mit der oben beschriebenen Methode vernetzt wird. Anschließend wird das generierte Netz so deformiert, daß es die ursprüngliche Form des zu vernetzenden Gebietes erhält. Das Verfahren ist ausführlich in [9] beschrieben.

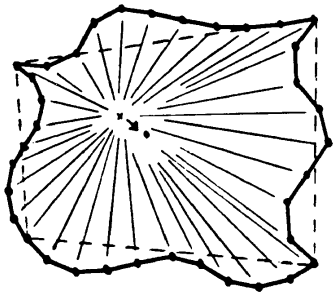


Bild 1.3: Beschreibung eines nicht-konvexen Gebietes durch eine grobe Näherung [9]

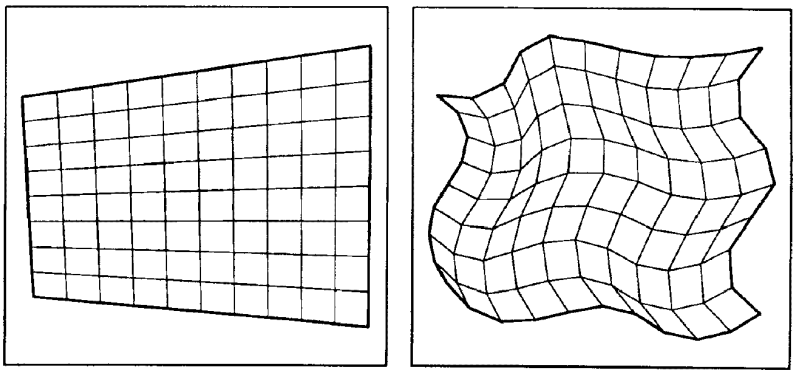


Bild 1.4: Vernetzung und Deformation [9]

Die Transport-Mapping Methode ist auf einfache Geometrien beschränkt. Große Probleme bereitet das Verfahren, wenn im Innern des zu vernetzenden Gebietes Aussparungen vorhanden sind. Die Abbildung der Punkte auf dem Rand der Aussparungen auf Punkte im Referenzsystem ist nicht eindeutig und erschwert eine geeignete Vernetzung des Referenzgebietes. Außerdem ist die Restriktion, daß auf gegenüber liegenden Kanten die gleiche Punktanzahl vorhanden sein muß, für beliebige Netzgebiete nicht immer zu erfüllen. Durch diese Nachteile ist die Methode nicht universell für beliebige Gebiete einsetzbar.

Obwohl das Verfahren sehr schnell ist, ist es aufgrund dieser Schwierigkeiten für die gestellte Aufgabe dieser Arbeit nicht geeignet.

1.2.2 Superposition eines regelmäßigen Rasters

- Ein regelmäßiges Raster aus Quadraten gleicher Größe wird über das zu vernetzende Gebiet Ω gelegt. Die Größe der Quadrate ist vom Abstand der Punkte auf dem Rand abhängig. Sie wird so gewählt, daß sich in jedem Quadrat höchstens ein Randpunkt befindet.

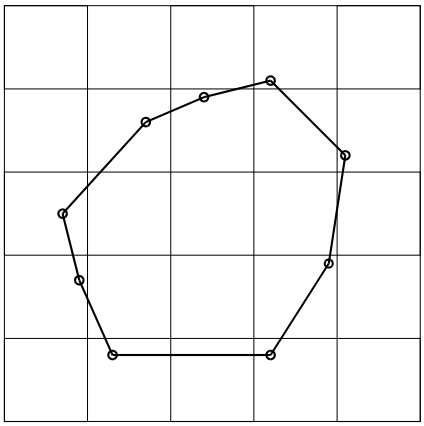


Bild 1.5: Erzeugen eines regelmäßigen Rasters

- Alle Elemente, die nicht vollständig zu Ω gehören oder nicht von einer Randkante geschnitten werden, werden entfernt.

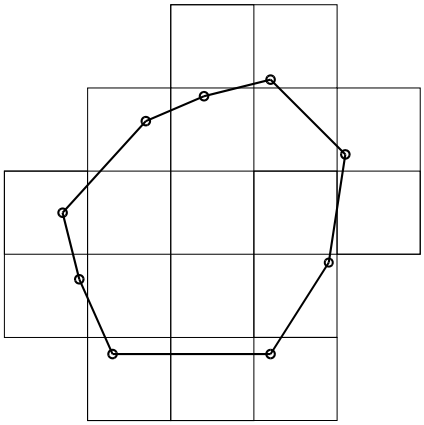


Bild 1.6: Entfernen der außenliegenden Quadrate

- Ein vollständig im Innern von Ω liegendes Quadrat wird zu einem Element des Netzes. Um Dreiecksnetze zu erhalten, werden diese Quadrate in zwei Dreiecke unterteilt.

  Befindet sich ein Knoten V in einem Quadrat, wird dieses wie in Bild 1.7 gezeigt, geteilt. Der Knoten V wird mit allen im Netzgebiet liegenden Knoten des Quadrates verbunden. Gebiete, die außerhalb Ω liegen, werden abgeschnitten.

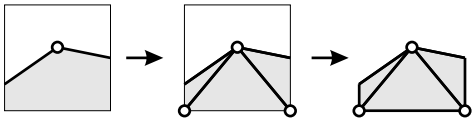


Bild 1.7: Unterteilung von Quadraten, in denen Randpunkte liegen

- Ein Element, das von einer Randkante geschnitten wird, wird in Abhängigkeit der Schnittführung in Elemente aufgeteilt. Bild 1.8 zeigt die verschiedenen Möglichkeiten, die auftreten können. An den Schnittpunkten der Kante mit dem Quadrat werden neue Randknoten des Netzes erzeugt. Das bedeutet, daß erst zu diesem Zeitpunkt die Randkanten und Randpunkte des endgültigen Netzes bekannt sind.

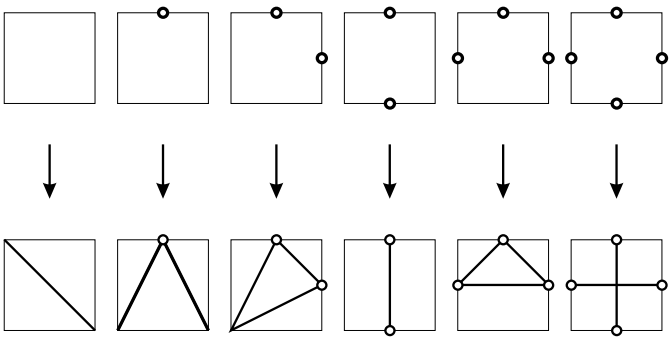


Bild 1.8: Unterteilung von Quadraten in Abhängigkeit der Schnittführung

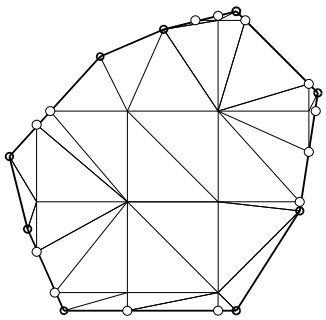


Bild 1.9: Durch Superposition eines regelmäßigen Rasters erzeugtes Dreiecksnetz

Schließlich erhält man ein Netz wie auf Bild 1.9 gezeigt.

Bei sehr unregelmäßig verteilten Randpunkten, d.h. wenn die Abstände der Randpunkte untereinander stark variieren, verschlechtert sich die Güte des Netzes sehr, da sich die Maschenweite des Netzes nach dem kleinsten Abstand zweier Randkantenpunkte richtet. Sind die Abstände in einem Bereich des Gebietes sehr klein, so wird ein sehr feines Raster erzeugt, welches in anderen Bereichen des Netzgebietes unangemessen sein

kann. Dieses Problem umgeht das Quadtree-Verfahren, das in Abschnitt 1.2.3 beschrieben ist.

1.2.3 Quadtree-Verfahren

Das Quadtree-Verfahren unterlegt kein regelmäßiges Raster, sondern verändert die Größe der Quadrate in Abhängigkeit von der Dichte der Randkantenpunkte. Dadurch kann man die Dichte des Netzes an die Verteilung der Punkte auf den Randkanten anpassen.

Der Algorithmus gliedert sich in drei Teile.

- Zuerst wird ein Quadrat Q_w definiert, das das gesamte zu vernetzende Gebiet Ω umfaßt. Dieses Quadrat ist die Wurzel eines Baumes, der pro Blatt vier Blätter oder kein Blatt haben kann.

  Ein Quadrat Q_i wird dann in vier gleich große Quadrate unterteilt, wenn in Q_i mehr als ein Randpunkt liegt. Mit den so erzeugten Quadraten verfährt man rekursiv wie mit Q_i. Man erhält durch dieses Vorgehen eine hierarchische Baumstruktur wie auf Bild 1.11 gezeigt.

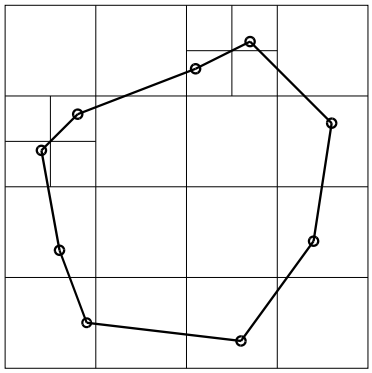


Bild 1.10: Festlegen eines Rasters beim Quadtree-Verfahren

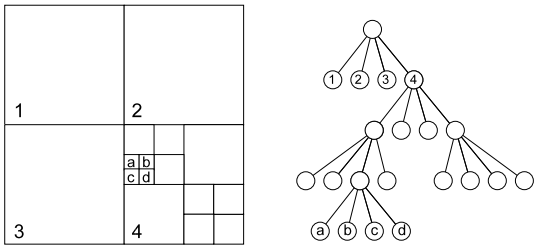


Bild 1.11: Hierarchische Struktur beim Quadtree-Verfahren

- Um ein ausgewogenes Netz zu bekommen, wird im zweiten Schritt ein Quadrat geteilt, wenn eines seiner Nachbarquadrate in der Hierarchiestruktur mehr als eine Stufe tiefer steht.

Quadrate, die nicht im Netzgebiet liegen oder nicht von einer Randkante geschnitten werden, werden, wie schon beim Verfahren mit einem regelmäßigen Netz, entfernt.

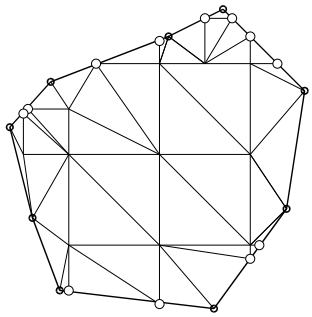


Bild 1.12: Mit dem Quadtree-Verfahren erzeugtes Netz

- Im dritten und letzten Schritt werden die Quadrate in Elemente des Netzes unterteilt. Quadrate, die vollständig in Ω liegen, werden in Dreiecke unterteilt. Diejenigen Quadrate, die keine Randkantenpunkte enthalten, aber von einer Randkante geschnitten werden, werden wie in Bild 1.8 gezeigt unterteilt. Liegt in einem Quadrat ein Randkantenpunkt, wird das Quadrat wie in Bild 1.7 gezeigt unterteilt.

Ein großer Vorteil dieses Verfahrens wird beim Vernetzen dreidimensionaler Strukturen deutlich. Wenn man die Quadrate zu Würfeln gleicher Kantenlänge erweitert, kann man mit demselben Algorithmus räumliche Gebilde vernetzen. In [9] ist dieses Verfahren ausführlich beschrieben.

Beide Methoden, die auf Unterlegung eines Rasters beruhen, sind für beliebige Netzgebiete anwendbar. Die erzeugten Dreiecke sind jedoch, gerade in der Nähe der Randkanten, degeneriert. Für den Fall, daß eine Randkante ein Quadrat sehr nahe an einem seiner Knoten schneidet, degenerieren die Elemente sehr stark. Möchte man dies vermeiden, so sind viele Sonderfälle zu prüfen, was den Aufwand drastisch erhöht. Auch ist es in vielen Fällen nicht erwünscht, neue Punkte auf dem Rand zu erzeugen.

Bessere Dreiecke erhält man mit dem Verfahren der Delaunay-Triangulation, die in den nächsten Abschnitten beschrieben ist.

1.2.4 Delaunay-Triangulation

Dieses Verfahren ist in erweiterter Form für die gegebene Aufgabenstellung zweckmäßig und wird deshalb im folgenden ausführlich beschrieben. Die Beschreibung des Verfahrens und die dazu nötigen Begriffsbestimmungen sind weitgehend [16] entnommen.

Um das Verfahren der Delaunay-Triangulation zu erläutern, sind vorab einige Begriffe zu definieren.

In einer Ebene E mit den Punkten X_i sei eine Punktmenge $P = \{P_1, ..., P_N\}$ von Bezugspunkten gegeben. Die Punkte P_i bilden die Knoten des zu bestimmenden Netzes. Die Teilfläche V, deren Punkte einen geringeren Abstand zu P_n als zu allen anderen Punkten von P hat, ist eindeutig zu bestimmen. Diese Tatsache ist Grundlage für die Delaunay-Triangulation.

Voronoï-Region: Die Menge der Punkte X_i in der Ebene E, deren Abstand zu einem Bezugspunkt P_n kleiner oder gleich ihrem Abstand zu allen anderen Bezugspunkten aus P ist, heißt Voronoï-Region des Bezugspunktes P_n. Die Voronoï-Region wird mit VR_n aus der Menge aller Voronoï-Regionen $VR = \{VR_1, ..., VR_N\}$ bezeichnet.

Besteht die Punktmenge P nur aus zwei Punkten P_1 und P_2, so teilt die Mittelsenkrechte der Verbindungsline $\overline{P_1P_2}$ die Ebene E in zwei Halbebenen $H_1(P_2)$, die P_1 enthält, und $H_2(P_1)$, die P_2 enthält. Die Halbebene $H_1(P_2)$ ist die Voronoï-Region VR_1 von P_1, die Halbebene $H_2(P_1)$ ist die Voronoï-Region VR_2 von P_2.

Besteht die Punktmenge P aus mehr als zwei Punkten, so kann für jedes Punktepaar (P_i, P_n) die Halbebene $H_i(P_n)$ und die Halbebene $H_n(P_i)$ bestimmt werden. Die Voronoï-Region VR_n des Punktes P_n ist der Durchschnitt aller Halbebenen von P_n gebildet mit allen anderen Punkten aus P:

$$VR(P_n) = \bigcap_{\substack{i=1 \\ i \neq n}}^{N} H_n(P_i). \tag{1.2}$$

Konvexität der Voronoï-Region: Man kann zeigen, daß die Voronoï-Region VR_n des Punktes P_n konvex ist. Deshalb ist es mit einfacher Delaunay-Triangulation nicht möglich, konkave Gebiete, wie sie bei eingeschränkter Triangulation vorkommen können, zu vernetzen. Eine erweiterte Delaunay-Triangulation für eingeschränkte Gebiete wird in Abschnitt 1.2.6 beschrieben.

Voronoï-Kante: Der gemeinsame Rand zweier Voronoï-Regionen VR_i und VR_n heißt Voronoï-Kante $VK(i, n)$. Jeder Punkt auf $VK(i, n)$ besitzt gleiche Abstände zu den Bezugspunkten P_i und P_n.

Voronoï-Knoten: Der Schnittpunkt zweier Voronoï-Kanten heißt Voronoï-Knoten V_k. Voronoï-Knoten besitzen folgende Eigenschaften:

- An einem Voronoï-Knoten treffen mindestens drei Voronoï-Kanten zusammen.
- An einem Voronoï-Knoten treffen mindestens drei Voronoï-Regionen zusammen.
- Ein Voronoï-Knoten besitzt den gleichen Abstand zu den Bezugspunkten aller Voronoï-Regionen, die an diesem Knoten zusammentreffen.

Voronoï-Kreis: Ein Voronoï-Kreis ist der Kreis durch die Bezugspunkte P_n aller Voronoï-Regionen, die bei einem Voronoï-Knoten V_k zusammentreffen. Der Voronoï-Knoten V_k ist Mittelpunkt dieses Kreises. Innerhalb des Voronoï-Kreises befinden sich keine weitere Bezugspunkte.

Voronoï-Dreieck: Treffen an einem Voronoï-Knoten drei Voronoï-Regionen zusammen, so sind die Bezugspunkte P_n die Eckpunkte eines Voronoï-Dreiecks. Da im Innern des Voronoï-Kreises keine weiteren Bezugspunkte liegen, liegen auch im Inneren des Voronoï-Dreiecks keine weiteren Bezugspunkte.

Treffen an einem Voronoï-Knoten mehr als drei Voronoï-Kanten zusammen, ist die Unterteilung in Voronoï-Dreiecke nicht eindeutig. Die Bezugspunkte dieser Region bilden die Eckpunkte eines Polygons.

Voronoï-Diagramm: Die Unterteilung der Ebene E in Voronoï-Regionen aller Bezugspunkte P_n aus einer gegebenen Punktemenge $P = \{P_1, ..., P_N\}$ heißt Voronoï-Diagramm von P. Das Voronoï-Diagramm ist eindeutig.

Es wird vorausgesetzt, daß auf einem Voronoï-Kreis nicht mehr als drei Bezugspunkte liegen. Liegen mehr als drei Bezugspunkte auf einem Voronoï-Kreis, wird willkürlich eine Triangulation des Voronoi-Polygons gewählt.

Unter diesen Voraussetzungen führt das Voronoï-Diaramm zu einer eindeutigen Vernetzung der Punktmenge. Diese Vernetzung nennt man Delaunay-Triangulation. Dieses Netz wird durch die geradlinige Verbindung aller Punkte konstruiert, die eine gemeinsame Voronoï-Kante besitzen. Der Rand des Delaunay-Netzes umschließt die konvexe Hülle [2] der Punktmenge P.

Da die Delaunay-Triangulation stets zu konvexen Netzen führt, ist die Bedingung, daß das Netz das zu vernetzende Kontinuum möglichst gut approximieren sollte, für konkave Strukturen verletzt.

1.2.5 Bowyer-Watson-Algorithmus

Um für eine gegebene Punktmenge P eine Delaunay-Triangulation zu ermitteln, ist der Algorithmus nach Bowyer-Watson geeignet. Man erzeugt ein beliebiges Dreieck, das alle zu vernetzenden Punkte enthält. Durch sukzessives Hinzufügen der Punkte P_i zum aktuellen Netz wird die Delaunay-Triangulation der Punktmenge P konstruiert. Das Einfügen eines Punktes geschieht nach folgendem Verfahren:

Einer gegebenen Delaunay-Triangulation sind Punkte hinzuzufügen. Auch nach dem Einfügen eines Punktes sei das Netz eine Delaunay-Triangulation.

- Gegeben sei eine Delaunay-Triangulation. Ein Punkt P_i wird im Innern der Triangulation hinzugefügt.
- Alle Dreiecke, deren Umkreise den Punkt P_i enthalten, werden gesucht.

  Dazu wird erst das Dreieck gesucht, das den Punkt P_i enthält. Danach werden die Nachbarn dieses Dreiecks ermittelt und geprüft, ob in deren Umkreis der Punkt P_i liegt. Mit den Nachbarn dieser Dreiecke verfährt man genauso. Dies wird solange

[2] es wird angenommen, daß das Netzgebiet somit eine Teilmenge der konvexen Hülle ist

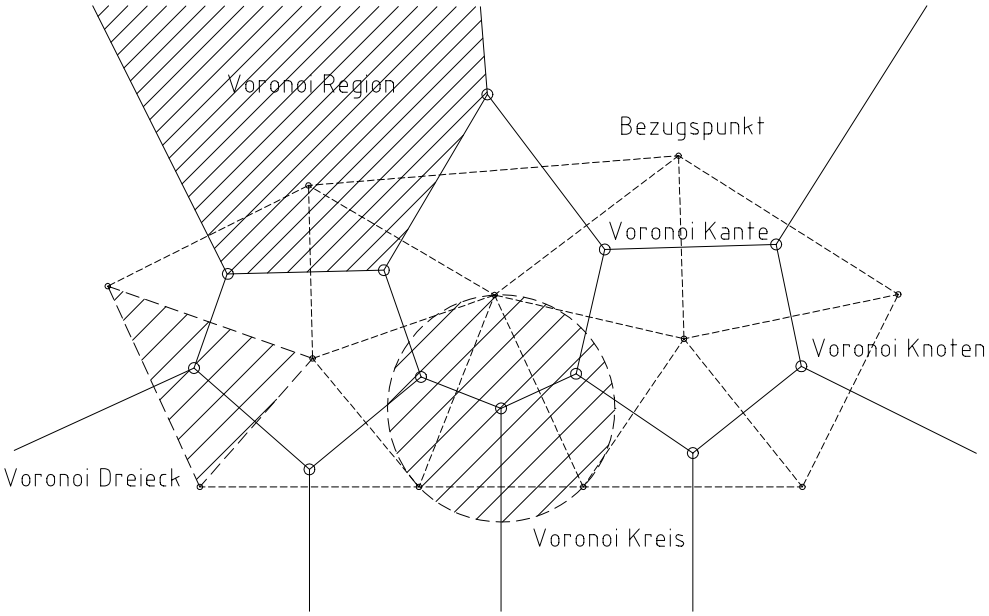


Bild 1.13: Delaunay-Triangulation und Voronoï-Diagramm

fortgeführt bis keine Dreiecke mehr gefunden werden, deren Umkreis den Punkt P_i enthalten.

- Die gefundenen Dreiecke werden gelöscht. Dadurch entstehen vorübergehend neue Randkanten R'. Die Delaunay-Triangulation stellt sicher, daß das gelöschte Gebiet konvex ist.

- Der Punkt P_i wird mit allen Bezugspunkten der Randkanten R' verbunden.

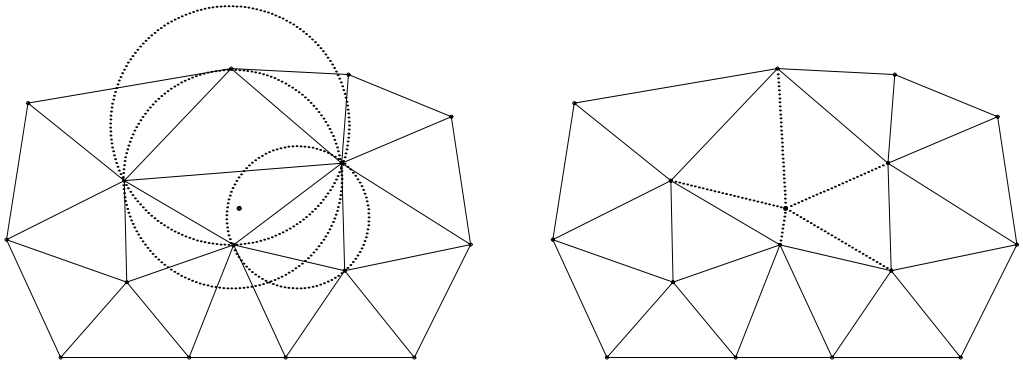


Bild 1.14: Einfügen eines Punktes nach dem Algorithmus von Bowyer-Watson

Nach Einfügen aller Punkte P_i in das Netz werden alle Dreiecke entfernt, die die Eckpunkte des am Anfang beliebig erzeugten Dreiecks enthalten.

1.2.6 Eingeschränkte Delaunay-Triangulation

Ist zusätzlich zur Punktmenge P eine Menge Randkanten R gegeben, so heißt die Triangulation eingeschränkt. Für die Vernetzung des Gebietes dürfen nur Kanten verwendet werden, die vollständig im Innern des Netzgebietes oder auf einer Randkante liegen. Es ist zu berücksichtigen, daß das zu vernetzende Gebiet nicht-konvex sein kann und deshalb die Delaunay-Triangulation, wie in Abschnitt 1.2.4 beschrieben, nicht unmittelbar angewendet werden kann.

Die Triangulation wird schrittweise durchgeführt. Der geschlossene Polygonzug der Randkanten bildet die aktuelle Front. Nach Hinzufügen eines neuen Dreiecks wird diese aktualisiert. In der Literatur [3] wird dieses Vorgehen auch als 'Advancing Front Methode' bezeichnet. Voraussetzung für dieses Verfahren ist ein geschlossenes Randgebiet.

Algorithmus für eine eingeschränkte Delaunay-Triangulation

Der folgende Algorithmus, der [16] entnommen ist, ist die Grundlage der in Kapitel 3 vorgestellten Implementierung in der Bibliothek `libMesh.a`. In Abschnitt 3.3 wird detailliert auf die Klassen und Methoden der Bibliothek eingegangen. An dieser Stelle soll der grundlegende Algorithmus zur eingeschränkten Triangulation dargestellt werden.

Gegeben sei die Menge der Punkte $P = \{P_1, ..., P_n\}$ und die Menge der gerichteten Randkanten $R = \{R_1, ..., R_n\}$. Das Netzgebiet Ω, das von R vollständig umschlossen ist, wird schrittweise vernetzt. Randkanten können auch im Innern von Ω liegen, aber auch diese müssen einen geschlossenen Polygonzug ergeben. In diesem Fall befinden sich im Netzgebiet Aussparungen.

Anfangs ist das aktuelle Netzgebiet gleich dem gegebenen Netzgebiet. In jedem Schritt wird das aktuelle Netzgebiet um ein Voronoï-Dreieck verkleinert.

1. Eine Randkante wird gewählt.
2. Die Endpunkte P_{r1} und P_{r2} der Randkante sind Elemente der Menge P. Ein Punkt P_3 wird aus P gewählt, der folgende Eigenschaften erfüllt.
   - Die Fläche des Dreiecks $P_{r1}P_{r2}P_3$ ist positiv. Damit ist sichergestellt, daß das Dreieck nicht ganz außerhalb Ω liegt und daß die Punkte P_{r1}, P_{r2} und P_3 nicht kollinear sind.
   - Die Kanten $P_{r1}P_3$ und $P_{r2}P_3$ schneiden keine Randkanten des aktuellen Netzgebietes. Durch diese Bedingung ist gewährleistet, daß das Dreieck nicht teilweise außerhalb des aktuellen Netzgebietes liegt.
   - Im Innern des Umkreises von $P_{r1}P_{r2}P_3$ liegen keine weiteren Punkte aus P. Dadurch ist sichergestellt, daß $P_{r1}P_{r2}P_3$ ein Voronoï-Dreieck ist.
3. In das neu generierte Dreieck werden die Knotenreferenzen in der Reihenfolge gegen den Uhrzeigersinn vermerkt.
4. Das aktuelle Netzgebiet wird um das neu generierte Voronoï-Dreieck verkleinert. Die Kante $P_{r1}P_{r2}$ ist fertig. Existieren die Kanten $P_{r1}P_3$ und $P_{r2}P_3$ schon, so sind auch diese fertig; wenn nicht, werden sie zu Randkanten des aktuellen Gebietes.

[3] siehe [9], [8]

Der Algorithmus endet nach dem Abarbeiten der letzten Randkante.

Das Prüfen auf einen Kantenschnitt der Seiten des Dreiecks mit der aktuellen Front und der damit verbundene Rechenaufwand verlangsamt das Verfahren sehr. Durch ein Unterteilen des Netzgebietes in Sektionen kann dieser Aufwand reduziert werden.

Bei diesem Algorithmus ist es vorteilhaft, daß auch nicht-konvexe Gebiete, die darüberhinaus auch Aussparungen besitzen dürfen, vernetzt werden können. Dadurch ist dieser Algorithmus universell für die Vernetzung beliebiger Gebiete einsetzbar und gut für das in dieser Arbeit zu untersuchende Problem geeignet.

1.3 Dichtefunktion

In vielen Fällen werden an die Größe der Elemente in einem Netz besondere Anforderungen gestellt. Die Größe der Elemente, und damit auch die Dichte der Knoten, ist den Randbedingungen des physikalischen Problems anzupassen. Dabei unterscheidet man grundsätzlich a priori und a posteriori Netzadaption.

Die a posteriori Netzanpassung erfolgt nach einer durchgeführten FE-Analyse und berücksichtigt deren Ergebnisse. Z.B. wird das Netz in Bereichen großer Spannungen verfeinert. Diese Strategie wird in dieser Arbeit nicht behandelt.

Die a priori Netzanpassung wird vor einer FE-Analyse durchgeführt. Sie soll dem Anwender erlauben, manuell die Dichte der Punkte beeinflussen zu können.

Zu diesem Zweck wird eine Dichtefunktion D definiert. Diese ist ein Maß für die Abstände der Punkte zu ihren Nachbarpunkten. Es soll gelten, daß kleine Werte für D kleine Werte für die Punktdichte ergeben. Deshalb wird D als der reziproke Wert des mittleren Abstandes eines Punktes zu seinen Nachbarpunkten definiert.

Der Wert der Dichte eines Punktes P_k ist definiert als

$$D(P_k) := \frac{n}{\sum_{i=1}^{n} d(P_k, P_{k_i})}, \tag{1.3}$$

mit n gleich der Anzahl der Nachbarpunkte und $d(P_k, P_{k_i})$ gleich dem geometrischen Abstand zwischen dem Punkt P_k und seinem Nachbarn P_{k_i}.

Um für jeden Punkt im Netzgebiet einen Soll-Wert für seine Dichte bestimmen zu können, wird das folgende Verfahren verwendet.

Sei $X = \{X_1, ..., X_n\}$ eine Menge von Punkten in der Ebene E. Die Delaunay-Triangulation für die Punktmenge X wird bestimmt. Das Netzgebiet Ω des zu untersuchenden Kontinuums sei eine Teilmenge der konvexen Hülle $H(X)$:

$$\Omega \subseteq H(X). \tag{1.4}$$

Diese Bedingung für die Menge X ist notwendig, da für jeden Punkt $\vec{x} \in \Omega$ ein Wert der Dichtefunktion durch Interpolation zu bestimmen sein muß.

An den Punkten X_i sei ein Wert $D(X_i)$ für die Dichte gegeben. Die Delaunay-Triangulation der Menge X wird dazu benutzt, Zwischenwerte $D(\vec{x})$ durch lineare Interpolation

zu bestimmen. Dabei bilden die Eckpunkte des Voronoï-Dreiecks $X_i X_j X_k$, in dem $\vec{x}$ liegt, die Stützstellen und die Werte $D(X_i)$, $D(X_j)$ und $D(X_k)$ die Stützwerte der Interpolation.

Das Netz der Dichtefunktion erstreckt sich über das gesamte zu untersuchende Kontinuum. Dadurch ist sichergestellt, daß zu jedem Punkt $P_k \in \Omega$ ein Dichtewert $D(P_k)$ durch lineare Interpolation bestimmt werden kann. Bild 1.15 zeigt ein Netzgebiet mit gestrichelt gezeichnetem Netz der Dichtefunktion.

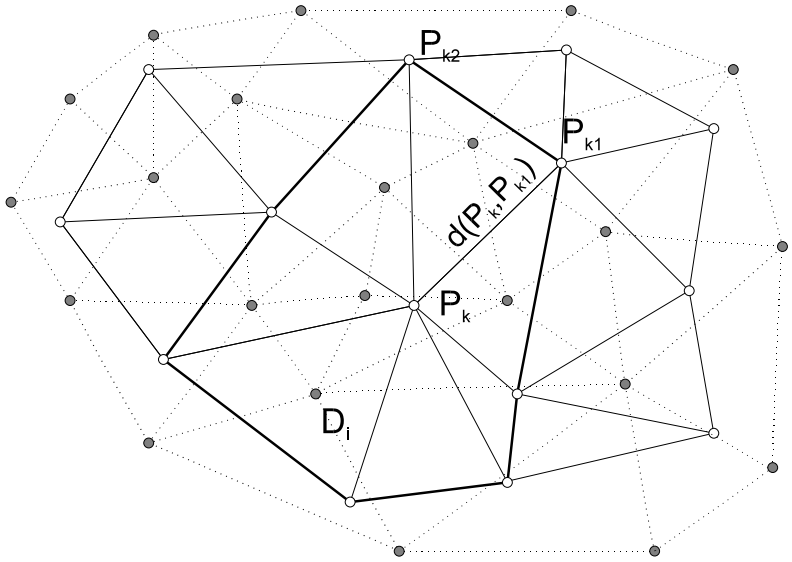


Bild 1.15: Dichtefunktion

Werden im Zuge der Optimierung des Netzes die Knotenkoordinaten eines Punktes P_k verändert, so verändert sich dadurch sowohl der aktuelle Wert der Dichte des Punktes, als auch der zu erzielende Wert der Dichtefunktion. Dies führt zu einer Mehrzieloptimierung, da neben den Bedingungen für die Qualität der Dreiecke auch alle Punkte des Netzgebietes den Wert der vorgegebenen Dichtefunktion erreichen sollen.

In Abnschnitt 2.3.7 wird auf das Problem von mehreren Zielen bei der Optimierung eingegangen und eine Lösungsmöglichkeit aufgezeigt.

1.4 Optimierung von Netzen

Die durch Delaunay-Triangulation erzeugten Netze sind bezüglich der gegebenen Punkte eindeutig und optimal. Die Netze können jedoch, bezüglich der Form der Dreiecke, durch Veränderung der Knotenkoordinaten der Innenpunkte für FE-Analysen verbessert werden. Im folgenden werden Kriterien für die Güte von Netzen definiert und klassische Methoden zur Verbesserung von gegebenen Netzen erläutert.

1.4.1 Kriterien für die Güte von Netzen

Die Ergebnisse von FE-Analysen hängen stark von der Güte des zugrunde liegenden Netzes ab. Das Netz muß die zu untersuchende Region möglichst gut approximieren. Das Netz muß fein genug sein, um eine akzeptable Lösung zu liefern. Trotzdem sollte die Anzahl der Elemente klein sein, um die Rechenzeit kurz zu halten. Prinzipiell sollten folgende Punkte beachtet werden:

1. Die Differenz der Größe zweier benachbarter Elemente sollte klein sein.

   Dies wird über den Vergleich der Fläche eines Dreiecks mit den Flächen seiner Nachbardreiecke berücksichtigt.

2. In Bereichen höherer Spannung sollte das Netz dichter bzw. feinmaschiger sein.

   Dies ist eine Bedingung, die erst nach einer erfolgten FE-Analyse überprüft werden kann. Deshalb wird die automatische Berücksichtigung dieser Bedingung in dieser Arbeit nicht behandelt. Trotzdem ist es über die in Abschnitt 1.3 vorgestellte Dichtefunktion möglich, manuell die Dichte eines Netzes, etwa bei geometrischen Unstetigkeiten, a priori zu beeinflussen.

3. Elemente sollten gleichförmig und nicht degeneriert sein.

   Diese Bedingung kann durch verschiedene Kriterien bewertet werden. In dieser Arbeit wird die Gleichmäßigkeit eines Dreiecks über die Größe der Innenwinkel bestimmt. Dabei wird ein gleichwinkliges Dreieck als das Beste bewertet.

   Denkbar ist aber auch eine Bewertung über das in der Literatur [4] als 'aspect ratio' bezeichnete Verhältnis vorzunehmen. Dieses ist das Verhältnis der längsten Ausdehnung einer geometrischen Figur zur ihrer kürzesten. Bei einem Dreieck wäre dies das Verhältnis der Hypotenuse zur Höhe des Dreiecks. Andere Autoren definieren diesen Wert als das Verhältis der Hypotenuse zum Inkreis des Dreiecks.

Für die Bewertung der Qualität von Dreieckselementen hinsichtlich ihrer Geometrie werden Bedingung 1. und 3. berücksichtigt. Dazu werden vier Kennwerte definiert.

Statistische Grundlagen

Für die Definition von Kennwerten, die eine Beurteilung der Güte von Netzen erlauben, werden folgende statistische Kenngrößen benötigt:

- Arithmetisches Mittel

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{1.5}$$

- Varianz

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2. \tag{1.6}$$

[4] siehe [19], [4]

- Standardabweichung

$$s = \sqrt{s^2}. \tag{1.7}$$

Kenngrößen für die Quantifizierung der Qualität von Dreiecksnetzen

Hinsichtlich der Optimierung von Knotenkoordinaten ist es zweckmäßig, Indikatoren für die Qualität eines Netzes zu definieren. Kenngröße Q_w berücksichtigt die Form der Elemente, Kenngröße Q_g berücksichtigt die Größe der Elemente bezüglich ihrer Nachbarn, Kenngröße Q_d gibt ein Maß für die Abweichung von der geforderten Dichte an einer Stelle des Netzes und Kenngröße Q_n ist eine Kombination der genannten Kenngrößen.

- Kenngröße Q_w

  Um die Form der Elemente eines Netzes zu bewerten, wird als Kenngröße die Standardabweichung der Innenwinkel aller Dreiecke definiert. Da die Summe der Innenwinkel eines Dreiecks 180° beträgt, ergibt sich das arithmetische Mittel zu 60°.

$$Q_w := \sqrt{\frac{1}{3n-1}\sum_{i=1}^{n}\sum_{j=1}^{3}(\alpha_{i,j} - 60°)^2}, \tag{1.8}$$

  mit n gleich der Anzahl der Dreiecke im Netz und $\alpha_{i,j}$ gleich dem Innenwinkel j im Dreieck i.

  Der optimale Wert liegt bei $Q_w = 0$ und ist dann erreicht, wenn nur gleichwinklige Dreiecke im Netz vorhanden sind.

- Kenngröße Q_g

  Um die Größenunterschiede der Elemente zu ihren Nachbarn zu bewerten, wird als Kenngröße die Standardabweichung des Verhältnisses der Größe eines Dreiecks zum arithmetischen Mittel der Größe seiner Nachbarn definiert.

  r_j ist das Verhältnis der Fläche A_j des Dreiecks D_j zum arithmetischen Mittel der Flächen A_i der Nachbardreiecke:

$$r_j = \frac{A_j}{\frac{1}{3}\sum_{i=1}^{3} A_i}. \tag{1.9}$$

  Der Wert liegt bei $r = 1$, wenn die Größe aller Dreiecke gleich ist.

  Die Kenngröße Q_g wird mit Gleichung 1.9 definiert als:

$$Q_g := \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(r_j - \overline{r})^2}. \tag{1.10}$$

Das Optimum liegt bei $Q_g = 0$ und ist dann erreicht, wenn alle Dreiecke die gleiche Größe besitzen. In Verbindung mit der in Abschnitt 1.3 definierten Dichtefunktion $D(\vec{x})$ entsteht ein Zielkonflikt zwischen der Dichte eines Punktes und dem Größenverhältnis eines Dreiecks zu seinen Nachbarn, wenn die Dichtefunktion nicht an jeder Stelle im Netzgebiet den gleichen Wert besitzt, da einerseits die Größe der Dreiecke möglichst gleich sein sollte, andererseits aber eine vorgegebene Dichte der Punkte angestrebt wird.

- Kenngröße Q_d

  Es wird eine Kenngröße für das Einhalten der in Abschnitt 1.3 beschriebenen Dichtefunktion definiert.

$$\Delta D = D_{P(\vec{x})} - D(\vec{x}). \tag{1.11}$$

$$Q_d := \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(\Delta D_{P_i} - \overline{\Delta D})^2}. \tag{1.12}$$

  mit $D_{P(\vec{x})}$ gleich dem Wert der Dichte des Punktes P an der Stelle $\vec{x}$, $D(\vec{x})$ gleich dem Wert der Dichtefunktion an der Stelle $\vec{x}$ und m gleich der Anzahl der Punkte im Netz.

- Kombinierte Kenngröße Q_n für die Quantifizierung der Qualität von Dreiecksnetzen

  Um alle oben definierten Kenngrößen gleichzeitig zu berücksichtigen, wird ein kombinierter Kennwert definiert, der die angesprochenen Kenngrößen vereint. Dazu wird die gewichtete Summe der Kenngrößen betrachtet.

  Mit den Gleichungen 1.8, 1.10 und 1.12 wird ein Maß für die Güte eines Netzes definiert als:

$$\boxed{Q_n := g_w \cdot Q_w + g_g \cdot Q_g + g_d \cdot Q_d \qquad \text{mit} \qquad g_i \geq 0.} \tag{1.13}$$

  Über die Gewichte g_i ist es möglich, die entsprechenden Ziele im Optimierungsprozeß zu betonen oder zu schwächen.

  Das Optimum liegt bei $Q_n = 0$ und ist dann erreicht, wenn alle Ziele ihr Optimum erreichen.

Für die in Abschnitt 3.3 gezeigte Implementierung des Netzgenerators wird diese Kenngröße für die Berechnung des Wertes der Zielfunktion verwendet.

1.4.2 Methoden zur Verbesserung von Netzen

Nachfolgend werden zwei Verfahren zur Verbesserung von FE-Netzen vorgestellt.

- Zentrierung des Innenpunktes (*Laplacian smoothing*)

  Die Menge aller Punkte P_{k_i}, die in der Nachbarschaft eines gegebenen Punktes P_k liegen und über eine Kante mit P_k verbunden sind, werden ermittelt. Die neuen Koordinaten von P_k werden über das gewichtete arithmetische Mittel der Koordinaten der Nachbarpunkte bestimmt.

$$P_k = \sum_{j=1}^{n} \alpha_j P_{k_j}, \tag{1.14}$$

  mit n gleich der Anzahl der Nachbarpunkte, die über Kanten mit P_k verbunden sind.

  α_j ist die zu P_{k_j} gehörende Gewichtung mit der Bedingung $\sum_j \alpha_j = 1$. Die Wahl einer Gewichtung ist z.B. auf Grundlage des reziproken Verhältnisses der Längen der an einen Punkt P_{k_j} anschließenden Kanten zur Gesamtlänge aller Kanten, die das umschließende Polygon bilden, möglich:

$$\alpha_j = \frac{1}{n-1} \cdot \left(1 - \frac{K_{j,j+1} + K_{j,j-1}}{2 \cdot \sum_{i=1}^{n-1} K_{i,i+1} + K_{n,1}} \right) \tag{1.15}$$

  $K_{i,j}$ ist die Länge der Kante von Punkt P_{k_i} nach Punkt P_{k_j}.

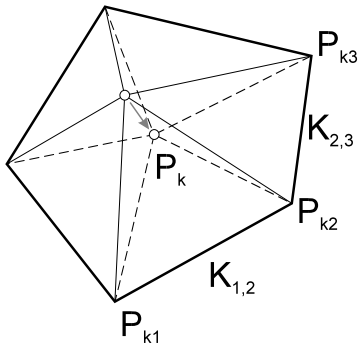


Bild 1.16: Laplacian smoothing

  Dadurch kann man ein ausgewogenes Netz erhalten. Allerdings wird die Güte eines Punktes und seiner anschließenden Nachbardreiecke durch das Variieren eines Punktes P_{k_j} in einem der nächsten Schritte wieder beeinträchtigt. Der Einfluß der Veränderung des Punktes auf die Dreiecke in der Nachbarschaft wird nicht berücksichtigt. Die Optimierung mit evolutionären Algorithmen soll diesen

Nachteil beheben, da alle Koordinaten als Parameter der Zielfunktion gleichzeitig berücksichtigt werden.

Nach [8] sind nur kleine Unregelmäßigkeiten mit diesem Verfahren zu korrigieren. Nach zwei bis fünf Iterationen konvergiert das Verfahren.

Ist das zu betrachtende Gebiet nicht konvex, so kann u.U. die Position des zentrierten Punktes außerhalb des aktuellen Gebietes liegen (siehe Bild 1.17). Da dieses nicht zulässig ist, ist deshalb nach jedem Schritt zu prüfen, ob die neuen Koordinaten im zulässigen Gebiet liegen.

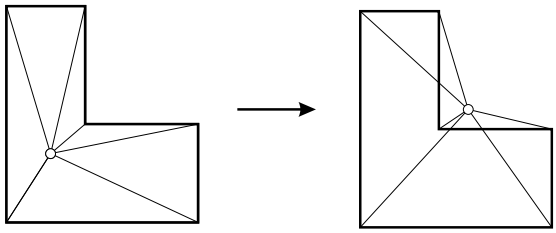


Bild 1.17: Laplacian smoothing, Punkt außerhalb des zulässigen Gebietes

- Iterative Methode

  Q sei ein definiertes Maß für die Güte der Dreiecke, die den Punkt P_k als Eckpunkt besitzen. Durch einen iterativen Prozeß verändert man die Koordinaten des Punktes P_k in verschiedene Richtungen und berechnet für die neue Geometrie der Dreiecke den Wert Q. Die Position, für die sich der beste Wert Q ergibt, ist Ausgangsposition für den nächsten Iterationsschritt. In jedem Schritt ist zu prüfen, ob sich P_k im zulässigen Gebiet befindet. Die Iteration ist entweder nach einer festgelegten Anzahl von Iterationsschritten abzubrechen oder wenn sich keine Verbesserung von Q ergibt.

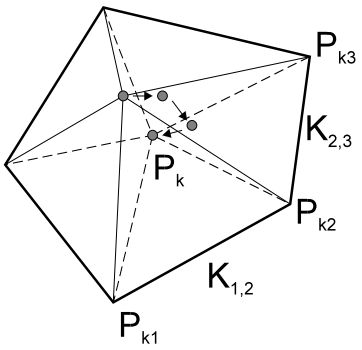


Bild 1.18: Iterative Methode

Auch bei diesem Verfahren wird die Auswirkung der Veränderung der Knotenkoordinaten auf die Güte der Nachbardreiecke nicht berücksichtigt. Das Verfahren ist dadurch weniger geeignet als die im folgenden beschriebenen evolutionären Algorithmen, die alle Knotenkoordinaten als Parameter gleichzeitig berücksichtigen.

Kapitel 2

Optimierung mit evolutionären Algorithmen

2.1 Einführung und Grundlagen der Optimierung

Viele Probleme in den Ingenieurwissenschaften lassen sich durch mathematische Modelle beschreiben. Diese zeichnen sich meist durch eine hohe Anzahl von Parametern aus. Ziel der Optimierung ist es, diejenige Kombination von Parametern zu finden, für die ein Gütekriterium für das mathematische Modell optimal wird.

In dieser Arbeit handelt es sich bei den Parametern um die Koordinaten der freien Punkte einer Triangulation, und das Gütekriterium für diese ist in Abschnitt 1.4.1 entwickelt worden.

Setzt man voraus, daß ein Wert für ein Gütekriterium zu minimieren[1] ist, läßt sich ein Optimierungsproblem allgemein wie folgt formulieren:

Finde einen Punkt $\hat{\vec{x}} \in X$, so daß gilt:

$$f(\hat{\vec{x}}) \leq f(\vec{x}), \qquad \forall \vec{x} \in X. \tag{2.1}$$

X ist das zulässige Lösungsgebiet. Die Funktion $f(\vec{x})$ heißt Zielfunktion und wird durch das mathematische Modell des zu optimierenden Systems bestimmt. Sie dient zur Bewertung der Parameter $\vec{x}$.

$f(\hat{\vec{x}})$ am Punkt $\hat{\vec{x}}$ heißt globales Optimum von $f(\vec{x})$.

Existiert eine ϵ-Umgebung, so daß gilt:

$$f(\hat{\vec{x}}) \leq f(\vec{x}), \qquad \forall \vec{x} \in U_\epsilon(\hat{\vec{x}}), \tag{2.2}$$

mit der ϵ-Umgebung

$$U_\epsilon(\hat{\vec{x}}) := \{\vec{x} \in \mathbb{R}^n \quad | \quad |\vec{x} - \hat{\vec{x}}| \leq \epsilon\}, \tag{2.3}$$

[1] Maximierung ist äquivalent

so spricht man von einem lokalen Optimum. In vielen Fällen konvergiert das Optimierungsverfahren gegen ein lokales Optimum. Nur in Sonderfällen kann mit Sicherheit davon ausgegangen werden, daß das gefundene Optimum ein globales Optimum ist. In den meisten Anwendungsfällen ist jedoch nur ein globales Optimum von Interesse.

Besitzt die Funktion $f(\vec{x})$ genau ein lokales Optimum, das gleichzeitig auch globales Optimum der Funktion ist, so heißt die Funktion unimodal. Existieren mehrere lokale Optima, dann heißt die Funktion multimodal.

Die Komplexität eines Optimierungsproblems wird gesteigert, wenn für die Zielfunktion Restriktionen gegeben sind.

$$\vec{x} = \{x_i, ..., x_n\} \in X. \tag{2.4}$$

$$g_i(\vec{x}) \leq 0, \qquad \forall i \in \{1, ..., m\}. \tag{2.5}$$

$$h_j(\vec{x}) = 0, \qquad \forall j \in \{1, ..., k\}. \tag{2.6}$$

Gl. 2.5 nennt man Ungleichungsrestriktionen, Gl. 2.6 heißen Gleichungsrestriktionen. Beide dienen dazu, den Bereich der zulässigen Lösungen für $f(\vec{x})$ einzuschränken.

Ist die Zielfunktion zweimal stetig differenzierbar, so lassen sich über die notwendige Bedingung

$$grad\ f(\vec{x}) = \left(\frac{\partial f(\vec{x})}{\partial x_1}, ..., \frac{\partial f(\vec{x})}{\partial x_n}, \right) = \vec{0} \tag{2.7}$$

und die notwendige und hinreichende Bedingung

$$\frac{\partial}{\partial x_j} \left(\frac{\partial f(\vec{x})}{\partial x_i} \right) \begin{matrix} > \\ = \\ < \end{matrix} 0 \qquad \text{mit} \qquad (1 \leq i, j \leq n) \tag{2.8}$$

die Extrema der Zielfunktion bestimmen. Der Vergleichsoperator bestimmt hierbei, ob es sich um ein Minimum oder ein Maximum handelt. Ist der Vergleichsoperator ein '=', besitzt die Funktion $f(\vec{x})$ kein lokales Extremum in $\vec{x}$.

Im Gegensatz zu anderen Optimierungsstrategien bestimmen evolutionäre Algorithmen ein Optimum nur mit Kenntnis des Funktionswertes der Zielfunktion und nicht über ihre Ableitungen. Man bezeichnet diese Strategien deshalb auch als Verfahren 0. Ordnung oder als direkte Verfahren. Für viele Anwendungsfälle, gerade für Simulationsaufgaben, ist es schwierig die Zielfunktion stetig differenzieren zu können. Für diese Fälle bieten sich evolutionäre Strategien an.

2.2 Einführung in die evolutionären Algorithmen

Schon seit Beginn des Computerzeitalters haben Wissenschaftler versucht, menschliches Verhalten, Intelligenz und biologische Evolution zu verstehen und auf Maschinen zu simulieren. In jüngster Vergangenheit haben Verfahren, auch außerhalb des Fachgebietes der Künstlichen Intelligenz, zunehmend an Bedeutung gewonnen, die sich die

Methoden der Natur und deren Mechanismen zum Vorbild nehmen. Evolutionäre Algorithmen (EA) für die Lösung von Optimierungsproblemen gehören zu dieser Gruppe von Verfahren.

EA orientieren sich an den Prinzipien der natürlichen Evolution, mit deren Hilfe man die Vielfältigkeit und Komplexität von Lebensformen durch wenige Mechanismen zu erklären versucht. Dazu gehört sowohl die Fortpflanzung, und die damit verbundene Weitergabe von Erbinformation, als auch die Mutation und Rekombination der Erbinformation. Durch Mutation und Rekombination kommt es zu einer Veränderung des Erbgutes der Nachkommen. Mutation verändert zufällig die Erbinformation eines Individuums, Rekombination dagegen kombiniert die Erbinformation zweier Eltern zum Erbgut eines Nachkommen. Auf diese Weise entstehen unterschiedliche Individuen, die in ihrer Umgebung unterschiedlich konkurrenzfähig sind. Durch Selektion wird sichergestellt, daß die gut angepaßten Individuen eine höhere Wahrscheinlichkeit haben ihre Erbinformation weitergeben zu können, als schlecht angepaße Individuen. So entwickelt sich die Anfangspopulation durch probabilistische Selektions-, Mutations- und Rekombinationsprozesse in einer Weise, daß die durchschnittliche Güte der Individuen auf Dauer zunimmt.

EA waren ursprünglich als universelle Optimierungs-Methoden gedacht, die für jedes Problem gleichermaßen anzuwenden seien. Heute haben sich EA als robuste, direkte, probabilistische Optimierungsverfahren bewährt. EA stellen für eine Vielzahl von Problemstellungen eine hervorragende Alternative zu den klassischen Verfahren dar.

Optimierungsverfahren werden grundsätzlich zwischen starken und schwachen Methoden unterschieden, die man nach den Kriterien der Allgemeinheit und Leistungsfähigkeit einer Methode beurteilt[2]. Starke Methoden machen von der Kenntnis des zu untersuchenden Problems Gebrauch und finden daher sehr effizient Lösungen. Schwache Methoden dagegen gelten als universelle Optimierungsverfahren. Sie sind bei einer Vielzahl von Problemen unverändert einsetzbar, sind aber dafür weniger leistungsfähig als eine für ein konkretes Problem angepaßte Methode. EA zählen, in ihrer reinen Form, in erster Linie zu den schwachen Verfahren, da die Algorithmen unverändert für verschiedene Probleme zu Lösungen führen. Berücksichtigt man jedoch Wissen aus dem Optimierungsproblem bei Anpassung des Mutations-, Rekombinations- oder Selektionsverfahren, kann man auch einen EA zu einer starken Methode machen.

EA sind hauptsächlich in vier Hauptrichtungen einzuteilen. Diese vier Hauptstömungen sind Genetische Algorithmen (GA), Genetische Programmierung (GP), Evolutionsstrategien (ES) und Evolutionäre Programmierung (EP).

Auch wenn sich diese Hauptrichtungen im Detail unterscheiden oder auf verschiedene Anwendungsgebiete abzielen, lassen sich folgende Gemeinsamkeiten feststellen:

- Die Parameter der Zielfunktion werden in String- oder Vektorform dargestellt.
- Es werden gleichzeitig mehrere Lösungsalternativen betrachtet.
- Es werden keine Ableitungen der Zielfunktion benötigt.
- Mutation, Rekombination und Selektion sind grundlegende Prinzipien des Optimierungsprozesses.

[2] siehe [14]

- Stochastische Elemente werden bewußt für die Suche nach dem Optimum verwendet, insbesondere bei der Veränderung der Parameter durch Mutation und Rekombination.

Der grundlegende Algorithmus der EA ist für alle vorgestellten Hauptströmungen gleich.

Man erzeugt eine zufällige Ausgangspopulation unterschiedlicher Lösungsalternativen. Durch Selektion bestimmt man diejenigen Lösungen, die Nachkommen für den nächsten Iterationsschritt erzeugen können. Die Nachkommen werden durch Mutation zufällig verändert und danach wiederum selektiert. Von den aktuellen Individuen werden dann wieder Nachkommen erzeugt. Die Iteration wird solange durchgeführt, bis eine maximale Anzahl von Iterationen durchlaufen wurde oder ein anderes vorgegebenes Abbruchkriterium erfüllt ist. Die verschiedenen EA-Strategien unterscheiden sich hauptsächlich durch verschiedene Mutations-, Rekombinations- und Selektionsverfahren.

Die erwähnten Methoden werden im folgenden vorgestellt und voneinander abgegrenzt. Für das Optimieren von Knotenkoordinaten eingeschränkter Triangulationen erscheinen aber nur Genetische Algorithmen und Evolutionsstrategien geeignet und werden deshalb ausführlich erläutert.

2.3 Genetische Algorithmen

Die ersten Grundlagen für Genetische Algorithmen (GA) wurden in den späten 60er Jahren von John Holland an der University of Michigan entwickelt. Im Gegensatz zur Evolutionsstrategie war die ursprüngliche Absicht von Holland nicht ein Optimierungsverfahren zu entwickeln, sondern die Mechanismen von adaptiven Systemen zu verstehen und erklären zu können. Schon bald wurden diese Erkenntnisse für die Lösung von Optimierungsproblemen eingesetzt. Heute ist diese Strömung der EA am weitesten verbreitet.

2.3.1 Terminologie

Die Informationen des Erbgutes sind bei einem Lebewesen im Zellkern in verschlüsselter Form gespeichert. Die verschlüsselte Erbinformation nennt man Genotyp. Die äußere Erscheinungsform eines Lebewesens, z.B. Augenfarbe, Größe und Haarfarbe, nennt man Phänotyp. Im Zusammenhang mit EA bezeichnet man als Genotyp die codierte Lösung und als Phänotyp die decodierte Lösung einer Optimierungsaufgabe.

Genetische Algorithmen betonen besonders die genetischen Mechanismen des evolutionären Prozesses. Sie arbeiten mit dem Genotyp eines Individuums. In Anlehnung an die Biologie werden die Parameter der Zielfunktion, die in einer Kette vereint dargestellt werden, als Chromosomen bezeichnet. Jedes dieser Chromosomen ist ein Array aus n Bits. In der ursprünglichen Form des Algorithmus ist die Reihenfolge festgelegt. Heute existieren Verfahren, die auch die Vertauschung der Reihenfolge in den Rekombinationsprozeß einbeziehen[3]. Die Länge des Chromosomes hängt vom konkreten An-

[3]In der Natur ist die Funktion eines Genes oft unabhängig von der Position im Chromosom. Durch

wendungsfall ab, ist aber während der Optimierung fest. Ein Chromosom gliedert sich in m Segmente. Jedes dieser Segmente wird einem bestimmten Parameter in der Zielfunktion des Optimierungsproblems zugeordnet. In Anlehnung an die Biologie werden die Bits als Gene und ihre Ausprägung als Allele bezeichnet.

Die Ausgangspopulation, bestehend aus einer Menge von Chromosomen, wird zufällig erzeugt. Der Optimierungsalgorithmus arbeitet mit einer binären Darstellung der Parameter der Zielfunktion. Deshalb ist es notwendig, Verfahren zur Codierung und zur Decodierung festzulegen.

2.3.2 Darstellungsformen der Parameter der Zielfunktion

Die Parameter der Zielfunktion sind in den meisten Anwendungsfällen Vektoren von reellen Zahlen. GA arbeiten i.d.R. jedoch mit einer binären Repräsentation der Parameter. Dies macht eine Codierung der Parameter erforderlich. Θ sei eine Vorschrift, die eine gegebene reelle Zahl x_i auf eine binäre Repräsentation a_i abbildet:

$$\Theta : x_i \mapsto a_i, \tag{2.9}$$

mit x_i gleich einer reellen Zahl im Dezimalsystem und a_i der äquivalenten Zahl in binär codierter Form.

Man kann als Codierung die in der Informatik übliche binäre Darstellung von Zahlen verwenden. Damit würde man, unter Verwendung des Datentyps `double`, mit 8 Byte/64 Bit bei einer Genauigkeit von 15 Nachkommastellen einen maximalen darstellbaren Bereich von -10^{308} bis +10^{308} erreichen. Dies führt jedoch für kleine Suchräume zu unangemessen großen Bitketten, die den Optimierungsprozeß unnötig erschweren und die darüberhinaus viele unzulässige Lösungen enthalten können. Besser ist es in diesem Fall, den Suchbereich durch zweckmäßige Codierung und Decodierung einzuschränken.

Die einfache binäre Codierung wurde um verschiedene Verfahren ergänzt. Zu nennen ist hier die 'Gray Codierung'. Der Schwerpunkt liegt jedoch auf der ursprünglichen Methode zur Codierung von reellen Zahlen.

2.3.2.1 Binäre Codierung der Parameter

Gesucht sei die Vorschrift Θ, die eine reelle Zahl x_i im Dezimalsystem auf eine binäre Ziffernfolge a_i abbildet. Der gesamte Definitionsbereich X der Zielfunktion $f(\vec{x})$ des Optimierungsproblems sei durch a_i darstellbar. Der Suchraum X sei auf eine für die Optimierung zweckmäßige Größe zu beschränken. Dazu ist eine notwendige Präzision bei der Darstellung und ein maximaler Darstellungsraum für die Ziffernfolge a_i zu bestimmen. Die meisten der heute angewandten EA basieren auf dem Codieren der Parameter in binärer Form mit fester Länge und festgelegter Reihenfolge.

Man erhält eine Abbildungsvorschrift durch folgendes Verfahren:

Identifizierung von Genen und Segmenten läßt sich dieser Umstand berücksichtigen. Man erweitert dann die Methoden zur Veränderung des Erbgutes zusätzlich um die Inversion. Für manche Problemstellungen mag dies vorteilhaft sein.

Gegeben sei ein Parameter x_i der Zielfunktion $f(\vec{x})$. Man wähle einen darstellbaren Lösungsbereich L mit einer oberen Grenzen L_o und einer unteren Grenze L_u, der als Teilmenge die Definitionsbereich der Zielfunktion X enthält.

$$X \subseteq L \tag{2.10}$$

Man wähle eine darstellbare Präzision p für die reellen Parameter x_i. Die Präzision p gibt die Anzahl der maximalen Nachkommastellen an.

Man teilt durch diese Festlegung den Lösungsbreich in eine feste Anzahl von Intervallen ein. Die Zahl der darstellbaren Intervalle ergibt sich zu:

$$n_I = (L_o - L_u) \cdot 10^p \qquad \text{mit} \quad n_I \in \mathbb{N}. \tag{2.11}$$

In Abhängigkeit der Anzahl der darstellbaren Intervalle n_I, läßt sich die Zahl der dazu nötigen binären Ziffern n_Z bestimmen.

Die Parameter $x_i \in [L_u, L_o]$ lassen sich auf ganze Zahlen abbilden:

$$N_{x_i} = (x_i - L_u) \cdot 10^p \qquad \text{mit} \quad N_{x_i} \in \mathbb{N}_0; \quad x_i \in [L_u, L_o]. \tag{2.12}$$

Durch die binäre Codierung der ganzen Zahl N_{x_i} erhält man eine binäre Zahl a_i, die das Intervall repäsentiert, in dem sich der Parameter x_i befindet.

Beispiel: Gegeben seien drei Parameter $x_1 = 1,43, x_2 = 2,21, x_3 = 2,4$, die in einem zulässigen Lösungsbereich von $L = [1,4; 2,5]$ liegen. Die Präzision der Lösung soll 2 Nachkommastellen betragen. Damit ergibt sich

$$p = 2 \quad \curvearrowright \quad n_I = (2,5 - 1,4) \cdot 10^2 = 110$$

Die Darstellung von 110 Intervallen erfordert mindestens 7 binäre Ziffern: n_Z = 7. Man beachte, daß der darstellbare Bereich mit einer bestimmten Anzahl von binären Ziffern meist größer ist als die geforderte Anzahl von Intervallen, da n_I immer eine Potenz von 2 ist. Es ist deshalb dafür Sorge zu tragen, daß Lösungen, die außerhalb des zulässigen Intervalls liegen, nicht im weiteren Verlauf der Optimierung berücksichtigt werden.

Die Abbildung der Parameter x_i auf ganze Zahlen und deren binäre Codierung a_i ergibt sich somit zu:

$$\begin{array}{lllll} x_1 = 1,43 & \Rightarrow & N_{x_1} = (1,43 - 1,4) \cdot 10^2 = 3 & \Rightarrow & a_1 = \texttt{0000011}, \\ x_2 = 2,21 & \Rightarrow & N_{x_2} = (2,21 - 1,4) \cdot 10^2 = 81 & \Rightarrow & a_2 = \texttt{1010001}, \\ x_3 = 2,40 & \Rightarrow & N_{x_3} = (2,4 - 1,4) \cdot 10^2 = 100 & \Rightarrow & a_3 = \texttt{1100100}. \end{array}$$

Mit $a_{i,j}$ wird im folgenden die j-te Ziffer der binären Zahl a_i bezeichnet.

2.3.2.2 Binäre Decodierung

Gesucht sei die Vorschrift Θ^{-1}, die eine binäre Repräsentation a_i auf die reelle Zahl x_i abbildet.

Die binäre Darstellung wird decodiert:

$$x_i = \Theta^{-1}(a_i) = L_u + \frac{L_o - L_u}{n_I} \sum_{z=0}^{n_Z-1} \left(a_{i,(n_Z-z)} \cdot 2^z\right), \tag{2.13}$$

mit $a_{i,k}$ gleich der binären Ziffer an der Stelle k der Zahl a_i.

Beispiel: Der reelle Wert x_2 der binären Zahl a_2 = `1010001` des Beispiels in Abschnitt 2.3.2.1 sei zu bestimmen:

$$x_2 = L_u + \tfrac{L_o - L_u}{n_I} \sum_{z=0}^{n_Z-1} \left(a_{2,(n_Z-z)} \cdot 2^z\right) =$$

$$1,4 + \tfrac{2,5-1,4}{110} \left((1 \cdot 2^0) + (1 \cdot 2^4) + (1 \cdot 2^6)\right) = 2,21.$$

2.3.2.3 Gray Codierung

Ein grundlegendes Prinzip der GA ist, daß geringfügige Veränderungen im Erbgut nur kleine Änderungen des decodierten Wertes bewirken sollen. Invertiert man jedoch ein Bit, das an einer der ersten Stellen im Segment steht, so hat dies eine zahlenmäßig große Auswirkung auf den reellen Parameter der Zielfunktion zur Folge.

Um dieses Problem zu umgehen, muß man andere Strategien bei der Codierung verfolgen. Eine erweiterte Form der binären Darstellung ist die 'Gray Codierung'. Die Gray Codierung hat den Vorteil, bei Änderung eines Bits nur eine kleine Änderung des decodierten Wertes hervorzurufen. Man konvertiert die binäre Darstellung in die Gray Codierung wie folgt (nach [14]):

Standard Binärcode zu Gray Code:

$$\gamma_{i,j} = \begin{cases} a_{i,j} & \text{für } j = 1 \\ a_{i,j-1} \oplus a_{i,j} & \text{für } j > 1 \end{cases} \tag{2.14}$$

Gray Code zu Standard Binärcode:

$$a_{i,j} = \bigoplus_{k=1}^{j} \gamma_{i,k} \tag{2.15}$$

mit $a_{i,j}$ gleich der binären Ziffer an der Stelle j im Standard Binärcode und $\gamma_{i,k}$ der binären Ziffer an der Stelle j im Gray Code. Der Operator $\oplus$ steht für "Addition modulo 2 im Binärraum". Es gilt $0 \oplus 0 = 0; 0 \oplus 1 = 1; 1 \oplus 0 = 1; 1 \oplus 1 = 0$.

Beispiele siehe Tabelle 2.1.

2.3.2.4 Nichtbinäre Darstellung

In jedem Schritt der Iteration des Optimierungsverfahrens wird die Bewertung der Individuen anhand der Zielfunktion durchgeführt. Dafür muß die binäre Repräsentation der Individuen decodiert werden. Um die aufwendige Codierung und Decodierung zu

Ganzzahl	Standard	Gray
0	0000	0000
1	0001	0001
2	0010	0011
11	1011	1110

Tabelle 2.1: Bsp. für Gray Codierung

vermeiden, ist es möglich, die Parameter in ihrem eigentlichen Datentyp zu speichern und die Chromosomen als Ketten von reellen Zahlen oder Buchstaben darzustellen.

Holland gab eine theoretische Rechtfertigung für die binäre Lösungsrepräsentation[4], die implizit besagt, daß GA ihre beste Leistungsfähigkeit entfalten, wenn sie mit binärer Lösungsrepräsentation arbeiten. Deshalb befürchten einige Autoren eine Abnahme der Leistungsfähigkeit des Optimierungsverfahren bei nicht-binärer Codierung der Parameter der Zielfunktion.

Verschiedene empirische Vergleiche zeigten jedoch, daß die Leistungsfähigkeit eng mit dem konkreten Problem und der Umsetzung des GA verbunden ist, und deshalb kein allgemein gültiges Urteil gefällt werden kann.

2.3.2.5 Diploidie

Paarweise zugeordnete Chromosomen, deren assoziierte Segmente die gleichen Parameter einer Zielfunktion repräsentieren, nennt man diploide Chromosomen. Beide Chromosomenstränge enthalten in den gleichen Segmenten Information für die gleichen Variablen der Zielfunktion, jedoch kann ihr Wert verschieden sein. Welches Gen sich beim Vererbungsprozeß durchsetzt hängt davon ab, welches Gen dominant und welches rezessiv ist.

Bei Optimierungsproblemen mit veränderlicher Zielfunktion wird dieses Verfahren gerne eingesetzt. Verschlechtern sich die Fitneßwerte eines Chromosoms dramatisch, so kann dieser Verschlechterung durch Änderung der Dominanz der einzelnen Gene entgegengewirkt werden.

2.3.3 Fitneßfunktion

Die Bewertung der Individuen wird mit der Fitneßfunktion Φ vorgenommen. Φ hängt von der Zielfunktion $f(\vec{x})$ des Optimierungsproblems ab. Die binären Parameter $\vec{a}$ werden decodiert:

$$\Phi(\vec{a}) = f\left(\Theta^{-1}(\vec{a})\right). \tag{2.16}$$

Die Fitneß eines Individuums muß sich nicht zwingend aus der Zielfunktion ergeben. Sie kann aber auch im Experiment oder in einer Simulation bestimmt werden.

In vielen Fällen ist es zweckmäßig, den Wert der Fitneßfunktion zu skalieren. Da die fitneßproportionalen Selektionsverfahren davon ausgehen, daß der Fitneßwert immer

[4] 'schema-counting argument' siehe [13]

positiv ist, ist der Wert der Fitneßfunktion entsprechend anzupassen. Die einfachste Methode transformiert die Fitneßwerte in den positiven Bereich, indem von jedem Fitneßwert der kleinste Fitneßwert der Population subtrahiert wird. In der Literatur sind weitere Möglichkeiten genannt, z.B. 'Sigma-Skalierung', 'Lineare dynamische Skalierung' und andere[5]. Es wird auf diese Methoden nicht weiter eingegangen. Im Beispiel des Abschnitts 2.3.10 und in der Implementierung der Software wird das einfachste Verfahren der Subtraktion des kleinsten Fitneßwertes gewählt.

2.3.4 Mutation

Bei den GA wird die Mutation, im Gegensatz zu anderen EA, als Operator von untergeordneter Bedeutung eingestuft. Die weitverbreitete Meinung ist, daß die Rekombination das wichtigste Instrument der GA sei, um Veränderungen im Erbgut zu bewirken. Mutation soll nur verhindern, daß ein Bit dauerhaft seinen Wert behält und damit der aktuelle Suchraum auf einen Unterraum des ursprünglichen Lösungsgebietes begrenzt wird.

Bei der Mutation wird jedes Bit eines Chromosoms mit einer Wahrscheinlichkeit p_m invertiert. Empfehlungen aus der Literatur liegen zwischen p_m = 0,01 und p_m = 0,001 (nach [14]):

$$a_{i,j} = \begin{cases} 1 - a_{i,j} & \text{für } U \leq p_m, \\ a_{i,j} & \text{für } U > p_m, \end{cases} \tag{2.17}$$

mit U gleich einer gleichverteilten Zufallszahl im Intervall [0,1[. U ist für jedes Gen neu zu bestimmen.

Die Schwierigkeit besteht darin, eine adäquate Mutations-Wahrscheinlichkeit p_m für das zu untersuchende Optimierungsproblem zu finden. Wünschenswert wäre eine zur Laufzeit stattfindende Selbstanpassung der Mutations-Wahrscheinlichkeit an den aktuellen Optimierungsprozeß. Dazu gibt es Untersuchungen[6] die den Mutationsoperator als Teil des Chromosoms dem Optimierungsprozeß unterwerfen. Dieser Bereich gehört zu den aktuellen Forschungsfeldern im Bereich der GA.

2.3.5 Rekombination (*Crossover*)

Die Rekombination ist der wichtigste Suchoperator der Genetischen Algorithmen. Rekombination erlaubt große Sprünge im Lösungsraum der Zielfunktion und erhöht dadurch die Wahrscheinlichkeit ein globales Optimum zu finden. Rekombination findet bei der Erzeugung von Nachkommen statt und wird zwischen zwei Eltern-Individuen durchgeführt. Die an einer Rekombination beteiligten Chromosomen werden zufällig und mit Zurücklegen aus der Menge aller Chromosomen gezogen. Dabei haben Individuen mit guten Fitneßwerten eine größere Wahrscheinlichkeit gezogen zu werden, als Individuen mit schlechter Fitneß.

Die in den folgenden Abschnitten beschriebenen Methoden sind weitgehend [14] entnommen.

[5] siehe [14] und [13]

[6] siehe [13]

2.3.5.1 Single-Point Crossover

Dieses Verfahren wurde von Holland in einem der ersten Versuche mit GA verwendet. Mittels einer gleichverteilten Zufallsgröße wird entschieden, an welcher Stelle im Chromosom ein Crossover stattfinden soll. Die Segmentaufteilung des Chromosoms wird dabei nicht berücksichtigt, d.h. ein Crossover kann auch innerhalb eines Segments stattfinden.

An der Schnittstelle werden die Teilstücke der Chromosomen ausgetauscht.

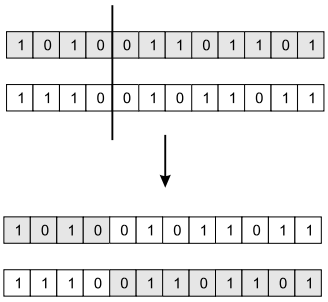


Bild 2.1: Single-Point Crossover

Man vertritt heute allgemein die Auffassung, daß Single-Point Crossover nicht die Leistungsfähigkeit besitzt wie die nachfolgend vorgestellten Methoden. Deshalb wird es auch nur in wenigen Fällen eingesetzt.

2.3.5.2 N-Point Crossover

Für jedes Elternpaar werden N Punkte zufällig bestimmt, an denen ein Crossover stattfinden soll. Dadurch teilt man die Chromosomen jeweils in N+1 Teilstücke. Jedes zweite Teilstück wird mit dem Teilstück des anderen Chromosoms ausgetauscht.

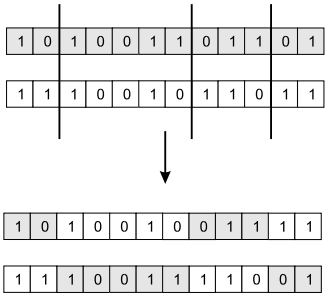


Bild 2.2: N-Point Crossover

2.3.5.3 Uniform Crossover

Für jedes Gen wird mittels einer gleichverteilten Zufallsgröße geprüft, ob es getauscht werden soll. Die übliche Größe für diese Wahrscheinlichkeit liegt zwischen 0,5 $\leq p_c \leq$ 0,8.

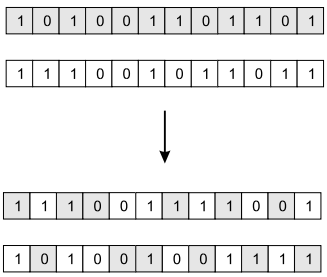


Bild 2.3: Uniform Crossover

2.3.5.4 Diagonal Crossover

Diese Crossover-Methode bezieht mehr als zwei Eltern in die Erzeugung von Nachkommen ein. Für N Eltern werden zufällig N-1 Schnittpunkte bestimmt. Die Abschnitte werden zyklisch vertauscht.

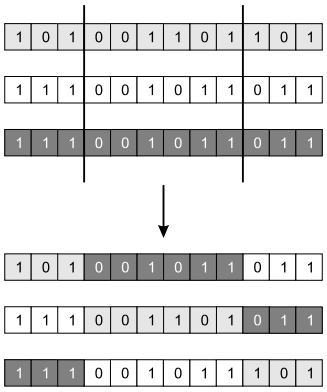


Bild 2.4: Diagonal Crossover

2.3.6 Selektion

Die Aufgabe der Selektion besteht darin, zu bestimmen, welche Individuen an der Erzeugung von Nachkommen im nächsten Iterationsschritt beteiligt sein werden. Dabei wird auch festgelegt, wieviele Nachkommen einzelne Individuen erzeugen werden. Die gut angepaßten Individuen werden dabei begünstigt.

Man kopiert zu diesem Zweck die Individuen, die für die Erzeugung von Nachkommen bestimmt sind, in einen sogenannten 'Mating Pool'. Welche Individuen, und in welcher

Anzahl dorthin kopiert werden, wird durch das Selektionsverfahren bestimmt.

Die Strenge der Selektionsstrategie ist zweckmäßig auf das Optimierungsproblem abzustimmen. Eine zu strenge Selektion würde suboptimale Lösungen begünstigen. Dann besteht die Gefahr, daß der Optimierungsprozeß gegen ein lokales Optimum konvergiert. Eine zu schwache Selektion führt dagegen zu einer langsamen Evolution.

2.3.6.1 "Roulette-Wheel"-Selektion, "Stochastic Universal Sampling"

Die Auswahl durch "Roulette-Wheel" Selektion und "Stochastic Universal Sampling" sind fitneßproportionale Verfahren. D.h., die Wahrscheinlichkeit eines Individuums in den Mating Pool kopiert zu werden ist direkt proportional zu seinem Fitneßwert.

Der Erwartungswert E_k eines Individuums sei sein Fitneßwert Φ_k im Verhältnis zur durchschnittlichen Fitneß seiner Population.

$$E_k = \frac{\Phi_k}{\frac{1}{N}\sum_{i=1}^{N}\Phi_i}. \tag{2.18}$$

N ist die Anzahl der Individuen in der Population. Der Erwartungswert ist eine Kenngröße für die zu erwartende Anzahl von Kopien eines Individuums im Mating Pool und damit in der nächsten Generation.

Da der Erwartungswert eine reelle Zahl ist, aber die Anzahl der Kopien nur ganze Zahlen sein können, ist mit einem zweckmäßigen Verfahren die Anzahl der Kopien zu bestimmen.

Man betrachte ein Glücksrad (Roulette Wheel), das in Teilbereiche aufgeteilt ist. Die Größe der Teilstücke ist proportional zum Erwartungswert der Individuen. Man dreht nun N-mal an dem Rad und kopiert jedes Individuum, das einen Treffer erhält, in den Mating Pool. Dieses Verfahren kann dazu führen, daß bei kleinen Populationen die Differenz zwischen der Anzahl der Kopien eines Individuums und seinem Erwartungswert groß ist.

Das "Stochastic Universal Sampling" (SUS) löst dieses Problem. Das Rad dreht sich nur einmal. Der Winkel der Drehung wird stochastisch bestimmt. Es gibt nun N Zeiger mit gleichem Abstand. Wieder wird jedes Individuum, auf das ein Zeiger zeigt, in den Mating Pool kopiert. Auf Bild 2.5 sind beide Verfahren illustriert.

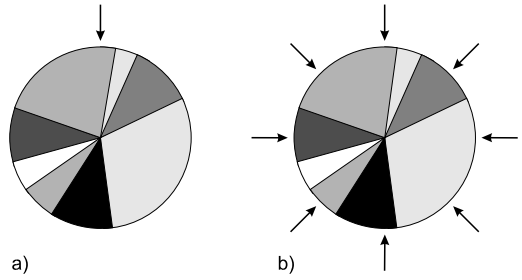


Bild 2.5: a) "Roulette Wheel", b) "Stochastic Universal Sampling"

"Stochastic Universal Sampling" ist heute eine gängige Methode und in den meisten Softwareprodukten implementiert.

2.3.6.2 Elite Selektion

Stellt man sicher, daß in jedem Iterationsschritt der Optimierung das beste Individuum auf jeden Fall in die neue Generation übernommen wird, nennt man dies 'Elite-Selektion'. Bei anderen Selektionsverfahren kann die bislang beste Lösung unter Umständen durch das zufällige Auswählen der Nachkommen aus der Population verloren gehen.

2.3.6.3 Wettkampfselektion

Zwei Individuen werden zufällig mit Zurücklegen gewählt. Über eine Zufallszahl z wird bestimmt, ob das Individuum mit dem besseren Fitneßwert in die neue Generation übernommen wird oder das Individuum mit dem schlechteren Fitneßwert. Um dies zu entscheiden wird die Zufallszahl mit einem vorher festgelegten Parameter k verglichen. Je nachdem ob z größer oder kleiner als k ist, wird eines der beiden Individuen in den Mating Pool kopiert.

2.3.6.4 Rangbasierte Selektion

Die Selektionswahrscheinlichkeit hängt von der Position eines Individuums in einer sortierten Liste ab. Als Sortierkriterium dient der Wert der Fitneßfunktion jedes Chromosoms. Der Unterschied zu anderen Selektionsverfahren ist, daß die Differenz der Fitneßwerte weniger Gewicht hat als bei anderen Methoden.

Man geht dabei so vor, daß das erste Objekt in der Liste den höchsten Erwartungwert und das letzte Objekt entsprechend den niedrigsten Erwartungwert erhält. Die Selektionswahrscheinlichkeit wird entsprechend dem Erwartungswert durch eine lineare Verteilung bestimmt.

2.3.6.5 "Generational Replacement" und "Steady-State" Selektion

Bisher ist davon ausgegangen worden, daß die aktuelle Population vollständig durch neue Individuen ersetzt wird. Diese Strategie nennt man "Generational Replacement". In manchen Selektionsstrategien, z.B. der Elite Selektion, sind jedoch manche Individuen der alten Generation auch in der nachfolgenden Generation enthalten. Werden nicht alle Individuen durch Nachkommen ersetzt, sondern nur die mit den schlechtesten Fitneßwerten, spricht man von 'Steady-State' Selektion.

2.3.7 Mehrzieloptimierung

Die meisten Optimierungsprobleme in der Praxis sind durch mehrere Ziele, die gleichzeitig zu optimieren sind, gekennzeichnet. Zudem sind diese Ziele oft einander gegensätzlich, d.h. die Optimierung des einen Ziels verschlechtert die Eigenschaften eines Individuums bezüglich eines anderen Ziels.

Bei der Optimierung von Knotenkoordinaten eingeschränkter Triangulationen sind ebenfalls mehrere Ziele zu berücksichtigen. Jede der in Abschnitt 1.4.1 definierten Kenngrößen gilt es gleichzeitig zu optimieren.

2.3.7.1 Aggregation

Die verschiedenen Ziele werden in gewichteter Form zu einer Zielgröße zusammengefaßt. In Gl. 1.13 wird eine kombinierte Kenngröße definiert, die alle zu verfolgenden Ziele gewichtet und summiert. Auf der Grundlage dieser Kenngröße wird die Ziel- und Fitneßfunktion ermittelt.

Das Mehrzielproblem kann dadurch in der gleichen Weise wie ein Einzielproblem gelöst werden. Durch die Gewichtung können für die Einzelziele Prioritäten vergeben werden.

2.3.7.2 Wechselnde Ziele

Gegeben sei ein Optimierungsproblem mit n Zielen und μ Individuen in der Population. Bei der Selektion werden μ Elternpaare gezogen, die Nachkommen erzeugen. Die Selektion wird n mal mit wechselnden Zielen durchgeführt. Die Zahl m der pro Selektionsdurchgang zu ziehenden Individuen beträgt:

$$m_i = \frac{\mu}{n} \cdot \gamma_i, \qquad \text{mit} \qquad m_i \in \mathbb{N}, \tag{2.19}$$

mit γ_i gleich der Gewichtung des Zieles i und der Bedingung $\sum_i \gamma_i = n$.

Durch die Gewichtung der Ziele ist sichergestellt, daß vorrangige Ziele bei der Selektion begünstigt werden.

2.3.7.3 Pareto-optimale Mengen

Von großer Bedeutung ist das Konzept der 'Pareto-Optimalität'. Definition aus [14]:

Eine Lösung $\vec{x}$ aus einem beliebigen Lösungsraum $L \neq \varnothing$ wird als Pareto-optimal (= effizient) bezeichnet, wenn es in L keine Lösung $\vec{x}_B$ gibt, die hinsichtlich des Zielerreichungsgrades bei keinem Zielkriterium schlechter, jedoch bei mindestens einem Zielkriterium besser ist als $\vec{x}_A$. Formal ergibt sich unter der Annahme einer Maximierung von:

$$\vec{f}(\vec{x}) = (f_1(\vec{x}), ..., f_g(\vec{x})), \quad \vec{x} \in L, \tag{2.20}$$

daß eine Lösung $\vec{x}_A \in L$ genau dann Pareto-optimal ist, wenn keine andere Lösung $\vec{x}_B \in L$ existiert, die $\vec{x}_B$ dominiert. Das heißt, es darf keine Lösung $\vec{x}_B \in L$ geben, so daß:

$$\forall z \in \{1, 2, ..., g\}, \quad f_z(\vec{x}_B) \geq f_z(\vec{x}_A) \quad \text{und} \tag{2.21}$$

$$\exists z \in \{1, 2, ..., g\}, \quad f_z(\vec{x}_B) > f_z(\vec{x}_A). \tag{2.22}$$

Der Ansatz der Mehrzieloptimierung basierend auf Pareto-optimalen Mengen ist eine rangbasierte Selektion. Es wird eine Liste erstellt, in der die Individuen nach Rang

sortiert sind. Der Rang eines Individuums ergibt sich aus der Zugehörigkeit zu einer Pareto-optimalen Menge und wird nach folgendem Prinzip ermittelt. Nicht-dominierte Individuen erhalten den Rang 1. Sie sind Element der Pareto-optimalen Menge bezüglich der gesamten aktuellen Population. Sie werden in die Rangliste kopiert und aus der Menge der Individuen entfernt. Die Individuen mit dem Rang 1 werden dadurch für die Bestimmung der nachfolgenden Positionen in der Rangliste nicht weiter berücksichtigt. Dadurch ändert sich auch der Status der restlichen Individuen, d.h. bislang dominierte Individuen können zu nicht-dominierten Individuen werden. Aus der Menge der restlichen Individuen werden diejenigen ausgewählt und entfernt, die nun zu nicht-dominierten Individuen geworden sind. Diese erhalten den Rang 2. Für jede nachfolgende Auswahl wird der Rang erhöht. Dieses Auswahlverfahren wird solange durchgeführt, bis alle Individuen in die Rangliste kopiert worden sind. Die Selektionswahrscheinlichkeit für die Auswahl zur Rekombination richtet sich nach dem Rang der Individuen. Individuen mit gleichem Rang haben auch gleiche Selektionswahrscheinlichkeit, Chromosomen mit dem Rang 1 wird die höchste Selektionswahrscheinlichkeit zugeordnet.

Die Menge der Pareto-optimalen Lösungen, auch als Kompromißmenge bezeichnet, enthält nur Lösungsvorschläge, die alle Ziele gleichermaßen berücksichtigt. Zur Auswahl einer konkreten Lösung ist eine Gewichtung der verfolgten Ziele zwingend erforderlich.

2.3.8 Restriktionen

Viele Probleme in der Praxis sind durch Restriktionen, die den zulässigen Lösungsbereich einschränken, gekennzeichnet. Es ist eine Vorgehensweise zu wählen, wie Lösungen, die nicht im zulässigen Lösungsbereich liegen, zu behandeln sind.

2.3.8.1 Elimination unzulässiger Lösungen

Die einfachste Möglichkeit ist es, die Lösungen, die Restriktionen verletzen, aus der aktuellen Population zu entfernen. Diese werden dann im Optimierungsprozeß nicht weiter berücksichtigt.

Dieses Vorgehen ist nicht effizient, da u.U. sehr viele Lösungen erzeugt werden könnten, die nicht zur Lösung des Optimierungsproblem beitragen. Trotzdem wird diese Methode häufig angewendet, da dieses Verfahren sehr einfach umzusetzen ist und dies den erwähnten Nachteil aufwiegt.

2.3.8.2 Straffunktion

Liegt eine Lösung nicht im zulässigen Lösungsgebiet, wird ein Strafterm zu dem Wert der Zielfunktion eines Individuums addiert. Der Fitneßwert des Individuums verschlechtert sich dadurch.

Unter der Voraussetzung der Minimierung gilt:

$$\Phi(\vec{a}) = \begin{cases} f(\vec{x}) & \text{für } \vec{x} \in L, \\ f(\vec{x}) + S(\vec{x}) & \text{für } \vec{x} \in \neg L, \end{cases} \tag{2.23}$$

wobei die Funktion $S(\vec{x})$ sowohl ein konstanter Wert sein kann, als auch eine Funktion in Abhängigkeit des Ortes $\vec{x}$.

In [14] wird die Empfehlung gegeben, die Größe des Strafterms vom Ausmaß der Ungültigkeit der Lösung abhängig zu machen. Hier kann der Abstand zum zulässigen Lösungsgebiet ein Kriterium für den Wert der Straffunktion sein.

Dieses Verfahren hat den Nachteil, daß unzulässige Lösungen im Optimierungsprozeß weiter berücksichtigt werden und dadurch Rechenzeit beanspruchen.

2.3.8.3 Zweckmäßige Codierung

Bei der Codierung der Parameter wird sichergestellt, daß alle möglichen Kombinationen der binären Codierung im zulässigen Lösungsgebiet liegen und somit alle Restriktionen erfüllen.

Schon bei der in Abschnitt 2.3.2.1 gezeigten binären Codierung gibt es binäre Zahlenkombinationen, die keine zulässige Lösung darstellen, da die Zahl der darstellbaren Ziffern größer ist als die Anzahl der Intervalle, in die der Lösungsbereich eingeteilt wird. Versucht man sicherzustellen, daß der zulässige Lösungsbereich vollständig auf die binäre Darstellung der Parameter eineindeutig abgebildet wird, muß ein ungleich größerer Aufwand bei der Erstellung der Codierungsvorschrift getrieben werden.

Im Allgemeinen ist es schwer eine Codierungsvorschrift zu finden, die diese Forderung erfüllt. Deshalb ist diese Methode auf nur wenige Ausnahmen beschränkt.

2.3.9 Basis-Algorithmus

Im folgenden ist der grundlegende Algorithmus der GA beschrieben.

1. Initialisiere stochastisch eine Ausgangspopulation von Individuen.
2. Bewerte jedes Individuum.
3. Selektiere stochastisch μ Individuen aus der aktuellen Population.

   Die Güte der Individuen wird berücksichtigt. D.h. Individuen mit guten Fitneßwerten haben eine höhere Wahrscheinlichkeit gezogen zu werden.
4. Erzeuge Nachkommen.

   Durch Mutation und Rekombination werden neue Individuen erzeugt.
5. Bewerte die erzeugten Nachkommen und füge die besten Individuen in die Population ein.

   Um die Größe der Population konstant zu halten, werden soviele Individuen der Population gelöscht, wie Nachkommen in die Population einzufügen sind. Individuen mit schlechten Fitneßwerten haben eine größere Wahrscheinlichkeit gelöscht zu werden.
6. Wenn eine gegebene Anzahl von Iterationen noch nicht erreicht ist oder ein anderes Abbruchkriterium nicht erfüllt ist, dann fahre mit Schritt 3. fort.

Als Abbruchkriterium ist das Über- oder Unterschreiten bestimmter Schranken denkbar. Als Schranke ist z.B. ein bestimmtes Verhältnis der mittleren Güte der Individuen zum besten Individuum einer Population sinnvoll oder die relative Änderung der Güte des besten Indiviuums in zwei oder mehreren aufeinanderfolgenden Iterationsschritten.

Im Laufe der Evolution werden die schlechten Individuen durch bessere Individuen ersetzt. Die durchschnittliche Fitneß der Chromosomen nimmt zu.

2.3.10 Beispiele für die Optimierung mit GA

Beispiel 1

Die Zielfunktion $f(x, y)$ sei mit einem Genetischen Algorithmus zu optimieren:

$$f(x, y) = \sin x + \sin y \qquad \begin{matrix} x \in [-\pi, \pi] \\ y \in [-\pi, \pi] \end{matrix} \tag{2.24}$$

Bild 2.6 zeigt die gegebene Funktion.
Das globale Optimum liegt bei $f(x = \frac{\pi}{2}, y = \frac{\pi}{2}) = 2$.

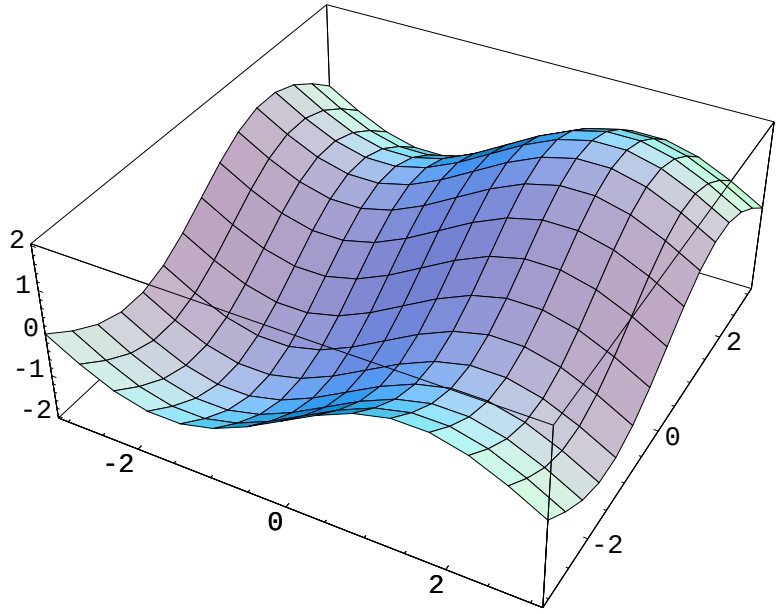


Bild 2.6: Zielfunktion: f(x,y) = sin x + sin y

Die Präzision wird mit p = 1 gewählt. Beide Parameter x, y seien im Bereich $[-\pi, \pi]$ darstellbar.

Die Anzahl der darzustellenden Intervalle ist:

$$n_I = (L_o - L_u) \cdot 10^p \rightarrow (3,141592 - (-3,141592)) \cdot 10^1 \rightarrow n_I = 63, \tag{2.25}$$

Dafür werden 6 binäre Ziffern benötigt. Für zwei Parameter ergibt sich somit die Länge der Chromosomen zu 12 Ziffern.

Es wird stochastisch eine Ausgangspopulation mit 8 Individuen erzeugt. Die Parameter in binärer Darstellung sind, um die Lesbarkeit zu verbessern, in den Tabellen 2.2, 2.3 und 2.4 durch einen Punkt getrennt.

Um ausschließlich positive Fitneßwerte zu erhalten, wird vom Fitneßwert Φ_i eines Individuums der kleinste Fitneßwert Φ_{min} der Population subtrahiert.

Der Erwartungswert E wird mit Gl. 2.18 bestimmt. Der gerundete Erwartungswert E gibt direkt die Anzahl der Kopien des Individuums in der nächsten Generation wieder. Ist die Anzahl der Individuen im Mating Pool aufgrund von Rundungsfehlern kleiner als die Größe der Population, so wird der Mating Pool mit dem besten Individuum der aktuellen Population aufgefüllt. Entsprechend werden die schlechtesten Individuen entfernt, sollte die Anzahl der Chromosomen im Mating Pool größer sein als die vorgegebene Populationsgröße.

1. Generation, F_{min} = -1,788, F_{max} = 1,333					
Nr.	Binärcode	Parameter im Dezimalsystem	Zielfunktion F	Fitneß $\Phi = F - F_{min}$	Erwartungswert E
1.	`100101.101001`	(0,549/0,947)	1,333	3,121	2,087
2.	`111001.001101`	(2,543/-1,845)	-0,399	1,389	0,929
3.	`001110.000100`	(-1,745/-2,743)	-1,373	0,415	0,277
4.	`011001.101010`	(-0,648/1,047)	0,262	2,050	1,371
5.	`111001.111001`	(2,543/2,543)	1,127	2,915	1,949
6.	`001010.010011`	(-2,144/-1,247)	-1,788	0,000	0,000
7.	`000110.000011`	(-2,543/-2,842)	-0,858	0,930	0,622
8.	`000111.111111`	(-2,443/3,142)	-0,643	1,145	0,766
			$\overline{F}$ = -0,292	$\overline{\Phi}$ = 1,496	$\sum E = 8$

Tabelle 2.2: Ausgangspopulation

Die Individuen werden entsprechend ihrem Erwartungswert in den Mating Pool kopiert. Durch Rekombination wird die nachfolgende Population bestimmt. Als Rekombinationsverfahren wird N-Point-Crossover mit 2 Punkten gewählt. Die Crossover-Punkte sind mit `'|'` gekennzeichnet. Das mittlere Teilstück wird jeweils ausgetauscht.

Mutation wird bei diesem Beispiel nicht durchgeführt.

Nr.	Binärcode	Nr.	Crossover
1.	`100101.101001 100101.101001`	1.	`100 | 101.101 | 001`
2.	`111001.001101`	2.	`111 | 001.001 | 101`
4.	`011001.101010`	4.	`01 | 1001.1 | 01010`
5.	`111001.111001 111001.111001`	5.	`11 | 1001.1 | 11001`
7.	`000110.000011`	7.	`00011 | 0.0000 | 11`
8.	`000111.111111`	5.	`11100 | 1.1110 | 01`
		8.	`000 | 111.11 | 1111`
		1.	`100 | 101.10 | 1001`

Tabelle 2.3: Mating Pool und Crossover

Die neue Population wird in Tabelle 2.4 bewertet.

2. Generation, F_{min} = -0,633, F_{max} = 1,333					
Nr.	Binärcode	Parameter im Dezimalsystem	Zielfunktion F	Fitneß $\Phi = F - F_{min}$	Erwartungswert E
1.	`100001.001001`	(0,150/-2,244)	-0,633	0,000	0,000
2.	`111101.101101`	(2,942/1,346)	1,173	1,806	1,595
3.	`011001.101010`	(-0,648/1,047)	0,262	0,895	0,791
4.	`111001.111001`	(2,543/2,543)	1,127	1,759	1,554
5.	`000111.111011`	(-2,443/2,743)	-0,254	0,378	0,334
6.	`111001.000001`	(2,543/-3,042)	0,464	1,097	0,969
7.	`000101.101111`	(-2,643/1,546)	0,521	1,154	1,020
8.	`100101.101001`	(0,549/0,947)	1,333	1,966	1,737
			$\overline{F}$ = 0,499	$\overline{\Phi}$ = 9,055	$\sum E = 8$

Tabelle 2.4: 2. Generation

Die nachfolgenden Generationen werden in gleicher Weise bestimmt. Bild 2.7 zeigt den Verlauf des minimalen und des maximalen Fitneßwertes, sowie den durchschnittlichen Fitneßwert einer Generation. In diesem Beispiel konvergiert die Iteration nach acht Schritten gegen das globale Optimum.

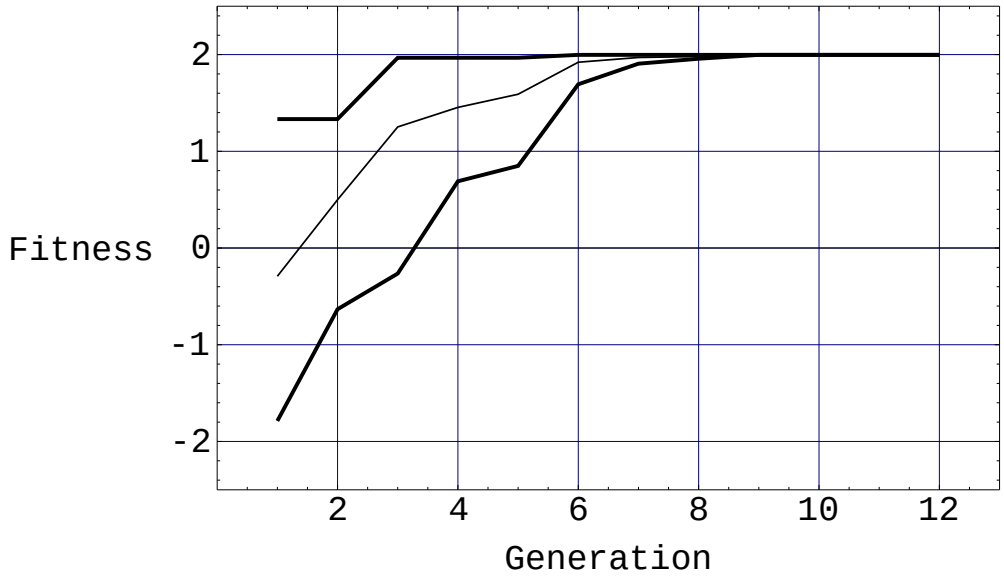


Bild 2.7: Konvergenz: Population = 8, Präzision = 1

In einem zweiten Versuch wird die Größe der Population auf vier Individuen reduziert. Wie in Bild 2.8 zu sehen, konvergiert auch diese Iteration, allerdings gegen den falschen Funktionswert. Da die Zielfunktion nur ein Optimum im Innern des zulässigen Lösungsbereichs besitzt und die Iteration gegen x = 0,91 und y = 1,11 konvergiert, kann es sich hierbei nicht um ein lokales Optimum handeln. In vielen anderen Testläufen wurde festgestellt, daß der Wert, gegen den die Iteration konvergiert, von den zufälligen Werten der Ausgangspopulation abhängt. Der korrekte Wert für das Optimum wurde von keiner durchgeführten Iteration erreicht. Das zeigt, wie wichtig die Wahl der Größe der Population ist. In der Literatur sind Werte für übliche Populationsgrößen zwischen 30 und 500 Individuen angegeben[7].

Um die Abhängigkeit der Güte des Ergebnisses von der Größe der Population qualitativ zu verdeutlichen, wurden folgende Untersuchungen durchgeführt. Es wurde die oben gegebene Funktion mit einem GA optimiert. Die Präzision wurde mit $p = 2$ vorgegeben. Der Lösungsbereich war für beide Parameter x und y auf das Intervall $[-\pi, \pi]$ beschränkt. Die Werte für x und y, gegen die die Iteration konvergierte, sind für beide Parameter in den Grafiken 2.9, 2.10 und 2.11 angegeben. Die maximale Anzahl der Iterationen betrug jeweils 20.

Bemerkenswert ist die Tatsache, daß gerade bei großen Populationen die Differenz zwischen der kleinsten und der größten Fitneß für eine lange Zeit des Optimierungsprozesses groß bleibt. Dies bedeutet, daß das Optimierungsverfahren die Suche nicht auf den Bereich lokaler Optima beschränkt. Trotzdem steigt die durchschnittliche Fitneß der

[7] siehe [14]

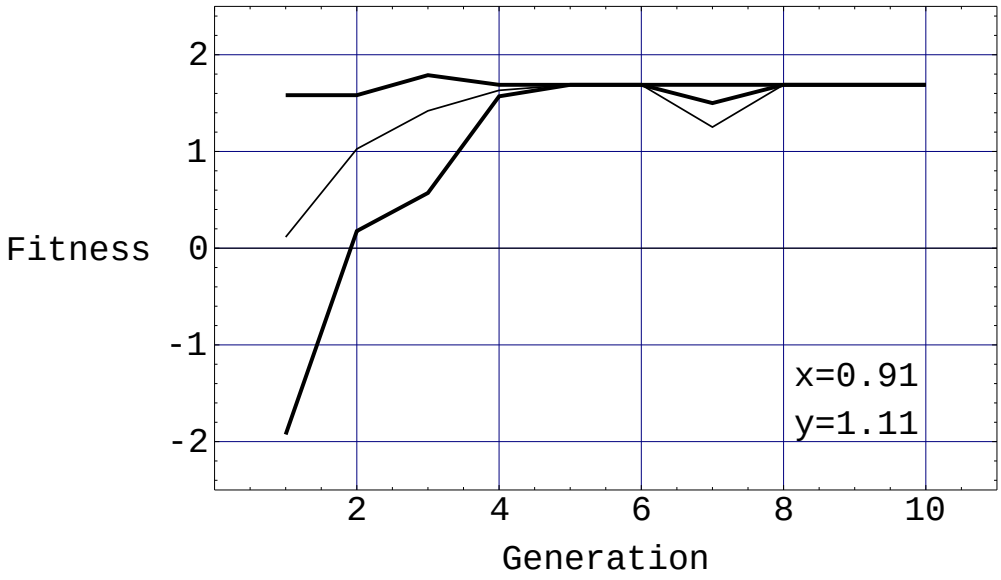


Bild 2.8: Konvergenz: Population = 4, Präzision = 1

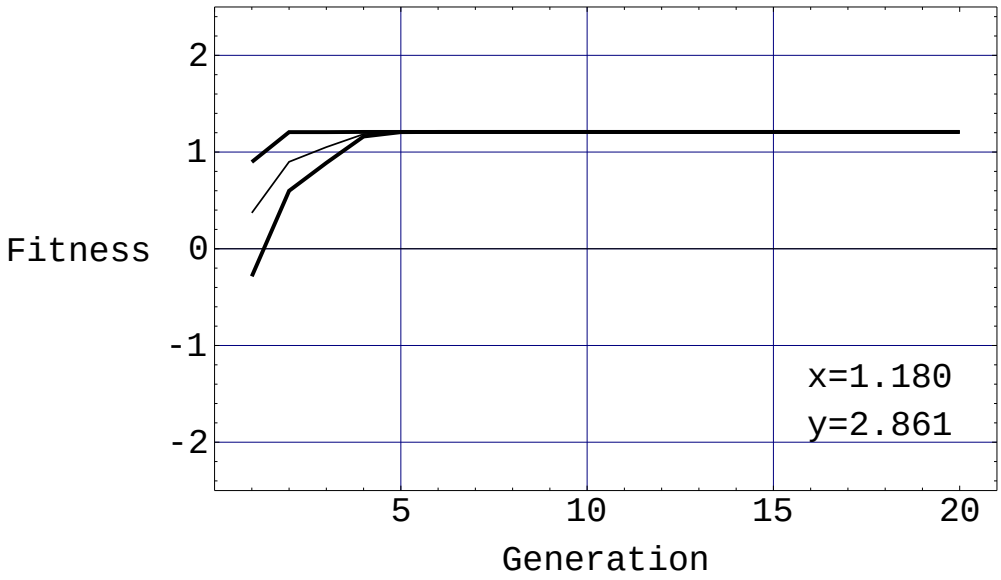


Bild 2.9: Konvergenz : Population = 4, Präzision = 2

Individuen im Laufe der Iteration an. Man erkennt daran, daß die Anzahl der guten Lösungen in der Population zunimmt. Die Wahrscheinlichkeit ein globales Optimum zu finden wird durch die breite Suche im Lösungsraum erhöht.

In Bild 2.12 ist dies deutlich zu sehen. Die Populationsgröße betrug 200, die Anzahl der Iterationen 40.

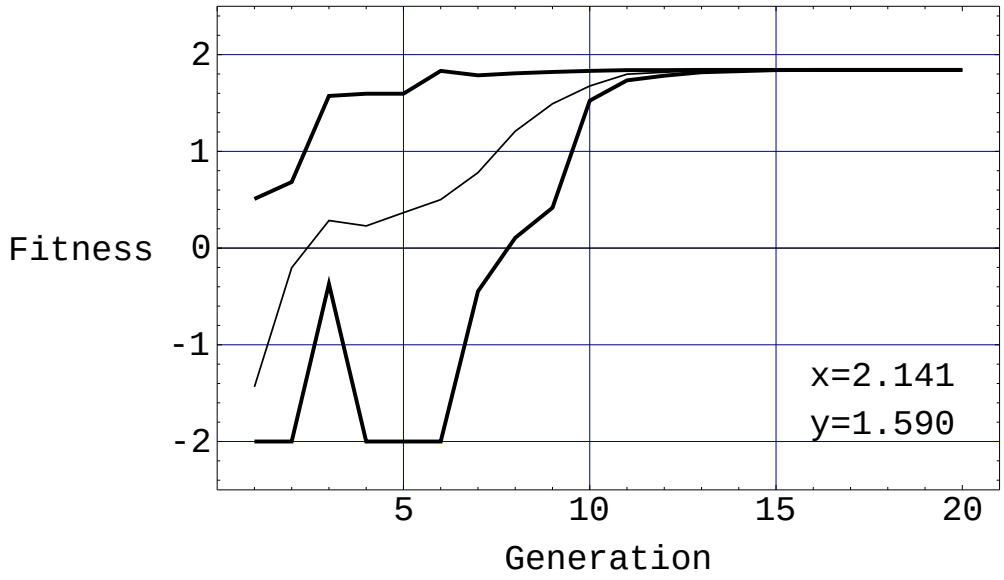


Bild 2.10: Konvergenz : Population = 10, Präzision = 2

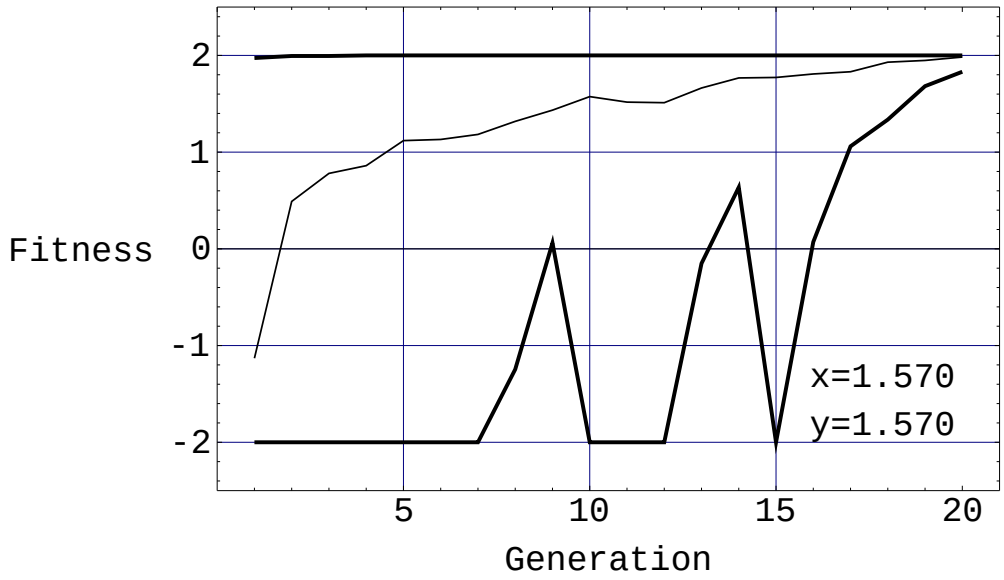


Bild 2.11: Konvergenz : Population = 100, Präzision = 2

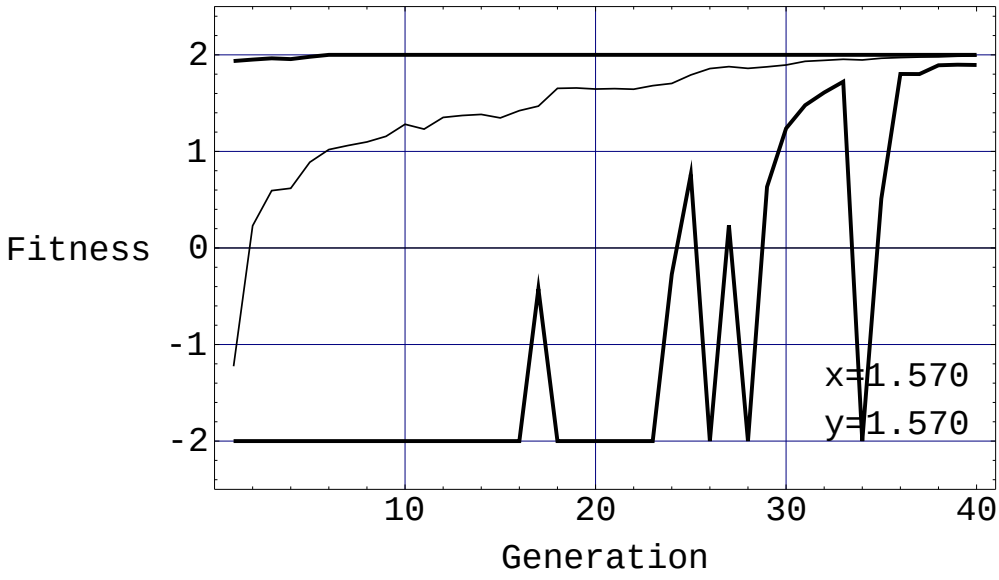


Bild 2.12: Konvergenz : Population = 200; Präzision 2

Beispiel 2

Die bisher untersuchte Funktion besitzt nur ein lokales Optimum im Innern des zulässigen Lösungsgebietes. Dieses lokale Optimum ist auch gleichzeitig das globale Optimum der Funktion. Die Funktion besitzt nur drei weitere lokale Optima, die sich auf dem Rand des zulässigen Lösungsgebietes befinden. Die nachfolgenden Optimierungen untersuchen eine Funktion mit mehreren lokalen Optima im zulässigen Lösungsbereich und nur einem globalen Optimum auf dem Rand des zulässigen Lösungsbereichs. Es soll festgestellt werden, ob das Verfahren in der Lage ist gegen dieses globale Optimum zu konvergieren.

Die folgende Funktion sei zu optimieren:

$$f(x,y) = \sin\left(x + \sin y \cdot x\right) + \frac{x+y}{8} \qquad \text{mit} \qquad \begin{matrix} x \in [-2\pi, 2\pi] \\ y \in [-2\pi, 2\pi] \end{matrix} \tag{2.26}$$

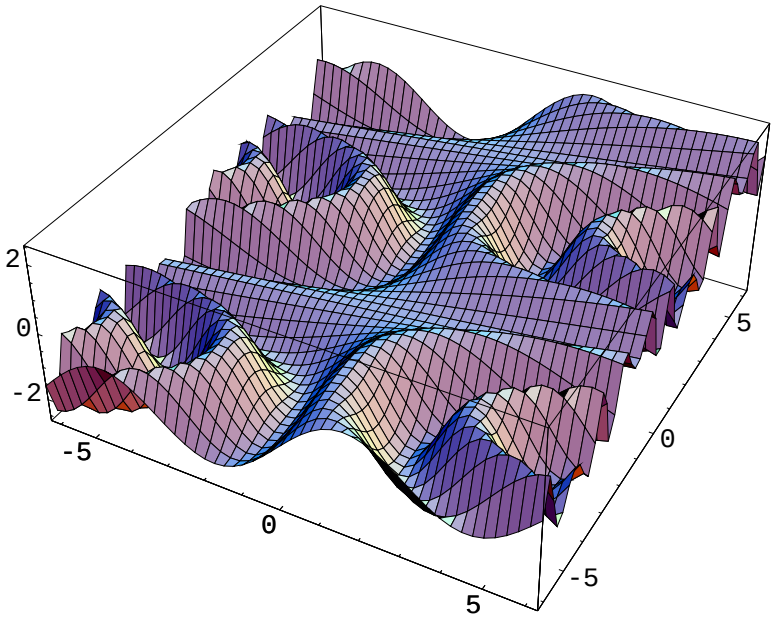


Bild 2.13: Zielfunktion Gl. 2.26

Das globale Optimum liegt bei:

$$f\left(2\pi, 2\pi - arcsin\left(\frac{3}{4}\right)\right) = f(6,283; 5,435) = 2,465. \tag{2.27}$$

Die Mutation wird diesmal berücksichtigt. Die Mutations-Wahrscheinlichkeit beträgt p_m = 0,01.

Die Iteration, die auf Bild 2.14 dargestellt ist, konvergiert gegen den Wert f(x,y) = 2,452. Die geforderte Präzision ist 1, d.h., die Iteration konvergiert gegen das globale Optimum mit gewünschter Präzision.

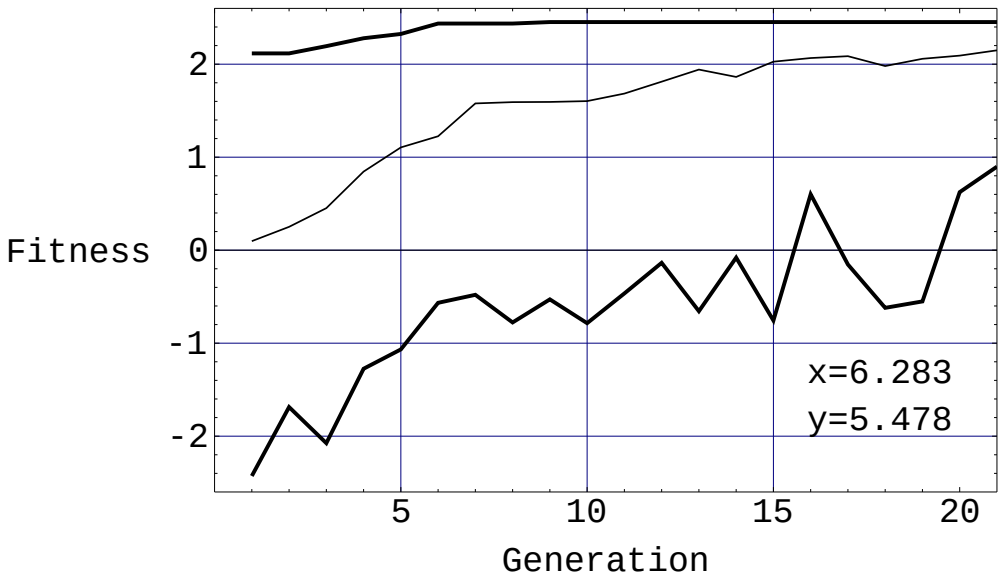


Bild 2.14: Zielfunktion Gl. 2.26 : Präzision = 1; Population = 100

Grundsätzlich ist zu bemerken, daß die vorgegebene Präzision der Zielfunktion anzupassen ist. Bei der Decodierung werden die binären Ziffernfolgen auf den reellen Wert des zugehörigen Intervallanfangs abgebildet. Deshalb können nur diejenigen Funktionswerte bei der Optimierung berücksichtigt werden, die die Zielfunktion an den Intervallgrenzen annimmt. Deshalb kann das Verfahren Optima der Zielfunktion, die innerhalb eines Intervalles liegen, nicht genau bestimmen.

Die Iteration in Bild 2.15 konvergiert gegen das globale Optimum.

Bild 2.16 zeigt eine Iteration mit nur 50 Individuen in der Population. Diese reichen jedoch für das Optimierungsverfahren aus, um gegen den korrekten Wert zu konvergieren. Allerdings erreichen beide Iterationen nicht die gewünschte Genauigkeit nach 50 bzw. 100 Iterationsschritten. Dabei ist zu berücksichtigen, wie groß der Unterschied im Wert der Zielfunktion ist. Ist nämlich der Unterschied nur sehr klein, können hier numerische Probleme auftreten, da sich der Wert der Fitneßfunktion innerhalb eines kleinen Bereichs nicht mehr ändert.

Die Beispiele zeigen, daß sowohl Populationsgröße als auch Intervallgröße von der Güte der Annäherung an das globale Optimum von Bedeutung sind. Durch eine breite Suche im zulässigen Lösungsraum ist die Wahrscheinlichkeit hoch, daß die Iteration gegen ein globales Optimum konvergiert.

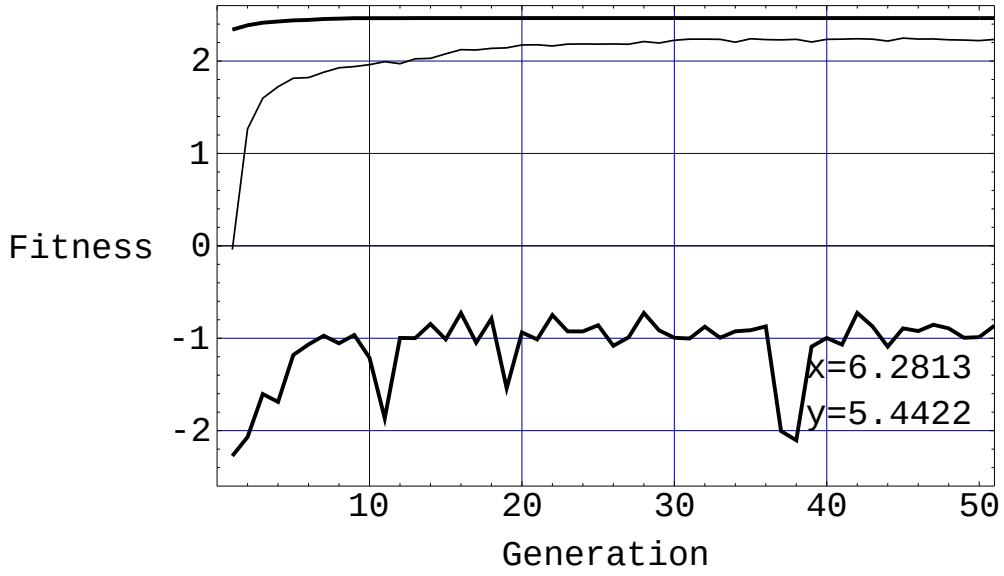


Bild 2.15: Zielfunktion 2.26 : Präzision = 5; Population = 500

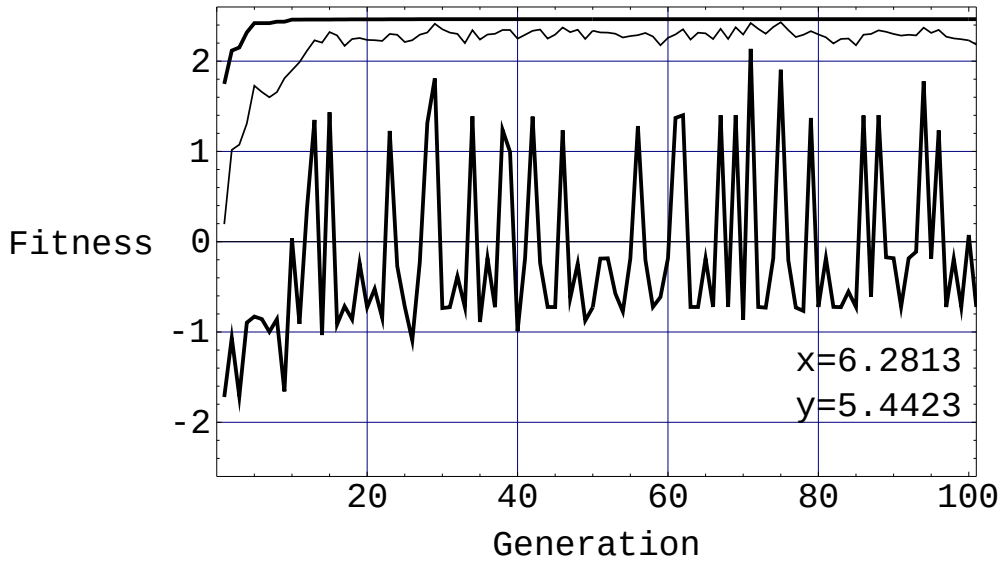


Bild 2.16: Zielfunktion 2.26 : Präzision = 5; Population = 50

2.4 Evolutionsstrategie

Die Evolutionsstrategie (ES) wurde von Rechenberg und Schwefel an der TU Berlin Mitte der 60er Jahre entwickelt. Sie sollte ein experimentelles Optimierungsverfahren für eine Vielzahl von Problemen aus den Ingenieurwissenschaften darstellen und sich an den Prinzipien der biologischen Evolution orientieren. Rechenberg vertritt die Hypothese, daß sich im Zuge der Evolution nicht nur die Arten an ihre Umgebung optimal angepaßt hätten, sondern auch die Optimierungsstrategie selbst optimiert sein müsste. Würde man also von den Prinizipien der biologischen Evolution lernen und diese auf geeignete Weise übertragen, müsste man ein universelles Optimierungsverfahren erhalten.

Rechenberg optimierte in einem seiner ersten Versuche den Luftwiderstand einer Gelenkplatte. Eine mathematische Lösung konnte für dieses Problem nicht gefunden werden. Sechs Platten waren an ihren Längskanten mit Gelenken verbunden, die einzeln verstellt werden konnten. Jedes Gelenk besaß 51 Einraststufen, die Anzahl aller möglichen Kombinationen betrug demnach 51^5 = 345 025 251 verschiedene Stellungen. Ein reines Probieren wäre also nicht in Frage gekommen.

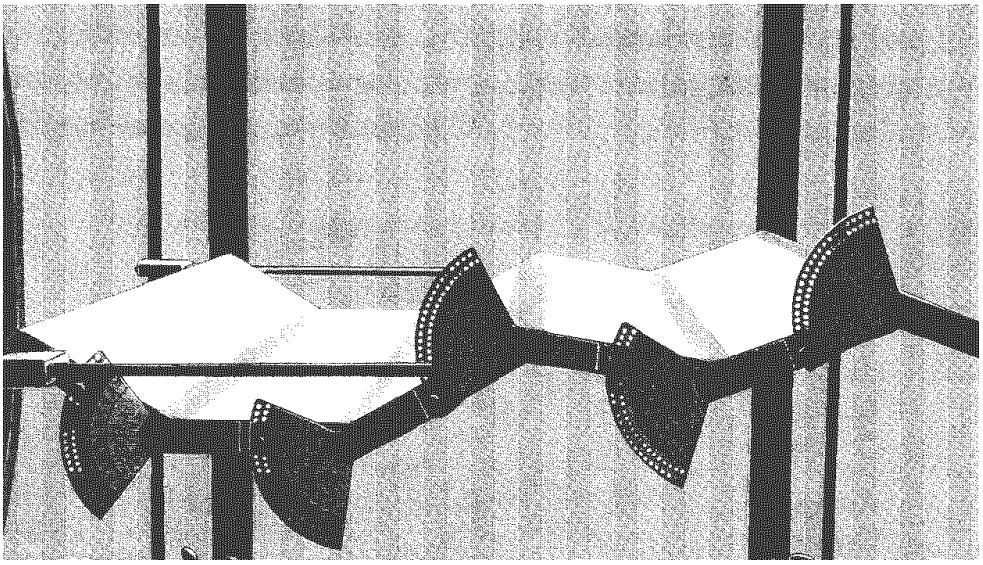


Bild 2.17: Experiment von I. Rechenberg

Gesucht war diejenige Stellung der Gelenke, für die der Luftwiderstand minimal wurde. Auf Bild 2.17 ist die Versuchsanordnung zu sehen. Rechenberg verwendete hierbei eine (1+1)-ES. Obwohl dies, nach heutiger Meinung, keine besonders effiziente Strategie darstellt, hatte Rechenberg die optimale Form schon nach ca. 320 Generationen gefunden.

Die ES ist von vornherein als Optimierungsverfahren für technische Systeme entworfen worden. Die Individuen der ES bestehen nicht nur aus den Parametern der Zielfunktion, sondern auch aus Strategieparametern, die im Laufe der Optimierung angepaßt werden. Dadurch besitzt die ES die Möglichkeit, nicht nur die Entwurfsvariablen zu optimieren, sondern auch die Optimierungstrategie. Anders als bei den GA werden die

Parameter nicht binär codiert, d.h. die ES arbeitet mit dem Phänotyp eines Individuums.

2.4.1 Grundform der Evolutionsstrategie

Im folgenden wird die für ES übliche Notation verwendet. Diese orientiert sich an der aus der Theoretischen Informatik bekannten Notation für deterministische, endliche Automaten.

Danach sei eine Evolutionsstrategie (ES) beschrieben durch ein 9-Tupel (in Anlehnung an [22]):

$$ES = (I, \Phi, m_{\Theta_m}, r_{\Theta_r}, s_{\Theta_s}, \Psi, \iota, \mu, \lambda), \tag{2.28}$$

mit:

- I gleich der Menge der Individuen $\vec{a}$ der Population.
- Φ gleich der Fitneßfunktion.
- $m_{\Theta_m} : I^\lambda \mapsto I^\lambda$ gleich dem stochastischen Mutationsoperator.
  Dieser wird durch die Menge $\Theta_m = \{\tau_1, \tau_2\}$ gesteuert[8].
- $r_{\Theta_r} : I^\mu \mapsto I^\lambda$ gleich dem Rekombinationsoperator.
  Dieser wird durch die Menge $\Theta_r = \{r_x, r_\sigma\}$ gesteuert.
- $s_{\Theta_s} : I^\lambda \cup I^{\lambda+\mu} \mapsto I^\mu$ gleich dem Selektionsoperator,
  mit der Menge $\Theta_s = \{\mu, \lambda, a\}, a \in \{« + »; «, »\}$.
- $\Psi : I^\mu \mapsto I^\mu$ Generationsübergangsfunktion. Ψ beschreibt den Übergang von einer Population $P(t)$ zur nächsten Population $P(t+1)$ und faßt damit die Schritte Rekombination, Mutation und Selektion zusammen:

$$\begin{aligned} \Psi\left(P(t)\right) &= s_{\Theta_s}\left(Q \cup m_{\{\tau_1,\tau_2\}}\left(r_{\{r_x,r_\sigma\}}(P(t))\right)\right), \\ & \begin{cases} \text{mit} \quad Q = \{\}, & \text{wenn} \quad \Theta_s = \{\mu, \lambda, «, »\}, \\ \text{mit} \quad Q = P(t), & \text{wenn} \quad \Theta_s = \{\mu, \lambda, « + »\}, \end{cases} \end{aligned} \tag{2.29}$$

  t ist dabei ein Generationszähler.
- ι ist ein Abbruchkriterium.
- $\mu \in \mathbb{N}$ gleich der Anzahl der Eltern bzw. der Größe der Population.
- $\lambda \in \mathbb{N}$ gleich der Anzahl der Nachkommen.

Die wichtigsten Parameter sind in den folgenden Abschnitten ausführlich erläutert.

[8] Korrelierte Mutation wird in dieser Arbeit nicht betrachtet. Deshalb sind die Rotationswinkel α_i, im Gegensatz zu den meisten Abhandlungen in der Literatur, nicht in der Menge Θ_m enthalten.

2.4.2 Individuen

Im Gegensatz zu GA arbeiten ES nicht mit dem Genotyp eines Individuums, sondern mit dem Phänotyp, das heißt mit den Parametern der Zielfunktion in reellen Zahlen. Deshalb besteht nicht die Notwendigkeit die Parameter der Zielfunktion binär zu codieren.

Ein Individuum besteht aus einem Vektor reeller Zahlen, die die Parameter der Zielfunktion darstellen, und einem Vektor $\vec{\sigma}$ mit Standardabweichungen, die den Optimierungsvorgang steuern.

$$a = (\vec{x}, \vec{\sigma}) \quad \text{mit} \quad \begin{cases} \vec{x} \in \mathbb{R}^n & \text{mit} \quad n \in \mathbb{N}, \\ \vec{\sigma} \in \mathbb{R}^{n_\sigma} & \text{mit} \quad (1 \leq n_\sigma \leq n), \end{cases} \tag{2.30}$$

$\vec{\sigma}$ ist ein Vektor mit n_σ Standardabweichungen. Der Wert für n_σ liegt üblicherweise bei 1 oder n. Liegt der Wert dazwischen, so werden die Parameter x_i bis zum Index n_σ mit den zugehörigen Werten σ_i assoziiert, für alle übrigen Parameter wird σ_{n_σ} verwendet.

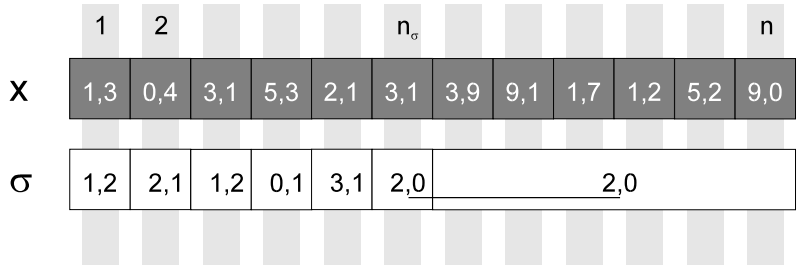


Bild 2.18: Evolutionsstrategie: Individuum

Die Werte $\vec{x}_i$ der Ausgangspopulation werden zufällig erzeugt. Für die Startwerte von σ_i wird in [14] eine Empfehlung von 3,0 und $n_\sigma = n$ Standardabweichungen gegeben.

2.4.3 Fitneßfunktion

Da die Decodierung nicht notwendig ist, ist die Fitneßfunktion üblicherweise gleich der Zielfunktion des Optimierungsproblems:

$$\Phi(\vec{a}) = f(\vec{x}). \tag{2.31}$$

2.4.4 Rekombination

Aus μ Eltern werden λ Nachkommen erzeugt. Dazu werden zufällig, gleichverteilt und mit Zurücklegen λ Elternpaare gezogen. Aus diesen wird je ein Nachkomme durch Rekombination erzeugt. Dazu dient der Rekombinationsoperator r_{Θ_r}:

$$r_{\Theta_r} : I^\mu \mapsto I^\lambda. \tag{2.32}$$

Die Menge $\Theta_r = \{r_x, r_\sigma\}$ legt fest, mit welcher Methode die Eltern zu rekombinieren sind. r_x steht dabei für die Rekombinationsvorschrift für die Entwurfsvariablen x_i, r_σ legt die Rekombinationsmethode für die Standardabweichungen σ_i fest.

Standardwerte für die Größe der Population und Anzahl der Nachkommen sind μ = 15 und λ = 100.

Man unterscheidet zwischen diskreter und intermediärer Rekombination. Aus der Literatur ist die Empfehlung zu entnehmen, für $\vec{x}$ und $\vec{\sigma}$ verschiedene Rekombinationsstrategien zu verwenden.

2.4.4.1 Diskrete Rekombination

In Abhängigkeit von einer gleichverteilten Zufallsgröße wird entweder der Wert des einen oder der Wert des anderen Elters gewählt und an den Nachkommen vererbt.

$$\begin{aligned} x_{K,i} &= x_{E1,i} \vee x_{E2,i}, \\ \sigma_{K,i} &= \sigma_{E1,i} \vee \sigma_{E2,i}. \end{aligned} \tag{2.33}$$

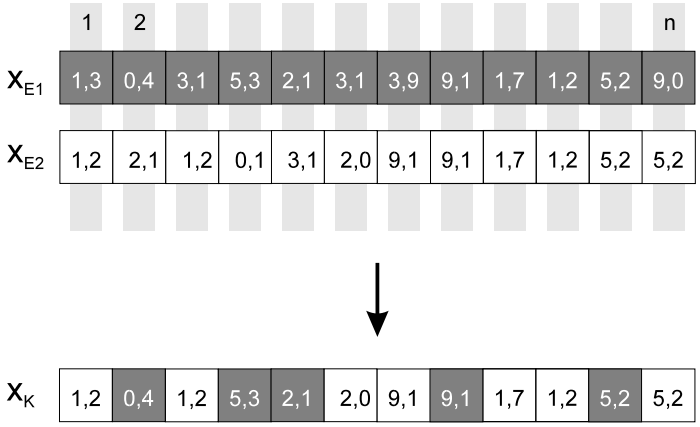


Bild 2.19: Diskrete Rekombination

2.4.4.2 Intermediäre Rekombination

Die Werte x_i, σ_i des Nachkommens ergeben sich aus dem Mittelwert der entsprechenden Werte der Eltern.

$$\begin{aligned} x_{K,i} &= \frac{1}{2} \cdot (x_{E1,i} + x_{E2,i}), \\ \sigma_{K,i} &= \frac{1}{2} \cdot (\sigma_{E1,i} + \sigma_{E2,i}). \end{aligned} \tag{2.34}$$

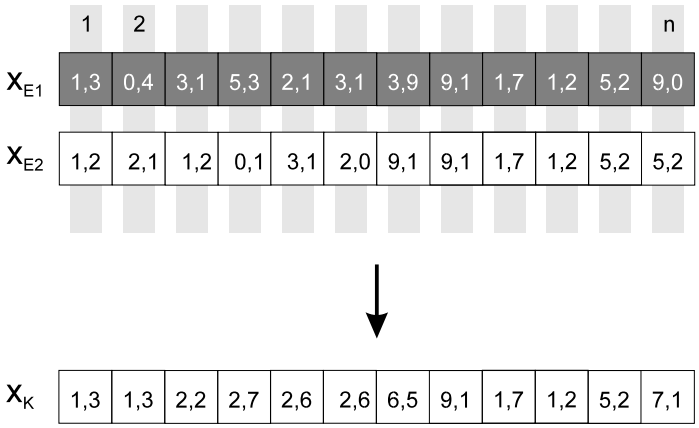


Bild 2.20: Intermediäre Rekombination

Es ist auch möglich, zwischen beiden Werten linear zu interpolieren, wobei für λ eine gleichverteilte Zufallsgröße zwischen 0 und 1 erzeugt wird.

$$\begin{aligned} x_{K,i} &= \lambda x_{E1,i} + (1-\lambda) x_{E2,i}, \\ \sigma_{K,i} &= \lambda \sigma_{E1,i} + (1-\lambda) \sigma_{E2,i}, \qquad \text{mit} \quad \lambda \in [0,1]. \end{aligned} \tag{2.35}$$

2.4.5 Mutation

Für die ES hat die Mutation eine größere Bedeutung als für die GA. Da die Mutationswahrscheinlichkeit p_m bei den GA gewöhnlich sehr klein gewählt wird, kommt es nur vereinzelt zu Mutationen. Bei der ES dagegen ist die Mutation die wichtigste Strategie, um Veränderungen im Erbgut zu bewirken. Darüber hinaus wird die Mutation für jeden Parameter einzeln gezielt gesteuert und angepaßt. Die Mutationsschrittweiten werden, wie die Parameter der Zielfunktion, dem Optimierungsprozeß unterworfen.

Nach dem Erzeugen neuer Nachkommen durch Rekombination werden diese mutiert:

$$m_{\Theta_m} : I^\lambda \mapsto I^\lambda, \tag{2.36}$$

mit der Menge der Parameter $\Theta_m = \{\tau_1, \tau_2\}$. Die Standardabweichungen σ_i werden durch Multiplikation mit einer logarithmisch-normalverteilten Zufallsgröße mutiert:

$$\sigma_i' = \sigma_i \cdot e^{\tau_1 N(0,1) + \tau_2 N_j(0,1)}. \tag{2.37}$$

$N(0,1)$ ist eine für ein Individuum einmal zu erzeugende standardnormalverteilte Zufallsgröße mit dem Erwartungswert 0 und der Standardabweichung 1. $N_j(0,1)$ ist für jeden Index neu zu erzeugen. Für die Strategiewerte τ_1 und τ_2 werden in der Literatur folgende Werte empfohlen:

$$\tau_1 = \frac{1}{\sqrt{2 \cdot n}}; \qquad \tau_2 = \frac{1}{\sqrt{2 \cdot \sqrt{n}}}, \tag{2.38}$$

mit n gleich der Anzahl der Parameter der Optimierungsaufgabe.

Der mutierte Wert für die Entwurfsvariable ergibt sich zu:

$$x_i' = x_i + \sigma_i' \cdot N_i(0,1). \tag{2.39}$$

Im Laufe des Suchprozesses wird die Mutationsschrittweite an das Optimierungsproblem angepaßt. Dadurch können Ergebnisse hoher Genauigkeit erreicht werden. GA waren durch die vorgegebene Präzision, und das damit vorgegebene Raster von Intervallen, auf eine bestimmte Genauigkeit beschränkt.

Erzeugung von normalverteilten Zufallsgrößen

Normalverteilte Zufallsgrößen N(0,1) mit dem Erwartungwert 0 und der Standardabweichung 1 können wie folgt bestimmt werden (nach [14]):

$$N = \sqrt{-2 \ln U_1} \cdot \sin 2\pi U_2. \tag{2.40}$$

U_1 und U_2 sind gleichverteilte Zufallszahlen im Intervall]0,1[. Zufallszahlen mit einem beliebigen Erwartungswert E und einer Standardabweichung σ lassen sich erzeugen durch folgende Transformation:

$$N(E,\sigma) = E + \sigma \cdot N(0,1). \tag{2.41}$$

2.4.6 Selektion

Im Vergleich der ES mit GA fällt auf, daß die Selektion für beide Methoden eine andere Bedeutung hat. Bei den GA hat die Selektion die Aufgabe, diejenigen Chromosomen auszuwählen, die bei der Rekombination Nachkommen für die nächste Generation erzeugen können. Bei der ES werden mutierte Individuen durch die Selektion deterministisch gewählt, die direkt in die nächste Generation übernommen werden.

Es werden aus den λ erzeugten Nachkommen die μ besten Individuen gewählt, die direkt in die nächste Generation übernommen werden. Der Selektionsoperator mit der Menge $\Theta_s = \{\mu, \lambda, a\} \quad \text{mit} \quad a \in \{«, », « + »\}$ bestimmt die Selektionsstrategie.

Es wird zwischen Plus- und Komma-Strategie unterschieden.

Plus-Strategie:

$$s_{\{\mu,\lambda,«+»\}} : I^{\mu+\lambda} \mapsto I^{\mu} \tag{2.42}$$

Zur Selektion kommen die Nachkommen und die Individuen der alten Population. Aus der Vereinigung beider Mengen werden die μ besten Individuen in die nächste Generation übernommen.

Komma-Strategie:

$$s_{\{\mu,\lambda,«,»\}} : I^{\lambda} \mapsto I^{\mu} \tag{2.43}$$

Nur die Nachkommen kommen zur Selektion. Die μ besten Individuen werden in die nächste Generation übernommen.

Eine ES, die auf der Plus-Selektion beruht, wird als $(\mu + \lambda)$-ES bezeichnet. Entsprechend wird als (μ, λ)-ES diejenige ES bezeichnet, die eine Komma-Strategie verwendet.

Über das Selektionsverfahren kann man beeinflussen, ob die bisher beste Lösung auf jeden Fall in den nachfolgenden Population enthalten bleiben soll. Dies erhöht allerdings die Wahrscheinlichkeit, daß der Optimierungsprozeß gegen ein lokales Optimum konvergiert. Wählt man statt der Plus- eine Komma-Strategie, verbreitert man zwar die Suche im Lösungsraum, aber es kann vorkommen, daß man nach der letzten Iteration nicht das beste Individuum aller Generationen erhält. Um diesen Konflikt zu lösen, verwendet man gewöhnlich eine Komma-Strategie, merkt sich aber gesondert das bislang beste Individuum. Auf diese Weise erhält man das während des Itertionsprozesses beste gefundene Individuum, ohne den Lösungsraum auf einen Subraum einzuschränken.

2.4.7 Abbruchkriterium

Um entscheiden zu können, wann eine Iteration abgebrochen werden darf, wird ein Abbruchkriterium ι definiert. Die Iteration wird abgebrochen, wenn ι wahr ist.

Die Wahl des Abbruchkriteriums beeinflußt direkt das Laufzeitverhalten der Optimierung. Es besteht ein Konflikt zwischer kurzer Rechenzeit und dem Bestreben ein globales Optimum zu finden. Bricht man den Optimierungsprozeß zu früh ab, besteht die Möglichkeit keine oder nur eine suboptimale Lösung gefunden zu haben. Andererseits hat man nur beschränkt Ressourcen und Zeit zur Verfügung und ist deshalb bestrebt, den Optimierungsprozeß nach möglichst wenig Iterationen zu beenden.

Üblicherweise beendet man den Optimierungsprozeß nach einer maximalen Anzahl von Iterationen. Dies berücksichtigt allerdings nicht die Qualität der bislang gefundenen Lösungen und ist deshalb nur als obere Begrenzung der Anzahl der Iterationen empfehlenswert.

Günstiger ist ein Abbruchkriterium, das sich an der Güte der Individuen orientiert:

Φ_I sei das bestes Individuum, Φ_{II} das schlechteste Individuum in der aktuellen Population. Das Abbruchkriterium wird auf Grundlage der Differenz der Fitneßwerte des besten und des schlechtesten Individuums gewählt. Die Iteration wird dann beendet, wenn die Differenz Φ_I - Φ_{II} kleiner als eine bestimmte Schranke c_i wird. Unter der Voraussetzung, daß gute Fitneßwerte größer sind als schlechte, kann diese Schranke absolut definiert sein als (in Anlehnung an [14]):

$$\iota = (\Phi_I - \Phi_{II} \leq c_1), \tag{2.44}$$

oder relativ zum Durschnittswert der Fitneß aller Individuen in der aktuellen Population:

$$\iota = (\Phi_I - \Phi_{II} \leq \frac{c_2}{\mu} \sum_{k=1}^{\mu} \Phi(\vec{x}_k)). \tag{2.45}$$

Es hat sich jedoch gezeigt, daß diese Kriterien nicht immer zuverlässig funktionieren. Im 2. Beispiel dieses Abschnitts kann man sehen, daß der schlechteste Fitneßwert einer Population bis zum Ende der Iteration konstant bleibt. Die Iteration würde bei Verwendung der oben genannten Abbruchkriterien zu spät beendet werden.

Als zweckmäßiges Abbruchkriterium hat sich der Wert für die mittlere Standardabweichung $\overline{\sigma}$ aller Individuen der Population herausgestellt. Die mittlere Standardabweichung konvergiert gegen Null. Fällt $\overline{\sigma}$ unter eine zu bestimmende Schranke ϵ, wird die Iteration abgebrochen.

$$\iota = (\overline{\sigma} < \epsilon). \tag{2.46}$$

2.4.8 Basis-Algorithmus

Der Ablauf der Optimierung ist dem von GA sehr ähnlich:

Es sei eine Funktion $f(\vec{x})$ mit n reellen Entwurfsvariablen x_i zu optimieren. Für jede dieser Variablen wird ein Satz von Strategieparametern bereitgestellt.

1. Wähle Strategieparameter μ, λ, {«+»,«,»}, n_σ, r_σ, r_x, τ_1, τ_2.

2. Initialisiere Ausgangspopulation. Die Ausgangspopulation wird stochastisch erzeugt und sollte möglichst gleichmäßig über den Lösungsraum verteilt sein. Dadurch wird die Wahrscheinlichkeit ein globales Optimum zu finden erhöht.

3. Bewerte Ausgangpopulation durch Bestimmen der Fitneßwerte für jedes Individuum.

4. Erzeuge Nachkommen. Aus μ Eltern werden λ Nachkommen erzeugt. Dazu werden jeweils zwei Individuen aus der Population mit Zurücklegen gezogen. Im Unterschied zu den GA ist die Wahrscheinlichkeit hierbei für alle Individuen in der Population gleich. Aus diesen wird durch Rekombination ein Nachkomme erzeugt.

5. Mutiere Nachkommen.

6. Selektiere die μ besten Individuen unter den λ Nachkommen.

7. Überprüfe Abbruchkriterium. Wenn dieses nicht erüllt ist, dann setze Prozeß mit Punkt 4 fort.

Aufgrund der Darstellung der Parameter der Zielfunktion als reelle Zahlen bietet sich die ES für die Optimierung von Systemen mit kontinuierlichen Entwurfsvariablen an.

2.4.9 Beispiele für die Optimierung mit ES

Beispiel 1

Es sei die Zielfunktion $f(x, y)$ des Beispiels 1 aus Abschnitt 2.3.10 mit einer Evolutionsstrategie zu optimieren:

$$f(x, y) = \sin x + \sin y \qquad \begin{matrix} x \in [-\pi, \pi] \\ y \in [-\pi, \pi] \end{matrix} \tag{2.47}$$

Das globale Optimum liegt bei $f(x = \frac{\pi}{2} = 1,5708, y = \frac{\pi}{2} = 1,5708) = 2$.

Es werden folgende Strategieparameter gewählt:

- $\Theta_r = \{r_x, r_\sigma\} = \{diskret, diskret\}$.
- $\Theta_m = \{\tau_1, \tau_2\} = \{\frac{1}{\sqrt{2 \cdot n}}, \frac{1}{\sqrt{2 \cdot \sqrt{n}}}\}$.
- $\Theta_s = \{5, 15, \text{« + »}\}$.
- $n_\sigma = n_x = 2$.

Die Ausgangspopulation wird zufällig erzeugt. Die Standardabweichung wird einheitlich für alle Parameter auf 3,0 gesetzt.

1. Generation, F_{min} = -0,6507, F_{max} = -0,0091			
Nr.	Parameter (x,y)	Standardabweichung σ_1/σ_2	Fitneß
1.	(-0,5382, 0,5276)	3,0000 / 3,0000	-0,0091
2.	(1,1946, -1,3033)	3,0000 / 3,0000	-0,0343
3.	(0,3928, -2,4543)	3,0000 / 3,0000	-0,2517
4.	(-2,3541, 0,2803)	3,0000 / 3,0000	-0,4320
5.	(-2,0465, 2,9010)	3,0000 / 3,0000	-0,6507
		$\overline{\sigma}$ = 3,00	$\overline{F}$ = -0,2756

Tabelle 2.5: Ausgangspopulation, (5+15)-ES

Durch Rekombination werden von den 5 Individuen 15 Nachkommen erzeugt. Die Nachkommen werden mutiert. Es wird eine Plus-Strategie verfolgt, das heißt, zur Selektion kommen sowohl die Individuen der Population als auch deren Nachkommen.

Individuen, die nicht im zulässigen Lösungsbereich liegen, erhalten willkürlich den Fitneßwert F = -2,6 zugeteilt. Es wird darauf verzichtet, den Individuen, die die Restriktion des eingeschränkten Lösungsbereichs verletzen, den Wert `Fitness::notvalid` zuzuordnen. Die unzulässigen Individuen werden somit nicht automatisch aus der Population gelöscht. Eine Reihe von durchgeführten Optimierungen hat gezeigt, daß es für dieses Beispiel unerheblich ist, ob unzulässige Individuen gelöscht werden oder nicht. Mit beiden Varianten konnte das globale Optimum gefunden werden.

Bei der Optimierung von Knotenkoordinaten hingegen stellte sich heraus, daß es zweckmäßig ist, die unzulässigen Individuen in der Population zu belassen, aber mit einem Strafwert zu belegen. Bei der Optimierung von Knotenkoordinaten ist es sehr wahrscheinlich, daß sich nur durch kleine Änderungen an der Geometrie weniger Knoten ein unzulässiges Netz ergibt. Es hat sich im Rahmen von Beispieloptimierungen gezeigt, daß die Iteration nicht konvergierte, wenn unzulässige Individuen aus der Population gelöscht wurden.

1. Generation, F_{min} = -2,6, F_{max} = -0,0091			
Nr.	Parameter (x,y)	Standardabweichung σ_1/σ_2	Fitneß
1.	(-0,5382, 0,5276)	3,0000 / 3,0000	-0,0091
2.	(1,1946, -1,3033)	3,0000 / 3,0000	-0,0343
3.	(0,3928, -2,4543)	3,0000 / 3,0000	-0,2517
4.	(-2,3541, 0,2803)	3,0000 / 3,0000	-0,4320
5.	(-2,0465, 2,9010)	3,0000 / 3,0000	-0,6507
6.	(0,1049, -1,6300)	2,4812 / 2,4812	-0,8936
7.	(-0,2276, -2,0788)	2,3927 / 2,3927	-1,0994
8.	(-1,5473, -0,6592)	2,5113 / 2,5113	-1,6122
9.	(-1,7028, -2,1689)	1,1241 / 1,1241	-1,8177
10.	(0,0139, 4,3666)	3,5736 / 3,5736	-2,6000
11.	(6,5505, -6,9609)	6,7854 / 6,7854	-2,6000
12.	(6,8980, 8,7227)	4,5368 / 4,5368	-2,6000
13.	(11,2172, 7,5588)	10,6546 / 10,6546	-2,6000
14.	(1,9796, -15,2041)	9,5898 / 9,5898	-2,6000
15.	(0,6447, 12,3392)	4,0315 / 4,0315	-2,6000
16.	(4,4500, -3,1486)	2,5265 / 2,5265	-2,6000
17.	(3,8238, -3,0375)	3,0310 / 3,0310	-2,6000
18.	(0,6934, 6,1774)	3,7805 / 3,7805	-2,6000
19.	(1,1148, 3,2785)	0,8501 / 0,8501	-2,6000
20.	(3,5519, 0,4252)	1,5910 / 1,5910	-2,6000
		$\overline{\sigma}$ = 3,723	$\overline{F}$ = -1,7700

Tabelle 2.6: Nachkommen + Population, (5+15)-ES

In gleicher Weise werden die nachfolgenden Generationen bestimmt. Nach 20 Iterationen ergeben sich die in Tabelle 2.7 abgebildeten Werte. Man erkennt deutlich, daß die Werte für die Standardabweichungen dort groß sind, wo die Lösungen weit vom Optimum entfernt sind. Nähert sich die Lösung dem Optimum, dann werden auch die σ-Werte kleiner. Dadurch paßt sich das Verfahren der Fitneßfunktion an und es können hohe Genauigkeiten erzielt werden.

Die folgenden Bilder zeigen Optimierungen mit verschiedenen Populationsgrößen und Nachkommenszahlen. Es werden die Werte für die kleinste Fitneß, die größte Fitneß und die durchschnittliche Fitneß der Population mit durchgezogener Linie geplotet. Der Bereich zwischen der maximalen und der minimalen Standardabweichung wird grau hinterlegt gezeichnet. Der Durchschnittswert für die Standardabweichungen in einer Generation ist als gestrichelte Linie gezeigt.

20. Generation, F_{min} = 1,9482, F_{max} = 2,000			
Nr.	Parameter (x,y)	Standardabweichung σ_1/σ_2	Fitneß
1.	(1.5751, 1.5667)	0.0254 / 0.0316	2.0000
2.	(1.5614, 1.5688	0.0086 / 0.0106	2.0000
3.	(1.5608, 1.5740	0.0332 / 0.0062	1.9999
4.	(1.5585, 1.5716	0.0677 / 0.0842	1.9999
5.	(1.5591, 1.5749	0.0531 / 0.0661	1.9999
6.	(1.5629, 1.5809	0.0401 / 0.0499	1.9999
7.	(1.5584, 1.5673	0.0195 / 0.0242	1.9999
8.	(1.5620, 1.5577	0.0177 / 0.0220	1.9999
9.	(1.5578, 1.5606	0.0060 / 0.0075	1.9999
10.	(1.5539, 1.5686	0.0170 / 0.0212	1.9999
11.	(1.5558, 1.5587	0.0022 / 0.0081	1.9998
12.	(1.5549, 1.5582	0.0040 / 0.0050	1.9998
13.	(1.5718, 1.6050	0.0644 / 0.0802	1.9994
14.	(1.5265, 1.5569	0.0369 / 0.0458	1.9989
15.	(1.5551, 1.5153	0.0067 / 0.0933	1.9983
16.	(1.5616, 1.6372	0.0254 / 0.0316	1.9978
17.	(1.4982, 1.5665	0.0549 / 0.0077	1.9974
18.	(1.5378, 1.4479	0.1225 / 0.1524	1.9919
19.	(1.3669, 1.6584	0.0783 / 0.0974	1.9755
20.	(1.2798, 1.4311	0.1632 / 0.1203	1.9482
		$\overline{\sigma}$ = 0,044	$\overline{F}$ = 1,9953

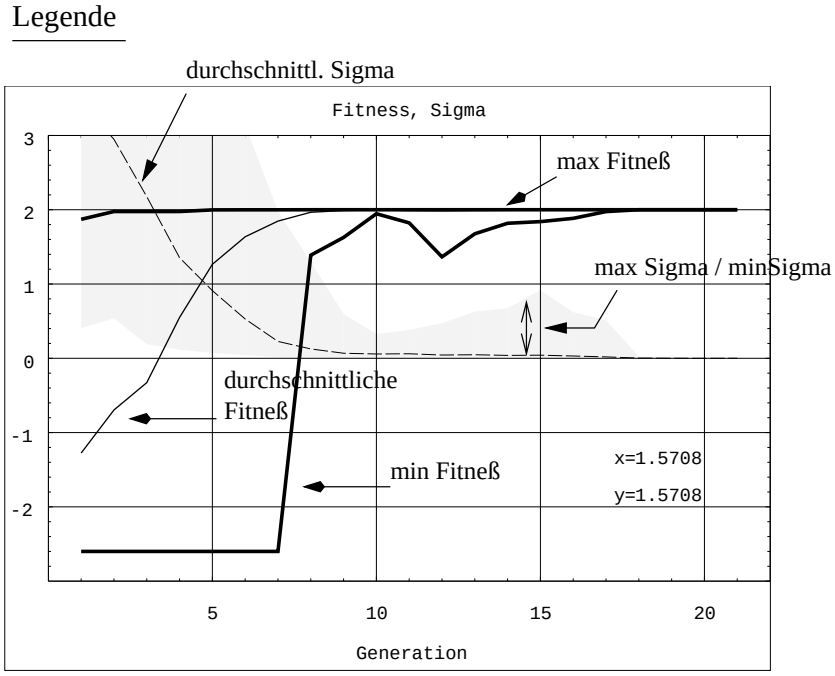
Tabelle 2.7: 20. Generation, Nachkommen + Population, (5+15)-ES

Man kann erkennen, daß die Werte für die Standardabweichungen gegen Null konvergieren. Aus diesen Beispielen läßt sich entnehmen, daß die mittlere Standardabweichung und die Differenz zwischen dem größten und dem kleinsten Fitneßwert ungefähr nach der gleichen Anzahl von Iterationen unter eine Schranke ϵ fallen. Daraus könnte man schließen, daß für beide Abbruchkriterien der Optimierungsprozeß ungefähr nach der gleichen Anzahl von Iterationen beendet werden würde. Es hat sich jedoch gezeigt, daß dies nicht immer Fall ist. Mit zunehmender Anzahl der Individuen steigt die Wahrscheinlichkeit einzelne Individuen zu erzeugen, die außerhalb des zulässigen Lösungsbereichs liegen. Im Beispiel 2. liegt das globale Optimum auf dem Rand des Lösungsbereichs. Die Wahrscheinlichkeit ist hier sehr hoch, sodaß in jedem Iterationsschritt mindestens ein Individuum erzeugt wird, das nicht im zulässigen Lösungsbereich liegt. Ein Abbruchkriterium, das sich an der mittleren Standardabweichung orientiert, hat sich hier besser bewährt[9]

Auf den nachfolgenden Diagrammen kann man erkennen, wie die Konvergenzrate mit steigender Nachkommenanzahl zunimmt. Dies ist auch zu erwarten, da mit zunehmen-

[9]Das Problem wird in diesem Beispiel noch dadurch verschärft, daß Individuen, die außerhalb des zulässigen Lösungsbereichs liegen, ein konstanter Fitneßwert zugewiesen wird. Das Verfahren ist dadurch nicht in der Lage, in den zulässigen Bereich zurückzuwandern, da die Selbstadaptivität hierdurch erschwert wird. Deshalb ist zu empfehlen, den Wert der Straffunktion von der Entfernung zum zulässigen Lösungsbereich abhängig zu machen (s. auch Abschnitt 2.3.8.2).

der Anzahl der Nachkommen die Zahl der möglichen Kombinationen steigt und damit auch die Wahrscheinlichkeit, Individuen mit guter Fitneß zu erzeugen.



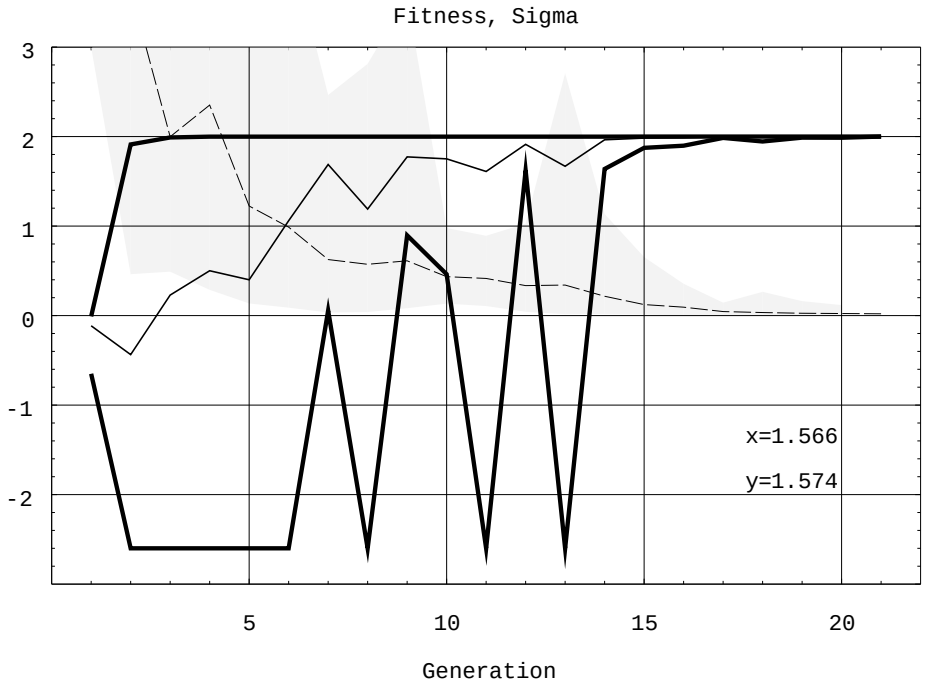


Bild 2.21: Konvergenz: (5+15)-ES

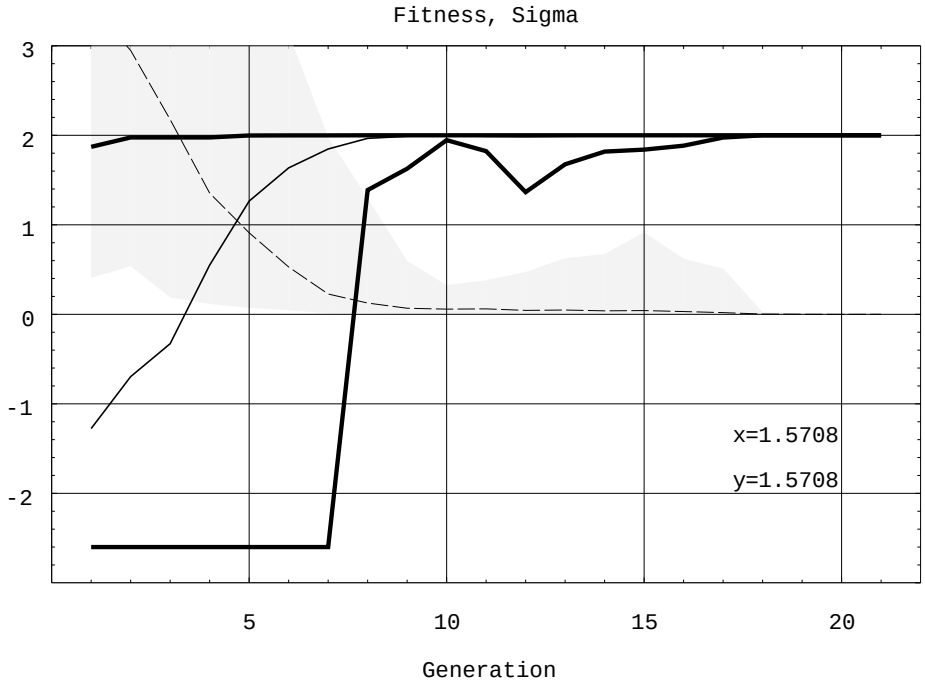


Bild 2.22: Konvergenz: (10+50)-ES

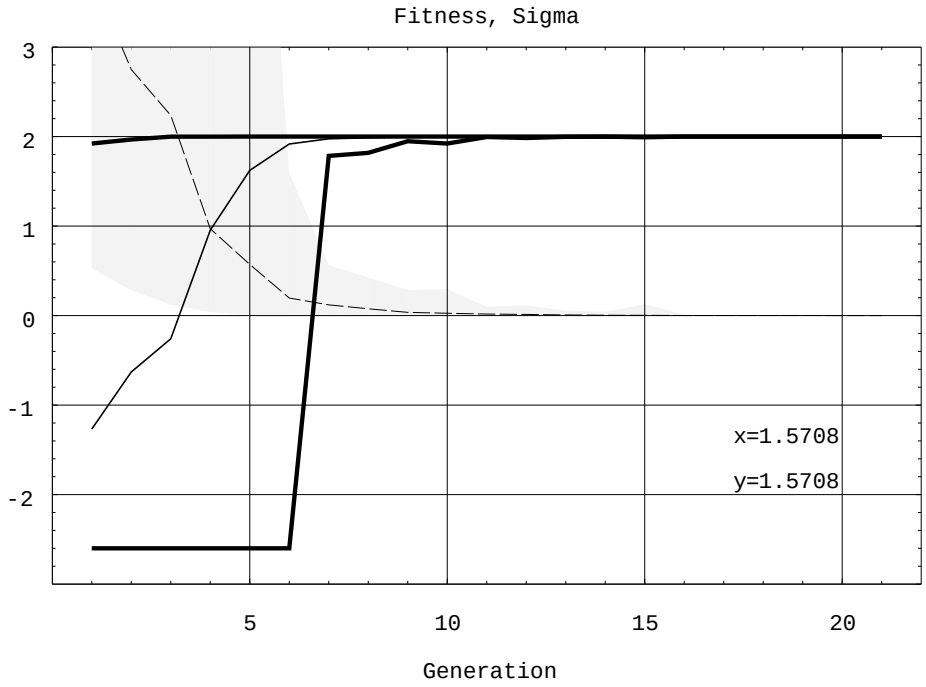


Bild 2.23: Konvergenz: (10+100)-ES

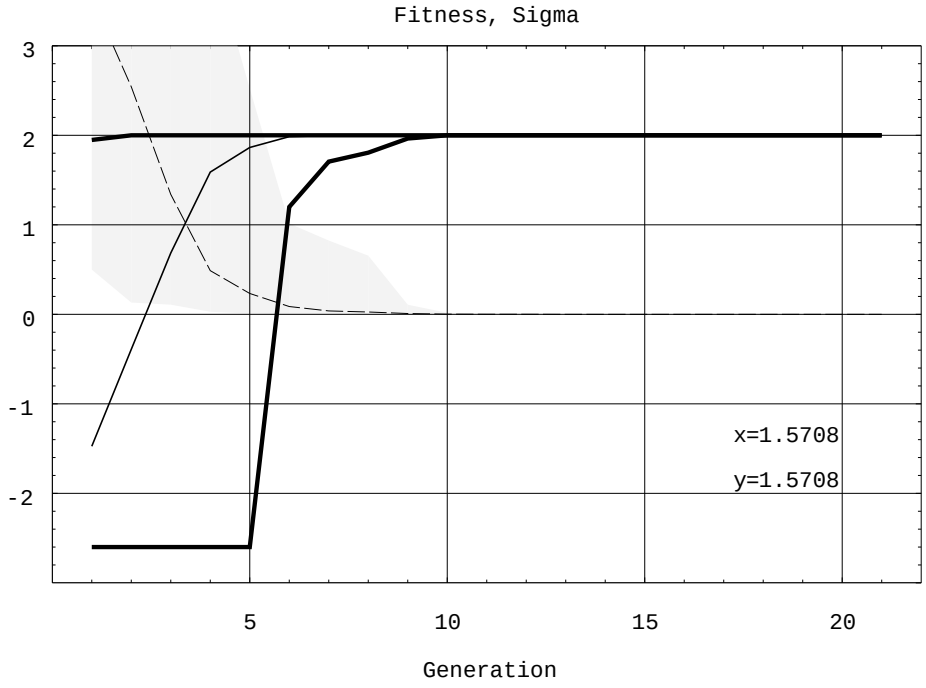


Bild 2.24: Konvergenz: (10+500)-ES

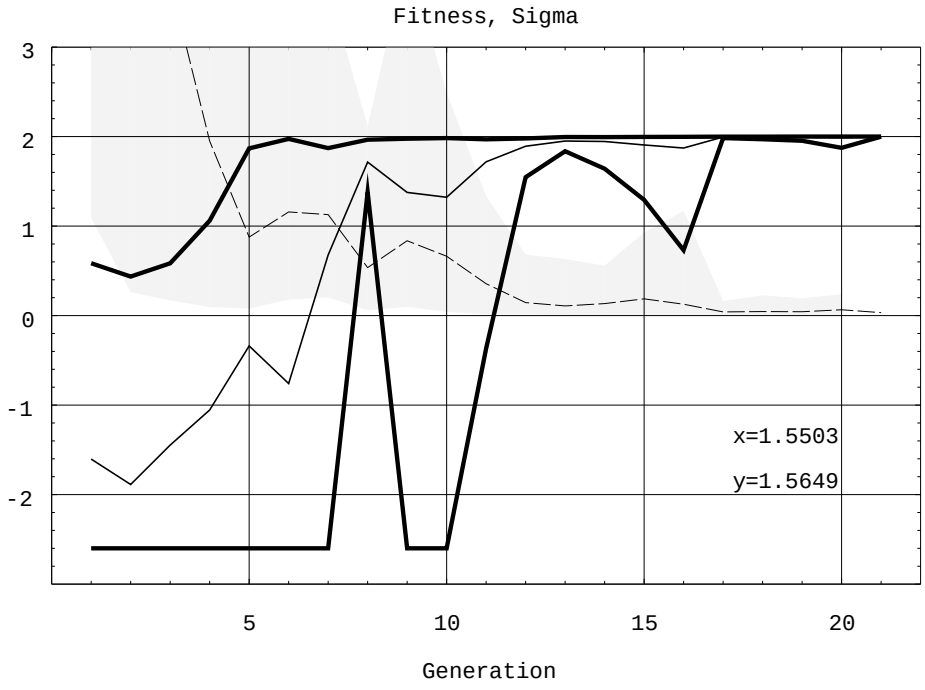


Bild 2.25: Konvergenz: (5,15)-ES

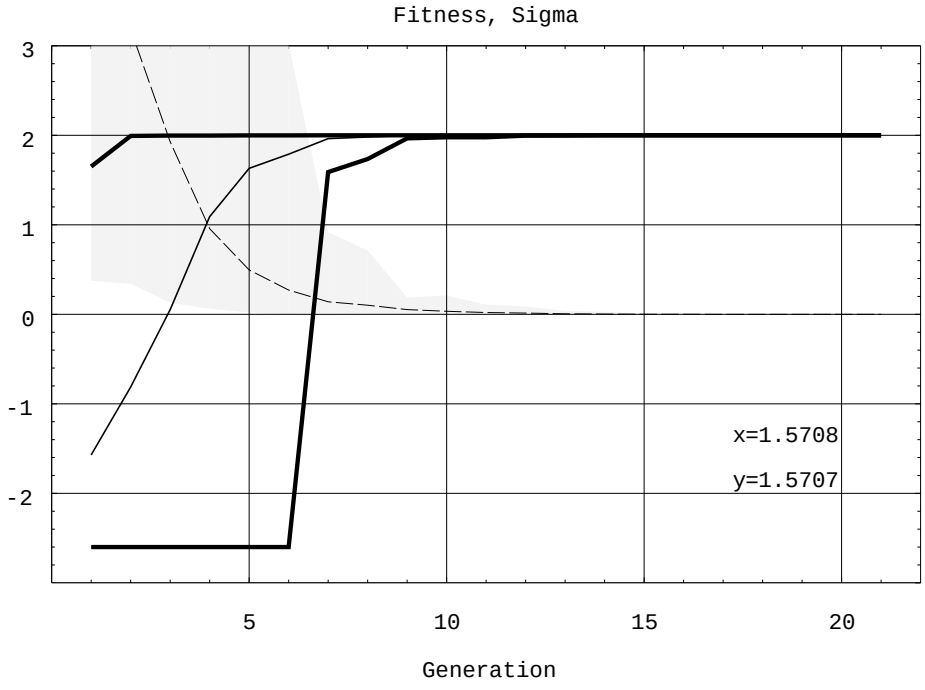


Bild 2.26: Konvergenz: (10,100)-ES

Beispiel 2

Es sei die Zielfunktion $f(x, y)$ des Beispiels 2. aus Abschnitt 2.3.10 mit einer Evolutionsstrategie zu optimieren:

$$f(x,y) = \sin\left(x + \sin y \cdot x\right) + \frac{x+y}{8} \qquad \text{mit} \qquad \begin{array}{l} x \in [-2\pi, 2\pi] \\ y \in [-2\pi, 2\pi] \end{array} \tag{2.48}$$

Das globale Optimum liegt bei:

$$f\left(2\pi, 2\pi - arcsin\left(\frac{3}{4}\right)\right) = f(6,283; 5,435) = 2,465. \tag{2.49}$$

Es werden verschiedene Strategien angewendet. Die Ergebnisse der Optimierungsprozesse zeigen die folgenden Bilder.

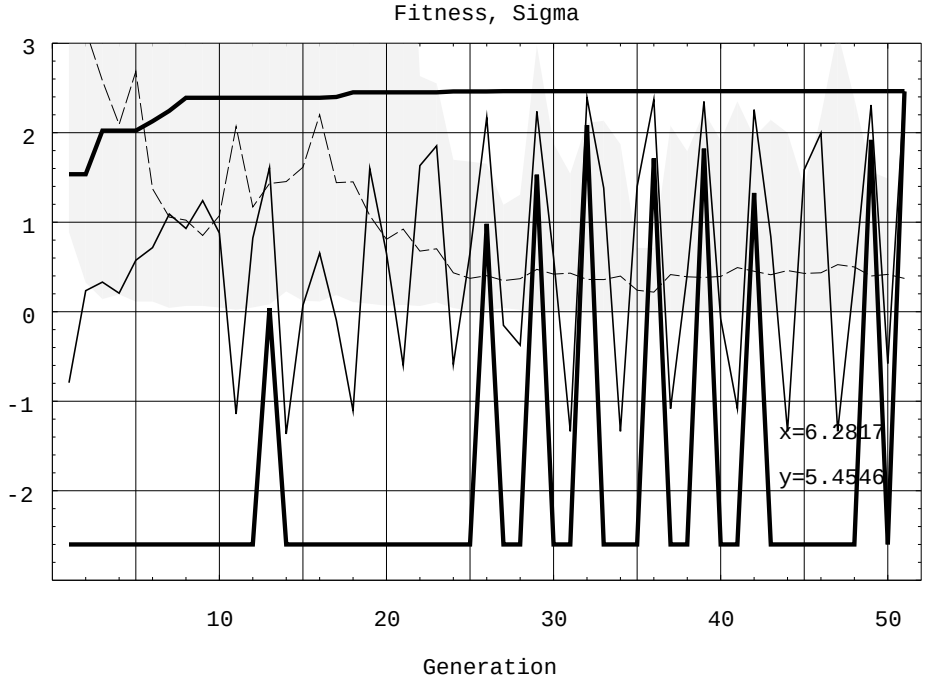


Bild 2.27: Konvergenz: (5+15)-ES

Das Verfahren konvergiert in allen gezeigten Fällen gegen das globale Optimum der Zielfunktion. Die Genauigkeit, die dabei erreicht wird, liegt über derjenigen, die in Abschnitt 2.3 zu GA erreicht wurde.

Darüberhinaus ist das hier verwendete Abbruchkriterium, das sich am Mittelwert der Standardabweichungen orientiert, ein robustes Kriterium, um die Qualität einer Lösung zu bewerten. Bei der ES kann, im Vergleich zu den GA, nach weniger Iterationen mit großer Wahrscheinlichkeit festgestellt werden, ob das Optimum gefunden wurde.

Es hat sich auch gezeigt, daß aufgrund der kleineren Populationsgrößen der durchgeführten Beispieloptimierungen die Laufzeiten im Vergleich mit GA, bei ähnlicher Ge-

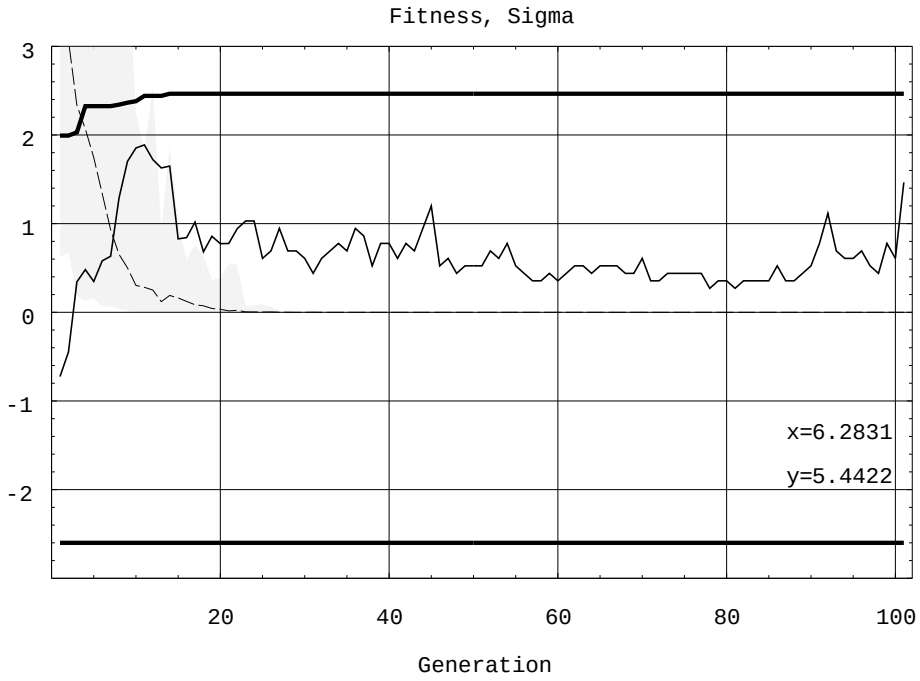


Bild 2.28: Konvergenz: (10+50)-ES

nauigkeit des Ergebnisses, erheblich kürzer waren. Da jedoch nur zwei spezielle Beispielfunktionen untersucht wurden und es auch nicht Absicht war, das Laufzeitverhalten beider Verfahren miteinander zu vergleichen, wird auf diesen Punkt nicht weiter eingegangen. In der Bibliothek `libEvoAlgorithms.a` sind beide Verfahren so implementiert, daß sie ohne Aufwand gegeneinander ausgetauscht werden können. So kann der Anwender entscheiden, welches Verfahren er benutzen möchte.

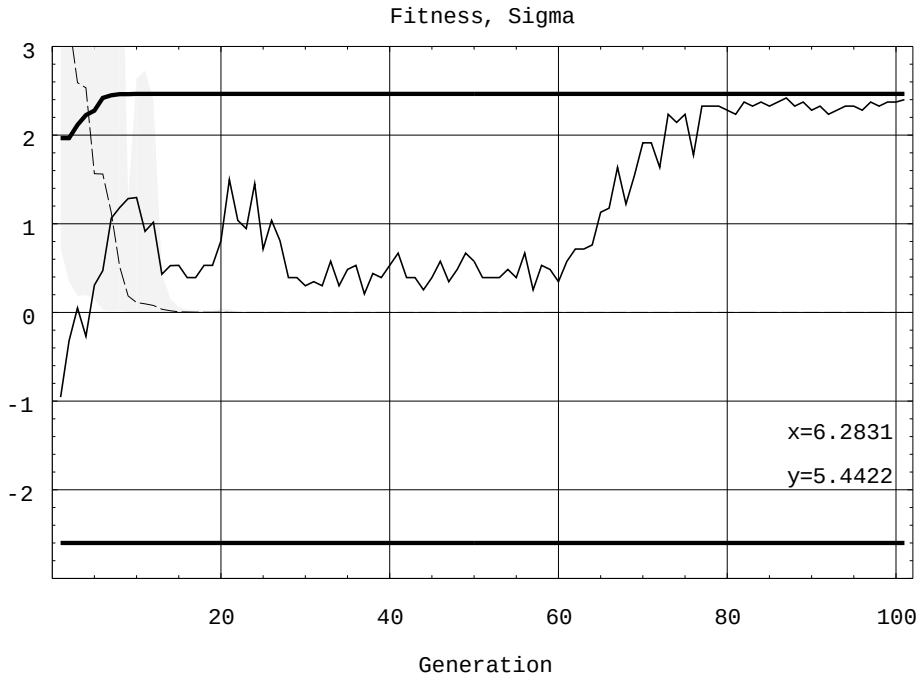


Bild 2.29: Konvergenz: (10+100)-ES

2.5 Weitere Methoden der Evolutionären Algorithmen

EA sind nicht auf klassische Optimierungsprobleme begrenzt. Auch im Bereich der künstlichen Intelligenz finden EA ihre Anwendung. Hier werden EA eingesetzt, um z.B. Computerprogramme zu entwickeln, die für beliebige Aufgaben automatisch Programme generieren können, welche eine gestellte Aufgabe lösen. Zu diesen Methoden gehören die Genetische Programmierung (GP) und die Evolutionäre Programmierung (EP). Da diese Methoden zu der Gruppe der EA gehören, werden diese Verfahren aus Gründen der Vollständigkeit kurz vorgestellt.

Neben den vier Hauptströmungen der EA gibt es weitere verwandte Verfahren zur Lösung von Optimierungsaufgaben. Sie gehören alle zur Gruppe der iterativen, heuristischen Strategien. Zu nennen sind hier Simulated Annealing, Threshold Accepting, Sintflut-Algorithmus und Record-to-Record-Travel. Es wird auf diese nicht weiter eingegangen und an dieser Stelle auf die Literatur, insbesondere [14], verwiesen.

Genetische Programmierung

Ziel ist es, einem Computer beizubringen ein Problem zu lösen, ohne ihn dafür explizit zu programmieren. Der Grundgedanke dabei ist, ein Programm automatisch generieren zu lassen, das die Fähigkeit besitzt, durch Anpassung des Algorithmus die gestellte Aufgabe zu lösen. Man kann dies als Optimierungsaufgabe betrachten und bedient sich hierbei evolutionärer Algorithmen.

Die Ausgangspopulation besteht aus μ zufällig erzeugten Programmen. Ein Programm besteht aus einer Menge problemspezifischer Funktionen und Variablen. Auf Grundlage beider Mengen werden Programme generiert, die die gestellte Aufgabe lösen sollen. Aufgrund von Trainingsdaten optimiert der EA den Programmcode in der Art, daß die gestellte Aufgabe bestmöglich gelöst wird. Als Fitneßfunktion dient die Auswertung von Trainingsdaten, als Parameter der Fitneßfunktion dient die Anordung und Reihenfolge der Funktionen und Variablen.

GP haben eine andere Zielsetzung als die in den Abschnitten 2.3 und 2.4 vorgestellten Methoden. Für das in dieser Arbeit zu behandelnde Problem der Optimierung von Knotenkoordinaten erscheint dieser Zweig der Evolutionären Algorithmen nicht zweckmäßig und wird deshalb nicht weiter verfolgt.

Evolutionäre Programmierung

Bei der EP ist es das Ziel, aufgrund von simulierter Evolution künstlich-intelligente Automaten (finite state machines) zu generieren. Diese sollen die ihnen gestellten Aufgaben auf innovative Weise lösen können. Die Vorgehensweise bei Mutation, Rekombination und Selektion ist dabei weitgehend identisch mit der Vorgehensweise bei der Evolutionsstrategie.

Dieser Zweig der EA findet seine Anwendung vorrangig auf dem Gebiet der künstlichen Intelligenz. Für das Lösen von Optimierungsproblemen ist dieses Verfahren weniger zweckmäßig und wird deshalb nicht weiter ausgeführt.

2.6 Bewertung evolutionärer Algorithmen

Die Optimierungen der Beispielfunktionen lieferten mit beiden Verfahren befriedigende Ergebnisse. Mit beiden Verfahren konnte das jeweilige globale Optimum in kurzer Rechenzeit gefunden werden.

Welches der beiden Verfahren man für das Lösen eines Optimierungsproblems wählt, und ob für dieses überhaupt ein EA ein zweckmäßiges Verfahren ist, muß anhand des konkreten Problems entschieden werden. Ist für ein Optimierungsproblem ein effizientes Spezialverfahren bekannt, so ist es zweckmäßig dieses anzuwenden, da die Lösung in den meisten Fällen schneller gefunden werden kann.

Ein großer Vorteil von EA besteht darin, daß diese Methoden, ohne Vorwissen über die zu optimierende Funktion, bei vielen Optimierungsproblemen anwendbar sind. Auch müssen die Ableitungen der Zielfunktion nicht bekannt sein und es können beliebige Restriktionen einfach in der Zielfunktion berücksichtigt werden.

ES und GA haben beide ihre Vor- und Nachteile. Für die Optimierung kombinatorischer Probleme sind die GA besonders gut geeignet, wenn man eine zweckmäßige Codierung aller möglichen Kombinationen findet.

ES bieten sich für die Optimierung von mathematischen und technischen Systemen an, wenn die Parameter der Zielfunktion reelle Zahlen sind. Es ist vorteilhaft, daß die Codierung der Parameter entfällt und damit die Genauigkeit der Lösung nicht von vornherein beschränkt ist. Durch die Selbstadaptivität der ES paßt sich das Verfahren der Zielfunktion an, das Verfahren ist somit für diese Problemgruppe universell einsetzbar.

Im Vergleich zu vielen anderen Verfahren zur Optimierung haben die EA den Vorteil, daß das Ergebnis nicht vom Startpunkt der Berechnung abhängig ist. Die Wahrscheinlichkeit ein globales Optimum zu finden steigt dadurch. Außerdem können durch die breite Suche im Lösungsraum lokale Optima wieder verlassen werden. Trotzdem garantieren EA nicht das Auffinden eines globalen Optimums.

Zweckmäßig sind EA für das Optimieren von dynamischen Systemen mit veränderlichen Optima. Ist das neue Optimum nicht weit vom alten entfernt, lokalisiert ein EA innerhalb weniger Generationen die Position des neuen Optimums. So können z.B. Regelungssysteme, bei denen sich gewöhnlich ständig die Eingabeparameter ändern, kontinuierlich optimiert werden.

Wegen der Vielzahl der gleichzeitig untersuchten Lösungsmöglichkeiten und ihrer notwendigen Bewertung, sind EA zeitkritisch bei Optimierungsaufgaben, bei denen das Bestimmen des Fitneßwertes einer Lösung rechenintensiv ist. In Zukunft wird dieses Problem jedoch weniger Gewicht haben, da durch die gute Parallelisierbarkeit der EA ein Einsatz auf Parallelrechnern und Clustern effizient zu realisieren ist.

EA sind probabilistische Verfahren und im Laufe der Beispielrechnungen, die im Rahmen dieser Arbeit durchgeführt wurden, hat sich herausgestellt, daß das Konvergenzverhalten von der Güte des Zufallszahlengenerators abhängt. Schlechten Zufallszahlen beeinträchtigen die Qualität der Lösung und erhöhen die benötigten Iterationsschritte. In vielen Softwarepaketen, die Module zum Optimieren mit EA anbieten, sind deshalb auch Generatoren für Zufallszahlen enthalten, die den gewöhnlichen Zufallszahlengenerator ersetzen.

Kapitel 3

Implementierung

3.1 Grundlagen und Basisklassen

Die in den folgenden Abschnitten vorgestellte Software ist in C++ implementiert. Die Wahl ist auf C++ als Programmiersprache gefallen, da C++ eine schnelle und leistungsfähige, objektorientierte Programmiersprache ist, für die die hervorragende Klassenbibliothek 'Qt' der Firma TrollTech zur Verfügung steht. 'Qt' ist für die Plattformen UNIX und Windows erhältlich. Die UNIX-Version ist im Quellcode verfügbar und für die nichtkommerzielle Verwendung kostenlos.

Qt ist eigentlich für die Programmierung von grafischen Oberflächen entwickelt worden, hat aber einige sehr leistungsfähige Funktionalitäten, die auch für nicht-grafische, objektorientierte Anwendungen gut zu gebrauchen sind.

Qt implementiert eine Technik, die als Signal/Slot-Mechanismus bezeichnet wird. Dabei hat ein Objekt die Möglichkeit ein Signal auszusenden, ohne Kenntnis davon zu besitzen, welches Objekt auf dieses Signal reagiert. Dadurch ist es möglich Softwaremodule als Komponenten zu implementieren, die weitgehend unabhängig voneinander sind. In der für diese Arbeit erstellten Software wird von dieser Technik häufig Gebrauch gemacht.

Die wichtigsten Klassen werden vorgestellt und erläutert.

3.1.1 Bibliothek `libFEutil.a`

Die Klassenbibliothek `FEutil` stellt Klassen für das Verwalten von Objekten zur Verfügung. Basisklasse ist `FEObject`. Als Containerklassen stehen `FEArray` und `FEHash` zur Verfügung.

Im folgenden sollen nur die wichtigsten Methoden erläutert werden.

3.1.1.1 Klasse `FEObject`

`FEObject` ist Basisklasse aller Objekte, die die von `FEutil` bereitgestellte Funktionalität verwenden sollen. Instanzen aller von `FEObject` abgeleiteten Klassen besitzen einen eindeutigen, persistenten ID. Dieser wird beim Erzeugen des Objektes automa-

tisch generiert und vergeben. Beim Löschen eines Objektes wird automatisch das Signal `sig_ObjectDeleted(FEObject* obj)` gesendet, das als Parameter einen Zeiger auf das Objekt selbst übergibt. Dadurch ist es möglich, alle Objekte, die das bestimmte Objekt referenzieren, darüber zu informieren, daß das Objekt gelöscht wird und die Referenz somit unzulässig ist. In den Containerklassen `FEArray` und `FEHash` wird diese Funktionalität benutzt um Objekte, die an anderer Stelle gelöscht werden, aus dem Container zu entfernen.

`FEObject` ist von `QObject`, der Basisklasse der Bibliothek Qt, abgleitet und hat dadurch die Fähigkeit Signale zu senden und zu empfangen. Darüberhinaus ist durch die Vaterklasse `QObject` eine beschränkte Selbstbeschreibung der abgeleiteten Klassen zur Verfügung gestellt. So ist es z.B. möglich den Klassennamen abzufragen.

Breschreibung der Methoden

- `FEObject(QString id = 0)`

  Ist der Parameter `id` gleich `0` oder wird kein Parameter übergeben, erzeugt der Konstruktor einen Identifikator für das Objekt. Wird dem Konstruktor ein ID übergeben, so wird dieser verwendet. Dies ist beim Wiederherstellen eines persistenten Objektes notwendig.

  Der ID setzt sich aus zwei Teilen zusammen, die durch einen Punkt voneinander getrennt sind. Der erste Teil ist ein Namensraum, der zweite Teil ist der objektspezifische Teil des ID.

  Beipiel : `model1.object_point_3a`, dieses Objekt mit dem ID `object_point_3a` befindet sich im Namesraum `model1`.

  Der Konstruktor trägt das Objekt in einen Container ein, der als Klassenvariable Zugriff über die Klassenmethode `getObject()` und den ID auf jedes Objekt gestattet.

- `~FEObject()`

  Der Destruktor sendet das Signal `sig_ObjectDeleted(FEObject*)`. Damit kann das Objekt vor seiner Destruktion alle Objekte, die dieses Signal aufnehmen, von seiner Zerstörung informieren.

- `QString getID()`

  Methode zum Lesen des Identifikators.

- `double order() : abstrakt`

  Dies ist eine virtuelle Methode, die einen reellen Wert für ein Ordnungskriterium zurückgibt. Durch das Reduzieren des Vergleichs zweier Objekte auf den Vergleich eines primitiven Datentyps ist es möglich, zwei Objekte verschiedener Klassen zu vergleichen ohne im voraus die Klassen zu kennen.

- `FEObject* getObject(QString id) : Klassenmethode`

  Über diese Klassenmethode hat man Zugriff auf alle Objekte, die von `FEObject` abgeleitet sind. Der Methode wird der ID des Objektes als Parameter übergeben. Die Methode gibt einen Zeiger auf das Objekt zurück. Existiert das Objekt nicht, wird der NULL-Zeiger zurückgegeben.

- `void deleteObject(QString id) : Klassenmethode`

  Der Aufruf dieser Klassenmethode erzwingt das Löschen des Objektes mit dem Identifikator `id`.

- `QString getNamespace(QString id) : Klassenmethode`

  Aus dem ID, der sich aus dem Namensraum und dem ID für das Objekt zusammensetzt, wird der Namesraum extrahiert und zurückgegeben.

- `QString getIDWithoutNSpc(QString id) : Klassenmethode`

  Der ID des Objektes wird aus dem zusammengesetzten ID extrahiert und zurückgegeben.

- `void dumpAllFEObjects() : Klassenmethode`

  Diese Methode gibt alle im aktuellen Prozeß befindlichen Objekte aus, die von der Klasse `FEObject` abgeleitet sind.

3.1.1.2 Klasse `FEArray`

Die Klasse `FEArray` ist eine Containerklasse zum Speichern von Objekten der Klasse `FEObject`. `FEArray` stellt eine Methode zum Sortieren der Objekte zur Verfügung.

Sowohl `FEArray` als auch `FEHash` werden mit dem Signal `sig_ObjectDeleted()` der eingetragenen Objekte verbunden. Wird ein Objekt, das in einem Container referenziert wird, an anderer Stelle gelöscht, so wird dieses Objekt automatisch aus dem Container ausgetragen. So wird sichergestellt, daß keine ungültigen Referenzen in `FEArray` und `FEHash` gespeichert werden.

Beschreibung der Methoden

- `void add(FEObject* obj)`

  Das Objekt `obj` wird dem Container hinzugefügt. Dabei wird überprüft, ob schon ein Objekt mit dem gleichen Identifikator im Array vorhanden ist. Es werden nur Objekte hinzugefügt, die noch nicht im Array vorhanden sind.

- `bool remove(FEObject* obj), bool remove(int i)`

  Das Objekt `obj`, bzw. das Objekt mit dem Index `i` im Array, wird aus dem Array herausgenommen. Das Objekt wird nicht gelöscht. Wenn das Objekt im Array vorhanden war und entfernt wurde, wird TRUE zurückgegeben, sonst FALSE.

- `void sort()`

  Die Objekte im Array werden in aufsteigender Reihenfolge sortiert. Als Vergleichsargument wird der Rückgabewert der Methode `double order()`, die alle von `FEObject` abgeleiteten Klassen implementieren müssen, verwendet.

- `FEObject* getObject(QString id)`

  liefert das Objekt mit dem Identifikator `id`, wenn es im Array vorhanden ist. Sonst gibt die Methode den NULL-Zeiger zurück.

- `bool containsObject(FEobject* obj)`

  gibt TRUE zurück, wenn das Objekt `obj` in dem Array vorhanden ist, sonst FALSE.

3.1.1.3 Klasse `FEHash`

Die Klasse `FEHash` ist eine Containerklasse zum Speichern von Objekten der Klasse `FEObject`. Der Zugriff ist wegen der Hash-Struktur sehr schnell. An den einzelnen Hashpositionen werden Arrays benutzt, um die Objekte aufzunehmen. Die Anzahl der Hashpositionen kann beim Erzeugen des Objektes bestimmt werden. Als Schlüssel für die Berechnung des Hashwertes eines Objektes wird sein ID verwendet.

Beschreibung der Methoden

- `void add(FEObject* obj)`

  Das Objekt `obj` wird dem Container hinzugefügt. Es werden nur Objekte hinzugefügt, die noch nicht im Hash vorhanden sind.

- `FEObject* getObject(QString id)`

  liefert das Objekt mit dem Identifikator `id`, wenn es im Hash vorhanden ist. Sonst gibt die Methode den NULL-Zeiger zurück.

- `bool remove(FEObject* obj)`

  Das Objekt `obj`, wird aus dem Hash herausgenommen. Das Objekt wird nicht gelöscht. Wenn das Objekt im Hash vorhanden war und entfernt wurde, wird TRUE zurückgegeben, sonst FALSE.

3.2 Bibliothek `libEvoAlgorithms.a`

Die Bibliothek `libEvoAlgorithms.a` stellt Klassen zum Lösen von Optimierungsaufgaben mit evolutionären Algorithmen zur Verfügung. In ihr sind sowohl GA als auch ES implementiert. Es war das Ziel, die Implementierung der Algorithmen weitgehend unabhängig von der konkreten Optimierungsaufgabe zu realisieren. Über eine definierte Schnittstelle, die das Anwendungsobjekt zu implementieren hat, greifen die Optimierungsobjekte auf die benötigten Daten zu. Die Parameter der Zielfunktion werden dabei als Vektor vom Datentyp `double*` übergeben. Bei der Optimierung wird versucht, den Fitneßwert zu maximieren.

In den folgenden Abschnitten wird mit ‚Optimierungsobjekt' dasjenige Objekt bezeichnet, das den Optimierungsalgorithmus enthält und das Optimierungsproblem löst.

Mit ‚Anwendungsobjekt' wird das Objekt bezeichnet, dessen Klasse von der Klasse `Opimizable` abgeleitet ist und damit die Zielfunktion implementiert.

3.2.1 Klassendiagramm

Bild 3.1 zeigt das Klassendiagramm der in `libEvoAlgorithms.a` enthaltenen Klassen.

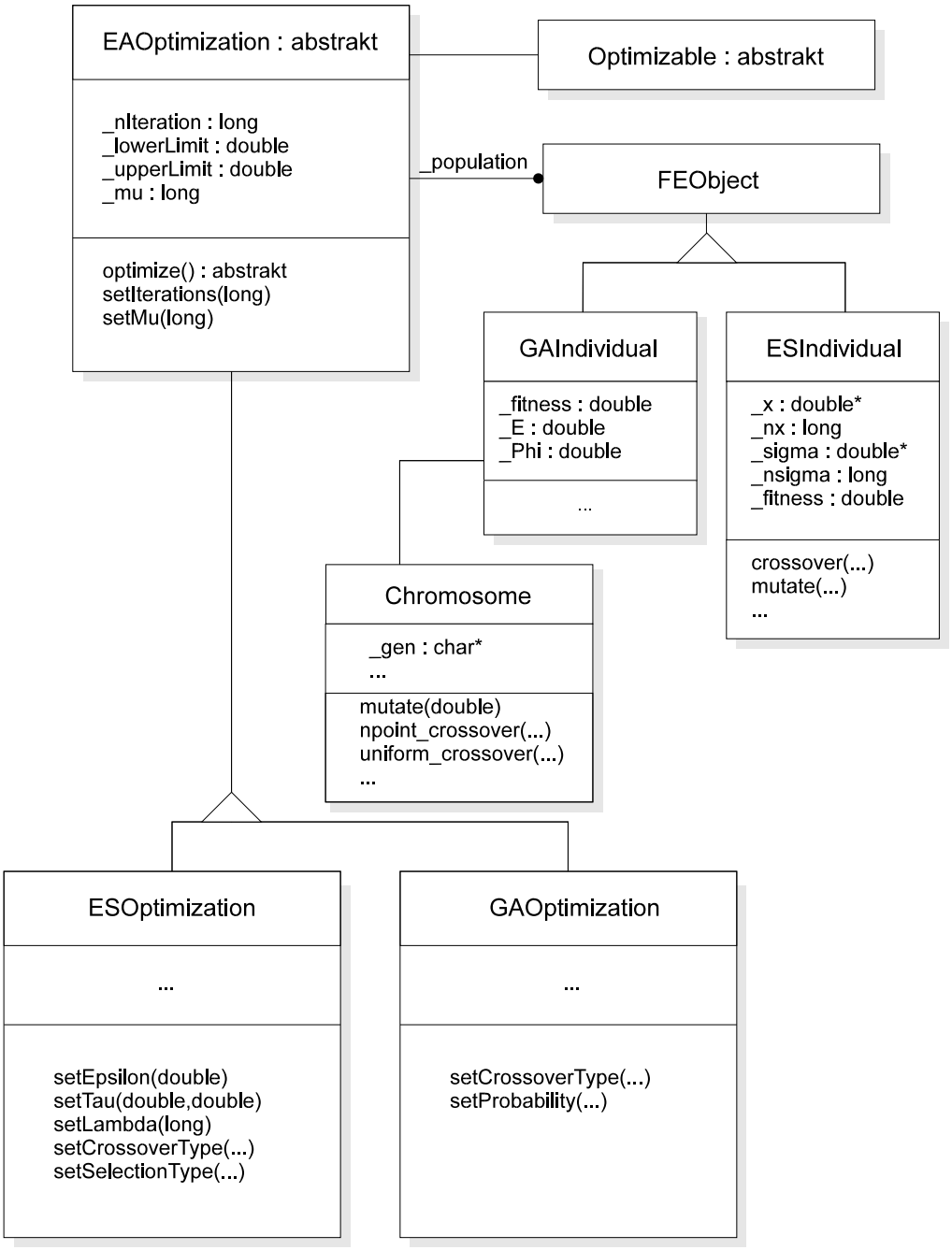


Bild 3.1: Klassendiagramm `libEvoAlgorithms.a`

3.2.2 Datentyp : `Fitness`

Der Datentyp `Fitness` kapselt den Fitneßwert eines Individuums und die Information über die Zulässigkeit der Parameter.

Ausschnitt aus der Datei `EvoAlgorithms.h`:

```
typedef struct
{
  enum Valid {
    valid,
    notvalid
  };

  double value;
  Valid state;

} Fitness;
```

Beschreibung des Datentyps:

- `double value`

  Der Fitneßwert des Individuums.

- `Valid state`

  Die Auzählung `Valid` kann den Status `valid` oder `notvalid` haben.

  Individuen, die den Status `Fitness::notvalid` haben, werden aus dem Optimierungsprozeß herausgenommen.

3.2.3 Klasse : `Optimizable`

Die Klasse `Optimizable` stellt die Schnittstelle des Anwendungsobjektes zu den Optimierungsalgorithmen dar. In Anlehnung an das Interface-Konzept der Programmiersprache JAVA, wurde die Klasse `Optimizable` als abstrakte Klasse realisiert, d.h. von ihr können keine Instanzen erzeugt werden. Sie enthält nur reine virtuelle Methoden, die vom Anwender in einer abgeleiteten Klasse zu implementieren sind. Über diese Methoden greifen die Optimierungsobjekte auf die von ihnen benötigten Daten zu.

Ausschnitt aus der Datei `EvoAlgorithms.h`:

```
//////////////////
class Optimizable
//////////////////
{

public:

  virtual ~Optimizable() {};

  /**
   *  compute and return fitnessvalue for given
   *  parameters x
```

```
   */

  virtual Fitness fitness(double* x) = 0;

  /**
   *  get number of parameters of optimization-problem
   */

  virtual long getNumberOfParameters() = 0;

  /**
   *  the result of optimization is stored in optimizable object,
   *  so memory must be allocated by this object. <br>
   *  this method returns ptr to this memory.
   */

  virtual double* getPtrParameters() = 0;

  /**
   *  a valid domain from lower limit to upper limit
   *  must be specified
   */

  virtual double getLowerLimit() = 0;
  virtual double getUpperLimit() = 0;

};
```

Beschreibung der Methoden

- `~Optimizable() : abstrakt`

  Es ist zweckmäßig für reine virtuelle Klassen einen virtuellen Destruktor zu definieren.

- `Fitness fitness(double* x) : abstrakt`

  Mit der Methode `fitness()` wird dem Optimierungsobjekt die Möglichkeit gegeben, den Fitneßwert eines Individuums zu ermitteln. Dazu werden die Parameter der Fitneßfunktion als Zeiger auf einen Vektor mit `double`-Werten übergeben.

  Restriktionen der Optimierungsaufgabe werden vom Anwendungsobjekt in der Methode `fitness()` überprüft. Es ist notwendig, das Optimierungsobjekt über die Zulässigkeit einer Lösung zu informieren. Da der gesamte Wertebereich des Datentyps `double` als Fitneßwert in Betracht kommen kann, ist es unmöglich, den Optimierungsalgorithmus nur durch den Wert einer Variablen über die Unzulässigkeit einer Lösung zu informieren. Aus diesem Grund wurde der Datentyp `Fitness` definiert, der sowohl Fitneßwert als auch Information über die Zulässigkeit einer Lösung enthält.

- `long getNumberOfParameters() : abstrakt`

  Über diese Methode kann das Optimierungsobjekt auf die Anzahl der Parameter der Zielfunktion zugreifen.

- `double* getPtrParameters() : abstrakt`

Über diese Methode kann das Optimierungsobjekt den Zeiger auf den Speicherbereich der Parameter des Anwendungsobjektes erfragen. Das Anwendungsobjekt hat dafür Sorge zu tragen, daß für die Parameter Speicherplatz angefordert wird. Das Optimierungsobjekt trägt das Ergebnis der Optimierung nach Ende der letzten Iteration dort ein.

- `double getLowerLimit() : abstrakt`

  Um die Initialisierung der Ausgangspopulation zu steuern, können die Werte der Parameter in festgelegten Grenzen erzeugt werden. Die hier gesetzten Grenzen werden nur am Anfang der Iteration abgefragt. Die Parameter der Ausgangspopulation werden zufällig erzeugt, liegen aber mit Sicherheit innerhalb des festgelegten Gebietes.

  Bei den GA legen die Grenzen darüberhinaus den darstellbaren Bereich der reellen Werte der Zielfunktion in binärer Codierung fest. Die ES hat, aufgrund der reellen Darstellung der Parameter, diese Einschränkung nicht.

  Mit der Methode `getLowerLimit()` kann das Optimierungsobjekt die untere Grenze abfragen.

- `double getUpperLimit() : abstrakt`

  Mit der Methode `getUpperLimit()` kann das Optimierungsobjekt die obere Grenze abfragen.

3.2.4 Evolutionäre Algorithmen

In der Bibliothek `libEvoAlgorithms.a` werden Klassen für zwei evolutionäre Optimierungsalgorithmen zur Verfügung gestellt. Dies ist eine Klasse für Genetische Algorithmen und eine Klasse, die eine Evolutionsstrategie implementiert. Beide Klassen sind von der gemeinsamen Vaterklasse `EAOptimization` abgeleitet.

3.2.4.1 Basisklasse `EAOptimization`

Die Klasse `EAOptimization` ist eine abstrakte Vaterklasse der Klassen `GAOptimization` und `ESOptimization`. In ihr sind gemeinsame Attribute und Methoden vereint.

Im Attribut `FEArray* _population` werden die Zeiger auf die Individuen der Population gespeichert. In Abhängigkeit von der verwendeten Optimierungsstrategie sind dies entweder Objekte vom Typ `ESIndividual` oder vom Typ `GAIndividual`. Beide Klassen sind von der Basisklasse `FEObject` abgeleitet und können somit in einem Container der Klasse `FEArray` gespeichert werden.

Beschreibung der Methoden

- `void optimize() : abstrakt`

  Die Methode startet den Optimierungsprozeß.

- `void setIterations(long it)`

  Über diese Methode ist die maximale Anzahl der Iterationen festzulegen.

- `void setMu(long mu)`

  Die Methode setzt die Populationsgröße für die Optimierung.

3.2.5 Genetische Algorithmen

Für die Implementierung von Genetischen Algorithmen wurden die drei Klassen `Chromosome`, `GAIndividual` und `GAOptimization` entworfen.

3.2.5.1 Klasse `Chromosome`

Die Klasse `Chromosome` ist eine interne Klasse, die für den Benutzer nicht sichtbar ist. Ein Chromosom besteht aus einer bestimmten Anzahl von Genen. Diese werden bei der Klasse `Chromosome` als Vektor des Datentyps `char` gespeichert. Die einzelnen Gene können den Wert `1` oder `0` annehmen.

Die wichtigsten Methoden werden im folgenden Abschnitt vorgestellt und erläutert.

Beschreibung der Methoden

- `Chromosome(long lengthSegment, long segments)`

  Bei der Erzeugung eines Chromosoms wird die Anzahl der Segmente und die Länge eines Segments festgelegt. Ein Segment entspricht einem reellen Parameter der Zielfunktion. Die Länge eines Segmentes wird von der Genauigkeit der Darstellung des reellen Parameters und des darstellbaren Bereichs beeinflußt.

- `Chromosome(Chromosome* chromo)`

  Die Werte der Attribute des neuen Objektes der Klasse `Chromosome` werden mit den Werten des Objektes `chromo` initialisiert.

- `void mutate(double pm)`

  Mit dieser Methode wird die Mutationswahrscheinlichkeit, mit der jedes Gen invertiert wird, gesetzt.

- `void npoint_crossover(Chromosome *chromosome, long n)`

  Es sind zwei Arten von Crossover implementiert. Die eine ist N-Point-Crossover. Diese Methode führt mit dem aktuellen Chromosom und dem Chromosom `chromosome` ein N-Point-Crossover mit `n` Punkten durch. Die Gene des aktuellen Chromosoms werden mit dem Ergebnis überschrieben.

- `void uniform_crossover(Chromosome *chromosome, double pu)`

  Uniform-Crossover ist der zweite Crossovertyp, der in der Klasse `Chromosome` implementiert ist. Mit `pu` wird die Crossoverwahrscheinlichkeit gesetzt. Die Gene des aktuellen Chromosoms werden mit dem Ergebnis überschrieben.

3.2.5.2 Klasse `GAIndividual`

Ein Objekt der Klasse `GAIndividual` stellt ein Individuum für einen Genetischen Algorithmus dar. In ihm werden ein Objekt der Klasse `Chromosome` der Wert der Zielfunktion, der Erwartungswert eines Individuums für die nächste Population, und der skalierte Fitneßwert gespeichert.

Beschreibung der Methoden

- `Chromosome* getChromosome()`
  `void setChromosome(Chromosome* c)`

  Methoden zum Setzen und Abfragen des Objektes vom Typ `Chromosome`.

- `double getFitness()`
  `void setFitness(double val)`

  Methoden zum Setzen und Abfragen der Fitneß des Individuums. Die Fitneß wird nicht in der Methode `double getFitness()` berechnet.

- `double getE()`
  `void setE(double val)`

  Mit diesen Methoden kann für das Individuum der Erwartungswert für die nächste Population gesetzt und abgefragt werden.

- `double getPhi()`
  `void setPhi(double val)`

  Mit diesen Methoden kann der skalierte Fitneßwert gesetzt oder abgefragt werden.

3.2.5.3 Klasse `GAOptimization`

Die Klasse `GAOptimization` implementiert einen Genetischen Algorithmus.

Beschreibung der Methoden

- Konstruktor

```
GAOptimization(Optimizable* obj,
               int precision, long mu,
               double pc = 0.1, double pm = 0.01,
               GACrossoverType co_type = npoint,
               long npoints = 2);
```

  Bei der Initialisierung können folgende Parameter übergeben werden:

  – `Optimizable* obj`

Es muß ein Zeiger auf das Anwendungsobjekt übergeben werden. Die Klasse des Anwendungsobjekts implementiert das Interface `Optimizable`. Dadurch kann das Optimierungsobjekt auf die benötigten Daten zugreifen.

Das Ergebnis der Optimierung wird im Anwendungsobjekt gespeichert.

- `int precision`

  Mit diesem Parameter wird die Präzision des darstellbaren Ergebnisses bestimmt. Die Variable `precision` gibt die Zahl der Nachkommastellen an.

- `long mu`

  `mu` legt die Größe der Population fest.

- `double pc = 0.1`

  `pc` legt die Crossover-Wahrscheinlichkeit für Uniform-Crossover fest. Wird kein Parameter übergeben wird standardmäßig `pc` auf 0.1 gesetzt.

- `double pm = 0.01`

  `pm` legt die Mutations-Wahrscheinlichkeit fest. Der Standardwert liegt bei 0.01.

- `GACrossoverType co_type = npoint`

  Mit diesem Parameter wird das Rekombinationsverfahren gewählt. Implementiert sind Uniform-Crossover und N-Point-Crossover. Entsprechend wird der Wert auf `GAOptimization::uniform` oder `GAOptimization::npoint` gesetzt. Standardmäßig wird N-Point-Crossover verwendet

- `long npoints = 2`

  `npoints` legt die Anzahl der Punkte für N-Point-Crossover fest. Der Standardwert ist zwei.

- `void setCrossoverType(GACrossoverType type, long npoints = 2)`

  Mit dieser Methode ist es möglich das Rekombinationsverfahren und die Anzahl der Punkte für N-Point-Crossover zu setzen.

  Als Crossover-Methode stehen `GAOptimization::uniform` und `GAOptimization::npoint` zur Verfügung.

  Für die Crossover-Methode `GAOptimization::uniform` wird der Wert `npoint` nicht benötigt und kann deshalb immer vernachlässigt werden.

- `void setProbability(double pc, double pm)`

  Die Methode setzt die Mutations-Wahrscheinlichkeit und die Crossover-Wahrscheinlichkeit für Uniform-Crossover.

3.2.6 Evolutionsstrategie

Es wurden zwei Klassen für die Implementierung der ES entworfen. Die Klasse `ESIndividual` stellt ein Individuum für eine Evolutionsstrategie dar, die Klasse `ESOptimization` enthält den Optimierungsalgorithmus und die Methoden um die Optimierung zu steuern.

3.2.6.1 Klasse `ESIndividual`

Die Klasse `ESIndividual` stellt ein Individuum einer Evolutionsstrategie dar. Es werden ein Vektor mit den Parametern der Zielfunktion und ein Vektor mit Strategieparametern vermerkt. Beide Vektoren sind vom Datentyp `double*`.

Es wird der Aufzählungstyp `ESDistribution` definiert, der entweder den Wert `equal` oder den Wert `normal` annehmen kann.

Nur die wichtigsten Methoden werden im folgenden erläutert.

Beschreibung der Methoden

- Konstruktor

  ```
  ESIndividual(long nx, long nsigma,
               double lowerL, double upperL,
               ESDistribution init = equal)
  ```

  Das Objekt wird beim Erzeugen mit den genannten Parametern erzeugt. `nx` setzt die Anzahl der Parameter, `nsigma` setzt die Anzahl der Standardabweichungen. `nx` ist üblicherweise gleich `nsigma`. Ist `nsigma` kleiner als `nx`, wird für die übrigen Parameter $\sigma_{i>nsigma}$ die Standardabweichung σ_{nsigma} verwendet.

  Die Parameter `lowerL` und `upperL` legen die untere und obere Grenze für die Initialisierung der Ausgangspopulation fest.

  Die Werte der Parameter der Zielfunktion in der Ausgangspopulation werden zufällig erzeugt. Über den Parameter `ESDistribution init` kann man eine normalverteilte oder eine gleichverteilte Erzeugung von Zufallswerten bestimmen. Für `init` wird entsprechend `ESIndividual::normal` oder `ESIndividual::equal` übergeben. Standardmäßig werden gleichverteilte Zufallsgrößen erzeugt. Es wird sichergestellt, daß die zufällig erzeugten Parameter innerhalb des durch `lowerL` und `upperL` begrenzten Bereichs liegen.

- Konstruktor

  ```
  ESIndividual(ESIndividual* ind)
  ```

  Der Konstruktor initialisiert das neue Objekt mit den Werten des Objektes `ind`.

- `void crossover(ESIndividual* ind,`
  `ESOptimization::ESCrossoverType rx,`
  `ESOptimization::ESCrossoverType rs)`

  Die Methode führt ein Crossover zwischen dem aktuellen Individuum und dem Individuum `ind` durch. Mit den Parametern `rx` und `rs` werden die Crossover-Strategien für die Parameter der Zielfunktion und für die Strategieparameter σ_i bestimmt. Es kann zwischen diskreter und intermediärer Rekombination gewählt werden.

  Die Werte der Variablen des aktuellen Objektes werden mit dem Ergebnis überschrieben.

- `void mutate(double tau1, double tau2)`

  Die Methode mutiert das Individuum mit den beiden Mutationsparametern `tau1` für τ_1 und `tau2` für τ_2.

- `double* getPtrX()`

  Die Parameter, die die Entwurfsvariablen der Optimierungsaufgabe darstellen, werden im Individuum gespeichert. Mit dieser Methode kann auf den Vekor der Parameter zugegriffen werden.

3.2.6.2 Klasse `ESOptimization`

Die Klasse `ESOptimization` implementiert eine Evolutionsstrategie. Der Optimierungsprozeß ist über Parameter steuerbar.

Es werden die Aufzählungstypen `ESCrossoverType` und `ESSelectionType` definiert.

Beschreibung der Methoden

- Konstruktor

```
ESOptimization(Optimizable* obj,
               long mu, long lambda, ESSelectionType sType,
               ESCrossoverType rx, ESCrossoverType rsigma,
               double epsilon = 0.01,
               double tau1 = 0, double tau2 = 0 );
```

  Der Konstruktor erzeugt ein Objekt der Klasse `ESOptimization`. Bei der Initialisierung können folgende Parameter übergeben werden:

  - `Optimizable* obj`
    Es muß ein Zeiger auf das Anwendungsobjekt übergeben werden. Die Klasse des Anwendungsobjekts implementiert das Interface `Optimizable`. Dadurch kann das Optimierungsobjekt auf die benötigten Daten zugreifen.
    Das Ergebnis der Optimierung wird im Anwendungsobjekt gespeichert.
  - `long mu`
    `mu` legt die Größe der Population fest.
  - `long lambda`
    `lambda` bestimmt die Anzahl der Nachkommen, die in jedem Iterationsschritt erzeugt werden sollen.
  - `ESSelectionType sType`
    legt die Selektionsstrategie fest. Möglich ist `ESOptimization::plus` für eine $(\mu + \lambda)$-ES und `ESOptimization::comma` für eine (μ, λ)-ES.
  - `ESCrossoverType rx`
    legt die Rekombinationsstrategie für die Parameter fest. Möglich ist `ESOptimization::uniform` für diskrete Rekombination und `ESOptimization::intermediate` für intermediäre Rekombination.

  - `ESCrossoverType rs`

    legt die Rekombinationsstrategie für die Standardabweichungen fest. Möglich ist `ESOptimization::uniform` für diskrete Rekombination und `ESOptimization::intermediate` für intermediäre Rekombination.

  - `double epsilon = 0.01`

    `epsilon` steuert das Abbruchkriterium für die Iteration. Fällt die mittlere Standardabweichung unter `epsilon`, wird die Itertion beendet. Wird beim Erzeugen des Optimierungsobjektes kein Parameter angegeben, wird der Wert auf 0.01 gesetzt.

  - `double tau1 = 0, double tau2 = 0`

    Beide Parameter steuern den Mutationsoperator. Wird beim Aufruf des Konstruktors kein Parameter angegeben oder der Wert auf Null gesetzt, werden τ_1 und τ_2 auf die in Abschnitt 2.4.5 empfohlenen Standardwerte gesetzt.

- `void setEpsilon(double e)`

  Die Methode setzt die Schranke für den Abbruch der Iteration. Fällt die mittlere Standardabweichung unter `epsilon`, wird die Iteration beendet.

- `void setTau(double t1, double t2)`

  Die Methode setzt die Parameter für den Mutationsoperator.

- `void setLambda(long lambda)`

  Die Methode setzt die Zahl der in jedem Iterationsschritt zu erzeugenden Nachkommen.

- `void setCrossoverType(ESCrossoverType rx, ESCrossoverType rsigma)`

  Die Methode setzt das Rekombinationsverfahren für die Parameter und für die Standardabweichungen.

  Möglich sind `ESOptimization::uniform` für diskrete Rekombination und `ESOptimization::intermediate` für intermediäre Rekombination.

3.2.7 Beispiel für die Anwendung von `libEvoAlgorithms.a`

Anhand einer Beispielimplemetierung wird die Benutzung der Bibliothek `libEvoAlgorithms.a` und der in ihr bereitgestellten Klassen verdeutlicht. Es sei ein Programm zu entwickeln, das die Optimierungsaufgabe des Beispiels 2. aus Abschnitt 2.4 mit einer Evolutionsstrategie löst.

Die Anwendung eines Genetischen Algorithmus kann analog mit einem Objekt der Klasse `GAOptimization` statt `ESOptimization` durchgeführt werden.

Programmcode

Das Programm ist in zwei Dateien aufgeteilt.

Die Datei `OptObject.h` enthält die Definition der Klasse `OptObject`. Diese implementiert das Interface `Optimizable` und enthält damit die Implementierung der Zielfunktion.

Die Datei `bspES.C` implementiert die Funktion `main()`, erzeugt in ihr ein Optimierungsobjekt und startet die Optimierung.

OptObject.h

```
 1 //
 2 //  file : OptObject.h
 3 //
 4
 5 #include "EvoAlgorithms.h"
 6 #include "stdlib.h"
 7 #include "stdio.h"
 8
 9 ////////////////////////////////////
10 class OptObject
11   : public virtual Optimizable
12 ////////////////////////////////////
13 {
14
15 public:
16
17   OptObject()
18   {
19     _countP = 2;
20     _x = new double[_countP];
21   }
22
23   ~OptObject()
24   {
25     delete [] _x;
26   }
27
28   Fitness fitness(double* x)
29   {
30     Fitness F;
31
32     if(x[0] > 2*M_PI || x[1] > 2*M_PI ||
33        x[0] < -(2*M_PI) || x[1] < -(2*M_PI) ) {
34       F.state = Fitness::notvalid;
35       return F;
36     }
37
38     F.value = sin(x[0] + sin(x[1]) * x[0]) + (x[0]+x[1])/8.;
39     F.state = Fitness::valid;
40
41     return F;
42
43   }
44
45   long getNumberOfParameters()
46   {
47     return _countP;
48   }
49
```

```
50   double* getPtrParameters()
51   {
52     return _x;
53   }
54
55   double getUpperLimit()
56   {
57     return (2*M_PI);
58   }
59
60   double getLowerLimit()
61   {
62     return -(2*M_PI);
63   }
64
65 protected:
66
67   long _countP;
68   double* _x;
69
70 };
71
```

Die Klasse `OptObject` implementiert das Interface `Optimizable`.

- Zeile 17-21 : Im Konstruktor wird Speicherplatz für die Parameter der Optimerungsaufgabe angefordert.
- Zeile 28-43 : Die Zielfunktion wird in der Methode `fitness()` implementiert.

  Der zulässige Lösungsbereich ist auf $x \in [-2\pi, 2\pi]$ und $y \in [-2\pi, 2\pi]$ beschränkt. In den Zeilen 32-36 wird dies überprüft. Liegen die Parameter nicht in diesem Bereich, wird der Variablen `F.state` der Wert `Fitness::notvalid` zugewiesen. Dadurch wird das Individuum automatisch aus der Population entfernt.

  In Zeile 38 wird der Wert der Zielfunktion berechnet. Sind die Parameter zulässig, wird der Variablen `state` der Wert `Fitness::valid` zugewiesen.

bspES.C In der Datei `bspES.C` wird die `main()`-Funktion implementiert.

```
 1 //
 2 //  file    : bspES.C
 3 //
 4
 5 #include "OptObject.h"
 6 #include "ESOptimization.h"
 7
 8
 9 ////////////////////////////////
10 int main(int argc, char **argv )
11 ////////////////////////////////
12 {
13   double* result;
14   long nParameters;
15
16   if(argc < 5) {
17    printf("usage : %s ", argv[0] );
```

```
18    printf("[iterations][pop. size][offspring size][epsilon]\n");
19    exit(1);
20   }
21
22   long iterations = atol(argv[1]);
23   long mu = atol(argv[2]);
24   long lambda = atol(argv[3]);
25   double epsilon = atof(argv[4]);
26
27   //  create optimizable object
28   OptObject* obj = new OptObject();
29
30   //  create object for ES
31   ESOptimization* esOpt =
32     new ESOptimization(obj,
33        mu, lambda,
34        ESOptimization::plus,
35        ESOptimization::uniform, ESOptimization::uniform,
36        epsilon );
37
38   esOpt->setIterations( iterations );
39   esOpt->optimize();
40
41   //  print results
42
43   result      = obj->getPtrParameters();
44   nParameters = obj->getNumberOfParameters();
45
46   printf("\n\n=========  result  ES  ==========\n");
47
48   for(int i=0; i<nParameters; i++)
49     printf("[%d] x = %.6f \n", i, result[i] );
50
51   printf("================================\n");
52   printf("fitness : %.6f \n\n", obj->fitness( result ).value);
53
54 }
```

- Zeile 22-25 : Bei Aufruf des Programms kann der Benutzer Parameter zur Steuerung der Optimierung übergeben. Diese werden in entsprechenden Variablen gespeichert. Dem Optimierungsobjekt werden diese Variablen bei seiner Erzeugung übergeben.
- Zeile 28 : Das Anwendungsobjekt wird als Instanz der Klasse `OptObject` erzeugt.
- Zeile 31ff : Eine Instanz der Klasse `ESOptimization` wird erzeugt. Diesem Objekt wird das Anwendungsobjekt und die Parameter zur Steuerung des Optimierungsprozesses übergeben.
- Zeile 39 : Die Optimierung wird gestartet. Das Ergebnis wird im Anwendungsobjekt gespeichert.
- Zeile 43-52 : Die Ergebnisse werden ausgegeben.

Möchte man statt einer ES einen GA verwenden, erzeugt man statt der Instanz der Klasse `ESOptimization` ein Objekt der Klasse `GAOptimization` (Zeile 31 ff). Dieses er-

wartet andere Steuerparameter bei seiner Erzeugung. Die Klasse `OptObject` muß nicht verändert werden.

3.3 Bibliothek `libMesh.a`

Die Bibliothek `libMesh.a` bietet Klassen für die Vernetzung von Punktmengen an.

3.3.1 Klassendiagramm

Bild 3.2 zeigt das Klassendiagramm der in `libMesh.a` enthaltenen und verwendeten Klassen.

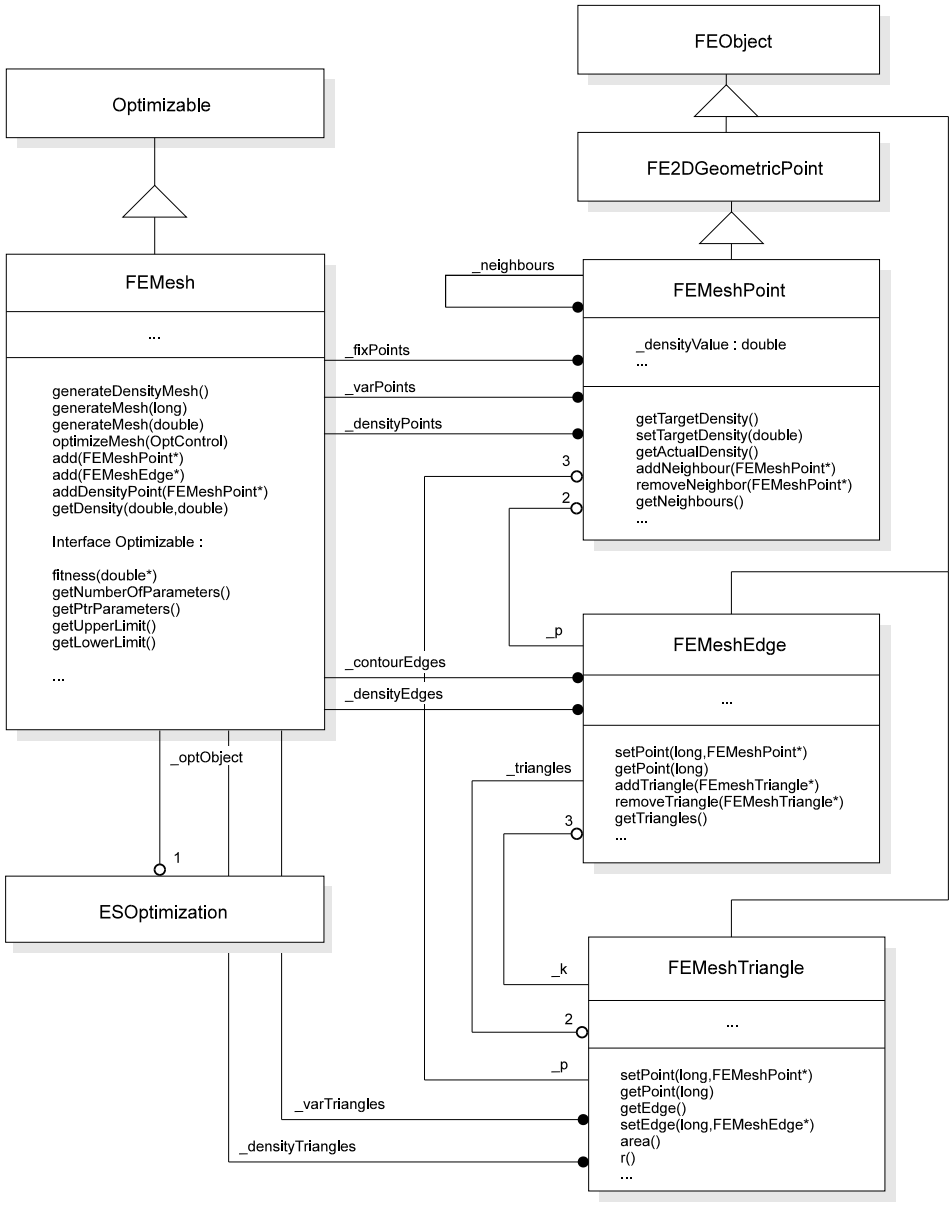


Bild 3.2: Klassendiagramm `libMesh.a`

3.3.2 Klasse `FEMeshPoint`

Die Klasse `FEMeshPoint` definiert einen 2-dimensionalen Punkt eines Netzes. Dem Punkt kann ein Soll-Wert für seine Dichte zugeordnet werden.

Die Klasse ist von der Basisklasse `FE2DGeometricPoint` abgeleitet. Diese stellt die elementaren Attribute und Methoden zum Speichern, Setzen und Abfragen von Koordinaten zur Verfügung. Die Klasse `FE2DGeometricPoint` ist in `libFEutil.a` definiert. Es wird nicht weiter auf diese Klasse eingegangen.

3.3.2.1 Beschreibung der Methoden

- `FEMeshPoint(double x, double y, QString id = 0)`

  Der Konstruktor erzeugt ein Objekt der Klasse `FEMeshPoint`. Die Koordinaten des neuen Punktes werden mit `x` und `y` initialisiert. Wird kein Identifikator als Parameter übergeben, wird ein ID erzeugt.

- `double getTargetDensity()`

  Über diese Methode kann der Soll-Wert für die Dichte des Punktes abgefragt werden.

- `void setTargetDensity(double value)`

  Mit dieser Methode kann der Soll-Wert für die Dichte gesetzt werden.

- `double getActualDensity()`

  Die Methode `getActualDensity()` berechnet den tatsächlichen Wert für die Dichte des Punktes aus dem Abstand des Punktes zu seinen Nachbarn.

- ```
  void addNeighbour(FEMeshPoint* point)
  void removeNeighbour(FEMeshPoint* point)
  void removeAllNeighbours()
  FEArray* getNeighbours()
  ```

  Die Nachbarn eines Punktes werden in der Objektvariablen `_neighbours` vermerkt. Über die genannten Methoden können Nachbarn hinzugefügt, entfernt und abgefragt werden.

3.3.3 Klasse `FEMeshEdge`

Die Klasse `FEMeshEdge` definiert die Kante eines 2-dimensionalen Netzes.

3.3.3.1 Beschreibung der Methoden

- `FEMeshEdge(FEMeshPoint* p1, FEMeshPoint* p2, QString id = 0)`

  Der Konstruktor erzeugt ein neues Objekt der Klasse `FEMeshEdge` und initialisiert die beiden Punkte mit `p1` und `p2`. Wird kein Identifikator als Parameter übergeben, wird ein ID erzeugt.
  ```

- `FEMeshPoint* getPoint(long index)`

  Über diese Methode kann der Punkt mit dem Index `index` abgefragt werden.

- `void setPoint(long index, FEMeshPoint* p)`

  Mit dieser Methode kann der Punkt mit dem Index `index` gesetzt werden.

- ```
  void addTriangle(FEMeshTriangle* tri)
  void removeTriangle(FEMeshTriangle* tri)
  FEArray* getTriangles()
  ```

  Die angrenzenden Dreiecke werden in der Kante vermerkt. Über die genannten Methoden können diese hinzugefügt, entfernt oder abgefragt werden.

3.3.4 Klasse `FEMeshTriangle`

Die Klasse `FEMeshTriangle` definiert ein Dreieck eines 2-dimensionalen Netzes.

3.3.4.1 Beschreibung der Methoden

- `FEMeshTriangle(QString id = 0)`

  Der Konstruktor erzeugt ein neues Objekt der Klasse `FEMeshTriangle`. Wird kein Identifikator als Parameter übergeben, wird ein ID erzeugt.

- `double alpha(long index)`

  Diese Methode berechnet den Innenwinkel am Punkt `index` im Bogenmaß und gibt diesen zurück.

- `double area()`

  Die Fläche des Dreiecks wird berechnet und zurückgegeben. Die berechnete Fläche wird im Dreieck vermerkt.

- `double getArea()`

  Die Methode gibt die vermerkte Fläche zurück. Die Fläche wird nicht neu berechnet.

- `FEMeshPoint* getPoint(long index)`

  Mit dieser Methode kann der Punkt mit dem Index `index` abgefragt werden.

- `void setPoint(long index, FEMeshPoint* p)`

  Mit dieser Methode kann der Punkt mit dem Index `index` gesetzt werden.

- `FEMeshEdge* getEdge(long index)`

  Mit dieser Methode kann die Kante mit dem Index `index` abgefragt werden.

- `void setEdge(long index, FEMeshEdge* e)`

  Mit dieser Methode kann die Kante mit dem Index `index` gesetzt werden.
  ```

- `double r()`

  Die Methode `r()` berechnet das Verhältnis der Fläche eines Dreiecks zu der durchschnittlichen Fläche seiner Nachbardreiecke.

- `bool isInside(double x, double y, double* z1 = 0, double* z2 = 0, double* z3 = 0)`

  Mit dieser Methode kann erfragt werden, ob der Punkt mit den Koordinaten (x,y) innerhalb des Dreiecks liegt. Liegt der Punkt im Innern, werden die normierten Koordinaten des Punktes (x,y) in Bezug auf das aktuelle Dreieck in den Variablen `z1`, `z2` und `z3` gesetzt.

3.3.5 Klasse `FEMesh`

Die Klasse `FEMesh` implementiert die Algorithmen für die Vernetzung einer Punktmenge und die Optimierung der Koordinaten ihrer Punkte.

`FEMesh` implementiert das Interface `Optimizable`, und kann damit auf die Funktionalität der Optimierungsalgorithmen der Bibliothek `libEvoAlgorithms.a` zurückgreifen.

Es werden nur die Randkanten und die festen Punkte auf den Randkanten und im Netzinneren dem Objekt der Klasse `FEMesh` übergeben. Die freien Punkte, die für die Optimierung relevant sind, werden im Netzinneren vom Netzgenerator erzeugt. Bei der Erzeugung von freien Punkten werden zwei Strategien angeboten. Die eine generiert eine bestimmte Anzahl von Innenpunkten, die andere Strategie generiert die Innenpunkte auf einem regelmäßigen Raster.

3.3.5.1 Beschreibung der Methoden

- `void generateDensityMesh()`

  Die Punkte, die zur Bildung des Netzes der Dichtefunktion übergeben wurden, werden mit einem Delaunay-Algorithmus trianguliert. Es entsteht dabei ein konvexes Netz. Dieses ist unabhängig von dem Netz für die FE-Analyse.

- `void generateMesh(long innerPoints)`

  Die Methode generiert ein Netz mit einer bestimmten Anzahl von Innenpunkten.

  Algorithmus:

  - Es wird ein Initialnetz erzeugt. Dieses wird bestimmt, indem die festen Punkte des Netzes unter Berücksichtigung der gegebenen Randkanten eingeschränkt trianguliert werden.
  - In Abhängigkeit von der Größe der Dreiecke des Initialnetzes werden zufällig im Innern dieser Dreieckecke eine gegebene Anzahl von Punkten erzeugt. Diese Punkte sind die freien Punkte des Netzes und ihre Koordinaten werden bei der Optimierung verändert. Das Verhältnis der zu erzeugenden Innenpunkte in einem Dreieck ist proportional zu seiner Größe.
  - Das Initialnetz wird gelöscht.

  - Es wird das endgültige Netz mit den festen und freien Punkten durch eine eingeschränkte Triangulation erzeugt.

- `void generateMesh(double griddistance)`

  Die freien Innenpunkte werden auf einem regelmäßigen Raster generiert. Der Abstand der Rasterpunkte wird durch den Parameter `griddistance` festgelegt.

  Algorithmus:

  - Es wird ein Initialnetz erzeugt. Dieses wird bestimmt, indem die festen Punkte des Netzes unter Berücksichtigung der gegebenen Randkanten eingeschränkt trianguliert werden.
  - Es werden die minimalen und maximalen Koordinaten des Initialnetzes ermittelt. Es werden auf den Knoten des Rasters, dessen Größe und Lage durch den Bereich der extremalen Koordinaten des Initialnetzes vorgegeben wird, Punkte erzeugt, wenn der zu untersuchenden Punkt zulässig ist. Ein Punkt ist dann zulässig, wenn er im Innern des Netzgebietes liegt. Dies wird überprüft, indem festgestellt wird ob der Punkt in einem der Dreiecke des Initialnetzes liegt. Punkte, die in einem dieser Dreiecke liegen, liegen auch im Innern des Netzgebietes.
  - Das Initialnetz wird gelöscht.
  - Es wird das endgültige Netz mit den festen und freien Punkten durch eine eingeschränkte Triangulation erzeugt.

- `void optimizeMesh(OptControl control)`

  Die Optimierung wird gestartet. Der Parameter `control` spezifiziert die Parmeter für die Optimierung der Knotenkoordinaten.

  Die Klasse `FEMesh` implementiert das Interface `Optimizable` und aggregiert ein Objekt der Klasse `ESOptimization`. Bei Aufruf des Konstruktors des Optimierungsobjektes wird das Objekt der Klasse `FEMesh` als Anwendungsobjekt übergeben.

- `FEArray* getTriangles()`

  Über die Methode `getTriangles()` kann auf das Array zugegriffen werden, in dem die Dreiecke des Netzes für die FE-Analyse gespeichert sind.

- `FEArray* getPoints()`

  Über die Methode `getPoints()` kann auf das Array zugegriffen werden, in dem die Punkte des Netzes für die FE-Analyse gespeichert sind.

- `FEArray* getDensityTriangles()`

  Über die Methode `getDensityTriangles()` kann auf das Array zugegriffen werden, in dem die Dreiecke des Dichtenetzes gespeichert sind.

- `FEArray* getDensityPoints()`

  Über die Methode `getDensityPoints()` kann auf das Array zugegriffen werden, in dem die Punkte des Dichtenetzes gespeichert sind.

- `void add(FEMeshEdge* obj)`

  Hinzufügen einer Randkante zum Netz für die FE-Analyse.

- `void add(FEMeshPoint* obj)`

  Hinzufügen eines Punktes zum Netz für die FE-Analyse.

- `void deleteAll()`

  Die Methode löscht alle im Objekt der Klasse `FEMesh` gespeicherten Objekte.

- `void addDensityPoint(FEMeshPoint* obj)`

  Hinzufügen eines Punktes zum Dichtenetz.

- `ESOptimization* getESOptimization()`

  Die Methode gibt das Optimierungsobjekt vom Typ `ESOptimization*` zurück.

- `double getDensity(double x, double y)`

  Die Methode bestimmt den Soll-Wert der Dichtefunktion am Punkt (x,y).

- `Fitness fitness(double* x)`

  Die Methode `fitness(double* x)` implementiert die Fitneßfunktion für die Optimierung.

  Ausschnitt aus der Datei `FEMesh_opt.C`:

```
 1 Fitness FEMesh::fitness(double* x)
 2 ////////////////////////////////////
 3 {
 4   Fitness F;
 5   FEMeshTriangle* tri;
 6   double area;
 7   double _Qw=0., _Qg=0., _Qd=0.;
 8
 9   //  init fitness-value
10   F.value = 0.;
11
12   /*
13    *  copy x to mesh
14    *  coords of points = initP + x
15    */
16
17   copyXToMesh( _initialParameters, x );
18
19   /*
20    *  calculate area of triangles
21    *  if area is negative -> penality
22    */
23
24   for(int j=0; j<_varTriangles->size(); j++) {
25
26     tri = (FEMeshTriangle*) _varTriangles->elementAt(j);
27
28     area = tri->area();
29
30     if( area <= 0 ) {
31       F.value += (100*area - 1.);
```

```
32      }
33    }
34
35     /*
36      *  Qw
37      */
38
39    if( _control.stateQw == OptControl::active )
40      _Qw = Qw() * _control.Gw;
41
42    /*
43     *  Qg
44     */
45
46    if( _control.stateQg == OptControl::active )
47      _Qg = Qg() * _control.Gg;
48
49    /*
50     *  Qd
51     */
52
53    if( _control.stateQd == OptControl::active )
54      _Qd = Qd() * _control.Gd;
55
56    //  accumulate fitness
57    F.value -= ( _Qw + _Qg + _Qd );
58    F.state = Fitness::valid;
59    return F;
60  }
```

In Zeile 17 werden mit der Methode `copyXToMesh()` die Parameter `x` in das Netz kopiert. Der Vektor `x` enthält nur die Veränderung der Koordinaten gegenüber den Koordinaten des Ausgangsnetzes.

Die Fitneß eines Individuums setzt sich aus der gewichteten Addition der drei Kenngrößen Q_w, Q_g und Q_d zusammen. Die Restriktion, daß Dreiecke nur positive Flächen besitzen dürfen, wird mit der Subtraktion eines Strafwerts vom Fitneßwert in Zeile 31 erfaßt. Die Gewichtung der negativen Flächen mit dem Wert 100 ist willkürlich gewählt. Es ist nur sicherzustellen, daß Netze, die Dreiecke mit negativen Flächen enthalten, einen schlechteren Fitneßwert aufweisen als die schlechtesten zulässigen Lösungen. Es wurde darauf verzichtet, Netze die Dreiecke mit negativen Flächen enthalten, den Status `Fitness::notvalid` zuzuordnen und somit aus dem Optimierungsprozeß zu entfernen. Dies hat folgenden Grund: In jedem Iterationsschritt werden die Parameter, d.h. die Knotenkoordinaten mit einer Zufallsgröße verändert. Da selbst bei kleinen Veränderungen die Wahrscheinlichkeit sehr hoch ist, daß mindestens ein unzulässiges Dreieck im Netz vorhanden ist, besteht die Möglichkeit, daß in einem Iterationsschritt nur unzulässige Lösungen erzeugt werden. Bei der Durchführung von Beispieloptimierungen hat sich gezeigt, daß in diesem Fall die Optimierung nicht konvergierte. Es ist daher zweckmäßig, unzulässige Lösungen im Oprimierungsprozeß beizubehalten, aber mit einem Strafwert zu belegen, der sich nach der Größe der negativen Fläche richtet. Dadurch hat der Optimierungsalgorithmus die Möglichkeit, durch Verbesserung einer unzulässigen Lösung in den zulässigen Lösungsbereich zu wandern.

Nicht alle Kenngrößen müssen berücksichtigt werden. Die Variablen `OptControl.stateQw`, `OptControl.stateQg` und `OptControl.stateQd` zeigen an, ob die entsprechende Kenngröße berechnet werden soll und somit in den Optimierungsprozeß einfließt.

Durch die Gewichtung der Kenngrößen ist eine Steuerung des Optimierungsprozesses gegeben.

- `long getNumberOfParameters()`
  `double* getPtrParameters()`
  `double getLowerLimit()`
  `double getUpperLimit()`

  Diese Methoden implementieren neben der Methode `fitness()` das Interface `Optimizable`.

3.3.6 Datentyp `OptControl`

Der Datentyp `OptControl` kapselt die Daten, die zur Steuerung des Optimierungsprozesses benötigt werden.

```
typedef struct
{
  double Gw, Gg, Gd;

  enum State{passive,active};

  State stateQw;
  State stateQg;
  State stateQd;

  double lowerLimit, upperLimit;

  long mu, lambda;
  long nIterations;

  double epsilon;

  ESOptimization::ESCrossoverType crossover_rx;
  ESOptimization::ESCrossoverType crossover_rsigma;
  ESOptimization::ESSelectionType selection;

} OptControl;
```

3.3.7 Klasse `OptControlDlg`

Die Klasse `OptControlDlg` stellt einen Dialog zur Verfügung, über den die Optimierung der Knotenkoordinaten gesteuert werden kann.

Bild 3.3 zeigt einen Bildschirmausdruck des Dialogs.

3.3.7.1 Beschreibung der Methoden

- `OptControlDlg(QWidget * parent=0)`

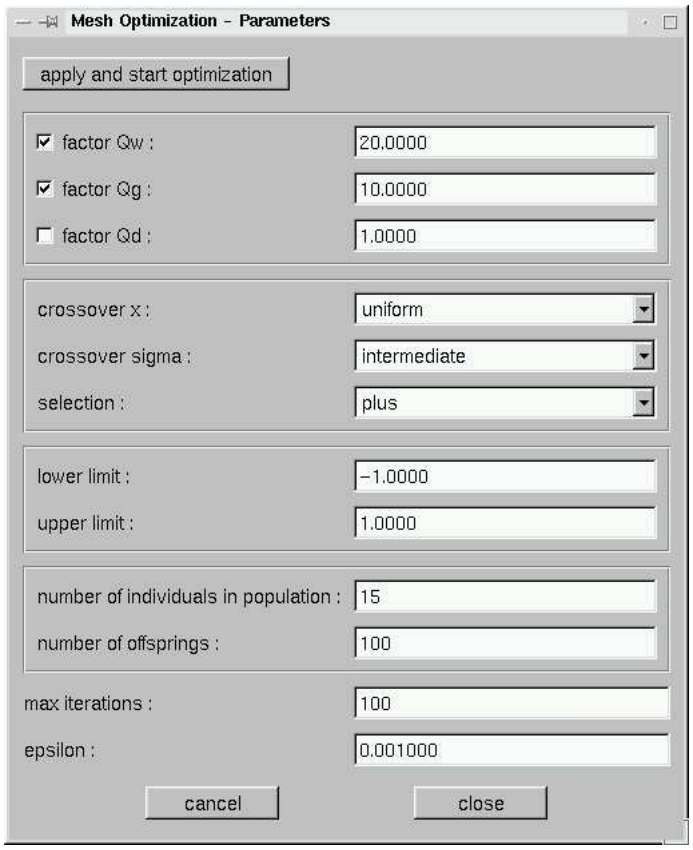


Bild 3.3: Dialog `OptControlDlg`

Der Konstruktor erzeugt ein neues Objekt der Klasse `OptControlDlg`. Der Parameter `parent` gibt an, an welcher Stelle das Dialogobjekt in die Hierarchie der grafischen Objekte eingeordnet werden soll. So ist es möglich, den Dialog in die Oberfläche eines Programms zu integrieren ohne für den Dialog ein eigenes Fenster erzeugen zu müssen.

- `OptControlDlg(OptControl optcontrol, QWidget * parent=0)`

  Der Konstruktor erzeugt ein neues Objekt der Klasse `OptControlDlg`. Die Werte der Attribute des Dialogobjekts werden mit den Werten des Parameters `optcontrol` initialisiert.

- `void setValues(OptControl optcontrol)`

  Die Attributwerte des Objektes werden mit den Werten des Parameters `optcontrol` überschrieben.

- `OptControl getValues()`

  Die Attributwerte des Dialogs werden ermittelt und zurückgegeben.

3.4 Pilotimplementierung MeshApp

In einer Beispielimplementierung werden die oben vorgestellten Module verwendet. Es sind die Daten eines Netzes einzulesen und das Netz grafisch darzustellen. Über Dialoge ist die Vernetzung und Optimierung des Netzes interaktiv zu steuern. Es soll sowohl das Ausgangsnetz als auch das optimierte Netz grafisch dargestellt werden.

Die Parameterstudien in Abschnitt 4 wurden mit dieser Applikation durchgeführt.

3.4.1 Klasse `MeshApplicationWindow`

Die Klasse `MeshApplicationWindow` enthält ein Objekt zur grafischen Darstellung von geometrischen Objekten, einen Netzgenerator vom Typ `FEMesh` und Objekte vom Typ `OptControlDlg` und `MeshControlDlg` zu Steuerung der Vernetzung und der Optimierung des Netzes.

Auf Bild 3.4 ist ein Bildschirmausdruck der Anwendung zu sehen.

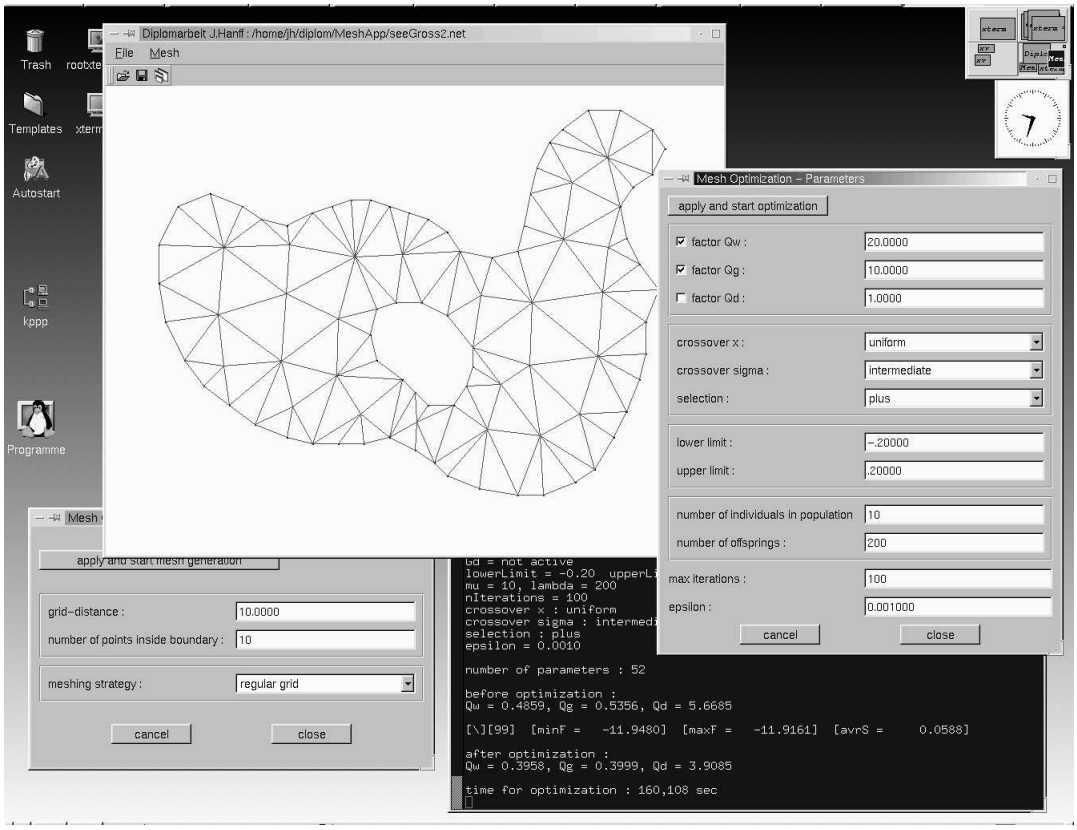


Bild 3.4: Bildschirmausdruck der Anwendung

3.4.2 Klasse `MeshControlDlg`

Die Klasse `MeshControlDlg` implementiert einen Dialog über den die Vernetzung gesteuert werden kann. Es kann über eine Auswahlliste die Strategie für die Vernetzung vom Anwender gewählt werden. Die Werte für die Anzahl der Innenpunkte und die Rasterweite können über Textfelder eingegeben werden.

Bild 3.5 zeigt einen Bildschirmausdruck des Dialogs.

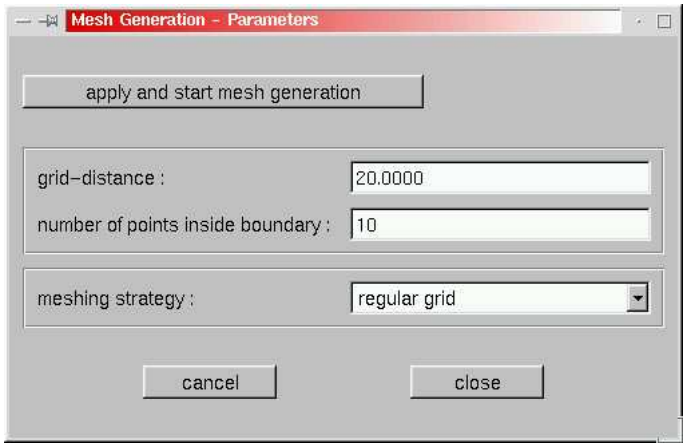


Bild 3.5: Dialog `MeshControlDlg`

Kapitel 4

Parameterstudien

4.1 Beispielfiguren

An drei Beispielen wird der Einfluß der verschiedenen Parameter auf den Verlauf und das Ergebnis der Optimierung untersucht. Die Beispiele 1 und 2 sind bewußt sehr einfach gewählt, um die Güte der Ergebnisse abschätzen zu können. Beispiel 3 ist ein Beipiel, das eine größere Anzahl von Punkten auf dem Rand besitzt und sich deshalb näher an der Praxis orientiert.

4.1.1 1. Beispiel: Quadrat, 4 Punkte

Das Beispiel 1 ist ein Quadrat mit 4 Punkten und einer Seitenlänge von 10 Einheiten.

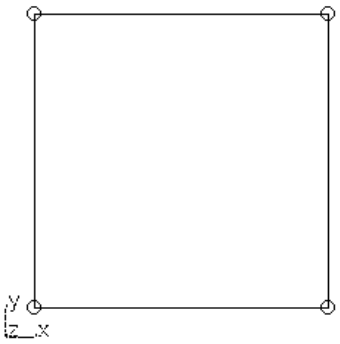


Bild 4.1: Beispielfigur 1: Quadrat 4 Punkte

4.1.2 2. Beispiel: Quadrat, 16 Punkte

Das Beispiel 2 ist ein Quadrat mit 16 Punkten und einer Seitenlänge von 10 Einheiten. Die Punkte auf den Kanten haben gleiche Abstände.

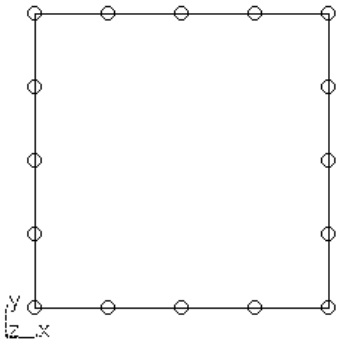


Bild 4.2: Beispielfigur 2: Quadrat 16 Punkte

4.1.3 3. Beispiel: Polygonzug mit 69 Punkten und Aussparung

Das Beispiel 3 ist ein polygonal umrandetes Gebiet mit einer Aussparung und insgesamt 69 Randpunkten.

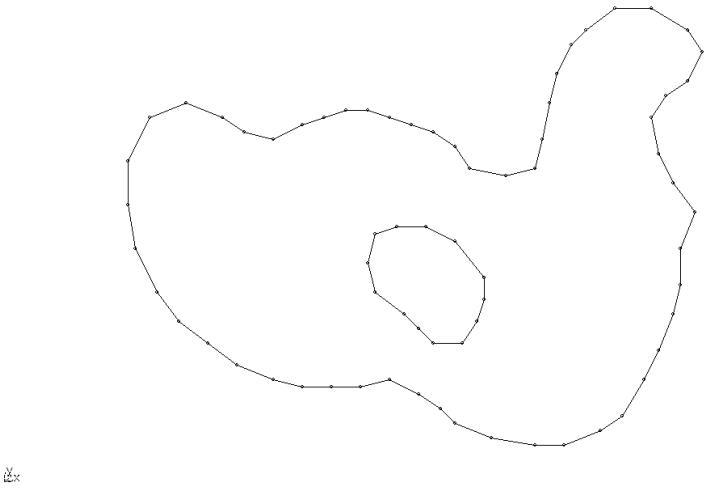


Bild 4.3: Beispielfigur 3: Polygon 69 Punkte

4.2 Anzahl ungültiger Individuen

In einer Testreihe sollte herausgefunden werden, wie die Zahl der freien Punkte, und damit die Zahl der Entwurfsvariablen für die Optimierungsaufgabe, sich auf die Zahl der ungültigen Lösungen in der Population auswirkt. Zu diesem Zweck wurde der Programmcode geringfügig verändert. Die ungültigen Lösungen wurden nicht, wie das normalerweise der Fall ist, gelöscht, sondern wurden statt dessen in der Optimierung weiterhin berücksichtigt und ihre Anzahl protokolliert.

4.2.1 Quadrat, 4 Punkte

Bild 4.4 zeigt die Anzahl der ungültigen Individuen für das Testbeispiel 1. Die Dicke der Linien ist proportional zur Anzahl der Entwurfsvariablen der Optimierungsaufgabe.

Quadrat, 4 Punkte; Bild 4.4	
Parameter	Wert
Anzahl Entwurfsvariablen	2, 10, 20, 40
Optimierungsstrategie	(15+100)-ES
G_w	10
G_g	10
G_d	/
Crossover-Strategie für Parameter	diskret
Crossover-Strategie für Standardabweichungen	diskret

Tabelle 4.1: Parameter 1. Beispielfigur

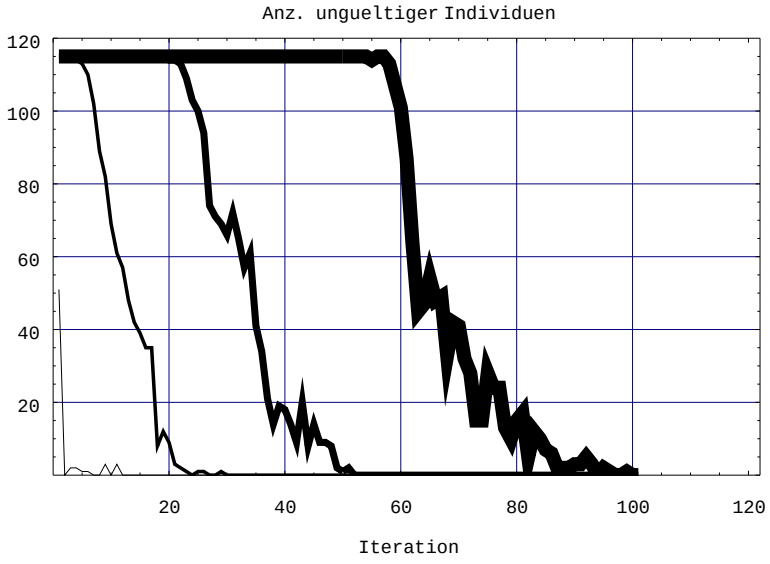


Bild 4.4: Anzahl ungültiger Individuen, (15+100)-ES, Quadrat 4 Punkte

4.2.2 Quadrat, 16 Punkte

Quadrat, 16 Punkte; Bild 4.5	
Parameter	Wert
Anzahl Entwurfsvariablen	2, 10, 20, 40
Optimierungsstrategie	(15+100)-ES
G_w	10
G_g	10
G_d	/
Crossover-Strategie für Parameter	diskret
Crossover-Strategie für Standardabweichungen	diskret

Tabelle 4.2: Parameter 2. Beispielfigur

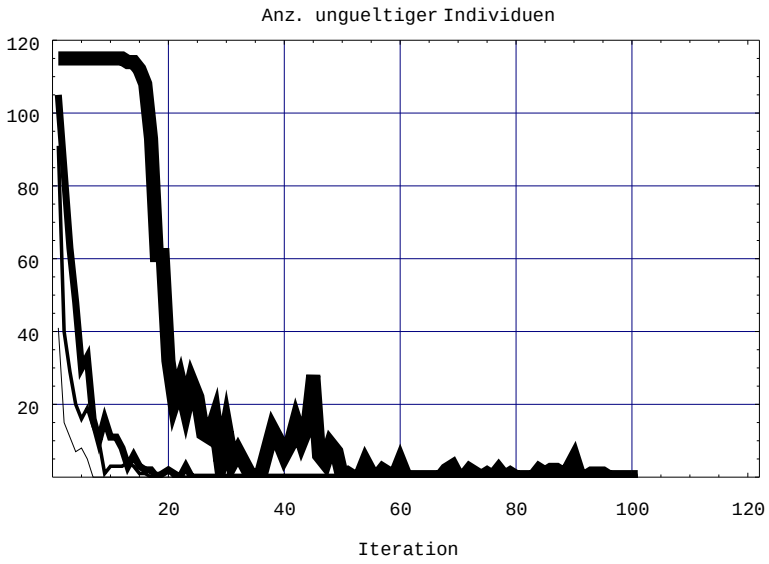


Bild 4.5: Anzahl ungültiger Individuen, (15+100)-ES, Quadrat 16 Punkte

4.2.3 Polygonal umrandetes Gebiet, 69 Punkte

Polygonal umrandetes Gebiet, 69 Punkte; Bild 4.6	
Parameter	Wert
Anzahl Entwurfsvariablen	10, 20, 40, 200
Optimierungsstrategie	(15+100)-ES
G_w	10
G_g	10
G_d	/
Crossover-Strategie für Parameter	diskret
Crossover-Strategie für Standardabweichungen	diskret

Tabelle 4.3: Parameter 3. Beispielfigur

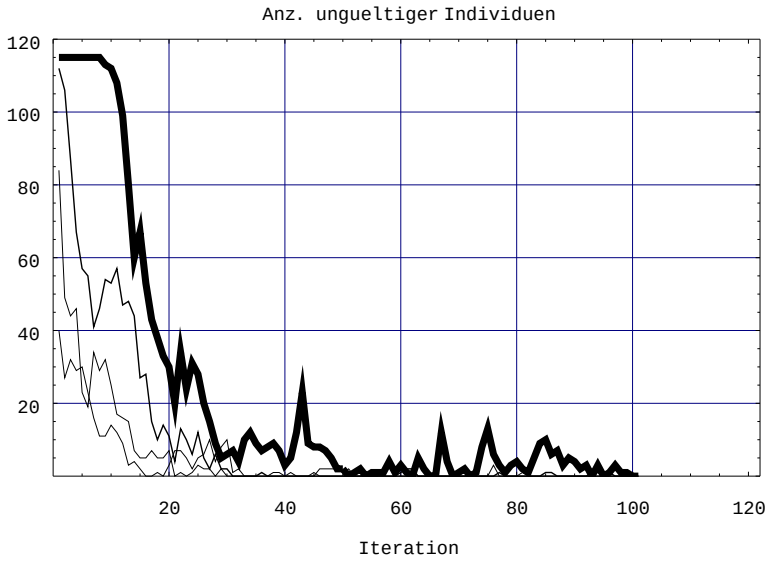


Bild 4.6: Anzahl ungültiger Individuen, (15+100)-ES, Polygon, 69 Punkte

Die Grafiken zeigen deutlich, daß mit zunehmender Zahl von Entwurfsvariablen, die Anzahl der Iterationen, in denen ungültige Lösungen existieren, zunimmt. Die ungültigen Lösungen werden jedoch schrittweise verbessert, bis Lösungen entstehen, die die Restriktionen nicht verletzen. Würde man den unzulässigen Lösungen in der Fitneßfunktion den Status `Fitness::notvalid` zuweisen und sie damit aus dem Optimierungsverfahren nehmen, würde der Optimierungsprozeß lange Zeit nur zufällige Individuen erzeugen, ohne eine Verbesserung zu erreichen. Aus diesem Grund wurden die unzulässigen Individuen zwar mit einem Strafwert belegt, aber nicht als unzulässig gekennzeichnet und damit nicht automatisch aus der Population gelöscht.

4.3 Beispieloptimierungen

Die nachfolgenden Beispiele sollen zeigen, ob das Verfahren ein Optimum findet. Es werden nur die Kenngrößen Q_w und Q_g berücksichtigt.

4.3.1 Quadrat, 4 Punkte, 1 Innenpunkt

Für den einfachsten Fall, daß nur ein Innenpunkt vorgegeben wird, ist für das Beispiel 1 das globale Optimum bekannt. Das Optimum für die Kenngröße Q_w und Q_g ist dann erreicht, wenn der Innenpunkt in der Mitte des Quadrats liegt.

Die nachfolgenden Bilder zeigen das Netz vor und nach einer beispielhaften Optimierung. Die Grafik 4.9 illustriert den Verlauf der Optimierung.

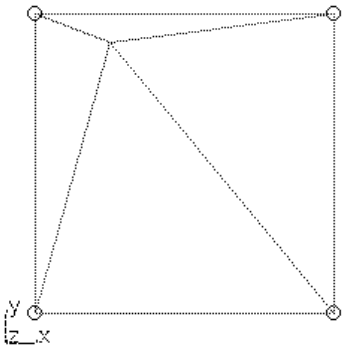


Bild 4.7: Netz vor der Optimierung, 1 Innenpunkt

Quadrat, 4 Punkte Optimierungsstrategie (15+100)-ES G_w = 10, G_g = 10 Zeit: 2,437 sec Schranke für Abbruchkriterium: 0.0010 Anzahl der Iterationen: 17		
Parameter	Wert vor Optimierung	nach Optimierung
Q_w	0.6878	0.3867
Q_g	0.7696	0.0000

Tabelle 4.4: Optimierung 1 Innenpunkt

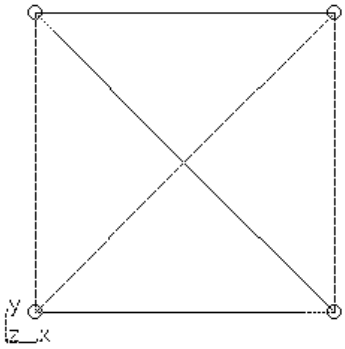


Bild 4.8: Netz nach der Optimierung, 1 Innenpunkt

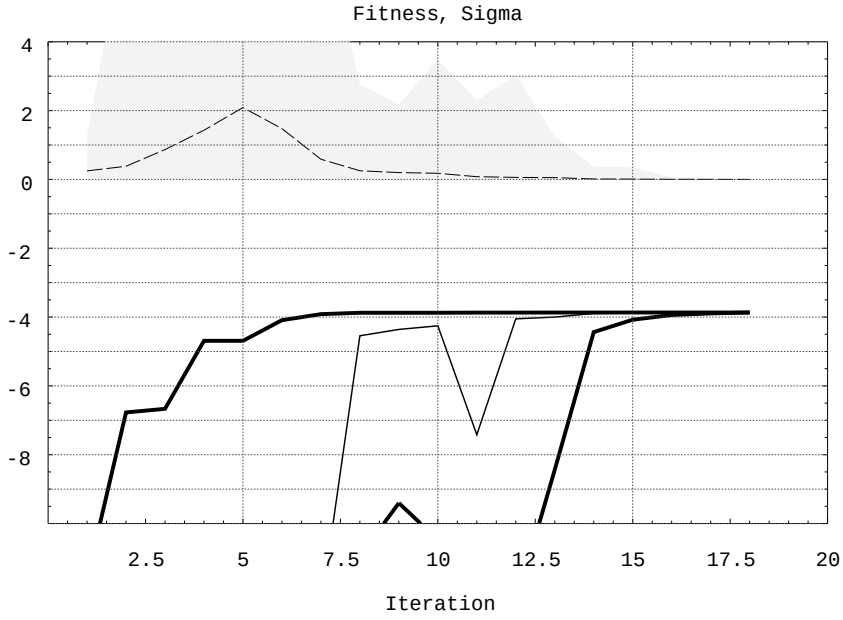


Bild 4.9: Verlauf der Optimierung, 1 Innenpunkt

4.3.2 Quadrat, 16 Punkte, 1 Innenpunkt

Das globale Optimum für das zweite Beispiel ist nicht von vornherein bekannt. Das Ergebnis hängt von der Initialvernetzung ab, da man, je nach Lage des Innenpunktes, eine anderer Topologie erhält.

Die nachfolgenden Bilder zeigen ein zufällig erzeugtes Netz vor und nach der Optimierung. Die Grafik 4.12 gibt den Verlauf der Optimierung wieder.

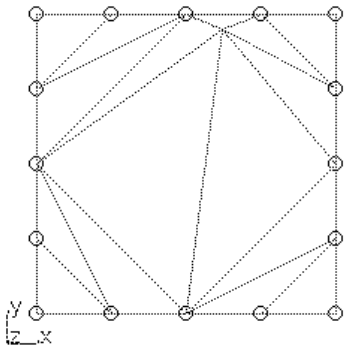


Bild 4.10: Netz vor der Optimierung, 1 Innenpunkt

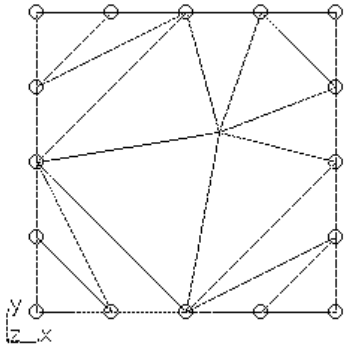


Bild 4.11: Netz nach der Optimierung, 1 Innenpunkt

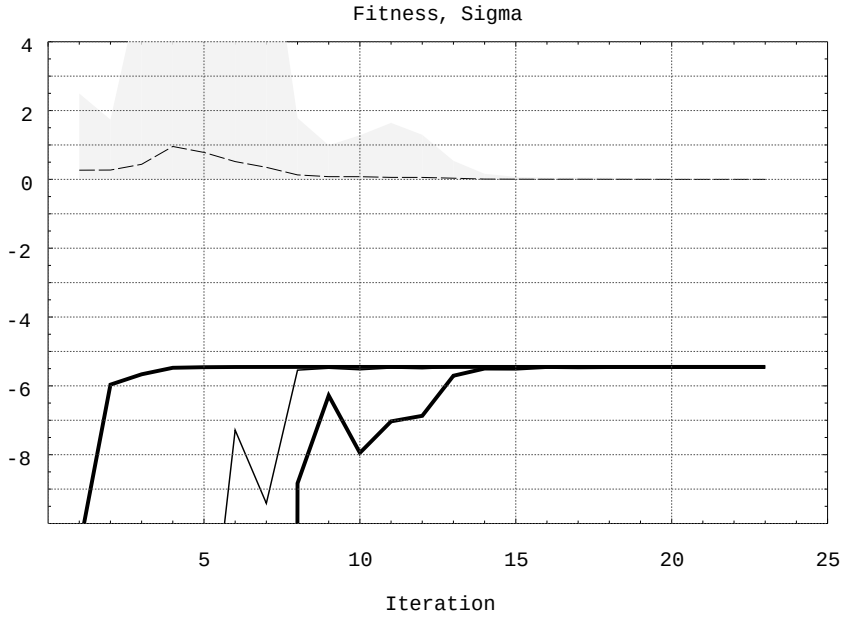


Bild 4.12: Verlauf der Optimierung, 1 Innenpunkt

Quadrat, 16 Punkte Optimierungsstrategie (15+100)-ES G_w = 10, G_g = 10 Zeit: 3,497 sec Schranke für Abbruchkriterium: 0.00010 Anzahl der Iterationen: 22		
Parameter	Wert vor Optimierung	nach Optimierung
Q_w Q_g	0.6785 1.0597	0.2917 0.2532

Tabelle 4.5: Quadrat 16 Punkte, Optimierung 1 Innenpunkt

4.3.3 Polygonal umrandetes Gebiet

Das auf den Bildern 4.13 und 4.14 gezeigte Gebiet mit einer Aussparung besitzt 69 Randpunkte und 301 zu optimierende Innenpunkte. Die Anzahl der Entwurfsvariablen beträgt somit 602.

Die Werte für die Kenngrößen Q_w und Q_g in Tabelle 4.6 zeigen, daß auf Kosten der Kenngröße Q_g die Kenngröße Q_w verbessert wurde. Dies hat mit der Gewichtung der Kenngrößen zu tun und wurde bewußt in Kauf genommen.

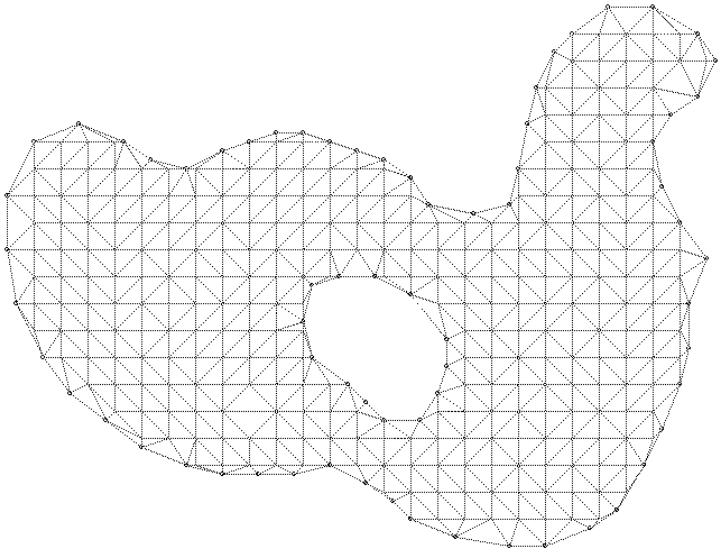


Bild 4.13: Polygonal umrandetes Gebiet, vor Optimierung, 301 Innenpunkte

Beispielfigur 3, 69 Randpunkte Optimierungsstrategie (15+100)-ES G_w = 20, G_g = 10 Zeit: 4316,497 sec Schranke für Abbruchkriterium: 0.00010 Anzahl der Iterationen: 725		
Parameter	Wert vor Optimierung	nach Optimierung
Q_w Q_g	0.4768 0.7516	0.3510 1.0997

Tabelle 4.6: Polygonal umrandetes Gebiet, 69 Randpunkte, Optimierung 301 Innenpunkte

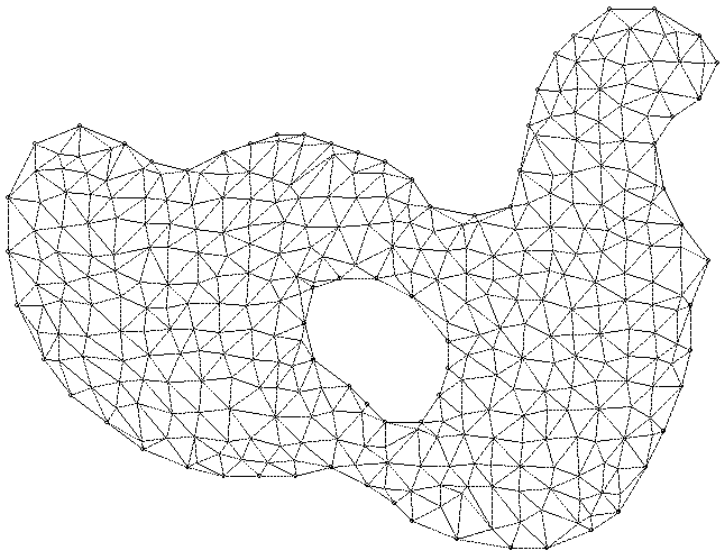


Bild 4.14: Polygonal umrandetes Gebiet, nach Optimierung, 301 Innenpunkte

4.4 Aufwand

In dieser Testreihe wird eine grobe Abschätzung für die Abhängigkeit der Anzahl der freien Punkte auf das Laufzeitverhalten gegeben und für die Anzahl der Iterationen, die nötig sind, um eine gegebene Schranke zu unterschreiten.

Das Problem beim Abschätzen des Aufwandes besteht darin, daß die Anzahl der nötigen Operationen nicht eindeutig zu bestimmen ist, da die Verfahren von Zufallszahlen abhängen.

Außerdem beeinflußt auch die Güte des Startnetzes den Verlauf der Optimierung.

Um diese Einflüsse auszugleichen, werden je 10 Testläufe für jede Anzahl von Entwurfsvariablen durchgeführt und aus den Werten für Laufzeit und die maximale Anzahl von Iterationen das arithmetische Mittel berechnet. Die Mittelwerte der Testläufe werden in den Diagrammen 4.15 und 4.16 gezeigt.

Die Ausgangsnetze für jeden dieser zehn Testläufe sind zufällig erzeugt und verschieden.

Beispielfiguren 1,2 und 3	
Optimierungsstrategie	(15+100)-ES
G_w	20
G_g	10
Schranke für Abbruchkriterium	0.0010

Tabelle 4.7: Parameter für die Abschätzung des Aufwandes

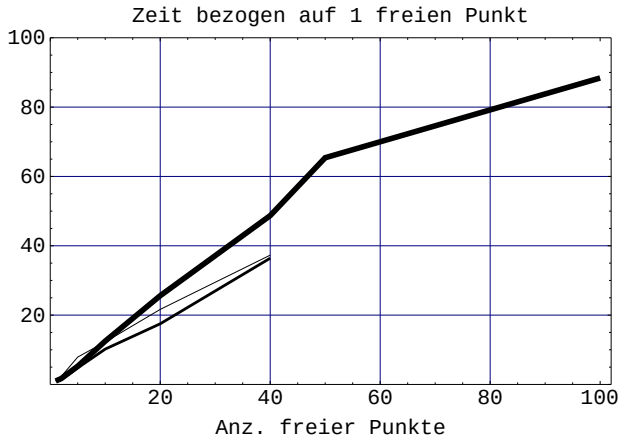


Bild 4.15: Zeitaufwand

Man kann aufgrund der oben gezeigten Diagramme abschätzen, daß der Zeitaufwand linear abhängig ist von der Anzahl der freien Punkte, die Anzahl der Iterationen ist dagegen nicht linear abhängig. Dies ist bemerkenswert und bedeutet, daß die Zeit für eine Iteration mit zunehmender Anzahl von freien Punkten abnimmt. Dies hat u.a. damit zu

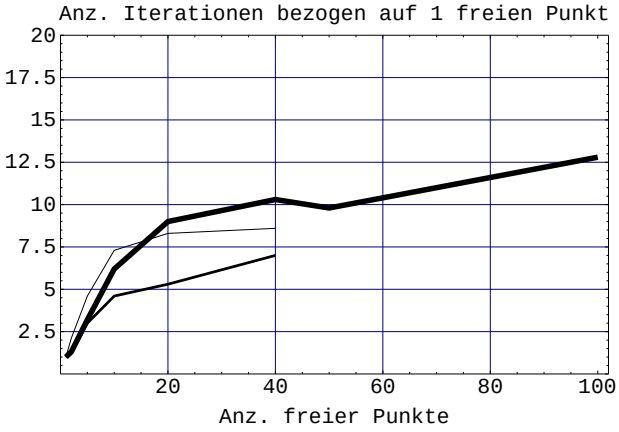


Bild 4.16: Anzahl Iterationen

tun, daß zwar die Zahl der freien Punkte zunimmt, aber die Zahl der variablen Dreiecke, deren Winkel und Flächen in jedem Iterationsschritt berechnet werden müssen, sich nicht linear erhöht.

4.5 Einfluß der Rekombinationsparameter

Für die Beispielfigur 2 wird der Einfluß der Rekombinationsstrategien untersucht. Es wird für jede Anzahl von freien Punkten je 5 Optimierungen durchgeführt und der Mittelwert für die Anzahl der Iterationen gebildet.

Auf dem Bild 4.17 sind die Ergebnisse für die vier möglichen Kombinationen zu sehen:

Beispielfigur 2		
Rekombination x	Rekombination σ	Strichstärke
intermediär	intermediär	0.01
diskret	intermediär	0.04
intermediär	diskret	0.08
diskret	diskret	0.12

Tabelle 4.8: Vergleich Rekombinationsparameter

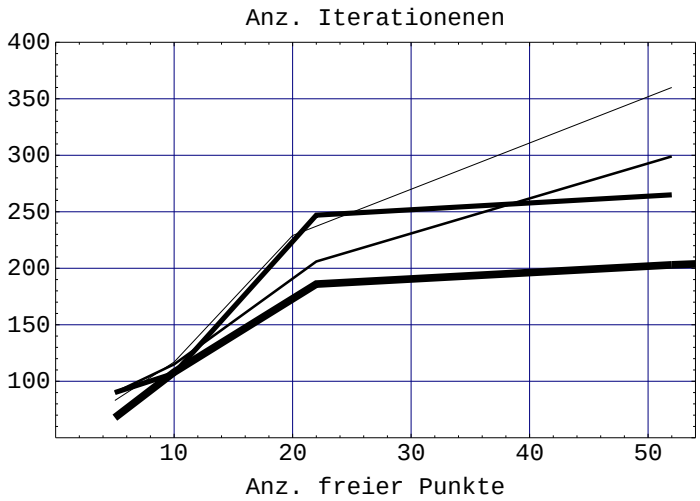


Bild 4.17: Vergleich verschiedener Rekombinationsstrategien

Um ein abschließendes Urteil geben zu können, ist die Anzahl der untersuchten Optimierungen zu klein. Es hat sich gezeigt, daß die Anzahl der Iterationen bei der Durchführung der Optimierungen großen Schwankungen unterworfen war. Trotzdem kann man qualitativ abschätzen, daß intermediäre Rekombination langsamer konvergiert.

Kapitel 5

Ausblick

Die vorliegende Arbeit zeigt, daß evolutionäre Algorithmen robuste Optimierungsverfahren sind, die zur Lösung von Problemen aus den Ingenieurwissenschaften verwendet werden können. Da die Ableitungen der Zielfunktion nicht bekannt sein müssen und beliebige Restriktionen in die Fitneßfunktion aufgenommen werden können, stellen die EA ein universelles Optimierungsverfahren für eine Vielzahl von Problemen dar.

Für die gewählten Beispiele konnten in akzeptabler Rechenzeit gute Lösungen gefunden werden, aber Optimierungen mit der Beispielfigur 3 zeigten auch, daß für eine große Anzahl von Entwurfsvariablen die Laufzeit kritisch werden kann.

Es bleibt zu untersuchen, wie sich die Rechenzeit durch das Generieren eines guten Initialnetzes verkürzen läßt. Es ist z.B. denkbar, durch die in Abschnitt 1.4.2 beschriebenen Methoden eine gute Ausgangsbasis zu schaffen, die den Aufwand für das Optimieren mit EA verkleinert.

Durch das Kombinieren der in Abschnitt 1.4.2 beschriebenen Methoden mit den EA lassen sich die Vorteile beider Verfahren verbinden. Laplacian Smoothing erzielt schnell gute Lösungen, berücksichtigt aber nicht den Einfluß der Veränderung eines Punktes auf seine Nachbarpunkte und -dreiecke. Dies kann mit einer anschließenden Optimierung mit einem EA erreicht werden, der aufgrund des verbesserten Ausgangsnetzes nur noch kurze Rechenzeit benötigt.

Eine weitere Möglichkeit zur Verbesserung des Laufzeitverhaltens stellt das Generieren von gleichwinkligen Dreiecken für das Ausgangsnetz dar. Das Verfahren zur Erzeugung eines regelmäßigen Rasters und das Unterteilen der Quadrate in rechtwinklige Dreiecke wird hierbei durch das Erzeugen von gleichwinkligen Dreiecken ersetzt. Es bleibt zu untersuchen, ob sich die Rechenzeit für die Optimierung hierdurch verkürzen und die Qualität der Ergebnisse verbessern läßt.

Bei der FE-Analyse dürfen keine Dreiecke mit negativen Flächen verwendet werden. Deshalb werden in der Funktion `fitness()` der Klasse `FEMesh` negative Flächen mit einem Strafwert belegt. Diese Bedingung kann vernachlässigt werden, wenn vor der FE-Analyse die Reihenfolge der Punkte im Dreieck zweckmäßig umgeordnet wird. Es bleibt dann nur die Restriktion, daß die Dreiecke, die eine Randkante enthalten, beim Optimierungsprozeß nicht negativ werden dürfen, da sonst der freie Punkt des Dreiecks außerhalb des zulässigen Netzgebietes liegen würde. Es bleibt zu untersuchen, wie sich diese Veränderung der Restriktionen für die Zielfunktion auf den Verlauf und das Er-

gebnis der Optimierung auswirkt.

Die Optimierung der Knotenkoordinaten von Triangulationen hat den Zweck die Ergebnisse einer FE-Analyse zu verbessern. Diese Verbesserung kann man u.U. auch durch eine Verfeinerung des Netzes erreichen. Von Interesse ist deshalb eine Abschätzung, welches der beiden Verfahren in kürzerer Zeit die besseren Ergebnisse liefert. Dazu müsste untersucht werden, wie sich die Qualität der Ergebnisse bei Veränderung der in dieser Arbeit definierten Gütekriterien ändert.

Literaturverzeichnis

[1] K.J. Bathe; Finite-Element-Methoden; Springer

[2] T. Beck, U. Hammel, H.-P. Schwefel; Modelloptimierung mit evolutionären Algorithmen; Universität Dortmund, Lehrstuhl für Systemanalyse

[3] Bronstein et al.; Taschenbuch der Mathematik; Verlag Harri Deutsch

[4] M. Bern, D. Eppstein, J. Gilbert; Provably Good Mesh Generation, Xerox Palo Alto Research Center and Department of Information and Computer Science, Univ. of California

[5] M. Bernreuther; Diskrete Optimierung mit der Evolutionsstrategie auf parallelen Systemen; Beitrag zum Tagungsband 'Forum Bauinformatik 1996'; VDI Verlag

[6] M. Berzins; Solution-Based Mesh Quality for Triangular and Tetrahedral Meshes; University of Utah

[7] J. Enseleit; Netzgenerator 2D; Institut für Bauingenieurwesen TU Berlin

[8] M. Filipiak; Mesh Generation; Edinburgh Parallel Computing Centre

[9] P.L. George; Automatic Mesh Generation; Wiley

[10] Ch. Hartmann; "Simulierte Evolution": Ein Lösungsansatz für Formfindungsprobleme?, Dissertation an der TU München

[11] M. Holder, J. Richardson; Genetic Algorithms, Another Tool for Quad Mesh Optimaziation?; Beitrag 7th Annual International Meshing Roundtable 1998

[12] A. Malanthara, W. Gerstle; Comparative Study of Unstructured Meshes Made of Triangles and Quadrilaterals; University of New Mexico

[13] M. Mitchell; An Introduction to Genetic Algorithms; MIT Press

[14] V. Nissen; Einführung in Evolutionäre Algorithmen; Vieweg

[15] S. Owen, S. Saigal; Neighborhood-Based Element Sizing Control for Finite Element Surface Meshing

[16] P.J. Pahl; Theoretische Methoden der Bau- und Verkehrstechnik II, Skript zur Lehrveranstaltung an der TU Berlin

[17] G. Prestifiloppo, J. Sprave; Optimal Triangulation by Means of Evolutionary Algorithms; Universtity of Dortmund

[18] I. Rechenberg; Evolutionsstrategie '94; Frommann-Holzboog

[19] J. Ruppert; A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation; Journal of Algorithms 18(3)

[20] G. Rudolph, H.P. Schwefel; Evolutionäre Algorithmen: Ein robustes Optimierungskonzept; Universität Dortmund, Lehrstuhl für Systemanalyse

[21] U. Schöning; Theoretische Informatik - kurzgefaßt; Spektrum Akademischer Verlag

[22] M. Schütz; Eine Evolutionsstrategie für gemischt-ganzzahlige Optimierungsprobleme mit variabler Dimension; Diplomarbeit am Lehrstuhl für Systemanalyse, Universität Dortmund

[23] J.R. Shewchuk; Triangle: Engineering a 2D Quality Mesh Generator and Delauny Triangulatior; Carnegie Mellon University

[24] J. Sprave, H.P. Schwefel; Evolutionäre Algorithmen auf Transputerfarmen zur Lösung schwieriger Optimierungsprobleme; Universität Dortmund, Lehrstuhl für Systemanalyse

[25] A. Steuer; Natürlich Optimiert; Beitrag iX Januar 1997; Heise-Verlag

[26] B. Stroustrup; Die C++ Programmiersprache; 2. Auflage; Addison-Wesley

Index