



Thomas Fober

**Experimentelle Analyse
Evolutionärer Algorithmen
auf dem CEC 2005
Testfunktionensatz**

Diplomarbeit

01.07.2006

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl XI
(Algorithm Engineering und Systemanalyse)
Fachbereich Informatik
Universität Dortmund

Gutachter:

Prof. Dr. Günter Rudolph
Dr. Thomas Bartz-Beielstein

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Motivation und Einleitung | 1 |
| 1.1 | Aufbau dieser Arbeit | 1 |
| 2 | Optimierung | 3 |
| 2.1 | Problembeschreibung | 3 |
| 2.1.1 | Beispiele | 4 |
| 2.1.2 | Auswahl spezieller Lösungsverfahren | 5 |
| 2.2 | Reellwertige Optimierungsprobleme | 6 |
| 2.2.1 | Analytisches Suchverfahren | 9 |
| 2.2.2 | CEC 2005 Testfunktionensatz | 11 |
| 2.3 | Zusammenfassung | 13 |
| 3 | Algorithmen | 15 |
| 3.1 | Hillclimbing Verfahren | 15 |
| 3.1.1 | Regula-falsi | 16 |
| 3.1.2 | Fibonacci-Suche | 16 |
| 3.1.3 | Suche entlang der Koordinatenachsen | 18 |
| 3.1.4 | Hooke-Jeeves Pattern Search | 19 |
| 3.1.5 | Nelder-Mead Simplex | 19 |
| 3.2 | Einfache stochastische Suche | 20 |
| 3.3 | Evolutionäre Algorithmen | 21 |
| 3.3.1 | Evolutionsstrategien | 23 |
| 3.3.2 | Genetische Algorithmen | 36 |
| 3.4 | Zusammenfassung | 46 |
| 4 | Matlab | 48 |
| 4.1 | Genetic Algorithm and Direct Search Toolbox | 48 |
| 4.1.1 | Kurze Einführung in die GADS Toolbox | 49 |
| 4.1.2 | Arbeitsweise des GA* | 51 |
| 4.2 | $(\mu, \kappa, \lambda, \rho)$ -Evolutionsstrategie | 52 |
| 4.2.1 | Implementierung | 52 |
| 4.2.2 | Verifikation | 63 |
| 4.2.3 | Validierung | 67 |
| 4.3 | Implementierung einer einfachen stochastischen Suche | 70 |
| 5 | Experimente | 72 |
| 5.1 | Eine „Historie“ experimenteller Untersuchungen der EA | 72 |
| 5.2 | Der neue Experimentalismus | 73 |
| 5.2.1 | Vorüberlegungen | 73 |
| 5.2.2 | Optimierung der Steuerparameter | 75 |
| 5.2.3 | Objektive Interpretierung der Ergebnisse | 79 |
| 5.3 | Parameteroptimierung | 81 |
| 5.3.1 | Anpassung der ES auf die 10-dimensionale sphere -Funktion | 81 |

| | | |
|----------|--|------------|
| 5.3.2 | Anpassung des GA auf die 10-dimensionale sphere -Funktion | 86 |
| 5.3.3 | Optimierung der ES-Parametrisierung auf den übrigen Problem instanzen | 88 |
| 5.3.4 | Optimierung der GA-Parametrisierung auf den übrigen Problem instanzen | 89 |
| 5.4 | Bewertung unterschiedlicher Suchalgorithmen | 89 |
| 5.4.1 | Berechnung der Fehler | 90 |
| 5.4.2 | Benötigte Funktionsauswertungen | 92 |
| 5.4.3 | Run-length distribution | 92 |
| 5.4.4 | Komplexität der Algorithmen | 92 |
| 6 | Interpretation der Experimente | 95 |
| 6.1 | ES-Parametrisierung | 95 |
| 6.1.1 | Parametrisierung der sphere -Funktion | 95 |
| 6.1.2 | Parametrisierung der Rosenbrock -Funktion | 96 |
| 6.1.3 | Parametrisierung der Rastrigin -Funktion | 96 |
| 6.2 | GA-Parametrisierung | 96 |
| 6.3 | Vergleich zwischen ES und GA | 97 |
| 7 | Zusammenfassung und Ausblick | 98 |
| 7.1 | Zusammenfassung | 98 |
| 7.2 | Ausblick | 98 |
| | Literaturverzeichnis | 102 |
| | Abbildungsverzeichnis | 103 |
| | Tabellenverzeichnis | 104 |
| | Algorithmenverzeichnis | 105 |
| | Abkürzungsverzeichnis | 106 |
| | Verzeichnis der verwendeten Symbole | 107 |

1 Motivation und Einleitung

Evolutionäre Algorithmen sind leistungsfähige Optimierverfahren, die auf Prinzipien der biologischen Evolution basieren. Zwei Vertreter dieser Algorithmen sind die *Evolutionsstrategien* und die *genetischen Algorithmen*. Erstere wurden von I. Rechenberg und H.-P. Schwefel in Berlin entwickelt, letztere von J. Holland in Michigan. Entsprechend der Herkunft sind Evolutionsstrategien vor allem im europäischen Raum bekannt, genetische Algorithmen hingegen erfreuen sich weltweiter Bekanntheit.

Beide Verfahren lassen sich über mehr oder weniger viele Parameter einstellen. Kennern beider Verfahren wird bewusst sein, dass mit unterschiedlichen Einstellungen unterschiedlichste Ergebnisse erreicht werden können. Umso erstaunlicher sind einige Vergleiche zwischen Evolutionsstrategie und genetischen Algorithmen, in denen beide Algorithmen scheinbar willkürlich eingestellt werden. Ergebnis solcher Untersuchungen sind dann Aussagen wie: „Evolutionsstrategien sind vorzuziehen, wenn unimodale Funktionen optimiert werden, genetischen Algorithmen hingegen bei der Optimierung multimodaler Funktionen.“

Diese Arbeit soll einen objektiven Vergleich beider Verfahren ermöglichen. Dazu bedarf es zweier Schritte: Im ersten Schritt wird zu einer gegebenen Probleminstanz eine optimale Einstellung der Algorithmen bestimmt. Im zweiten Schritt werden dann beide Verfahren auf derselben Probleminstanz verglichen. Evolutionäre Algorithmen sind zwar leicht verständliche direkte Suchverfahren, sie können aber auch sehr rechenaufwendig werden (beispielsweise, wenn die Populationsgröße sehr groß gewählt wird). In einem weiteren Test soll deshalb untersucht werden, ob dieser Aufwand berechtigt ist. Dazu wird ein sehr einfaches stochastisches Suchverfahren betrachtet, welches nur auf einem Lösungspunkt arbeitet und praktisch parameterfrei ist. Ziel ist es jedoch nicht ausschließlich Datensätze zu erzeugen. Vielmehr sollen die Ergebnisse der Experimente auch erklärt werden.

Die Implementierung einer Evolutionsstrategie ist ein weiteres Ziel dieser Diplomarbeit, die in Matlab erfolgen soll. Matlab stellt bereits eine Toolbox zu Verfügung, in der genetische Algorithmen implementiert sind. Durch Integration der Evolutionsstrategien in diese Toolbox wird ein Paket geschaffen, in dem die zwei bekanntesten Arten evolutionärer Algorithmen vereint sind. So kann es gelingen, dass die Evolutionsstrategien auch über den europäischen Raum hinweg bekannter werden.

1.1 Aufbau dieser Arbeit

Im *Kapitel 2* werden Optimierungsprobleme definiert und weiter unterteilt. Der Schwerpunkt wird dann auf der reellwertigen Parameteroptimierung liegen. Diese Art Optimierungsproblem kann eine Menge an Eigenschaften besitzen, die in diesem Kapitel eingeführt werden. Aus einigen dieser Eigenschaften leitet die Mathematik eine notwendige und eine hinreichende Bedingung für ein Optimum ab. Auf beiden Bedingungen basiert ein analytisches Verfahren, das das Optimum einer Funktion bestimmt. Dieses Verfahren wird auch hier vorgestellt.

Zum Vergleich von Evolutionsstrategien und genetischer Algorithmen bedarf es geeigneter Probleme, auf denen diese Algorithmen angewendet werden können. Diese Probleme sollten möglichst heterogen sein, also möglichst viele und unterschiedliche Eigenschaften aufweisen. Eine Auswahl solcher Testprobleme liefert der CEC 2005 Testfunktionensatz, der ebenfalls in diesem Kapitel vorgestellt wird.

Das *Kapitel 3* führt anschließend Algorithmen ein, die zu Optimierung reellwertiger Parameter eingesetzt werden können. Das sind zum einen die Hillclimbing Verfahren und zum anderen stochastische Suchverfahren, zu denen auch die evolutionären Algorithmen gehören. Hillclimbing Verfahren werden nur kurz eingeführt und es werden ihre Schwächen aufgezeigt. Im zweiten Teil des Kapitels wird eine besonders einfache Art der stochastischen Suche eingeführt, die, wie sich im weiteren Verlauf dieser Arbeit zeigen wird, zu nicht besonders guten Ergebnissen führen wird. Deshalb werden im letzten Teil evolutionäre Algorithmen ausführlich eingeführt. Diese Algorithmen gelten als besonders leistungsfähig und lassen sich über eine größere Anzahl an Steuerparametern an eine Problemstellung anpassen. Die Einstellung dieser Parameter hat dann großen Einfluss auf das Ergebnis der Optimierung, so dass die Parametereinstellung für jede neue Problemstellung kritisch zu betrachten ist. Ein Ziel der Diplomarbeit wird es sein, eine, im Vergleich zur Standardeinstellung, bessere Parametrisierung der evolutionären Algorithmen für ausgewählte Problemstellungen zu ermitteln.

Im *vierten Kapitel* wird schließlich die benötigte Experimentierumgebung geschaffen und so ein essentieller Teil der Diplomarbeit bearbeitet. Wie bereits oben beschrieben, liefert Matlab über die *Genetic algorithm and direct search toolbox* einen genetischen Algorithmus. Diese Toolbox soll so erweitert werden, dass sie auch die Ausführung einer Evolutionsstrategie oder einer einfachen stochastischen Suche ermöglicht. So wird die gesamte Software, die einer experimentellen Betrachtung unterzogen werden soll, in einer einzigen Toolbox zusammengefasst. Über eine solche universelle Toolbox soll die Evolutionsstrategie mit der Zeit auch einen größeren „Markt“ finden.

Anschließend können im *Kapitel 5* die Experimente durchgeführt werden, die sich in zwei Teile aufspalten. In einem ersten Teil, wird für Evolutionsstrategien, als auch genetische Algorithmen eine optimale Einstellung gesucht. Dazu wird eine Teilmenge der im CEC 2005 Testfunktionensatz enthaltenen Probleme ausgewählt. Auf diesen Problemen und der gefundenen optimalen Einstellung wird im zweiten Teil ein objektiver Vergleich zwischen Evolutionsstrategie, genetischem Algorithmus und einer einfachen stochastischen Suche ermöglicht. Die so gewonnenen Resultate werden danach im *Kapitel 6* interpretiert.

Das *Kapitel 7* fasst schließlich die Ergebnisse dieser Arbeit zusammen und gibt einen weiteren Ausblick in die Thematik.

Iserlohn / Dortmund im Juni 2006

Thomas Fober

2 Optimierung

Die Optimierung ist ein Suchproblem, das sich in vielen Bereichen wieder findet und von uns unbewusst mehrmals täglich zu lösen ist (beispielsweise bei der Bestimmung einer Wegstrecke, bei der man schnellstmöglich am Ziel ankommen möchte, oder der Planung eines Arbeitsablaufes, wie dieser Diplomarbeit, dessen Fertigstellung frühest- und bestmöglich passieren sollte). In diesem Kapitel werden Optimierungsprobleme zu Beginn formal beschrieben und klassifiziert. Anschließend werden Optimalitätsbedingungen und weitere Eigenschaften von Optimierungsproblemen angegeben, auf denen einige Lösungsverfahren ansetzen. Abschließend werden spezielle Funktionen vorgestellt, die sich zur experimentellen Analyse von Optimierungsalgorithmen besonders gut eignen.

2.1 Problembeschreibung

Eine Optimierungsaufgabe ist dann zu lösen, wenn eine Problemstellung mehrere, oder gar unendlich viele Alternativen zur Lösung bietet. Dann wird stets versucht die beste Alternative zu wählen. Neben einer Menge an Entscheidungsalternativen benötigt ein Optimierungsproblem somit ein Bewertungsmaß und eine Funktion, die eine Entscheidungsalternative auf eine Bewertung abbildet.

Formal ist ein Optimierungsproblem gegeben durch einen Suchraum X , ein Bewertungsmaß Y und eine Bewertungsfunktion

$$f : X \rightarrow Y .$$

Ziel der Optimierung ist das Auffinden eines $x \in X$, für das f einen optimalen Wert liefert. Gewöhnlich wird dies entweder der maximale Wert (Maximierungsproblem) oder der minimale Wert (Minimierungsproblem) sein.

Optimierungsprobleme können unterschiedlich klassifiziert werden. Je nach Suchraum kann z.B. ein *reellwertiges*, *ganzzahliges* oder *kombinatorisches* Problem vorliegen. Reellwertige oder ganzzahlige Optimierungsprobleme suchen für eine Menge Parameter eine optimale Belegung. Der Suchraum bei n Parametern ist somit der n -dimensionale reelle oder ganzzahlige Raum. Zudem kann der Suchraum solcher Probleme durch Nebenbedingungen der Form $g_j(x) \leq 0$ weiter beschränkt werden, so dass das Optimierungsproblem als restringiert bezeichnet wird. Die Form $g_j(x) \leq 0$ erlaubt das Verwenden aller (zu einem geschlossenen Bereich führenden) Nebenbedingungen, denn es gilt:

- $g_j(x) \geq 0 \Leftrightarrow -g_j(x) \leq 0$
- $g_j(x) = 0 \Leftrightarrow -g_j(x) \leq 0 \wedge g_j(x) \leq 0$
- $g_j(x) \leq a \Leftrightarrow g_j(x) - a \leq 0$ mit $a \in \mathbb{R}$.

Neben den reinen Formen existieren auch gemischte Optimierungsprobleme, wie beispielsweise gemischt ganzzahlige Probleme, bei denen unterschiedliche Variablen ganzzahlig oder reell sein müssen.

Ist an Stelle einer optimalen Parameterbelegung eine Menge oder Ordnung (Struktur) aus einer endlichen Anzahl von Objekten gesucht, so handelt es sich um ein *kombinatorisches* Optimierungsproblem.

Weiter kann man Optimierungsprobleme nach der Bewertungsfunktion unterscheiden. Ist der funktionale Zusammenhang zwischen dem Lösungsraum X und dem Bewertungsmaß Y nicht bekannt, so handelt es sich um eine *experimentelle* Optimierung (Schwefel, 1994). Dann müssen verschiedenen Entscheidungsalternativen am realen System oder einem Modell simuliert werden, um eine Bewertung für diese Einstellungen zu erhalten. Zudem können Optimierungsprobleme *verrauscht* sein, wenn die Bewertungsfunktion (oder die Simulation) nicht die exakten Werte, sondern einen gestörten zurückgibt. Ein *dynamisches* Optimierungsproblem liegt vor, wenn die Bewertung nicht nur von der Lösung, sondern auch von der Zeit abhängt. Schwefel (1994) gibt eine mögliche Ursache für dynamische Optimierungsprobleme an. Diese können entstehen, wenn nicht alle Parameter kontrolliert werden können und sich die nicht kontrollierbaren Parameter mit der Zeit ändern. Verfolgt die Optimierung mehrere Ziele, so spricht man von einer *multikriteriellen* Optimierung. Die Bewertungsfunktion setzt sich dann aus m einzelnen Funktionen f_i zusammen und Ziel der Optimierung ist das Auffinden einer Menge pareto-optimaler Lösungen $P = \{p \mid p \in X\}$, so dass für jeden Punkt $p \in P$ gilt:

$$\nexists p' \in X : \forall i \in \{1, \dots, m\} : f_i(p') \leq f_i(p) \wedge \exists k \in \{1, \dots, m\} : f_k(p') \prec f_k(p),$$

wobei $a \prec b$ bedeutet, dass a besser ist als b , und $a \leq b$, dass a besser oder gleich b ist. P enthält nur solche Punkte, die nicht von anderen Punkten übertroffen (dominiert) werden, also objektiv nicht vergleichbar sind. Im Allgemeinen besitzen multikriterielle Optimierungsprobleme mehr als nur einen pareto-optimalen Punkt, so dass die endgültige Lösung nur subjektiv ermittelt werden kann.

Ein *lineares* Optimierungsproblem liegt schließlich vor, wenn sowohl alle Nebenbedingungen, als auch die Zielfunktion linear sind, ansonsten heißen sie *nichtlinear*.

2.1.1 Beispiele

Beispiel 2.1.1 *In der Betriebswirtschaft werden Produktionsprogramme gesucht, in denen aus m Ressourcen n Güter so herzustellen sind, dass der Erlös maximiert wird. Das zugehörige Maximierungsproblem hätte dann folgendes Aussehen:*

$$\begin{array}{ll} X = \mathbb{N}^n & \text{Welche Mengen sollen produziert werden?} \\ Y = \mathbb{R} & \text{Erlös} \\ f : \mathbb{N}^n \rightarrow \mathbb{R} & \text{mit } f(\mathbf{x}) = \mathbf{c}^T \cdot \mathbf{x} \text{ und } c_i \text{ ist der Preis für das } i\text{-te Produkt} \end{array}$$

Dieses Optimierungsproblem ist sehr leicht zu lösen. Je mehr produziert wird, desto mehr Erlös wird eingenommen. In der Realität sind die für die Produktion benötigten Ressourcen beschränkt, so dass X weiter eingeschränkt werden muss. Dann muss

$$\forall \text{ beschränkten Ressourcen } i: \mathbf{b}_i \cdot \mathbf{x} \leq B_i$$

gelten. Der Vektor \mathbf{b}_i gibt in der j -ten Komponente an, wie viele Mengeneinheiten von Ressource i durch eine produzierte Mengeneinheit des Produktes j verbraucht werden, so dass $\mathbf{b}_i \cdot \mathbf{x}$ den Gesamtverbrauch angibt, der maximal die vorhandene Menge B_i der Ressource i erreichen darf.

Da die Zielfunktion, wie auch die Nebenbedingungen linear sind, handelt es sich bei diesem Beispiel um ein lineares Optimierungsproblem.

■

Beispiel 2.1.2 Beim *Traveling-Salesperson-Problem (TSP)* ist ein kantenbewerteter Graph $G = (V, E)$ mit $|V| = n$, $E = V \times V$ und $c : E \rightarrow \mathbb{R}$ gegeben, dessen n Knoten genau einmal besucht werden dürfen und das aufsummierte Gewicht der dabei durchlaufenden Kanten minimiert wird. Mögliche Lösungen können hier nur als Permutation der Knoten angegeben werden, so dass der Lösungsraum wie folgt dargestellt werden kann:

$$X = \{(k_1, \dots, k_n) \in \{1, \dots, n\}^n \mid \forall i, j \in \{1, \dots, n\} \wedge i \neq j : k_i \neq k_j\}$$

Die Zielfunktion $f : X \rightarrow \mathbb{R}$, die es zu minimieren gilt, muss die besuchten Kanten aufsummieren, so dass gilt:

$$f(x) = \sum_{i=1}^n c((k_i, k_{i+1})) \quad \text{mit } k_{n+1} = k_1 \quad \text{und } c((i, j)) = \infty \text{ für } (i, j) \notin E$$

Aufgrund des Suchraums handelt es sich bei diesem Problem um ein kombinatorisches Optimierungsproblem. ■

Beispiel 2.1.3 Ziel der Regression ist die Bestimmung eines funktionalen Zusammenhangs zu einer gegebenen Menge Ein-/Ausgabepunkte (\bar{x}_i, \bar{y}_i) . Im Fall eines linearen Modells $y(x) = a \cdot x + b$ sind die Koeffizienten $(a, b) \in \mathbb{R}^2$ so zu bestimmen, dass der Fehler minimiert wird. Die Zielfunktion kann somit den quadratischen Fehler $(y(\bar{x}_i) - \bar{y}_i)^2$ aufsummieren und es gilt die Summe zu minimieren.

Dieses Optimierungsproblem ist unbeschränkt. Die zwei Parameter a und b können mit reellen Werten belegt werden, so dass es sich hier um ein reellwertiges Optimierungsproblem handelt. ■

2.1.2 Auswahl spezieller Lösungsverfahren

Je nach Optimierungsproblem können und müssen verschiedene Lösungsverfahren eingesetzt werden, da es nicht das eine universelle Verfahren zur Lösung von Optimierungsproblemen gibt. Exemplarisch sollen hier Lösungsverfahren für die oben genannten Beispiele skizziert werden. Lineare Optimierungsprobleme, wie das aus Beispiel 2.1.1, lassen sich mit dem Simplexverfahren (Neumann & Morlock, 1993) lösen. Dieses Verfahren macht sich die Eigenschaft linearer Optimierungsverfahren zu nutze, dass die optimale Lösung stets auf einem Eckpunkt des durch die Nebenbedingungen aufgespannten Lösungspolyhedrons liegt. Zum Finden einer optimalen Lösung läuft das Simplexverfahren solche Kanten des Lösungspolyhedrons ab, die zu verbesserten Funktionswerten führen (verbessernde Kanten), bis es in einem Eckpunkt keine verbessernde Kante mehr gibt. Dann ist mit diesem Punkt die optimale Lösung gefunden. Das Simplexverfahren wurde von Gomory so erweitert, dass auch ganzzahlige lineare Optimierungsprobleme gelöst werden können (Neumann & Morlock, 1993). Sollte das Simplexverfahren eine nicht ganzzahlige Lösung x^* liefern, fügt Gomory dem linearen Optimierungsproblem so eine neue Nebenbedingung hinzu, dass x^* nicht mehr zulässig ist, der Lösungsraum des ganzzahligen linearen Problems aber nicht verändert wird. Auf diesem neuen linearen Optimierungsproblem kann das Simplexverfahren erneut angewendet werden. Dieser Vorgang wird bis zum Auffinden einer ganzzahligen Lösung fortgeführt.

Für das TSP aus Beispiel 2.1.2 kann es nach heutigem Stand kein effizientes Lösungsverfahren geben, da das Entscheidungsproblem des TSP *NP*-vollständig ist. *NP* leitet sich vom englischen *Nondeterministic Polynomial time* ab und die gleichnamige Komplexitätsklasse beinhaltet solche Probleme, die sich, stark vereinfacht dargestellt, dann in polynomieller Zeit lösen lassen,

wenn ein Orakel existiert, das stets den richtigen Rechenschritt vorgibt. Ein solches Orakel ist in der Realität nicht gegeben. Deshalb kann eine optimale Lösung für NP -vollständige Probleme nur ermittelt werden, in dem alle Entscheidungsalternativen bewertet, und die beste ausgewählt wird. Im Fall des TSP mit n Knoten müssten somit $n!$ mögliche Lösungen betrachtet werden. Effiziente Lösungen können hier nur Approximationsalgorithmen liefern, deren Güte aber den der optimalen Lösung um maximal einen Faktor r verfehlen.

Lösungsverfahren für das in Beispiel 2.1.3 beschriebene reellwertige Optimierungsproblem werden schließlich in Abschnitt 2.2.1 und Kapitel 3 ausführlich behandelt. Zuvor werden reellwertige Optimierungsprobleme genauer beschrieben und besondere Eigenschaften angegeben, auf denen einige der im Kapitel 3 beschriebenen Verfahren ansetzen.

2.2 Reellwertige Optimierungsprobleme

Ein reellwertiges Optimierungsproblem liegt vor, wenn n Parameter so mit reellen Werten belegt werden sollen, dass eine maximale oder minimale Bewertung erreicht wird. Ist der funktionale Zusammenhang zwischen Suchraum $X \subseteq \mathbb{R}^n$ und Bewertungsmaß \mathbb{R} bekannt, so hat man das Optimierungsproblem gegeben als

$$f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$$

mit der Zielfunktion f . Ziel der Optimierung ist das Auffinden eines $x^* \in X$ für den f einen minimalen oder maximalen (oder den gewünschten) Wert liefert. Für einige Optimierverfahren ist der Begriff der *zulässigen Lösung* interessant. Eine zulässige Lösung ist dann gegeben, wenn ein (nicht unbedingt optimaler) Lösungspunkt im Suchraum (auch *zulässiger Bereich* genannt) liegt.

Im folgenden werden nur noch Minimierungsprobleme betrachtet. Dies ist keine wesentliche Einschränkung, da gilt:

$$x^* \text{ minimiert } f \Leftrightarrow x^* \text{ maximiert } -f .$$

Definition 2.2.1 (Globales Minimum)

Sei $x^* \in X \subseteq \mathbb{R}^n$. x^* heißt globaler Minimalpunkt, wenn gilt:

$$\forall x \in X : f(x^*) \leq f(x) .$$

■

Definition 2.2.2 (Lokales Minimum)

$x' \in X \subseteq \mathbb{R}^n$ heißt lokaler Minimalpunkt, wenn gilt:

$$\exists \epsilon > 0 : \forall x \in U_\epsilon(x') \cap X : f(x') \leq f(x)$$

mit

$$U_\epsilon(x') := \left\{ x \in \mathbb{R}^n \mid \sqrt{\sum_{i=1}^n (x_i - x'_i)^2} < \epsilon \right\} .$$

■

Ein x^* ist somit globaler Minimalpunkt, wenn es kein anderes Element im Suchraum gibt, das auf einen kleineren Funktionswert abgebildet wird. Erfüllt an Stelle des gesamten Suchraums nur eine kleine Umgebung (ϵ -Umgebung) um den Punkt x diese Anforderung, so liegt nur ein lokaler Minimalpunkt vor.

Die Abbildung 2.1 stellt die verschiedenen Minima einer Funktion $f : [0, 17] \rightarrow \mathbb{R}$ dar. Sie hat in

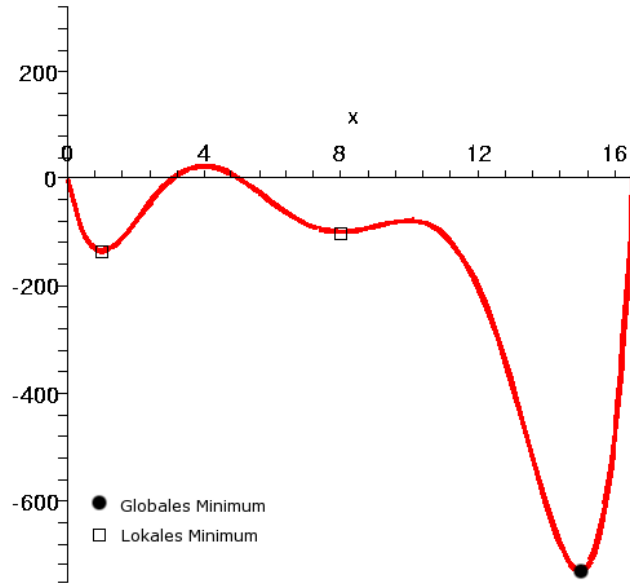


Abbildung 2.1: Minima einer Funktion

$x = 15$ die globale Minimalstelle und zwei lokale Minimalstellen in $x = 1$ und $x = 8$. Die Stelle $x = 15$ ist ein globales Minimum, da keine anderes $x \in [0, 17]$ auf einem geringeren Funktionswert abgebildet wird. Die Stellen $x = 1$ und $x = 8$ stellen hingegen nur lokale Minima da, da sie in einer ϵ -Umgebung (z.B. $\epsilon = 0.1$) zwar die kleinsten Funktionswerte aufweisen, insgesamt aber andere Stellen existieren, die auf noch kleineren Funktionswerten abgebildet werden. Die meisten Lösungsverfahren können nicht zwischen lokalem und globalem Minimum unterscheiden. Aus diesem Grund sind konvexe Funktionen interessant, deren lokale und globale Minima zusammenfallen.

Definition 2.2.3 (Konvexe Funktion)

Eine Funktion $f : X \rightarrow \mathbb{R}$ heißt konvex, wenn X eine konvexe Menge ist und gilt:

$$f(\lambda \cdot x_1 + (1 - \lambda) \cdot x_2) \leq \lambda \cdot f(x_1) + (1 - \lambda) \cdot f(x_2) .$$

Sie heißt streng konvex, wenn die Ungleichung strikt erfüllt wird. ■

Die Ungleichung kann nur dann erfüllt werden, wenn f' monoton steigend (bzw. im strikt konvexen Fall streng monoton steigend ist). Dann müssen aber die lokalen und globalen Minima zusammenfallen (Neumann & Morlock, 1993). Jede streng konvexe Funktion ist zudem unimodal. Eine Funktion f heißt auf einem Intervall $[a, b]$ unimodal, wenn ein Punkt x^* existiert, so dass f in $[a, x^*]$ streng monoton fallend und in $[x^*, b]$ streng monoton steigend ist. Dann ist x^* die einzige Minimalstelle (Neumann & Morlock, 1993).

Ist der Suchraum X eines reellen Optimierungsproblems beschränkt, so ist der Begriff der zulässigen Richtung interessant.

Definition 2.2.4 (Zulässige Richtung; Neumann & Morlock, 1993)

Ein Vektor $z \in \mathbb{R}^n$ heißt zulässige Richtung im Punkt \bar{x} , wenn ein $\delta' > 0$ existiert, so dass

$\bar{x} + \delta \cdot z \in X$ für $0 \leq \delta \leq \delta'$ gilt. Die Menge aller zulässigen Richtungen im Punkt \bar{x} wird mit $Z(\bar{x})$ bezeichnet. ■

Die Menge $Z(\bar{x})$ enthält alle Richtungen, in die man den Lösungspunkt (wenn auch noch so gering) bewegen kann, ohne dass der Lösungsraum verlassen wird. Liegt der Punkt \bar{x} im Inneren des Lösungsraums X , so ist $Z(\bar{x}) = \mathbb{R}^n$ und alle Richtungen sind zulässig. Liegt der Punkt auf dem Rand des Lösungsraum X , so enthält $Z(\bar{x})$ nur solche z , die in den Lösungsraum X zeigen. Für unbeschränkte Optimierungsprobleme mit Suchraum $X = \mathbb{R}^n$ gilt daher in allen Punkten $Z(\bar{x}) = \mathbb{R}^n$, da jeder Punkt ein innerer Punkt und somit eine zulässige Lösung ist.

Ist die Funktion f einmal stetig differenzierbar, so kann der Gradient berechnet werden, der die Richtung des lokal steilsten Anstiegs in einem Punkt x angibt. Einige Lösungsverfahren benutzen diese Information und können so schneller zum Optimum konvergieren (Schwefel, 1994). Analytische Verfahren leiten aus dem Gradienten sogar eine notwendige Bedingung für einen Minimalpunkt ab und können so das Minimum in einem Schritt finden.

Definition 2.2.5 (Gradient)

Der Gradient einer Funktion f in einem Punkt x ist definiert als:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix}$$

mit der partiellen Ableitung $\frac{\partial f}{\partial x_i}$ nach der Variablen i . ■

Der Gradient $\nabla f(x)$ zeigt ausgehend vom Punkt x in Richtung des lokal stärksten Anstiegs, folglich $-\nabla f(x)$ in Richtung des lokal steilsten Abstiegs. Eine Lösung kann ausgehend vom Punkt x verbessert werden, wenn man diesen Punkt entgegen der Richtung $\nabla f(x)$ verschiebt. Dies ist nur dann möglich, wenn diese Richtung in der Menge der zulässigen Richtungen ist, da die so entstehende Lösung sonst nicht mehr zulässig wäre.

Formal muss das Skalarprodukt $z^T \nabla f(x) < 0$ sein. Dann bilden die beiden Vektoren z und $\nabla f(x)$ einen stumpfen Winkel und zeigen in Anteilen in entgegengesetzte Richtung.

Eine notwendige Bedingung für einen lokalen Minimalpunkt x^* ist somit erfüllt, wenn es keine zulässige Richtung gibt, in die der Lösungspunkt so bewegt werden kann, dass der Funktionswert weiter minimiert wird. Dann muss aber $z^T \nabla f(x^*) \geq 0 \quad \forall z \in Z(x^*)$ gelten. Ist der Lösungsraum unbeschränkt, so ist ausreichend, dass der Gradient in x^* verschwindet, weil hier alle Richtungen zulässig sind.

Die soeben beschriebene Bedingung bildet nur eine notwendige Optimalitätsbedingung. Tatsächlich besitzen beispielsweise auch Wendestellen einen verschwindenden Gradienten. Punkte, die die notwendige Optimalitätsbedingung erfüllen, werden als stationäre Punkte bezeichnet. Zur Formulierung einer notwendigen Bedingung kann die Hessematrix und der Begriff der positiven Definitheit herangezogen werden. Zur Bestimmung der Hessematrix einer Funktion f muss diese Funktion zweimal stetig differenzierbar sein.

Definition 2.2.6 (Hesse-Matrix)

An der Stelle x ist die Hessematrix der Funktion f definiert als

$$H_f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix}$$

■

Definition 2.2.7 (Definitheit)

Sei A eine symmetrische $n \times n$ Matrix und $x \in \mathbb{R}^n$, $x \neq \mathbf{0}$. A ist

- positiv definit, wenn $xAx^T > 0$
- positiv semidefinit, wenn $xAx^T \geq 0$
- negativ definit, wenn $xAx^T < 0$
- negativ semidefinit, wenn $xAx^T \leq 0$
- sonst indefinit.

■

Ist x^* ein stationärer Punkt von f und $H_f(x^*)$ positiv definit (Neumann & Morlock (1993) stellen Verfahren vor, mit denen Matrizen auf Definitheit hin überprüft werden können), so ist x^* ein lokales Minimum. Ist zudem die Funktion konvex, so liegt ein globales Minimum in x^* vor, und im Fall einer unimodalen Funktion ist x^* das globale Minimum. (Ein lokales Maximum liegt vor, wenn die Hessematrix negativ definit und ein Sattelpunkt, wenn die Hessematrix indefinit ist.) Sollte $H_f(x^*)$ semidefinit sein, so kann keine eindeutige Entscheidung getroffen werden. Dann muss auf weitere Methoden zurückgegriffen werden.

Eine weitere interessante Eigenschaft einer Funktion, ist die Separierbarkeit.

Definition 2.2.8 Eine Funktion f heißt separierbar, wenn sie sich in Teilfunktionen mit nur einer (oder wenigen) Variable(n) zerlegen lässt und die Summe dieser Teilfunktionen wieder f bildet. Die Funktion f muss sich somit wie folgt zerlegen lassen:

$$f(x_1, \dots, x_n) = f_1(x_1) + \cdots + f_n(x_n).$$

■

Separierbare Funktionen sind im Allgemeinen sehr leicht zu minimieren, in dem die einzelnen Teilfunktionen minimiert werden. Im Extremfall kann ein n -dimensionales Optimierungsproblem dann durch Minimieren von n eindimensionalen Funktionen gelöst werden.

2.2.1 Analytisches Suchverfahren

Für unrestringierte Optimierungsprobleme wurden soeben eine notwendige und eine hinreichende Bedingung für ein Minimum angegeben. Demnach war ein Punkt x^* dann ein Minimum, wenn der Gradient $\nabla f(x^*)$ an diesem Punkt null und die Hessematrix $H_f(x^*)$ positiv definit sind. War das Optimierungsproblem zudem konvex, so war der Punkt sogar ein globales Minimum.

Analytische Verfahren machen sich diese Eigenschaft zu nutze und bestimmen das Minimum in einem Schritt, in dem sie

1. einen Punkt x^* bestimmen, für den $\nabla f(x^*) = 0$ und
2. $H_f(x^*)$ positiv definit ist.

Tatsächlich müssen solche Verfahren aber noch den Gradienten und die Hessematrix bestimmen, und in 1. ein Gleichungssystem lösen. Ist die Hesse-Matrix in x^* semidefinit, so müssen weitere Methoden zur Anwendung kommen. Zudem können solche Verfahren nur angewendet werden, wenn die Bewertungsfunktion zwei mal stetig differenzierbar ist.

Für restringierte Probleme muss ein Punkt x^* bestimmt werden, für den in allen zulässigen Richtungen z gilt, dass $z^T \cdot \nabla f(x^*) \geq 0$ und $H_f(x^*)$ positiv definit ist. Hier ist es nicht mehr möglich, durch Lösen eines Gleichungssystem das x^* zu bestimmen, da sich die Menge der zulässigen Richtungen in einem Punkt nicht durch die Nebenbedingungen ausdrücken lässt (Neumann & Morlock, 1993).

Für restringierte Probleme kann aber die Lagrange-Funktion verwendet werden, die hier eine einfachere Berechnung des Minimums ermöglicht, weil sie die Nebenbedingungen mit der Zielfunktion verknüpft. Vorausgesetzt, dass eine Funktion f zu minimieren ist und die Nebenbedingungen in der Form $g_i(\mathbf{x}) \leq 0$ vorliegen (vgl. Abschnitt 2.1), kann die Lagrange-Funktion $\mathcal{L} : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$ definiert werden als:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) + \sum_{i=1}^m u_i \cdot g_i(\mathbf{x})$$

mit je einem Lagrange-Multiplikator u_i für die i -te Nebenbedingung zusammengefasst zum Vektor \mathbf{u} .

Der Punkt \mathbf{x}^* ist ein Minimum der Funktion f , wenn $(\mathbf{x}^*, \bar{\mathbf{u}})$ für ein $\bar{\mathbf{u}} > 0$ ein Sattelpunkt von \mathcal{L} ist. Der Beweis kann Neumann & Morlock (1993) entnommen werden. Das Paar $(\mathbf{x}^*, \bar{\mathbf{u}})$ mit $\bar{\mathbf{u}} > 0$ heißt Sattelpunkt, wenn

$$\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}_+^m : \mathcal{L}(\mathbf{x}^*, \mathbf{u}) \leq \mathcal{L}(\mathbf{x}^*, \bar{\mathbf{u}}) \leq \mathcal{L}(\mathbf{x}, \bar{\mathbf{u}}).$$

In der vorliegenden Form $\mathcal{L} : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$ ist der Sattelpunkt noch schwer zu bestimmen. Sind jedoch die Zielfunktion und die m Nebenbedingungen konvex und stetig differenzierbar und ist zudem ein $\mathbf{x}' \in \mathbb{R}^n$ vorhanden, so dass für alle nichtlinearen Nebenbedingungen $g_i(\mathbf{x}') < 0$ gilt („der Lösungsraum ist nicht leer“), können zum Ermitteln des globalen Minimums die Karush-Kuhn-Tucker Bedingungen verwendet werden.

Gelten in einem Punkt \mathbf{x}^* die Karush-Kuhn-Tucker Bedingungen

1. $\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \bar{u}_i \nabla g_i(\mathbf{x}^*) = 0$
2. $\sum_{i=1}^m \bar{u}_i \cdot g_i(\mathbf{x}^*) = 0$
3. $g_i(\mathbf{x}^*) \leq 0$ für $i = \{1, \dots, m\}$
4. $\bar{\mathbf{u}} \geq 0$

so ist \mathbf{x}^* globales Optimum (Neumann & Morlock, 1993).

Analytische Optimierungsverfahren, so wie hier beschrieben, können zur Anwendung kommen, wenn das Optimierungsproblem konvex und die Zielfunktion, sowie die Nebenbedingungen stetig differenzierbar sind. Zwar können der Gradient und die Hesse-Matrix einer nicht differentierbaren Funktion approximiert werden (Differenzbildung), dennoch ist die Einsetzbarkeit solcher Verfahren stark beschränkt. Sie sind für die meisten technischen Anwendungen ungeeignet. Hier ist die Zielfunktion meistens unbekannt und mögliche Lösungen können nur durch Simulation

bewertet werden (beispielsweise beim Entwurf technischer Systeme). Sollte die Zielfunktion bekannt sein, liegen bei technischen Aufgabenstellungen zudem oft nicht konvexe und nicht stetig differenzierbare Funktionen vor (Schwefel, 1994). Hier müssen analytische Verfahren versagen. Eine Lösung können die in Kapitel 3 vorgestellten Algorithmen liefern.

2.2.2 CEC 2005 Testfunktionensatz

Im diesem Kapitel wurden reellwertige Optimierungsprobleme vorgestellt, zu denen in Kapitel 3 eine Reihe an Lösungsverfahren eingeführt werden. Vor allem für den späteren Vergleich von Evolutionsstrategie, Genetischem Algorithmus und einer einfachen stochastischen Suche, die zu den globalen Optimierverfahren (also Verfahren, die das globale Optimum finden sollen) gehören, ist eine geeignete Testumgebung notwendig.

Bisher haben standardisierte Testfunktionen und ein standardisiertes Verfahren zur experimentellen Untersuchung gefehlt, was oft zur Folge hatte, dass verworrene Forschungsergebnisse vorgestellt wurden (Suganthan et al., 2005). So wurden beispielsweise Testfunktionen verwendet, zu denen ein bestimmter Algorithmus konstruiert wurde und dort entsprechend eindrucksvolle Ergebnisse lieferte. Für andere Problemstellungen wird ein solcher Algorithmus aber zu beliebig schlechten Ergebnissen führen. Zudem sind viele Testfunktionen nicht skalierbar und können nicht in der Problemgröße (Anzahl der Variablen) eingestellt werden. Solche Funktionen haben dann die Eigenschaft, dass sie eine geringe Dimension aufweisen und eher einfach zu lösen sind. Ein weiterer Nachteil bisheriger Testfunktionen ist ihre Separierbarkeit, die dazu führt, dass das Problem in eine Menge eindimensionaler Funktionen zerlegt und ebenfalls entsprechend leicht gelöst werden kann.

Whitley et al. (1995) stellen eine Richtlinie auf, die die minimalen Anforderungen an eine Testfunktion zusammenfasst. Demnach sollten Testfunktionen so komplex sein, dass zumindest einfache Hillclimbing Verfahren (vgl. Kapitel 3) an ihnen scheitern. Weiter sollten Testfunktionen nicht-linear und nicht-separierbar sein. Zudem wird von Testfunktionen verlangt, dass sie skalierbar sind. Bisherige Funktionen waren oft nicht skalierbar und zudem niedrig dimensioniert.

Ein weiterer Kritikpunkt bisheriger experimenteller Untersuchungen ist, dass sie untereinander nicht vergleichbar sind, da sie auf unterschiedlichen Testfunktionen durchgeführt wurden. Für die Untersuchung genetischer Algorithmen wird häufig der *De Jong* Testfunktionensatz angewendet (Whitley et al., 1996). Er enthält fünf Funktionen, von denen eine zwei-dimensional ist. Die restlichen sind zwar skalierbar, dafür aber auch separierbar. Evolutionsstrategien hingegen werden oft auf der Sphere-Funktion, der Rosenbrock-Funktion (beide sind auch im De Jong Testfunktionensatz enthalten), der Rastrigin-Funktion, der Schwefel-Funktion und der Griewangk-Funktion untersucht. Auch diese Funktionen sind nicht optimal. Die Schwefel-Funktion ist in ihrer Originalversion nur eindimensional, die Griewangk-Funktion verliert mit zunehmender Dimension lokale Minima und vereinfacht sich somit (Whitley et al. 1995). Zudem existieren verschiedene Realisierungen der Funktionen. Schöneburg et al. (1994) führen beispielsweise für die Rosenbrock-Funktion drei unterschiedliche Implementierungen auf.

Eine Lösung zu den hier aufgezählten Problemen bietet der Testfunktionensatz, der auf dem *Congress on Evolutionary Computation* (CEC) 2005 vorgestellt wurde. Dieser Testfunktionensatz enthält 25 heterogene Testfunktionen mit unterschiedlichen Eigenschaften. Er unterteilt diese Testfunktionen in 5 unimodale und 20 multimodale Funktionen, von denen wiederum 11 Funktionen hybrid sind (sich also aus mehreren unterschiedlichen Funktionen zusammensetzen). Die Testfunktionen sollen hier nicht im vollen Umfang vorgestellt werden. Die vollständige Dokumentation kann Suganthan et al. (1995) entnommen werden. An dieser Stelle soll nur auf die Vorzüge des Testfunktionensatzes eingegangen werden. Dieser Testfunktionensatz liefert eine Reihe an heterogenen Funktionen, von denen alle skalierbar sind. Zu Vermeidung der Symmetrie

und des Standard-Optimums $\mathbf{0}$ ist das Optimum, sowie der Funktionswert dieser Funktionen verschoben. Zudem wird ein starres Regelwerk gegeben, nach dem die experimentellen Untersuchungen durchzuführen sind, so dass im Ergebnis eine vergleichbare Studie ermöglicht wird. Dieses Regelwerk beschreibt ausführlich, wie die Optimierverfahren zu initialisieren und wann sie abzubrechen sind. Um einheitliche Ergebnisse zu liefern (und so eine Vergleichbarkeit verschiedener Untersuchungen zu ermöglichen), wird auch genau beschrieben, wie die Ergebnisse darzustellen sind. Schließlich wird aufgrund der Skalierbarkeit aller Funktionen vorgegeben, dass die Untersuchungen auf 10, 30 und 50 Dimensionen durchzuführen sind. Somit müssten die Untersuchungen auf 75 Problem instanzen durchgeführt werden, die den Rahmen einer Diplomarbeit sprengen würden. Deshalb wird hier eine Auswahl von Testfunktionen getroffen, die möglichst heterogen sind (die Nummerierungen in der folgenden Auflistung beziehen sich auf die Reihenfolge im CEC 2005 Testfunktionensatz):

Shifted Sphere-Funktion: Diese unimodale und separierbare Funktion findet sich in den meisten Testfunktionensätzen, und gilt als sehr einfach. Solche Testfunktionen sollten dennoch betrachtet werden, weil sie wertvolle Informationen über ein Optimierverfahren liefern können (Whitley et al. 1995). Die **sphere**-Funktion ist definiert als:

$$f_1(\mathbf{x}) = \sum_{i=1}^n (x_i - o_i)^2 + f_{\text{bias}} .$$

Sie ist um den Vektor \mathbf{o} verschoben und nimmt dort ihr Minimum an. Dieses Minimum hat den Wert $f_1(\mathbf{o}) = f_{\text{bias}} = -450$. Die Funktion hat den Definitionsbereich $[-100, 100]^n$, so dass Optimierverfahren in diesem Bereich gleichverteilt initialisiert werden sollen. Diese Methode zur Initialisierung wird im Folgenden auch als *gleichverteilt zufällige Initialisierung* (engl. uniform random starts) bezeichnet und mit UNIRND abgekürzt.

Diese Funktion gilt (in niedriger Dimension) als sehr leicht zu lösen, so dass jedes Optimierverfahren auf dieser Funktion gute Ergebnisse liefern sollte. Diese Funktion hat keine böartigen Eigenschaften und eignet sich deshalb vor allem zu Bestimmung der Genauigkeit der Optimierverfahren.

Weil im weiteren Verlauf komplexere Funktionen betrachtet werden, kann diese Funktion auch benutzt werden, um zu zeigen, wie sich die Parametrisierungen der Optimierverfahren verändern, wenn nicht mehr einfache, sondern eher komplexe Funktionen minimiert werden.

Shifted Rosenbrock's-Funktion: Diese Funktion ist komplizierter zu lösen, als die **sphere**-Funktion, weil sie multimodal und nicht separierbar ist (wie alle nun folgenden Funktionen). Definiert ist diese Funktion wie folgt:

$$f_6(\mathbf{x}) = \sum_{i=1}^{n-1} (100 \cdot (z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f_{\text{bias}} , \quad \text{mit } \mathbf{z} = \mathbf{x} - \mathbf{o} + \mathbf{1} .$$

Sie hat ihr globales Minimum in $\mathbf{x}^* = \mathbf{o}$ mit $f_6(\mathbf{x}^*) = f_{\text{bias}} = 390$ und den Definitionsbereich $[-100, 100]^n$, in dem Optimierverfahren gleichverteilt zu initialisieren sind.

Diese Funktion, die auch als „Bananenfunktion“ bezeichnet wird, zeichnet sich durch ein schmales gekrümmtes Tal aus, das iterative Optimierverfahren durchqueren müssen, um schließlich das globale Optimum zu erreichen. Die Verfahren können deshalb meistens nur kleine Schritte zurücklegen, um zum Minimum zu gelangen. Da Evolutionsstrategie und genetischer Algorithmus die Schrittweiten unterschiedlich einstellen (vgl. Kapitel 3), ist es interessant zu beobachten, welche Verfahren warum besser abschneiden.

Shifted rotated Rastrigin's-Funktion: Wie die Rastrigin-Funktion besitzt auch die gedrehte Variante eine hohe Anzahl lokaler Minima, ist aber nicht mehr separierbar. Definiert ist diese Funktion als:

$$f_{10}(\mathbf{x}) = \sum_{i=1}^n (z_i^2 - 10 \cdot \cos(2\pi \cdot z_i) + 10) + f_{\text{bias}}, \quad \text{mit } \mathbf{z} = (\mathbf{x} - \mathbf{o}) \cdot M$$

und M ist eine Rotationsmatrix. Der Definitionsbereich dieser Funktion ist $[-5, 5]^n$ und das globale Minimum befindet sich in \mathbf{o} mit $f_{10}(\mathbf{o}) = f_{\text{bias}} = -330$. Die Initialisierung erfolgt gleichverteilt zufällig im Definitionsbereich.

In vielen Arbeiten (z.B. Schöneburg et al., 1994) wird publiziert, dass die Evolutionsstrategie auf unimodalen und die genetischen Algorithmen auf multimodalen Funktionen bevorzugt anzuwenden sind, weil sie jeweils dort die bessere Performance liefern. Es wird interessant sein, zu beobachten, ob diese Aussage bestätigt werden kann. Mit dieser Funktion besitzt die gewählt Teilmenge sowohl eine unimodale, als auch multimodale Funktionen, mit wenigen und extrem vielen lokalen Minima. Diese Funktion ist aber auch in einem anderen Zusammenhang interessant. Die bisherigen Funktionen haben mit $X = [-100, 100]^n$ einen eher großen Suchraum. Diese Funktion besitzt mit $X = [-5, 5]^n$ einen kleineren Suchraum, so dass Erkenntnisse darüber gewonnen werden können, wie sich Optimierverfahren auf unterschiedlich großen Suchräumen verhalten.

Bemerkenswert an dieser Stelle ist noch ein neuer, für den CEC 2006 gedachter Testfunktionensatz (Liang, et al., 2006). Dieser enthält Testfunktionen für die restringierte reellwertige Optimierung. Erstaunlicherweise lässt dieser Satz alle Anforderungen, die Whitley et al. (1995) an Testfunktionen stellen, außer Acht. Die dort enthaltenen Probleme sind niedrig-dimensional, separierbar und zum Teil linear. Aus diesem Grund wird dieser Testfunktionensatz nicht weiter betrachtet und die Untersuchungen auf dem CEC 2005 Satz durchgeführt.

2.3 Zusammenfassung

1. Optimierungsprobleme können unterschiedlich klassifiziert werden. Es ist zwischen
 - a) reellwertigen – ganzzahligen – kombinatorischen
 - b) mono – multikriteriellen
 - c) statischen – dynamischen
 - d) restringierten, verrauschten oder experimentellen

Optimierungsproblemen zu unterscheiden, die entsprechende Optimierverfahren erfordern.

2. Reellwertige Optimierungsprobleme setzen sich aus einem Suchraum (zulässigen Bereich), einem Bewertungsmaß und einer Bewertungs- oder Zielfunktion zusammen. Das Ziel eines reellwertigen Optimierungsproblems ist das Auffinden einer zulässigen Lösung mit vorgegebenem Wert. Gewöhnlich wird das entweder das Minimum oder aber das Maximum sein.
3. Optimierungsprobleme verfügen über verschiedene Eigenschaften. Beispielsweise können sie konvex oder unimodal sein. Dann ist im ersten Fall jedes lokale Minimum auch ein globales bzw. es existiert im zweiten Fall exakt ein Minimum.
4. Ist der Gradient berechenbar, kann die Richtung ermittelt werden, entlang der ein Punkt bewegt werden kann, um einen lokal maximalen Abstieg zu erzielen. Zudem kann der Gradient verwendet werden, um einen stationären Punkt zu berechnen.

5. Die Hessematrix liefert eine hinreichende Bedingung für ein Minimum. Ist die Hessematrix im stationären Punkt positiv definit, so ist dieser Punkt ein Minimum.
6. Für restringierte Optimierungsprobleme ist es schwieriger einen stationären Punkt zu errechnen. Hier müssen alle zulässigen Richtungen betrachtet und ein Punkt x gefunden werden, in dem das Skalarprodukt aus Gradient und allen zulässigen Richtungen in x positiv ist. Die Berechnung eines solchen x ist aufgrund der Nebenbedingungen aufwendig.
7. Die Lösung bietet die Lagrange-Funktion, die die Nebenbedingungen mit der Zielfunktion verknüpft. Dennoch ist auch so die Ermittlung des Minimums schwierig. Schließlich bieten die Karush-Kuhn-Tucker Bedingungen, wenn bestimmte Voraussetzungen erfüllt sind, eine einfache Möglichkeit, das Minimum eines restringierten Optimierungsproblems zu berechnen.
8. Der CEC 2005 Testfunktionensatz stellt eine Reihe an Testfunktionen zu Verfügung, die sich besonders zu experimentellen Untersuchung und Bewertung von Optimierungsverfahren eignen, weil sie sehr heterogen sind und viele Eigenschaften tatsächlicher Problemstellungen kombinieren.

3 Algorithmen

Im Kapitel 2 wurden verschiedene Arten von Optimierungsproblemen vorgestellt und exemplarisch für drei verschiedene Arten Lösungsverfahren skizziert. Schon hier war festzustellen, dass es für Optimierungsprobleme nicht das universelle Verfahren gibt. Aber selbst für reellwertige Optimierungsprobleme ist ein universelles Verfahren nicht vorhanden. Das im Kapitel 2.2.1 vorgestellte analytische Verfahren eignet sich nur bedingt für die Optimierung, da es sich nur auf Funktionen anwenden lässt, die stetig differenzierbar sind.

Die meisten der in diesem Kapitel vorgestellten Algorithmen gehören zu den direkten Suchverfahren. Für sie ist es ausreichend, einen Punkt aus dem Suchraum bewerten zu können. Somit stellen direkte Suchverfahren die geringsten Anforderungen an das Optimierungsproblem und eignen sich auch für solche Problemstellungen, für die ein Modell (Zielfunktion) f nicht vorhanden ist (experimentelle Optimierung). Ist die Zielfunktion bekannt und einmal stetig differenzierbar, können Gradientenverfahren zum Einsatz kommen, die schneller als direkte Suchverfahren zu einem Optimum konvergieren, weil ihnen die Richtung des lokal steilsten Abstiegs bekannt ist.

Sowohl direkte Suchverfahren als auch Gradientenverfahren können zu *Hillclimbing* Verfahren zusammengefasst werden, deren Ziel es ist, über eine deterministische Vorschrift Lösungspunkte zu erzeugen, die stetig zu einem Minimum bewegt werden. Dieses ist bei nicht konvexen Funktionen in der Regel ein lokales Optimum. Zudem kann die deterministische Vorschrift je nach Zielfunktion besonders gute, oder aber weniger gute Ergebnisse liefern.

Einen anderen Weg wählen stochastische Suchalgorithmen. Solche Algorithmen haben keine deterministische Vorschrift und verfolgt daher eine zufällige Suchrichtung. Schwefel (1994) merkt jedoch an, dass solche Verfahren durchaus nach intelligenten Regeln (und nicht völlig willkürlich) das globale Optimum suchen. Zu stochastischen Suchalgorithmen gehören die Evolutionären Algorithmen (EA), die auf Prinzipien der biologischen Evolution basieren. Die zwei bekanntesten Vertreter der EA sind die Evolutionsstrategien (ES) und die genetischen Algorithmen (GA). Eine besonders einfache Variante stochastischer Suchalgorithmen ist die einfache stochastische Suche, engl. Randomised Iterative Improvement (RII). Diese soll hier zu Vergleichszwecken betrachtet werden. Alle drei Verfahren gehören zu den globalen Optimierungsverfahren, also solchen Verfahren, die ein globales Optimum auffinden können, wenn ihnen genügend Zeit zu Verfügung gestellt wird.

Der Schwerpunkt dieses Kapitels soll auf den Evolutionären Algorithmen und der einfachen stochastischen Suche liegen. Vorab werden noch einige Hillclimbing Verfahren skizziert. Eine weitergehende Beschreibung dieser Verfahren findet sich in Schwefel (1994).

3.1 Hillclimbing Verfahren

Hillclimbing Verfahren verdanken ihren Namen der Metapher eines Bergsteigers, der sich stetig dem Gipfel nähert. Die hinter den Hillclimbing Verfahren stehende Vorschrift geht ähnlich vor, mit dem Unterschied, dass sie sich stetig dem Tal nähert. Sie arbeitet auf einem Lösungspunkt und bewegt ihn stetig zum Minimum. Allgemein wird dazu in einer Iteration t ein Punkt $x^{(t)}$ variiert und nur dann in die Folgeiteration übernommen, wenn er einen kleineren Zielfunktionswert aufweist. Eine Ausnahme bildet das *Nelder-Mead Simplexverfahren*, welches nicht auf einem Punkt, sondern einem Simplex arbeitet, sich aber ebenfalls stetig dem Minimum nähert.

Vor der Beschreibung einiger Hillclimbing Verfahren für mehrdimensionale Funktionen werden vorab zwei Verfahren zur Minimierung eindimensionaler Funktionen vorgestellt. Diese eignen sich dann zur Bestimmung optimaler Schrittweiten für die n -dimensionalen Verfahren.

3.1.1 Regula-falsi

Das Regula-falsi Verfahren wird verwendet, um die Nullstelle einer eindimensionalen Funktion zu bestimmen. Es kann zur Bestimmung eines globalen Minimums der Funktion f verwendet werden, wenn die Funktion stetig differenzierbar und konvex ist. Dann liefert die Nullstelle der Ableitung f' eine globale Minimalstelle von f .

Das Verfahren arbeitet auf zwei Punkten $(a^{(t)}, f'(a^{(t)}))$ und $(b^{(t)}, f'(b^{(t)}))$, die eine Sekante definieren, die linear die Funktion f' approximiert. Die Nullstelle der Geraden kann sehr einfach durch

$$c^{(t)} = a^{(t)} - f'(a^{(t)}) \cdot \frac{b^{(t)} - a^{(t)}}{f'(b^{(t)}) - f'(a^{(t)})}$$

errechnet werden, so dass ein neuer Punkt $(c^{(t)}, f'(c^{(t)}))$ entsteht. Je nach Vorzeichen in der y -Koordinate der drei Punkte können die folgenden Fälle unterschieden werden:

1. Die Vorzeichen von $f'(a^{(t)})$ und $f'(b^{(t)})$ sind verschieden. Dann liegt die Nullstelle x^* der ersten Ableitung im Intervall $[a^{(t)}, b^{(t)}]$, das durch $c^{(t)}$ geteilt wird. Hier sind die folgenden zwei Fälle zu unterscheiden:
 - a) Die Vorzeichen von $f'(a^{(t)})$ und $f'(c^{(t)})$ stimmen überein. Dann liegt die Nullstelle x^* im Intervall $[c^{(t)}, b^{(t)}]$
 - b) Sind die Vorzeichen von $f'(b^{(t)})$ und $f'(c^{(t)})$ identisch, dann liegt die Nullstelle x^* im Intervall $[a^{(t)}, c^{(t)}]$
2. Sind die Vorzeichen von $f'(a^{(t)})$ und $f'(b^{(t)})$ identisch, so liegt die Nullstelle ausserhalb des Intervalls $[a^{(t)}, b^{(t)}]$. Hier sind wiederum zwei Fälle zu unterscheiden:
 - a) $f'(c^{(t)})$ hat ein von $f'(a^{(t)})$ (und damit auch $f'(b^{(t)})$) verschiedenes Vorzeichen. Dann liegt die Nullstelle von f' im kleinsten zusammenhängenden Intervall, das den Punkt $c^{(t)}$ und einen der anderen zwei Punkte $a^{(t)}$ oder $b^{(t)}$ enthält.
 - b) Stimmen die Funktionswerte der ersten Ableitung in allen drei Punkten überein, so kann kein Intervall bestimmt werden, das die Nullstelle von f' enthält. Das Intervall kann aber in Richtung betragsmäßig abnehmender Funktionswerte bewegt und so einer potentiellen Nullstelle angenähert werden. Dann muss $c^{(t)}$ das Intervallende mit betragsmäßig größtem Funktionswert ersetzen.

Durch Fortsetzung des Verfahrens kann das so neu entstandene Intervall im ersten Fall weiter verkürzt werden, bis der Punkt $c^{(t)}$ irgendwann nah genug an der tatsächlichen Nullstelle x^* der Ableitung von f liegt. Liegt die Nullstelle von f' nicht im betrachteten Intervall, so wird im zweiten Fall das Intervall i.d.R. in Richtung einer Nullstelle bewegt.

Ist die Funktion f konvex, so wird ein globales Minimum gefunden. Ähnlich wie bei den analytischen Verfahren muss die Funktion f stetig differenzierbar sein, da sonst die Ableitung nicht bestimmt werden kann. Das Verfahren muss daher auf vielen technischen Anwendungen ebenfalls scheitern. Ein anderes Verfahren, das ebenfalls auf einem Intervall arbeitet, ist die Fibonacci-Suche, die weniger Anforderungen an das Optimierungsproblem stellt.

3.1.2 Fibonacci-Suche

Die Fibonacci-Suche wird auch zum Auffinden eines Minimums einer eindimensionalen Funktion f verwendet. Anders als das Regula-falsi Verfahren kommt die Fibonacci-Suche jedoch ohne

erster Ableitung aus. Für die Fibonacci-Suche ist es ausreichend, eine Stelle auf der Abszisse bewerten zu können. Sie eignet sich somit auch für die experimentelle Optimierung.

Die Fibonacci-Suche arbeitet wie auch das Regula-falsi Verfahren auf einem durch zwei Punkte definierten Intervall, dass aber im Gegensatz zum Regula-falsi Verfahren das Minimum enthalten muss. Dieses Intervall wird in jeder Iteration um einen Faktor $q^{(t)}$ verkürzt, bis das Minimum ausreichend eingegrenzt ist. Der Faktor ergibt sich aus den Fibonacci-Zahlen. Die n -te Fibonacci-Zahl F_n ist rekursiv definiert als

$$F_0 = F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{für } n \geq 2$$

und der Faktor, um den das Intervall in der t -ten von T Iterationen verkürzt wird als

$$q^{(t)} = \frac{F_{T-t+1}}{F_{T-t+2}}$$

mit $\frac{1}{2} \leq q^{(t)} < 1$. Die Länge, um die das Intervall verkürzt wird beträgt somit in Iteration t

$$r^{(t)} = (b^{(t)} - a^{(t)}) \cdot q^{(t)} .$$

Sei nun $[a^{(t)}, b^{(t)}]$ das in Iteration t gegebene Intervall, dass das Minimum enthält. Dann können zwei Punkte $c^{(t)} = a^{(t)} + r^{(t)}$ und $d^{(t)} = b^{(t)} - r^{(t)}$ berechnet werden. Wegen $\frac{1}{2} \leq q^{(t)} < 1$ bildet $[a^{(t)}, d^{(t)}]$ das Anfangstück und $[c^{(t)}, b^{(t)}]$ das Endstück des Intervalls. Ist $f(c^{(t)}) > f(d^{(t)})$, so liegt bei einer unimodalen Funktion das Minimum im Intervall $[a^{(t)}, c^{(t)}]$, sonst in $[d^{(t)}, b^{(t)}]$. In jedem Fall wird das Intervall aber um die Länge $r^{(t)}$ verkürzt und das Minimum weiter eingegrenzt.

Schließlich ist in der letzten Iteration T der Faktor $q^{(T)} = \frac{1}{2}$ und die beiden Punkte $c^{(T)}$ und $d^{(T)}$ fallen zusammen und bilden die Lösung der Suche.

Schwefel (1994) gibt an, dass die Anzahl der Iterationen T für eine Genauigkeit ϵ so zu wählen ist, dass die Ungleichung $F_T > \frac{b^{(0)} - a^{(0)}}{\epsilon} \geq F_{T-1}$ erfüllt ist. Problematisch ist, dass in jeder Iteration eine neue Fibonacci-Zahl zu bestimmen ist. Die Fibonacci-Zahlen haben zwar den Vorteil, dass aufgrund ihres Teilungsverhältnisses ab der zweiten Iteration nur noch einer der Punkte $c^{(t)}$ oder $d^{(t)}$ neu bestimmt werden muss, da der andere Punkt aus der letzten Iteration übernommen werden kann (Schwefel, 1994). Dennoch muss in jeder Iteration eine neue Fibonacci-Zahl bestimmt werden, die die Laufzeit des Verfahrens verzögert.

Teilung nach dem goldenen Schnitt

Den gleichen Effekt, dass ab der zweiten Iteration nur noch ein neuer Punkt bestimmt werden muss, erzeugt auch ein auf dem goldenen Schnitt basierendes Unterteilungsverhältnis. Der Goldene Schnitt ist definiert als

$$\lim_{(T-t) \rightarrow \infty} \frac{F_{T-t+1}}{F_{T-t+2}} = \frac{2}{1 + \sqrt{5}} .$$

Somit sind bei großen T zu Beginn der Iteration die Teilungspunkte bei der Fibonacci-Suche identisch zu den Teilungspunkten, die bei der Unterteilung nach dem Goldenen Schnitt entstehen. Zum Ende der Suchen unterscheiden sich beide Unterteilungsarten in sofern, dass das Ergebnis bei der Fibonacci-Suche zu einem Punkt konvergiert, während bei Teilung nach dem goldenen Schnitt stets ein Intervall verbleibt.

Bestimmung des Startintervalls

Ist kein a-priori Wissen über die Position des Minimums bekannt, so kann ohne weiteres kein Startintervall vorgegeben werden, das das Minimum enthält. Ein einfaches Verfahren, das ein solches Intervall erzeugt, beginnt mit einem vom Anwender oder einem zufällig bestimmten Punkt $x^{(0)}$, und schreitet nach rechts solange fort, bis der Funktionswert des neu erzeugten Punktes $x^{(r)}$ größer ist als der des Vorgängers. So ist sichergestellt, dass ein u. U. rechts vom Startpunkt liegendes Minimum überschritten wird. Dann kann $x^{(r)}$ das rechte Ende des Intervalls bilden. Entsprechend analog kann beginnend vom Startpunkt $x^{(0)}$ auch das linke Ende des Intervalls $x^{(l)}$ bestimmt werden.

Um das Verfahren bei größeren Intervallen zu beschleunigen, kann die zu Beginn klein gewählte Schrittweite in jeder Iteration verdoppelt werden.

3.1.3 Suche entlang der Koordinatenachsen

Zum Minimieren mehrdimensionaler Optimierungsprobleme $f : \mathbb{R}^n \rightarrow \mathbb{R}$ kann die Suche entlang der Koordinatenachsen verwendet werden. Dieses Verfahren erzeugt in Iteration t einen Punkt $x^{(t+1)}$, in dem es den Vorgängerpunkt $x^{(t)}$ entlang seiner n Koordinatenachsen einer Verbesserung unterzieht. Die Abbildung 3.1 zeigt ausgehend von einem Startpunkt $x^{(0)}$ die ersten vier

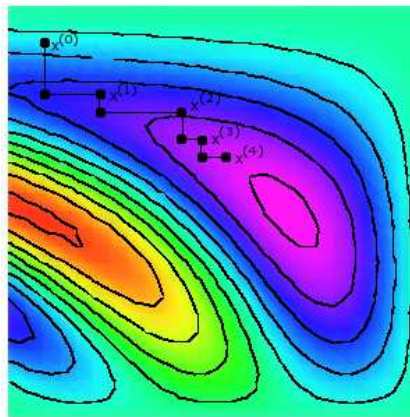


Abbildung 3.1: Exemplarischer Ablauf der Koordinatenstrategie

Iterationen im \mathbb{R}^2 .

In der einfachsten Variante können feste Schrittweiten vorgegeben werden, um die die Punkte entlang der Koordinaten verschoben werden können. Hier müssen beide Richtungen (sowohl $x_i + s_i$, als auch $x_i - s_i$ und s_i ist Schrittweite für die i -te Koordinate) in Betracht gezogen werden, da die Abstiegsrichtung im Allgemeinen nicht bekannt ist. Die Schrittweiten sind zudem abhängig von Problemstellung und Optimierungsfortschritt und müssen zu Beginn, aber auch während der Optimierung entsprechend eingestellt werden.

Anstelle fester Schrittweiten, kann aber entlang der einzelnen Koordinaten ein eindimensionales Optimierungsproblem gelöst werden. Dann wählt man entlang der einzelnen Koordinaten immer eine optimale Schrittweite. Dazu werden im Fall der i -ten Koordinate mit Ausnahme der i -ten Variable alle Variablen festgehalten, so dass das Optimierungsproblem in der Dimensionalität um $(n - 1)$ reduziert wird. Hier können dann die Fibonacci-Suche, oder das Regula-falsi Verfahren verwendet werden. Diese Vorgehensweise bringt wiederum den Nachteil mit sich, dass ein hoher Rechenaufwand entsteht und das Verfahren somit langsam wird. Daher sollte ein Mittelweg gewählt werden, der zwar geeignete Schrittweiten wie oben beschrieben ermittelt, diese jedoch nicht exakt, in dem den eindimensionalen Suchverfahren eine maximale Rechenzeit

gegeben wird.

Zwar konvergiert die Suche entlang der Koordinatenachsen immer, wenn die Funktion f stetige partielle Ableitungen hat (Schwefel, 1994), jedoch nur dann garantiert zu einem globalen Minimum, wenn f konvex ist. Wie in Abbildung 3.1 zu ersehen ist, können zum Ende der Optimierung die zurückgelegten Schritte sehr klein und der Optimierungsvorgang dem entsprechend langsam werden. Dies tritt vor allem auch dann auf, wenn der Gradient nicht annähernd parallel zu den Koordinatenachsen verläuft. Dann ist es viel versprechend an Stelle der Koordinatenachsen entlang des Gradienten zu optimieren. Dazu muss f bekannt und stetig differenzierbar sein. Zudem kann die Berechnung des Gradienten sehr aufwendig sein (Gerdes et al., 2004). Gradientenverfahren sollen hier nicht weiter betrachtet werden. Sie werden in Schwefel (1994), Neumann & Morlock (1993) und Nocedal & Wright (1999) beschrieben. Das *Hooke-Jeeves Pattern Search*- und das *Nelder-Mead Simplex* Verfahren kommen ohne Berechnung des Gradienten aus, können aber dennoch eine Abstiegsrichtung ermitteln.

3.1.4 Hooke-Jeeves Pattern Search

Das Hooke-Jeeves Pattern Search Verfahren setzt sich aus zwei Komponenten zusammen: Der Suche entlang der Koordinatenachsen (Suche) und einer Extrapolation. Beginnend mit einer Suche führt dieses Verfahren, solange erfolgreich, zyklisch eine Extrapolation gefolgt von einer Suche und anschließender Bewertung durch. Die Schrittweiten der Suche werden fest vorgegeben, die der Extrapolation ergeben sich als gesamter, im aktuellen Zyklus zurückgelegter Weg. Die Richtung der Extrapolation bestimmt sich aus der Verbindungslinie der Punkte aus jetziger und Vorgängeriteration. Diese verläuft annähernd entlang des Gradienten (Schwefel, 1994), so dass Extrapolation entlang dieser Linie den größten Erfolg verspricht.

Sobald die Bewertung am Ende einer Iteration ergibt, dass eine Verschlechterung zum letzten Punkt erfolgt ist, wird sowohl die letzte Suche als auch die letzte Extrapolation rückgängig gemacht. Hier ist anzunehmen, dass ein zu großer Schritt erfolgt ist und der Lösungspunkt über das Minimum hinaus bewegt wurde. Dann wird vom letzten Punkt aus der Zyklus aus Suche, Extrapolation und Bewertung neu gestartet und die Schrittweite der Extrapolation zurückgesetzt. Findet die erste Suche im neuen Zyklus keinen besseren Punkt, kann die Schrittweite der Suche reduziert, oder, falls die Schrittweite einen bestimmten Wert unterschreitet, abgebrochen werden.

Ist die Funktion strikt konvex und stetig differenzierbar, so wird das globale Minimum gefunden. Durch die Extrapolation, die nach jeder erfolgreichen Iteration größere Schrittweiten annimmt und annähernd dem Gradienten folgt, wird die Optimierung beschleunigt. Dennoch kann auch dieses Verfahren, aufgrund der Suche entlang der Koordinatenachsen, vorzeitig abbrechen.

3.1.5 Nelder-Mead Simplex

Das Nelder-Mead Simplex Verfahren (oder kurz Simplex Verfahren) minimiert mehrdimensionale Funktionen $f : \mathbb{R}^n \rightarrow \mathbb{R}$, in dem es einen n -Simplex über den Suchraum bewegt. Ein n -Simplex ist die minimale konvexe Hülle, die $n + 1$ affin unabhängige Punkte enthält. Für die Minimierung greift das Simplexverfahren auf diese Punkte zurück, die gerade ausreichend sind, um die Abstiegsrichtung bestimmen zu können (Nelder & Mead, 1965).

Die Grundidee des Verfahrens ist, ausgehend von einem Startsimplex den Punkt mit schlechtestem Zielfunktionswert zu finden und diesen zu entfernen. Dann bilden die restlichen Punkte eine Hyperfläche, um deren Schwerpunkt der gerade entfernte Punkt reflektiert werden kann. Die Abbildung 3.2 (links) stellt einen Ablauf der Grundidee dieses Verfahrens graphisch dar. Ausgehend von dem Startsimplex 1 werden neue Simplexe erzeugt, die sich dem Minimum annähern, bis schließlich ab Simplex 9 keine Verbesserungen mehr erreicht werden können. Das

Verfahren muss daher noch erweitert werden, in dem nicht nur Reflektiert wird, sondern auch noch expandiert und kontrahiert. Im ersten Fall wird versucht einen noch weiteren Schritt als bei der Reflexion zu gehen. So kann ein weiter außerhalb des Simplex liegender, besserer Punkt gefunden werden. Ist das der Fall, so wird das Simplex vergrößert und damit die Schrittweite erhöht. Ist der reflektierte Punkt der schlechteste im neu entstandenen Simplex, so muss aber

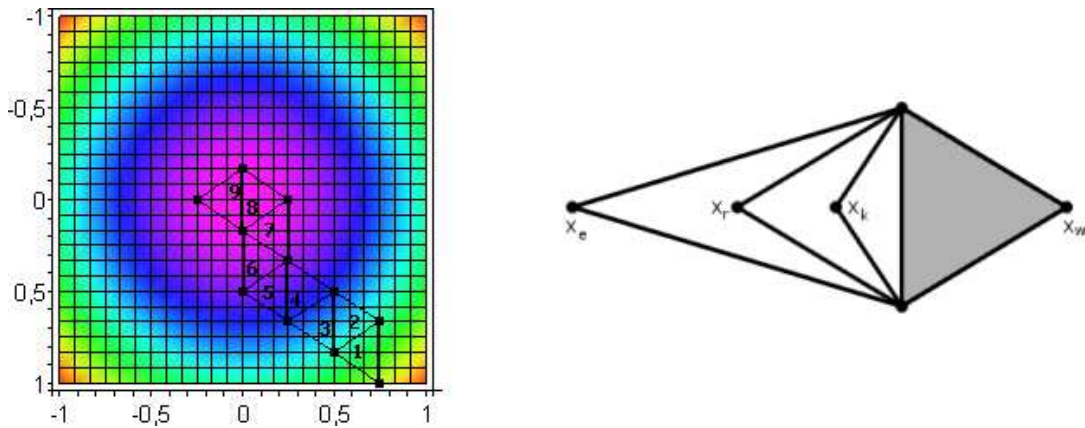


Abbildung 3.2: Ablauf (links) und Operatoren (rechts) des Simplexverfahrens

ein besserer im Innern des Simplex liegen, da die Reflexionsrichtung eine Abstiegsrichtung war. Dann kann die Kontraktion einen besseren Punkt im Inneren des neuen Simplex finden und das Simplex an die Erfordernisse anpassen, in dem sie es verkleinert. Schließlich müssen, besonders in der Nähe des Minimums, die Längen aller Kanten reduziert werden, was durch einen Reduktionskoeffizienten realisiert werden kann. Die so entstehenden Punkte im zwei-dimensionalen Raum zeigt die Abbildung 3.2 (rechts). Ausgehend vom grau eingefärbten Simplex wird der qualitativ schlechteste Punkt x_w reflektiert, so dass x_r entsteht. Expansion und Kontraktion erzeugen die Punkt x_e bzw. x_k . Die genaue Position dieser Punkte wird durch zwei Parameter ($p_e > 1$ für die Expansion bzw. $0 < p_k < 1$ für die Kontraktion) eingestellt. Schließlich wird, wie bei allen bisher vorgestellten Algorithmen ein weiterer Parameter benötigt, der angibt, wie die Schrittweiten reduziert werden.

Das Nelder-Mead Simplex Verfahren konvergiert nur dann sicher zum globalen Minimum, wenn die Funktion strikt konvex ist und nur von einer Variablen abhängt. Hängt sie von zwei Variablen ab, so kann gerade noch bewiesen werden, dass die Schrittweiten gegen null konvergieren. McKinnon (1996) zeigt für eher einfache zweidimensionale Funktionen, dass das Nelder-Mead Simplex Verfahren zu einem nicht stationären Punkt (also einem Punkt mit nicht verschwindenden Gradienten) konvergiert und somit versagt.

3.2 Einfache stochastische Suche

Wie bereits bei den oben beschriebenen Verfahren ist das Ziel der einfachen stochastischen Suche (RII) das Erzeugen einer Punktfolge, die zum Minimum konvergiert. Die RII setzt sich aus zwei Komponenten zusammen: Einem stochastischen Hillclimbing Verfahren, das sich, wie seine deterministischen Varianten, stetig einem lokalen Minimum nähert, und einem random-walk, welches zufällig einen neuen Lösungspunkt erzeugt. Auf den ersten Blick stehen sich beide Komponenten entgegen. Durch geschickte Verknüpfung beider Komponenten kann aber die daraus resultierende RII das Auffinden eines globalen Minimums garantieren (wenn der RII genügend Zeit gegeben wird). Sie zählt somit zu den globalen Optimierverfahren.

Wie die deterministischen Vertreter konstruiert auch das stochastische Hillclimbing Verfahren

eine Punktfolge, die sich zwar stetig, aber in der Regel nur einem lokalen Minimum annähert, und zwar dem, zu dem das Gefälle ausgehend vom Startpunkt führt. Das random-walk Verfahren, das mit einer Wahrscheinlichkeit p_w angewendet wird, verschiebt den Lösungspunkt zufällig und kann so zum Verlassen des lokalen Minimums führen und im besten Fall den Punkt in ein Gebiet des Lösungsraums bewegen, von dem aus das Gefälle zum globalen Minimum führt. Dann wird das stochastische Hillclimbing Verfahren dieses Minimum auffinden. Ein algorithmische Umsetzung dieses Verfahrens ist dem Algorithmus 3.1 zu entnehmen. Der

Algorithmus 3.1 : Einfache stochastische Suche

```

1  $t := 0$ ;
2 Initialisiere  $x^{(0)}$ ;
3 Initialisiere  $\sigma$ ;
4 while Abbruchbedingung nicht erfüllt do
5    $u := U[0, 1]$ ;
6    $z := N(0, 1)$ ;
7   if ( $u \leq p_w$ ) then
8      $x^{(t+1)} := x^{(t)} + \sigma \cdot z$ ;
9   else
10    if ( $f(x^{(t)} + \sigma \cdot z) < f(x^{(t)})$ ) then
11       $x^{(t+1)} := x^{(t)} + \sigma \cdot z$ ;
12    else
13       $x^{(t+1)} := x^{(t)}$ ;
14    $t := t + 1$ ;
15 Anpassung der Schrittweite  $\sigma$ ;
```

Algorithmus setzt sich wesentlich aus einer Schleife zusammen, die während der Optimierung eine bedingte Anweisung abarbeitet. In den Zeilen 7 – 8 wird mit Wahrscheinlichkeit p_w das random-walk Verfahren durchgeführt und ansonsten mit Wahrscheinlichkeit $(1 - p_w)$ in Zeilen 10 – 13 das stochastische Hillclimbing Verfahren. Dieses Verfahren erzeugt in der Nachbarschaft des aktuellen Punkts zufällig einen neuen Lösungspunkt und übernimmt diesen nur, wenn er einen besseren (hier: kleineren) Zielfunktionswert aufweist.

Die RII wird im Folgenden mit der Einstellung $p_w = 1$ und $s = 0$ durchgeführt, so dass ein minimaler Suchalgorithmus entsteht. Ein solcher sehr einfacher Algorithmus kann Aufschluss darüber liefern, ob die Verwendung komplexerer Verfahren Sinn macht.

3.3 Evolutionäre Algorithmen

Evolutionäre Algorithmen nehmen sich die Natur zum Vorbild, die auch in vielen anderen Bereichen (der Technik) eine Quelle der Inspiration ist. Man denke beispielsweise an Beschichtungen, die die Oberfläche einer Lotusblüte aufweisen und nicht verschmutzen können, oder an eine in Bremen entwickelte künstliche Haifischhaut (Spiegel-Online vom 18. Juli 2005), die zukünftig Schiffsrümpfe vor dem gefürchteten Bewuchs durch Muscheln und Seepocken schützen soll. Evolutionäre Algorithmen sind direkte, stochastische Suchverfahren, die auf Prinzipien der natürlichen Evolution basieren. Aus diesem Grund verwenden sie einen zur „klassischen“ Informatik abweichenden Wortschatz, der in Tabelle 3.1 zusammengestellt ist.

EA kodieren eine mögliche Lösung über eine passende Datenstruktur in einem so genannten Individuum. Diese Datenstruktur gibt dann genau einen Punkt im Suchraum an. Neben der Datenstruktur kann ein Individuum noch über weitere Zusatzinformationen, wie z.B. der Güte

| BEZEICHNUNG | BEDEUTUNG | SYMBOL |
|-------------------------|---|--|
| Generation | Iteration eines Evolutionären Algorithmus | t |
| Individuum | i -ter Lösungskandidat in Generation t | $I_i^{(t)}$ |
| Objektkomponente | Kodiert im Individuum eine mögliche Lösung | \mathbf{x} |
| Strategiekomponente | Evolutionstrategien kodieren im Individuum weitere Informationen in einer so genannten Strategiekomponente. In dieser Arbeit werden ausschließlich die Schrittwerte für die Mutation in der Strategiekomponente abgelegt. | \mathbf{s} |
| Elter | Individuum, aus dem Nachkommen erzeugt werden | |
| Nachkomme | In einer beliebigen Generation neu erzeugtes Individuum | |
| Population | Menge an Lösungskandidaten in Generation t | $P^{(t)}$ |
| Populationsgröße | Anzahl der Individuen in einer Population | $\mu \in \mathbb{N}$ |
| Anzahl der Nachkommen | Anzahl der Individuen, die zusätzlich zur Elternpopulation in einer Generation erzeugt werden | $\lambda \in \mathbb{N}$ |
| Selektionsdruck | Verhältnis aus λ und μ | $\nu = \frac{\lambda}{\mu}$ |
| Rekombinationsparameter | Menge an Individuen, die an der Rekombination teilnehmen | $\rho \in \mathbb{N}$ |
| Rekombination | Erzeugung neuer Individuen durch Austausch oder Mitteln einzelner Gene der Eltern | $rek : I^\rho \rightarrow I$ |
| Mutation | Zufällige (und geringe) Veränderung eines Individuums. | $mut : I \rightarrow I$ |
| Umweltselektion | Die Umweltselektion verringert den potentiellen Überschuss an Individuen und betont dabei die besonders guten Lösungskandidaten. Sie gibt einem EA somit seine Richtung. | $sel : I^k \rightarrow I^\mu$ mit $k \in \mathbb{N}$ |
| Paarungsselektion | Bestimmt aus der Population die Individuen, die über die Rekombination einen Nachkommen erzeugen sollen | $mar : I^\mu \rightarrow I^\rho$ |
| Fitness(funktion) | Funktion zur Bewertung eines Individuums | $\Phi : I \rightarrow \mathbb{R}$ |
| Genotyp | Interne Kodierung der Lösungskandidaten | $(x_1, \dots, x_n) =: I$ |
| Gen | Zeichen der internen Kodierung | x_i |
| Phänotyp | Interpretation der internen Codierung eines Lösungskandidaten (z.B. Binärcode) | $\Gamma(x_1, \dots, x_n) = \sum_{i=1}^n 2^{i-1} \cdot x_i$ |
| Lebensdauer | Maximale Anzahl der Generationen, in denen ein Individuum in der Population mitgeführt wird | $\kappa \in \mathbb{N}$ |

Tabelle 3.1: Bedeutung der Biologischen Begriffe in der Optimierung mit EA

des Lösungspunktes, verfügen. Evolutionsstrategien kodieren zudem die Schrittweiten für die Mutation im Individuum, so dass sich ein solches Individuum aus Objekt und Strategiekomponente zusammensetzt.

Die Optimierung erfolgt dann über einen iterativen Prozess, dessen Iteration als Generation bezeichnet wird. Ausgehend von einer aktuellen Population wird in jeder Generation über die genetischen Operatoren Rekombination und Mutation eine neue Population erzeugt. Basierend auf geringen Veränderungen der einzelnen Individuen und dem „Survival of the fittest“ Prinzip bewegen sich die Populationen, die im Laufe des Verfahrens erzeugt werden, sukzessive zum Optimum, da stets die besseren Individuen in einem Zyklus übernommen werden.

Beinahe zeitgleich wurden drei verschiedene Formen evolutionärer Algorithmen entwickelt. Dies waren die von Holland im Jahr 1962 vorgestellten Genetischen Algorithmen, die im Jahr 1965 von Rechenberg und Schwefel entwickelten Evolutionsstrategien und das 1962 von Fogel vorgestellte Evolutionäre Programmieren. Alle drei Verfahren werden benutzt, um Parametermengen zu optimieren. Eine vierte Form evolutionärer Algorithmen sind die Genetischen Programme, die 1992 von Koza entwickelt wurden. Anders als die drei obigen Arbeiten optimieren Genetische Programme keine Parametermengen, sondern approximieren Funktionen (Gerdes et al., 2004). Streng genommen handelt es sich bei genetischen Programmen aber ebenfalls um genetische Algorithmen, dessen Individuen Programme kodieren.

Im Folgenden wird auf solche Genetische Algorithmen und Evolutionsstrategien näher eingegangen, die zur Optimierung reellwertiger Funktionen eingesetzt werden können. Tatsächlich müssen EA auf die zu lösende Problemstellung genau angepasst werden. Dies betrifft insbesondere die Datenstruktur der Individuen und damit die Operatoren Mutation und Rekombination. Es wäre beispielsweise nicht viel versprechend, eine reellwertige Kodierung der Individuen zum Lösen des TSP zu verwenden. Vielmehr sollten bei einem TSP die Individuen in der Lage sein, eine Permutation der Knoten anzugeben und die Operatoren sicherstellen, dass stets gültige Permutationen erzeugt werden.

3.3.1 Evolutionsstrategien

Evolutionsstrategien (ES) wurden in den 60er Jahren von Rechenberg und Schwefel entwickelt und stellten zu Beginn nur eine Ablaufplan für die Durchführung von Experimenten aus dem ingenieur-technischen Bereich dar. Erwähnenswert ist hier die erfolgreiche Optimierung einer Heißwasser-Düse. Für diese Problemstellung mussten alle traditionellen Methoden (beispielsweise die Verfahren aus Abschnitt 2.2.1 oder 3.1) scheitern und nur eine sehr einfache Form der ES führte zum Erfolg. Dazu wurde ein Versuchsaufbau gering modifiziert und der so entstandene neue Aufbau bewertet. Ausgehend vom besseren der beiden Aufbauten setzte sich der Vorgang fort, bis ein optimales Versuchsmodell dieser Düse vorhanden war.

So war eine der ersten ES geboren, die auf einem Elter arbeitete und aus diesem über Mutation einen Nachkommen erzeugte. Der Bessere der Beiden wurde Elter der nächsten Generation. Schwefel setzte dieses Verfahren dann mit Einführung der ersten Rechner für die Optimierung von Funktionen ein.

Diese einfache ES ließ viele Eigenschaften einer auch noch so stark vereinfachten Annahme der Evolution außer Acht. Sie arbeitete auf keiner Population, sondern benutzt nur einen Elter, der asexuell, über Mutation, genau einen Nachkommen erzeugte. Ziel der ES war und ist es nicht, die Evolution so genau wie möglich nachzuahmen. Rechenberg sieht sie vielmehr als Vorbild und zugleich Anfangspunkt, von dem aus eine Weiterentwicklung auf die jeweilige Problemstellung durchgeführt werden kann (Rechenberg, 1973). Dennoch hat sich gezeigt, dass eine näher an der Evolution angesiedelte ES bessere Resultate bei der Optimierung aufweist, so dass, nach einigen Zwischenschritten, heute die $(\mu/\rho, \lambda)$ -ES bevorzugt angewendet wird. Sie arbeitet nun auf einer Menge an Individuen, die zusätzlich über einen sexuellen Mutationsoperator variiert

werden. Eine neuere Entwicklung stellt die $(\mu, \kappa, \lambda, \rho)$ -ES dar.

Ein stark vereinfachter Ablauf einer ES wird in Abbildung 3.3 dargestellt. Ein ES-Lauf beginnt mit der Initialisierung einer Population der Größe μ und einer anschließenden Schleife, in der die Evolution simuliert wird. Aus der aktuellen Population werden Individuen selekt-

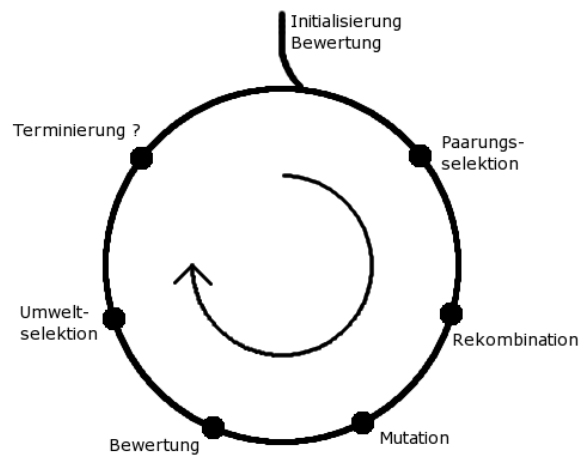


Abbildung 3.3: Zyklus eines EA

tiert (Paarungsselektion), aus denen über Rekombination, Mutation und Bewertung neue und veränderte Individuen erzeugt werden. Die Anzahl dieser Individuen wird über den Parameter λ ebenfalls über den gesamten Optimierungsvorgang konstant gehalten. Entsprechend werden Individuen in diesen Mengen Eltern bzw. Nachkommen genannt. Entweder aus der Menge der Nachkommen, oder aber aus beiden Mengen werden dann über einen Selektionsoperator (Umweltselektion) Individuen ausgewählt, um die Population für die nächste Generation zu erzeugen. Der Vorgang setzt sich dann iterativ weiter fort, bis eine Abbruchbedingung greift. Die Idee ist es, durch ein solches Vorgehen genau so gut an eine Problemstellung angepasste Individuen zu erzeugen, wie auch die Natur perfekt an ihre Umwelt angepasste Kreaturen geschaffen hat.

Beginnend mit der Repräsentation und dem Grundalgorithmus wird die ES nachfolgend genauer beschrieben. Anschließend werden die im Grundalgorithmus verwendeten Operatoren genauer erläutert.

Repräsentation

Individuen einer Evolutionsstrategie bestehen in der Regel aus mehreren Komponenten. Die wichtigste ist die Objektkomponente, da sie über eine passende Datenstruktur eine (nicht unbedingt optimale) Lösung angibt. Für Probleme, bei denen eine reellwertige Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ zu minimieren ist, ist die Objektkomponente ein Vektor \mathbf{x} , bestehend aus n reellwertigen Variablen. Die i -te Variable x_i gibt dabei die Position an der i -ten Koordinate an, so dass die gesamte Objektkomponente einen Lösungspunkt darstellt.

Neben der eigentlichen Lösung, der Objektkomponente, verfügen viele ES noch über eine zusätzliche Strategiekomponente zur Steuerung. In diesem Zusammenhang kann zwischen verschiedene Arten von Parametern zur Einstellung einer ES unterschieden werden. *Exogene Parameter* müssen vor Beginn eines ES Laufs vom Anwender eingestellt werden und werden über den gesamten Optimiervorgang konstant gehalten. *Endogene Parameter* hingegen werden in der Strategiekomponente abgelegt. Einer Einstellung durch den Anwender bedarf es nicht; diese Parameter werden während der Optimierung von der ES eingestellt. Dieses Prinzip wird als

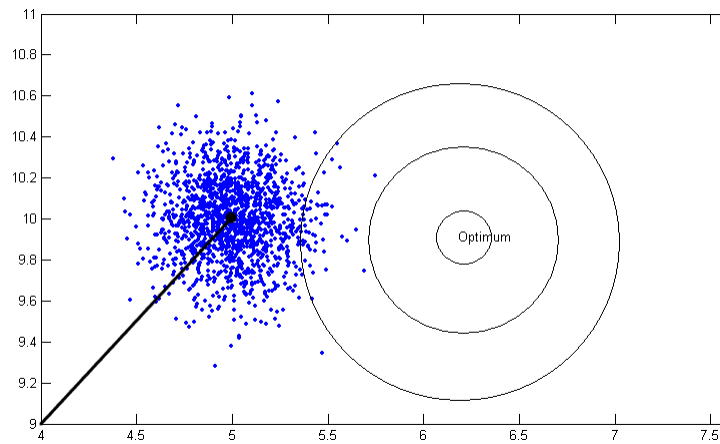
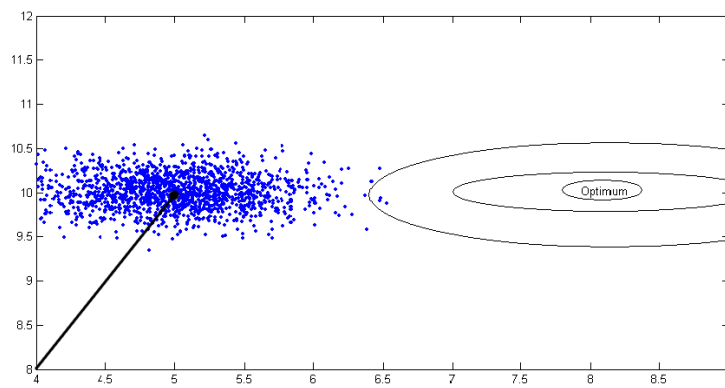


Abbildung 3.4: Mögliche Verteilung der Nachkommen bei einer Schrittweite

Abbildung 3.5: Mögliche Verteilung der Nachkommen bei n Schrittweiten

Selbstadaptation bezeichnet und im Folgenden noch genauer eingeführt.

Die Strategiekomponente codiert in dieser Arbeit die Schrittweiten. Es kann zwischen einer Schrittweite für das gesamte Problem und n Schrittweiten, je einer pro Koordinate, unterschieden werden. Wird eine Schrittweite verwendet, so verteilt die Mutation die Nachkommen symmetrisch um eine Hyperkugel, wie in Abbildung 3.4 dargestellt. Eine Schrittweite benötigt nur eine reellwertige Zahl im Individuum und lässt sich schnell rekombinieren als auch mutieren. Verlaufen die Höhenlinien der Fitnessfunktion jedoch elliptisch, so verteilen sich bei nur einer Schrittweite viele Nachkommen so, dass sie ein zu ihrem Elter schlechteren Wert erhalten. Hier kann dann für jede Koordinate eine Schrittweite verwendet werden, so dass bei einem n -dimensionalen Problem n Schrittweiten betrachtet werden (im Folgenden wird daher zwischen *einer* und n Schrittweiten unterschieden). Die Strategiekomponente muss somit Platz für n reelle Zahlen enthalten, um die Schrittweiten entsprechend ablegen zu können. Die Abbildung 3.5 zeigt die Höhenlinien der zugrunde liegenden Funktion und die um ein Hyperellipsoid angeordneten Individuen.

Man könnte noch weitere Fälle konstruieren, für die die Strategiekomponente zusätzlich erweitert werden könnte. Verläuft beispielsweise der Gradient nicht annähernd parallel zu den Koordinatenachsen, so können zu den n Schrittweiten noch zusätzlich Lagewinkel in der Strategiekomponente codiert werden (Beyer & Schwefel, 2002). Hildebrand (2001) codiert zusätzlich zu Schrittweiten und Lagewinkeln noch die Schiefe in der Strategiekomponente, um Aufgaben

im Bereich der *Wissenstransformation und Expertensysteme* besser lösen zu können. Der so entstandene Mutationsoperator verabschiedet sich aber von den Anforderungen, die Beyer an die Mutation stellt (vgl. Abschnitt über die Mutation), da er einem Bias unterliegt und je nach Schiefe nicht mehr symmetrisch ist. Somit ist im ersten Fall die Strategiekomponente ein Skalar $\sigma \in \mathbb{R}_+$ und im zweiten Fall ein n -dimensionaler Vektor $\sigma \in \mathbb{R}_+^n$, der in Anlehnung an klassische Optimierverfahren Schrittweite genannt wird und die Standardabweichung für die Mutation angibt. Da die Standardabweichung per Definition nur positive Werte annehmen kann, müssen die einzelnen σ_i stets positiv sein. Sie müssen daher positiv initialisiert werden und dürfen durch Rekombination oder Mutation nur derart verändert werden, dass sie ihre positiven Werte beibehalten. Mit der Aufnahme der Schrittweiten in ein Individuum soll erreicht werden, dass die ES nicht nur die Objektkomponente entsprechend der Problemstellung optimiert, sondern auch die Schrittweiten an die jeweiligen Erfordernisse angepasst werden. Je nach Problemstellung aber auch je nach Fortschritt der Optimierung sind unterschiedliche Schrittweiten erforderlich.

Neben der Objekt- und der Strategiekomponente wird in einem Individuum auch sein Fitnesswert abgelegt. Der Grund hierfür liegt an der Einfachheit einer ES, bei der die Hauptrechenleistung auf der Auswertung der Fitnessfunktion liegt. So kann man bei Individuen, die über eine längere Zeitspanne überleben, die mehrfache Auswertung der Fitnessfunktion vermeiden und Rechenzeit einsparen. Im hier vorliegenden Fall ist die Fitnessfunktion Φ äquivalent zur Zielfunktion f und der Fitnesswert eines Individuums i somit $\Phi(\mathbf{x}_i) = f(\mathbf{x}_i)$.

Als letzte Komponente wird bei ES, die an Stelle der Plus- oder Kommaselektion, die κ -Selektion verwenden, ein Integer-Wert K benötigt, der das aktuelle Alter eines Individuums angibt. Bei der κ -Selektion können nur solche Individuen überleben, die ein gewisses Alter, das mit dem Parameter κ angegeben wird, nicht überschreiten. Sie stellt somit ein Kompromiss zur bekannten Plus- bzw. Komma-Selektion dar, bei denen die Individuen unbegrenzte bzw. eine Lebensdauer von genau einer Generation haben.

Somit hat ein Individuum i in Generation t , welches n Schrittweiten adaptiert und die κ -Selektion verwendet, die folgende Repräsentation:

$$I_i^{(t)} = \left(\underbrace{x_{i,1}^{(t)}, \dots, x_{i,n}^{(t)}}_{\text{Objektkomponente } \mathbf{x}_i^{(t)}}, \underbrace{\sigma_{i,1}^{(t)}, \dots, \sigma_{i,n}^{(t)}}_{\text{Strategiekomponente } \sigma_i^{(t)}}, \Phi(\mathbf{x}_i^{(t)}), K_i \right) = \left(\mathbf{x}_i^{(t)}, \sigma_i^{(t)}, \Phi(\mathbf{x}_i^{(t)}), K_i \right)$$

oder kürzer ohne Indizierung der Individuen:

$$I^{(t)} = \left(\underbrace{x_1^{(t)}, \dots, x_n^{(t)}}_{\text{Objektkomponente } \mathbf{x}^{(t)}}, \underbrace{\sigma_1^{(t)}, \dots, \sigma_n^{(t)}}_{\text{Strategiekomponente } \sigma^{(t)}}, \Phi(\mathbf{x}^{(t)}), K \right) = \left(\mathbf{x}^{(t)}, \sigma^{(t)}, \Phi(\mathbf{x}^{(t)}), K \right)$$

Beide Schreibweisen werden in dieser Arbeit Anwendung finden und können leicht aus dem Kontext unterschieden werden.

(Für den Fall, dass nur eine Schrittweite in der Strategiekomponente betrachtet wird, vereinfacht sich der Vektor σ zu dem Skalar σ .)

Ablauf einer Evolutionsstrategie

Der iterative Ablauf einer Evolutionsstrategie orientiert sich an Abbildung 3.3 und wird über das Tupel $(\mu, \kappa, \lambda, \rho)$ eingestellt. Eine solche ES arbeitet dann in jeder Generation auf μ Individuen, die zusammen die Population bilden, die zu Beginn geeignet initialisiert werden muss. Solange eine Abbruchbedingung nicht greift werden aus dieser Population ρ Individuen ausgewählt, aus denen über Rekombination eine neue Strategie- und eine neue Objektkomponente

erzeugt wird. Die Position beider neuer Komponenten im Suchraum¹ orientiert sich an der Position ihrer Eltern, so dass alleine über die Rekombination in der Regel das globale Minimum nicht gefunden werden kann (Gerdes et al., 2004). Aus diesem Grund werden beide Komponenten zusätzlich mutiert, wodurch sie zufällig und bevorzugt gering über den Suchraum verschoben werden. Dabei wird zuerst die Strategiekomponente und anschließend die Objektkomponente mutiert, um eine Selbstadaptation der Strategiekomponente zu gewährleisten.

Um ein vollständiges neues Individuum zu erzeugen, wird im letzten Schritt noch seine Fitness berechnet und ihm sein Alter zugewiesen.

Insgesamt sind λ Nachkommen zu erzeugen, so dass sich der obige Vorgang λ -mal wiederholen muss. Danach existieren zwei Mengen (die der Eltern und die der Nachkommen), aus denen die besten μ Individuen, die das mit κ angegebene Alter nicht überschreiten, in die Folgegeneration übernommen werden, und dort die neue Population darstellen. Erst diese Selektion macht die ES zu einem „vollwertigen“ Optimierverfahren. Bis dahin wurden zufällig neue Individuen erzeugt, die sich symmetrisch um die Individuen der aktuellen Population verteilten. Durch Übernahme der besten dieser Individuen wird sich die neue Population i.d.R. einem Minimum annähern. Nach Inkrementierung des Alters der Individuen in der neuen Population setzt sich der Vorgang, beginnend mit Prüfung der Abbruchbedingung, fort.

Der daraus resultierende Pseudocode ist in Algorithmus 3.2 dargestellt. In den Zeilen 1 und

Algorithmus 3.2 : $(\mu, \kappa, \lambda, \rho)$ -Evolutionstrategie

```

1  $t := 0$ ;
2 Initialisiere  $P^{(t)} := \{ I_i^{(t)} = (\mathbf{x}_i^{(t)}, \sigma_i^{(t)}, \Phi(\mathbf{x}_i^{(t)}), 1), i = 1, \dots, \mu \}$ ;
3 while ( $\text{term}(P^{(t)}) \neq \text{true}$ ) do
4   for ( $l = 1, \dots, \lambda$ ) do
5      $Z_l := \text{mar}(P^{(t)}, \rho)$ ;
6      $\sigma_l^{(t)} := \text{rec}_\sigma(Z_l)$ ;
7      $\mathbf{x}_l^{(t)} := \text{rec}_x(Z_l)$ ;
8      $\sigma_l''^{(t)} := \text{mut}_\sigma(\sigma_l^{(t)}, \tau_0, \tau)$ ;
9      $\mathbf{x}_l''^{(t)} := \text{mut}_x(\mathbf{x}_l^{(t)}, \sigma_l''^{(t)})$ ;
10    Bewerte  $I_l^{(t)} : \Phi(\mathbf{x}_l''^{(t)})$ ;
11     $K_l := 0$ ;
12     $P^{(t)} := \{ (\mathbf{x}_l''^{(t)}, \sigma_l''^{(t)}, \Phi(\mathbf{x}_l''^{(t)}), K_l), l = 1, \dots, \lambda \}$ ;
13     $P^{(t+1)} := \text{sel}_\kappa(\{ I_i^{(t)} \in P^{(t)} \cup P^{(t)} \mid K_i < \kappa \}, \mu)$ ;
14    for ( $I_i^{(t)} \in P^{(t+1)}$ ) do
15       $K_i^{(t)} := K_i^{(t)} + 1$ ;
16     $t := t + 1$ ;

```

2 wird die Initialisierung durchgeführt, in der der Generationenzähler auf null und die Start-Population $P^{(0)}$ bestimmt wird. Falls die Abbruchbedingung in Zeile 3 von der aktuellen Population $P^{(t)}$ nicht erfüllt wird, werden in den Zeilen 4 – 12 durch λ -fache Anwendung von Rekombination, Mutation und Bewertung neue Individuen mit Alter 0 erzeugt, die schließlich in Zeile 12 zu Population der Nachkommen vereinigt werden. Danach erfolgt in Zeile 13 die κ -Selektion, die die besten μ Individuen, die das Alter κ noch nicht erreicht haben, in die neue Population $P^{(t+1)}$ aufnehmen. Nach Inkrementierung des Alters der Individuen in $P^{(t+1)}$ in

¹Dabei ist zwischen zwei Suchräumen zu unterscheiden: Dem Suchraum \mathbb{R}^n der Problemstellung und dem Suchraum \mathbb{R}_+^n bzw. \mathbb{R}_+ der Schrittweiten

der Zeile 15 sowie des Generationenzählers in Zeile 16 setzt sich der Vorgang mit Prüfung der Abbruchbedingung in Zeile 3 fort.

Die im Algorithmus 3.2 verwendeten Operatoren *Initialisierung*, *Paarungsselektion*, *Rekombination*, *Mutation*, *Umweltselektion* und *Abbruchbedingung* werden im Folgenden genau eingeführt. Anschließend wird das hinter der Selbstadaptation stehende Prinzip vorgestellt und eine Empfehlung für die exogenen Parameter der ES gegeben.

Initialisierung

In der bisherigen Betrachtung der Optimierungsalgorithmen wurde die Initialisierung völlig außer Acht gelassen, obwohl sie maßgeblich zum Erfolg dieser Algorithmen beiträgt. Es ist intuitiv klar, dass die Lösungspunkte (im Fall der ES die Objektkomponenten) in der Nähe des globalen Optimums initialisiert werden sollten, falls Informationen über die (ungefähre) Lage des globalen Optimums vorhanden sind. Ansonsten sollte sie gleichverteilt über den Suchraum verteilt werden. Schwefel und Rudolph (1995) initialisieren die Objektkomponenten der Startpopulation in dem die ausgehend von einer Objektkomponente $\mathbf{x}_0^{(0)}$ die übrigen $\mu - 1$ Komponenten nach folgender Vorschrift erzeugen:

$$x_{k,i}^{(0)} := x_{0,i}^{(0)} + c \cdot N_k(0, 1) \quad \text{für } i = 1, \dots, n \text{ und } k = 2, \dots, \mu.$$

Für die spätere experimentelle Analyse eignen sich andere Verfahren zur Initialisierung besser. Diese werden im weiteren Verlauf dieser Arbeit vorgestellt.

Um ein komplettes Individuum zu erzeugen, muss noch die Strategiekomponente initialisiert werden. Schwefel (1977) empfiehlt die Strategiekomponente wie folgt zu initialisieren:

$$\sigma_i = \frac{\Delta x_i}{\sqrt{n}}$$

wobei Δx_i die geschätzte Distanz zwischen dem Startwert und der i -ten Koordinate x_i des Optimum ist. Bäck (2000) schlägt aufgrund experimenteller Untersuchungen $\sigma_i = 3$ für $i = 1, \dots, n$ vor. Er bemerkt, dass zu große Anfangs-Schrittweiten dazu führen können, dass die ES divergiert. Werden sie hingegen eher klein gewählt, so werden sie durch die Selbstadaptation schnell an die Anforderungen angepasst. Aber auch diese Empfehlung ist kritisch zu betrachten. Beispielsweise ist intuitiv klar, dass bei sehr kleinen Suchräumen, die Schrittweite $\sigma_i = 3$ zu groß sein kann.

Paarungsselektion

Die Paarungsselektion (mar: $I^\mu \rightarrow I^\rho$) wählt aus der Elternpopulation die für die Rekombination benötigten ρ Individuen aus. Im Fall der ES sind alle Individuen in der Population gleichberechtigt und ihre Auswahlwahrscheinlichkeit beträgt $\frac{1}{\mu}$. Durch ziehen einer gleichverteilten, diskreten Zufallszahl z aus der Menge $\{1, \mu\}$ werden einzelne Individuen I_z aus der aktuellen Elternpopulation in eine Menge Z eingefügt, bis sie die angegebene Größe ρ erreicht. Bereits gezogene Individuen werden zurückgelegt, so dass sie auch mehrfach gezogen werden können. Der Grund dafür ist nach (Beyer, 2001) vor allem in der leichteren Implementierung zu suchen.

Rekombination

Die Rekombination erzeugt ausgehend von ρ Individuen ein Neues, die dem sie die Information der ρ Individuen in einer geeigneten Form miteinander verbindet. Sie setzt sich aus zwei

Operatoren rec_σ und rec_x zusammen, von denen einer die Strategie- und der Andere die Objektkomponenten rekombiniert. Im Fall der reellwertig kodierten Individuen sind die zwei bekanntesten Operatoren die diskrete und die intermediäre Rekombination, die sowohl auf die Objekt-, als auch die Strategiekomponente angewendet werden können. Sie werden hier daher nur für die Objektkomponente eingeführt. Die Rekombination der Strategiekomponente erfolgt analog. Dem Rekombinationsoperator stehen in jedem Fall ρ Individuen aus Z mit verschiedenen Ausprägungen der einzelnen Koordinaten zu Verfügung.

Die *intermediäre Rekombination* mittelt die ρ Ausprägungen der i -ten Koordinate und erzeugt dabei eine Neue. Insgesamt sind Belegungen für n Koordinaten zu erzeugen, die dann die neue Objektkomponente x' bilden. Die intermediäre Rekombination berechnet somit:

$$\forall i \in \{1, \dots, n\} : \quad x_{\text{neu},i} := \frac{1}{\rho} \sum_{k=1}^{\rho} x_{k,i}$$

Der Nachkomme der intermediären Rekombination wird eindeutig durch die Menge Z bestimmt, da diese Rekombinationsvariante eine Objektkomponente erzeugt, die Schwerpunkt der Eltern in Z ist.

Die *diskrete Rekombination* (auch dominante Rekombination genannt) verzichtet auf das Mitteln der ρ Ausprägungen der i -ten Koordinate und wählt dafür gleichverteilt einen der vorhandenen ρ Werte. Somit ist

$$\forall i \in \{1, \dots, n\} : \quad x_{\text{neu},i} := x_{m_i,i} \quad \text{mit } m_i \in U_\rho.$$

zu berechnen, um eine neue Objektkomponente x' zu erzeugen. Hier wird (anders als bei der intermediären Rekombination) der Nachkomme nicht alleine durch die Menge Z bestimmt, sondern zusätzlich noch durch die Zufallszahlen m_i .

Die Abbildung 3.6 stellt die Unterschiede beider Rekombinationsmethoden mit $\rho = 2$ im \mathbb{R}^2 dar. Ausgehend von zwei Elternindividuen, die als schwarze Quadrate dargestellt sind, können

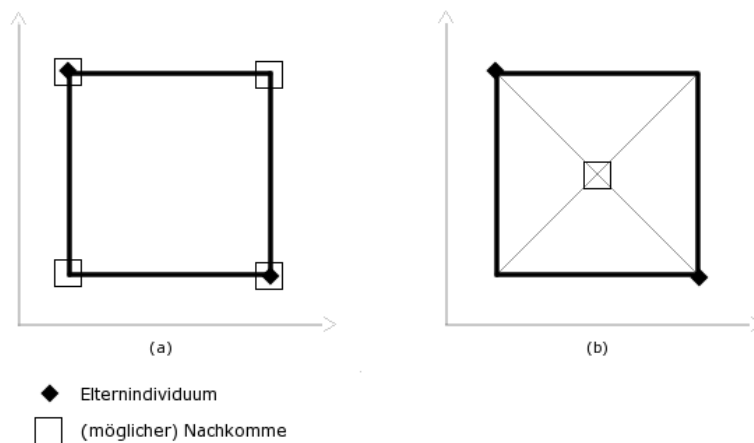


Abbildung 3.6: Visualisierung der (a) diskreten Rekombination, (b) intermediären Rekombination

in Fall der diskreten Rekombination (a) je nach gezogener Zufallszahl eines der vier möglichen Nachkommen erzeugt werden. Im Fall der intermediären Rekombination (b) wird in jeder Koordinate der Mittelwert gebildet, so dass nach Bestimmung der Menge Z der Nachkomme

eindeutig definiert ist.

Der Grund, dass nicht zwischen unterschiedlichen Operatoren für Objekt- und Strategiekomponente unterschieden werden muss, liegt darin, dass beide Varianten nicht zu negativen Werten in der Strategiekomponente führen können, wenn die Strategiekomponenten aller Eltern vor der Rekombination positiv waren. Dies ist aber in der Initialisierung und der Mutation sichergestellt. Aus den positiven Elementen der Strategiekomponenten werden im Fall der diskreten Rekombination einzelne Elemente entnommen, bzw. im Fall der intermediären Rekombination einzelne Elemente gemittelt. Da aber alle Elemente positiv waren, können durch Entnehmen oder Mitteln selbst nur positive Werte entstehen. Somit erzeugen beide Operatoren stets gültige Strategiekomponenten.

Bereits 1977 stellte Schwefel durch experimentelle Untersuchungen fest, dass durch Einsatz der Rekombination eine Beschleunigung der Optimierung zu verzeichnen ist. Er konnte damals aber noch nicht erklären, wodurch dieser Effekt zu Stande kommt (Schwefel, 1977). Beyer (2001) und Goldberg (1989) versuchen den Nutzen der Rekombination mit der *Genetic repair*- bzw. der *Building block*-Hypothese zu erklären. Beide Hypothesen sollen hier aber nicht weiter betrachtet werden.

Mutation

Die Mutation ist der Operator, der es den Individuen ermöglicht, jeden Punkt im Suchraum zu erreichen, und trägt nach (Gerdes et al., 2004) hauptsächlich zum Auffinden des globalen Optimums bei. An die Mutation werden die folgenden Bedingungen gestellt (Beyer, 2001):

- Erreichbarkeit: Die Mutation muss es erlauben, jeden Punkt im Suchraum nach endlicher Zeit zu erreichen. Diese Bedingung ist zwingend für die Konvergenz der ES zum globalen Optimum, da ansonsten das Optimum ein Punkt sein könnte, der nach endlicher Zeit nicht erreicht wird.
- Skalierbarkeit: Die Mutation sollte so beschaffen sein, dass die aus ihr resultierenden Schrittweiten angepasst werden können. Je nach Fortschritt der Optimierung, aber auch je nach Problemstellung, können unterschiedliche Schrittweiten erforderlich werden.
- Maximale Entropie: Die Mutation soll keinen Annahmen unterliegen und ziellos sein. Nur die Umweltselektion soll der ES ihr Ziel geben.
- Symmetrie: Der Mutationsoperator soll ferner isotrop sein. Somit soll er in alle Richtungen gleiche Veränderungen verursachen. Als Folge muss der Erwartungswert der Verteilung, auf der die Mutationsfunktion basiert, null sein.

Aus der ersten Bedingung, der Erreichbarkeit, folgt direkt, dass hier zwingend zwischen je einem Mutationsoperator für die Strategiekomponente $\sigma \in \mathbb{R}_+^n$ (bzw. $\sigma \in \mathbb{R}_+$) und einem für die Objektkomponente $\mathbf{x} \in \mathbb{R}^n$ zu unterscheiden ist, da beide Operatoren auf unterschiedlichen Suchräumen arbeiten. Die Mutation setzt sich somit aus den Operatoren mut_σ und mut_x zusammen, von denen einer die Strategie- und der andere die Objektkomponente mutiert. Wie sich im Folgenden noch zeigen wird, ist auf die Reihenfolge beider Operatoren zu achten: Die Strategiekomponente muss vor der Objektkomponente mutiert werden, um eine Selbstadaptation der Strategiekomponente zu ermöglichen.

Beyer und Schwefel (2002) verwenden logarithmisch normalverteilte Zufallszahlen und Multiplikation, um die Strategiekomponente zu mutieren. Ihr Mutationsoperator mut_σ greift dann

auf die durch Rekombination erzeugten Individuen σ' und zwei exogene Parameter τ und τ_0 zu, und wird wie folgt realisiert:

$$\sigma'' = e^{\tau_0 \cdot N(0,1)} \cdot (\sigma'_1 \cdot e^{\tau \cdot N_1(0,1)}, \dots, \sigma'_n \cdot e^{\tau \cdot N_n(0,1)}) .$$

Wird an Stelle n Schrittweiten nur eine Schrittweite verwendet vereinfacht sich die Mutation zu

$$\sigma'' = e^{\tau \cdot N(0,1)} \cdot \sigma' .$$

Die logarithmische Normalverteilung $e^{N(0,\tau)} = e^{\tau \cdot N(0,1)}$, deren Dichte in Abbildung 3.7 dargestellt ist, erzeugt nur Zahlen aus dem Intervall $]0, \infty[$. So ist sichergestellt, dass die Elemente

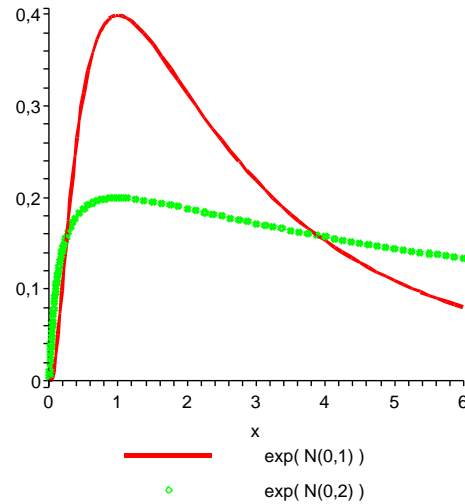


Abbildung 3.7: Dichte einer logarithmischen Normalverteilung

der Strategiekomponente stets positiv bleiben, es ihnen dennoch ermöglicht wird, alle erlaubten Werte anzunehmen. Der Erwartungswert der logarithmischen Normalverteilung ist 1 und die „Wahrscheinlichkeit“ für eine Zahl k ist gleich der „Wahrscheinlichkeit“ für $\frac{1}{k}$ und nimmt mit steigendem k ab². Somit ist der Mutationsoperator isotrop und symmetrisch. Weil die logarithmische Normalverteilung sich über die Standardabweichung τ einstellen lässt, sind schließlich alle Bedingungen, die Beyer an die Mutation stellt, erfüllt.

Der obige Operator mut_σ verwendet n unabhängige logarithmisch normalverteilte Zufallszahlen $e^{\tau N_i(0,1)}$, $i = 1, \dots, n$ mit Standardabweichung τ von denen je eine dieser Zufallszahlen genau eine Koordinate der Strategiekomponente mutiert. So werden die einzelnen Koordinaten lokal an die Problemstellung angepasst. Eine weitere unabhängige logarithmisch normalverteilte Zufallszahl mit Standardabweichung τ_0 wird zu Mutation aller Koordinaten verwendet und passt sie global an die Problemstellung an (beispielsweise durch insgesamt größere oder kleinere Schrittweiten). Beyer und Schwefel (2002) schlagen für den Parameter $\tau_0 \propto \frac{1}{\sqrt{2n}}$ und für den Parameter $\tau \propto \frac{1}{\sqrt{2\sqrt{n}}}$ vor, falls n Schrittweiten adaptiert werden. Im Falle der Adaptation nur einer Schrittweite wird $\tau \propto \frac{1}{\sqrt{n}}$ vorgeschlagen.

Zur Mutation der Objektkomponente werden normalverteilte Zufallszahlen und Addition verwendet. Die Dichte der Normalverteilung $N(0, \sigma) = \sigma \cdot N(0, 1)$ ist in Abbildung 3.8 dargestellt. Der Mutationsoperator mut_x greift dann auf die bereits mutierte Strategiekomponente, sowie

²Tatsächlich ist bei kontinuierlichen Zufallsgrößen die Wahrscheinlichkeit für eine bestimmte Zahl gleich 0. Stattdessen müssen hier endliche Intervalle betrachtet werden.

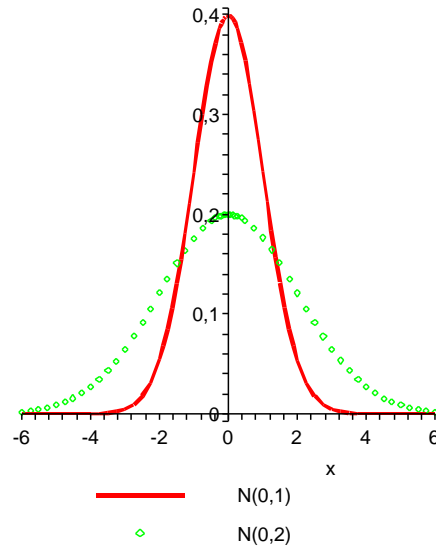


Abbildung 3.8: Dichte der Normalverteilung

die durch Rekombination gebildeten Objektkomponente \mathbf{x}' zurück und wird wie folgt realisiert:

$$\mathbf{x}'' = \mathbf{x}' + (\sigma_1'' \cdot N_1(0, 1), \dots, \sigma_n'' \cdot N_n(0, 1)) .$$

Wiederum kann der Ausdruck bei nur einer Schrittweite zu

$$\mathbf{x}'' = \mathbf{x}' + \sigma'' \cdot (N_1(0, 1), \dots, N_n(0, 1))$$

reduziert werden.

Die Schrittweiten des Mutationsoperators werden somit durch die bereits mutierte Strategiekomponente skaliert. Aufgrund der Addition der Objektkomponente mit dem Vektor z erfüllt dieser Mutationsoperator auch die weiteren von Beyer gestellten Anforderungen an die Mutation. Die Normalverteilung hat den Erwartungswert 0 und ist symmetrisch. Sie erzeugt Zufallszahlen aus dem Intervall $[-\infty, +\infty]$ und ermöglicht es somit jeden Punkt im hier betrachteten Suchraum zu erzeugen.

Prinzip der Selbstadaptation

Die hier vorgestellte ES stellt ihre Schrittweiten, wie bereits beschrieben, eigenständig ein. Sie hat somit zwei Optimierungsprobleme zu lösen: Die gegebene Problemstellung, bei der sie auf eine Fitnessfunktion zurückgreifen kann und das Problem der optimalen Schrittweite, zu dem keine Fitnessfunktion vorliegt. Somit ist es objektiv nicht möglich, die durch Rekombination und Mutation manipulierten Schrittweiten zu bewerten. Beyer und Schwefel (2002) mutieren daher zuerst die Schrittweiten und anschließend die Objektkomponente, wobei sie hier auf die bereits mutierte Strategiekomponente zurückgreifen. Gute Strategiekomponenten werden i.d.R. zu guten Objektkomponenten führen. Schlechte Strategiekomponenten führen hingegen eher zu schlechten Objektkomponenten. Durch anschließende Bewertung der Objektkomponente ist so aber eine indirekte Bewertung der Strategiekomponente möglich, denn die Güte der Objektkomponente hängt stark von der Güte der Strategiekomponente ab.

Umweltselektion

Die Umweltselektion (sel: $I^m \rightarrow I^\mu$) dient zur Reduktion des Überschusses an Individuen und betont dabei besonders gute Individuen. Durch die Operatoren Rekombination und Mutation wurden neue Individuen geschaffen, die sich annähernd symmetrisch um ihre Eltern verteilen. Beide Operatoren verfolgen daher kein bestimmtes Ziel, insbesondere nicht das Auffinden des globalen Minimums. Diese Aufgabe übernimmt die Selektion. Sie wählt aus m gegebenen Individuen deterministisch die besten, für die Population der Folgegeneration benötigten, μ Individuen aus. Das werden i.d.R. die Individuen sein, die die Rekombination und Mutation zu einem Minimum hin bewegt hat. Individuen, die in die falsche Richtung verschoben wurden, werden in die neue Generation nicht übernommen und verworfen. Insgesamt wird sich die neue Population so einem Minimum annähern.

Die Anzahl der gegebenen m Individuen variiert dabei je nach Art der Selektion. Die zwei bekanntesten Arten sind die Plus- und die Komma-Selektion. Die κ -Selektion ist eine neuere Entwicklung, die ein Kompromiss zwischen Plus- und Komma-Selektion darstellt.

Die Plus-Selektion (symbolisiert durch $(\mu/\rho + \lambda)$) wählt sowohl aus der Eltern- als auch der Nachkommenpopulation die besten μ Individuen aus. Somit stehen $m = \mu + \lambda$ Individuen für die Selektion zu Verfügung. Die Plus-Selektion hat Vor- wie auch Nachteile. Es ist offensichtlich, dass das beste Individuum nicht *aussterben* kann, so dass eine ES mit Plusselektion eine notwendige Bedingung für die Konvergenz zu einem Optimum erfüllt. Die Fitness des besten Individuums verläuft hier monoton fallend, so dass eine $(\mu/\rho + \lambda)$ -ES stark dazu neigt, in einem lokalen Minimum stecken zu bleiben. Ihr gelingt es nur dann ein lokales Minimum zu überwinden, wenn durch Rekombination und Mutation ein Nachkomme erzeugt wird, der in ein tieferes Tal fällt, als das bisher schlechteste Individuum.

Die Komma-Selektion (symbolisiert durch $(\mu/\rho, \lambda)$) gibt den Individuen hingegen eine maximale Lebensdauer, die genau eine Generation beträgt. Nach der Erzeugung der Nachkommen werden alle Eltern vergessen und nur die λ Nachkommen der Selektion in die nächste Generation unterzogen. Dies setzt zwingend voraus, dass $\lambda > \mu$ ist. Wäre λ kleiner, so würde die Populationsgröße degenerieren. Aber auch die Gleichheit von μ und λ reicht nicht aus, da sonst keine Selektion stattfinden und die ES nur ein *random-walk* darstellen würde. Alle Nachkommen müssten in die nächste Generation übernommen werden und die ES würde ziellos Punkte im Suchraum erzeugen. Die Vor- und Nachteile kehren sich zur Plus-Selektion um. Hier kann es einer ES zwar eher gelingen aus einem lokalen Optimum zu entkommen, dafür konvergiert sie aber nicht zwangsläufig gegen das Optimum. Daher sollte bei Anwendung der Komma-Selektion nach (Gerdes et al., 2004) das beste Individuum zusätzlich gespeichert werden.

Die κ -Selektion versucht die Vorteile aus Plus- und Kommaselektion zu vereinen, in dem sie eine variable, über den Parameter κ einstellbare Lebensdauer der Individuen zulässt. Die Menge der für die Selektion zu Verfügung stehender Individuen ist gegeben durch $\{I_i^{(t)} \in P^{(t)} \cup P^{(t)} \mid K_i < \kappa\}$ und ist somit in ihrer Größe variabel. Mit Hilfe der κ -Selektion können die Plus-Selektion ($\kappa = \infty$), sowie die Kommaselektion ($\kappa = 1$) modelliert werden, so dass in der späteren Matlab-Implementierung aufgrund der Flexibilität nur die κ -Selektion verwendet wird.

Die Parameter μ und λ sind bei der κ -Selektion mit $\kappa < \infty$ so zu wählen, dass gilt $\lambda > \mu$. Somit ist sichergestellt, dass die Populationsgröße μ nicht degenerieren kann, da selbst im Fall des *Aussterbens* der gesamten Population $P^{(t)}$ genügend Nachkommen erzeugt wurden, um aus ihnen die Population $P^{(t+1)}$ zu bilden.

Abbruchkriterium

Wie bei den vorhergehenden Suchalgorithmen kann auch bei den ES nicht eindeutig bestimmt werden, wann die Suche abgebrochen werden kann. Abbruchkriterien stellen hier immer Heuristiken dar, die zu beliebt guten Ergebnissen führen können. Typische Abbruchkriterien sind (Gerdes et al., 2004), (Schwefel, 1994):

1. Überschreitung einer maximalen Rechenzeit
2. Überschreitung einer maximalen Anzahl von Generationen
3. Finden einer Lösung mit vorgegebener Mindestgüte
4. Keine Verbesserung der Lösung in den letzten T Generationen
5. Unterschreitung einer vorgegebenen mittleren Verbesserung in den letzten Generationen
6. Unterschreiten einer Mindestschrittweite
7. Die Differenz zwischen dem besten und den schlechtesten Individuum in einer Generation unterschreitet eine Schranke

Die ersten zwei Abbruchkriterien können keine Konvergenz zu einem Optimum garantieren. Sie werden vor allem bei beschränkten Ressourcen benötigt, wie z.B. einer zeitlich beschränkten Rechenzeit.

Das dritte Kriterium ist eher von praktischem Interesse. Für Anwendungen, bei denen es ausreichend ist, eine Lösung mit bestimmter Güte zu erreichen, kann bei Auffinden einer solchen Lösung abgebrochen werden.

Die anderen vier Kriterien versuchen abzubrechen, wenn objektive Anhaltspunkte für ein Minimum eintreten. Die Abbruchkriterien 4 – 6 können aber dennoch vorzeitig konvergieren. Bewegt sich die Population durch ein flaches Plateau, werden die Verbesserungen pro Generation sehr gering, in engen Schluchten hingegen wird die Selbstadaptation die Schrittweiten anpassen und sehr klein wählen, obwohl in beiden Fällen das Minimum noch weit entfernt sein kann.

Das letzte oben aufgeführte Abbruchkriterium basiert auf der Annahme, dass sich die Individuen einer Generation hinsichtlich der Objektkomponente, aber auch hinsichtlich der Fitnesswerte solange unterscheiden, solange das Minimum noch nicht gefunden ist. Zum Ende des Optimiervorgangs werden sich schließlich die Individuen um das Minimum sammeln, so dass dann abgebrochen werden kann, wenn die Differenz zwischen dem besten und den schlechtesten Individuum in einer Generation eine Schranke unterschreitet.

Wie bereits die Initialisierung ist auch das Abbruchkriterium problemabhängig zu wählen, so dass eine generelle Empfehlung für eine Methode nicht gegeben werden kann.

Behandlung von Nebenbedingungen

Nebenbedingungen können sehr einfach behandelt werden. Sind λ Nachkommen zu erzeugen, so wird der Prozess aus Paarungsselektion, Rekombination und Mutation so lange fortgesetzt, bis λ Nachkommen erzeugt werden, die die Nebenbedingungen einhalten. Erzeugte Nachkommen, die mindestens eine Nebenbedingung verletzen, werden sofort verworfen.

Zum Auffinden eines zulässigen Startpunkts kann die Nullstelle der Funktion

$$G(x) = \sum_{j=1}^m g_j(x) \cdot \delta_j(x) \quad \text{mit}$$

$$\delta_j(x) = \begin{cases} 1 & \text{falls } g_j(x) > 0 \\ 0 & \text{sonst} \end{cases}$$

gesucht werden. Jeder Punkt x für den $G(x)$ zunimmt, nähert sich dem zulässigen Bereich an, bis schließlich für ein x gilt $G(x) = 0$. Dann ist eine zulässige Lösung gefunden, die als ein Startpunkt verwendet werden kann (Schwefel & Rudolph, 1995).

Eine andere Varianten der ES

Yao & Liu stellten 1997 in ihrem Aufsatz (Yao & Liu, 1997) mit dem eindrucksvollen Namen „Fast Evolution Strategies“ eine neue ES (im Folgenden mit Cauchy-ES bezeichnet) vor. Diese ES tauscht lediglich den Mutationsoperator für die Objektkomponente aus und verfährt sonst wie eine klassische ES. Wollte man den Algorithmus 2 um die neue Mutation erweitern, so müsste nur der Operator in Zeile 9 ausgetauscht werden. Für klassische ES ist er realisiert als

$$\mathbf{x}'' = \mathbf{x}' + \sigma'' \cdot (N_1(0, 1), \dots, N_n(0, 1)) .$$

Die Cauchy-ES tauscht diesen Operator aus gegen

$$\mathbf{x}'' = \mathbf{x}' + \sigma'' \cdot (C_1(0, 1), \dots, C_n(0, 1)) .$$

Dieser Operator verwendet nun keine normalverteilten, sondern cauchyverteilte Zufallszahlen mit Zentrum 0 und Breitenparameter 1 ($C(0, 1) = \frac{1}{\pi \cdot (1+x^2)}$), die pro Koordinate neu erzeugt werden.

Yao & Liu benutzen somit eine Cauchyverteilung anstelle der Normalverteilung zur Mutation der Objektkomponente. Der Unterschied zwischen beiden Verteilungen kann der Abbildung 3.9 entnommen werden. Im Gegensatz zu Normalverteilung ist die Varianz der Cauchyverteilung unendlich, was gut an der sich sehr langsam an die Abszisse annähernden Kurve zu erkennen

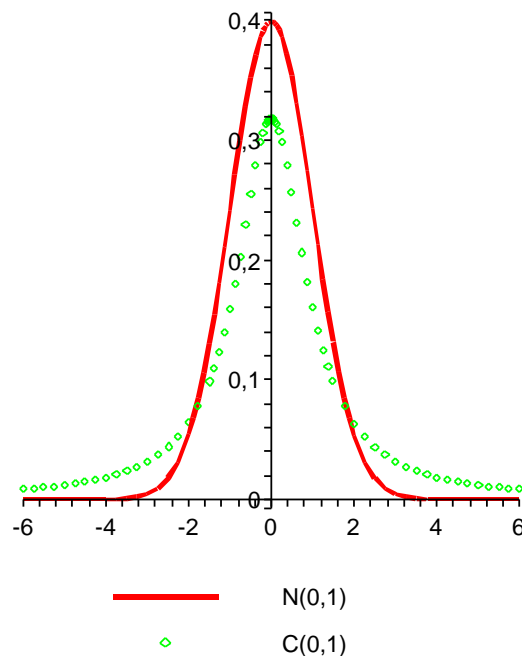


Abbildung 3.9: Gegenüberstellung der Dichten von Cauchy- und Normalverteilung

ist. Die Cauchyverteilung erzeugt somit offensichtlich mit einer größeren Wahrscheinlichkeit Zufallszahlen, welche weit entfernt vom Ursprung sind.

Beide Autoren führten experimentelle Untersuchungen mit der klassischen ES und der Cauchy-ES durch und stellten fest, dass die Cauchy-ES auf multimodalen Problemen mit vielen lokalen Minima die klassischen ES übertrifft. Auch beim Umgang mit Plateaus schnitt die Cauchy-ES besser ab. Für multimodale Probleme mit nur wenigen lokalen Minima unterschieden sich beide Strategien kaum. Die klassischen ES konnten nur bei unimodalen Problemen die Cauchy-ES übertreffen.

Die Autoren erklärten sich dieses Ergebnis damit, dass die Cauchy-ES größere Sprünge ermöglicht und somit von einem lokalen Minimum schneller in das globale Minimum springen kann. Betrachtet man aber die Adaptation der Schrittweiten, so sollte es auch der klassischen ES gelingen, bei Bedarf größere Schrittweiten einzustellen und ihrerseits auch große Sprünge durchzuführen. Dieses Thema soll hier aber nicht mehr weiter betrachtet werden. Es wird nochmals im Kapitel 5 aufgegriffen, in dem experimentelle Untersuchungen beider ES durchgeführt werden.

Zusammenfassung der Evolutionsstrategie

Evolutionsstrategien sind Optimierverfahren, deren Herkunft aus dem ingenieur-technischen Bereich stammt und die auf Prinzipien der biologischen Evolution basieren. Die hier beschriebene ES kann über eine Anzahl an Steuerparametern eingestellt werden, so dass diese ES vollständig über das folgende Tupel beschrieben werden kann.

$$ES = (P^{(0)}, \mu, \kappa, \lambda, \rho, \tau, \tau_0, r_x, r_\sigma, n_\sigma, \text{term}) .$$

Betrachtet man die Initialisierung sowie das Abbruchkriterium als Bestandteil der Problemstellung, und ersetzt man die Parameter τ und τ_0 durch einen Parameter c_τ , so reduziert sich das Tupel auf:

$$ES = (\mu, \kappa, \nu, \sigma_i^{(0)}, \rho, c_\tau, r_x, r_\sigma, n_\sigma) ,$$

mit den Parametern, $\tau = c_\tau \cdot \frac{1}{\sqrt{2\sqrt{n}}}$ und $\tau_0 = c_\tau \cdot \frac{1}{\sqrt{2n}}$ im Fall von n Schrittweiten und $\tau = c_\tau \cdot \frac{1}{\sqrt{n}}$ im Fall nur einer Schrittweite, sowie $\lambda = \nu \cdot \mu$.

Die Einstellung dieser Parameter hat wesentlichen Einfluss auf die Optimierung und ist problemabhängig zu wählen. In vielen praktischen Anwendungen kann ein Individuum nur durch Simulation bewertet werden (vgl. Kapitel 2), so dass nur wenige Generationen für die ES zu Verfügung stehen und die ES somit optimal eingestellt sein muss. Dann ist eine zusätzliche Optimierung des Optimierungsalgorithmus erforderlich. Durch experimentelle Untersuchungen wurden aber Standardwerte ermittelt (die jedoch für jede neue Problemstellung kritisch betrachtet werden sollten), die in Tabelle 3.2 angegeben sind. Die empfohlenen Standardwerte sind Bäck (1994) entnommen. Zusätzlich fasst die Tabelle 3.2 die Bedeutung und den Wertebereich der einzelnen Parameter nochmals zusammen.

3.3.2 Genetische Algorithmen

Genetische Algorithmen sind eine Entwicklung von John Holland, die zeitgleich, aber völlig unabhängig von der Entwicklung der Evolutionsstrategien an der Universität Michigan erfolgte. Ziel der Untersuchungen von Holland war die Abstraktion und Erklärung der adaptiven Prozesse natürlicher Systeme, sowie die Entwicklung künstlicher Systeme, die die elementaren Mechanismen ihrer natürlichen Vorbilder beinhalteten. Hollands Interesse war stark auf die adaptiven Prozesse fokussiert, die aus dem genetische Code und den genetischen Mechanismen hervorgehen. Er bemerkte schnell, dass diese Adaptation auch in praktischen Anwendungen

| SYMBOL | BEDEUTUNG | WERTEBEREICH | STANDARDWERT |
|------------------|--|------------------------------|-------------------|
| $P^{(0)}$ | Startpopulation | $X^n \subseteq \mathbb{R}^n$ | — |
| μ | Populationsgröße | \mathbb{N} | 15 |
| κ | Lebensdauer eines Individuums | \mathbb{N} | 1 |
| ν | Verhältnis von Nachkommen zur Population ($\nu = \frac{\lambda}{\mu} \Leftrightarrow \lambda = \nu \cdot \mu$) | \mathbb{R} | 7 |
| $\sigma_i^{(0)}$ | Initial-Schrittweiten | \mathbb{R}_+ | 3 |
| ρ | Rekombinationsparameter | $\{1, \dots, \mu\}$ | 2 |
| c_τ | Konstante für den Mutationsoperator | \mathbb{R}_+ | 1 |
| r_x | Rekombinationsoperator für die Objektkomponente | $\{i, d\}$ | d (diskret) |
| r_σ | Rekombinationsoperator für die Strategiekomponente | $\{i, d\}$ | i (intermediär) |
| n_σ | Anzahl der Schrittweiten | $\{1, n\}$ | n |
| term | Abbruchbedingung | — | — |

Tabelle 3.2: Steuerparameter einer ES

Anwendung finden kann. Goldberg (1989), ein Schüler Hollands schreibt dazu: „*Thus, we are drawn to an interesting conclusion: where robust performance is desired (and where is it not?), nature does it better; the secrets of adaptation and survival are best learned from the careful study of biological example*“.

Genetische Algorithmen nehmen sich die biologische Evolution somit wesentlich stärker zum Vorbild, als es die Evolutionsstrategien tun. Sie unterscheiden zwischen einem Geno- und einem Phänotyp, also zwischen einer internen Codierung und deren Interpretation. Diese Unterscheidung ist aus dem genetischen Code motiviert, der über die DNA die Bestandteile des Lebens (Proteine) codiert und auf dem alleine die genetischen Mechanismen ansetzen. Klassische GA verwenden daher zur Repräsentation einen Binärcode, den eine Dekodierungsfunktion in Elemente aus dem Suchraum transformiert. Ein weiterer Unterschied zwischen den ES und den GA ist die Selektion, die nicht deterministisch (im Fall der Umweltselektion) bzw. nicht gleichverteilt (im Fall der Paarungselektion) erfolgt, sondern sich aus einer Wahrscheinlichkeitsverteilung ergibt, die auf der Fitness der einzelnen Individuen basiert. Zudem wird völlig auf die Umweltselektion verzichtet.

Ähnlich zur ES, die sich im Laufe der Zeit von einer sehr einfachen (1+1)-ES zu einem komplexeren Optimierverfahren entwickelt hat, haben sich auch die GA weiterentwickelt und nähern sich zunehmend den ES an. Heute werden auch reellwertige GA (rcGA) verwendet, die die folgenden Vorteile aufweisen:

- Binär codierte GA (bcGA) benötigen eine Funktion, um den Binärcode in Elemente aus dem Suchraum zu transformieren. Für reellwertige Optimierungsprobleme ergibt sich somit beim Einsatz der rcGA ein Beschleunigungseffekt aufgrund des Entfallens dieser Umrechnung.
- Der Suchraum muss vor Anwendung eines bcGA bekannt sein, um eine geeignete Codierung und die entsprechende Dekodierungsfunktion zu definieren; rcGA erfordern vom Anwender weniger Wissen über die Problemstellung.
- Die binäre Codierung weist ein weiteres Problem auf: Hamming Cliffs. Auch bei geringen Veränderungen eines binär codierten Individuums wird durch invertieren des signifikanten

Bits ein größerer Sprung im Suchraum verursacht (obwohl er ggf. nicht gewünscht ist, z.B. zum Ende der Optimierung).

- Die reellwertige Codierung erlaubt eine geringfügige Verschiebung der Lösungspunkte im Suchraum, so dass Schritt für Schritt eine Annäherung zum Optimum erfolgt.

Im Folgenden werden rcGA analog zur Evolutionsstrategien beginnend mit der Repräsentation und dem allgemeinen Ablauf eingeführt. Anschließend werden die Operatoren des rcGA vorgestellt und ein Exkurs in die bcGA unternommen. Für rcGA können Initialisierung und Abbruchkriterium analog zur ES realisiert werden, so dass sie hier nicht mehr behandelt werden. Die folgende Beschreibung des rcGA orientiert sich dabei an Herrera et al. (1998).

Repräsentation

Im Gegensatz zur ES, die in einem Individuum eine Menge an Informationen ablegt, stellt ein Individuum eines GA nur einen Punkt aus dem Suchraum dar. Im Fall der hier betrachteten n -dimensionalen Funktionen handelt es sich bei Individuen eines rcGA um einen Vektor bestehend aus n reellwertigen Variablen, je eine pro Koordinate des Suchraums. Das Individuum ist somit wie folgt gegeben:

$$I_i^{(t)} = \mathbf{x}_i = (x_{i,1}^{(t)}, \dots, x_{i,n}^{(t)}) .$$

Weitere Komponenten (wie beispielsweise eine Strategiekomponente) kennen die Individuen eines GA nicht. Daher müssen vom Anwender die Schrittweiten vor der Optimierung genau spezifiziert werden. Dennoch kennen auch rcGA eine Selbstadaptation, die über einen speziellen Rekombinationsoperator realisiert wird.

Allgemeiner Ablauf

Der Ablauf der GA orientiert sich ebenfalls an der Abbildung 3.3, jedoch nicht exakt. Zu Beginn eines GA muss die Population der Größe μ initialisiert werden. Anschließend erfolgt die Paarungsselektion, die im Gegensatz zur ES fitnessbasiert oder fitnessproportional erfolgt. Nach der Paarungsselektion können die genetischen Operatoren Rekombination und Mutation angewendet werden, die die Nachkommen erzeugen. GA erzeugen dabei genau die benötigten Individuen, so dass eine Umweltselektion nicht mehr erforderlich ist. Die Nachkommen können direkt der Folgepopulation zugeordnet werden. Anschließend kann mit Prüfung der Abbruch-

Algorithmus 3.3 : Genetischer Algorithmus

```

1  $t := 0;$ 
2 Initialisiere  $P^{(t)} := (I_1^{(t)}, \dots, I_\mu^{(t)});$ 
3 Bewerte  $P^{(t)} : (\text{scal}(f(I_1^{(t)})), \dots, \text{scal}(f(I_n^{(t)})));$ 
4 while ( $\text{term}(P^{(t)}) \neq \text{true}$ ) do
5    $P := \text{sel}(P^{(t)}, 2\mu);$ 
6    $P' := \text{rek}(P, p_{rek});$ 
7    $P'' := \text{mut}(P', p_{mut});$ 
8   Bewerte  $P'' : (\text{scal}(f(I_1''^{(t)})), \dots, \text{scal}(f(I_\mu''^{(t)})));$ 
9    $P^{(t+1)} := P'';$ 
10   $t := t + 1;$ 

```

bedingung fortgesetzt werden. Der Algorithmus 3 stellt den Ablauf ausführlich dar.

In den Zeilen 1 – 3 wird der Algorithmus initialisiert, in dem der Generationenzähler auf 0 gesetzt und die Startpopulation initialisiert und bewertet wird. Danach erfolgt in den Zeilen 4 – 9 zyklisch, so lange die Abbruchbedingung nicht greift, die Erzeugung einer neuen Population, die in der nächsten Generation die neue Elternpopulation bildet. Dazu wird eine Hilfspopulation P der Größe 2μ durch Selektion gebildet auf der die Rekombination angewendet wird und eine weitere Hilfspopulation P' der Größe μ erzeugt³, auf der die Mutation durchgeführt werden kann, die schließlich P'' erzeugt, die exakt μ Individuen enthält. Diese Individuen stellen dann die Population für die nächste Generation dar.

Bewertung

Die Bewertung eines Individuums wird aufgrund der hier betrachteten Funktionen $f : \mathbb{R}^n \rightarrow \mathbb{R}$ durch $f(I_i^{(t)})$ gegeben. Da das Ziel der Optimierung hier das Auffinden des globalen Minimums ist, ist ein Individuum umso fitter, je geringer es bewertet wird. Diese Form der Bewertung führt im Zusammenhang mit der fitnessproportionalen Selektion zu zwei Problemen:

- Individuen mit höherer Fitness haben eine größere Wahrscheinlichkeit selektiert zu werden – somit ergibt sich der für Minimierungsprobleme zutiefst unerwünschte Effekt, dass maximiert und nicht minimiert wird.
- Die hier betrachteten Funktionen liefern sowohl positive als auch negative Funktionswerte. Die fitnessproportionale Selektion kann aber nur auf positiver Fitness arbeiten.

Aus diesen Gründen müssen die Funktionswerte weiter bearbeitet werden, um den Individuen nur echt positive Fitnesswerte zu zuweisen. Diese Aufgabe wird von der Skalierung übernommen.

Skalierung

Soeben wurde die Notwendigkeit einer Skalierung motiviert. Allgemein handelt es sich bei der Skalierung um eine Funktion $scal : \mathbb{R} \rightarrow \mathbb{R}_+ \cup \{0\}$, die die Funktionswerte auf positive Fitnesswerte abbildet und fitteren Individuen (in unserem Fall also solchen, die kleinere Funktionswerte haben) die höhere Fitness zuweist. Ein Problem der Skalierung ist das Verfälschen der tatsächlichen Funktionswerte. Dies sollte mit ein Grund dafür sein, dass eine große Anzahl an Skalierungsfunktionen vorhanden ist, da es wohl nicht die optimale Funktion gibt. Bäck (1994) stellt eine Auswahl solcher Verfahren zusammen. Hier sollen nur zwei Skalierungsfunktionen vorgestellt werden:

Das *linear static scaling* benötigt zwei exogene Parameter c_0 und c_1 und wird wie folgt realisiert:

$$scal(f(I_i), \{c_0, c_1\}) = c_0 \cdot f(I_i) + c_1 .$$

Über den Parameter $c_0 \in \mathbb{R} \setminus \{0\}$ ist es möglich zwischen Maximierungs- und Minimierungsproblemen zu unterscheiden. Negative c_0 müssen gewählt werden, wenn minimiert, positive, wenn maximiert wird. Dennoch entstehen so weiterhin negative Fitnesswerte. Über den weiteren Parameter $c_1 \in \mathbb{R}$ werden alle Fitnesswerte nach oben geschoben, mit dem Ziel, dass der kleinste Fitnesswert positiv wird.

Das *linear dynamic scaling* kommt ohne den Parameter c_1 aus und ersetzt ihn dynamisch durch den kleinsten Funktionswert der aktuellen Population. Es wird wie folgt realisiert:

$$scal\left(f(I_i), \{c_0, P^{(t)}\}\right) = c_0 \cdot f(I_i) - \min\left\{f(I_j) \mid I_j \in P^{(t)}\right\} .$$

³Die Rekombination eines GA arbeitet auf genau zwei Individuen und erzeugt ein neues Individuum.

Weitere Ansätze verwenden komplexere Funktionen, die z.B. auf einen Exponenten zurückgreifen. Wird der Exponent zu Beginn kleiner 1 gewählt, hat die Fitness der einzelnen Individuen geringen Einfluss und der Suchraum kann weiträumig durchsucht werden. Zum Ende der Optimierung können Exponenten größer 1 den Selektionsdruck stark erhöhen, so dass nur noch lokal die Spitze des Minimums gesucht werden kann. In Verbindung mit der Rank-Selektion existiert eine weitere Methode der Skalierung, die auf eine Umrechnung der Zielfunktionswerte der Individuen verzichtet. Dieses Verfahren wird im folgenden Abschnitt über die Selektion beschrieben.

Selektion

Die Selektion selektiert ausschließlich 2μ Individuen aus der Elternpopulation in eine Hilfspopulation P , auf der die Rekombination angewendet wird, und wählt dabei fittere Individuen mit einer höheren Wahrscheinlichkeit aus. Dazu weist die Selektion jedem Individuum I_i eine Wahrscheinlichkeit $p_s(I_i)$ zu, die angibt, mit welcher Wahrscheinlichkeit das i -te Individuum selektiert wird. Eine typische Wahrscheinlichkeitsverteilung ergibt sich aus

$$p_s(I_i) = \frac{\text{scal}(f(I_i))}{\sum_{j=1}^{\mu} \text{scal}(f(I_j))} \quad \text{für alle } I_i \in P^{(t)}$$

auf der dann der Selektionsalgorithmus angewendet werden kann. Die zwei bekanntesten Algorithmen sind die Roulettrad Selektion und das stochastische universelle Sampling, die in Algorithmus 3.4 bzw. Algorithmus 3.5 dargestellt werden. Die Roulettrad Selektion gibt je-

Algorithmus 3.4 : Roulettrad Selektion

Eingabe : Zufällige Permutation der $p_s(I_i)$

Ausgabe : Selektiertes Individuum I_i

```
1 sum := 0;
2 z :=  $U[0, 1]$ ;
3 i := 1;
4 while (sum < z) do
5   | sum = sum +  $p_s(I_i)$ ;
6   | i = i + 1;
7 Return  $I_i$ ;
```

dem Individuum entsprechend seiner Selektionswahrscheinlichkeit Platz auf einem Roulettrad (bildlich gesehen), durch dessen Drehen je ein Individuum selektiert wird. Der Algorithmus 3.4 realisiert diese Idee, in dem es in Zeile 2 einen Zeiger z zufällig platziert und dann in Zeilen 4 – 6 das Roulettrad aufbaut. Sobald der Zeiger erreicht wird, kann das aktuelle Individuum zurückgegeben werden.

Dieser Ansatz hat den Nachteil, dass sehr fitte Individuen überdurchschnittlich oft selektiert werden. Der Algorithmus muss pro zu selektierendem Individuum neu aufgerufen werden und setzt dabei den Zeiger jedes mal neu. Die Wirkung ist, dass das fitteste Individuum in jedem Aufruf der Roulettrad Selektion die größte Wahrscheinlichkeit hat selektiert zu werden. Dieses Problem wird auch als Dominanzproblem bezeichnet. Die Lösung kann das stochastische universelle Sampling liefern. Hier werden alle benötigten Individuen durch einen Aufruf selektiert. Anstelle eines Zeigers verwendet das stochastische universelle Sampling bei μ zu selektierenden Individuen μ Zeiger, die in einem äquidistanten Abstand zueinander angeordnet sind. Die Position des ersten Zeiger wird im Algorithmus 3.5 in Zeile 2 zufällig bestimmt. Über eine Schleife werden in Zeile 10 dann die restlichen Zeiger äquidistant verteilt. Ansonsten verhält sich der

Algorithmus 3.5 : Stochastisches universelles sampling

Eingabe : Zufällige Permutation der $p_s(I_i)$ und Anzahl der Individuen μ , die selektiert werden sollen

Ausgabe : Menge der selektierten Individuen $L = \{I_{s_1}, \dots, I_{s_\mu}\}$

```

1  $sum := 0;$ 
2  $z := U[0, \frac{1}{\mu}];$ 
3  $i := 1;$ 
4  $L := \emptyset;$ 
5 for ( $i = 1, \dots, \mu$ ) do
6   while ( $sum < z$ ) do
7      $sum = sum + p_s(I_i);$ 
8      $i = i + 1;$ 
9    $L := L \cup I_i;$ 
10   $z := z + \frac{1}{\mu};$ 
11 Return  $L;$ 

```

Algorithmus wie die bereits beschriebene Roulettrad Selektion.

Die Unterschiede beider Verfahren werden in Abbildung 3.10 dargestellt. Unter der Annahme, dass 4 Individuen zu selektieren sind, hat bei Anwendung der Roulettrad Selektion (links) das hier „überfite“ Individuum 5 in allen 4 Aufrufen die höchste Wahrscheinlichkeit ausgewählt zu werden. Das universelle stochastische Sampling verteilt die Zeiger gleichmäßig (aber zufällig) über das Rad, so dass weniger gute Individuen eine höhere Wahrscheinlichkeit haben, ausgewählt zu werden. Eine andere Lösung des Dominanzproblems ist die Verwendung ei-

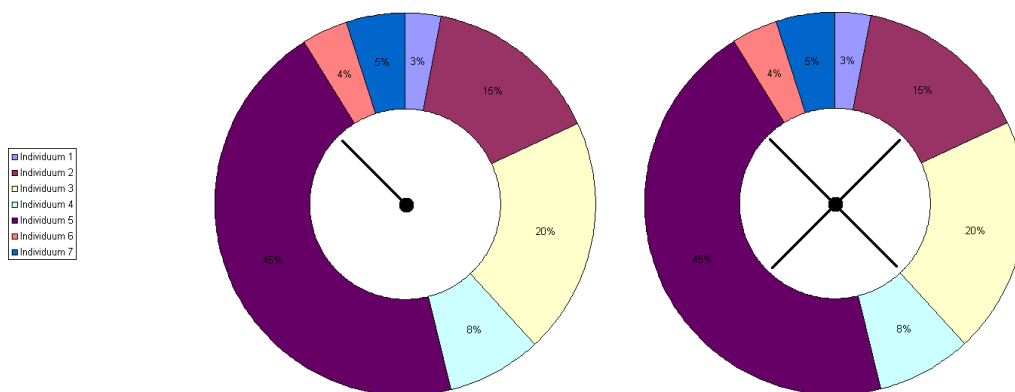


Abbildung 3.10: Roulettrad Selektion (links), uniform stochastische Selektion (rechts)

ner anderen Skalierung. Die Rangselektion erfordert keine Skalierungsfunktion. Sie sortiert die Individuen entsprechend ihrer Fitness und weist dem fittesten Individuum die höchste Auswahlwahrscheinlichkeit zu, die bis hin zum schlechtesten Individuum stetig abnimmt. Auf der so entstandenen Wahrscheinlichkeitsverteilung kann eine fitnessproportionale Selektion durchgeführt werden. So kann der Effekt der „überfitten“ Individuen beseitigt werden, der dann eintritt, wenn ein Individuum eine besonders hohe Auswahlwahrscheinlichkeit erhält und die übrigen Individuen die restliche (geringe) Wahrscheinlichkeit unter sich aufteilen müssen.

Eine ebenfalls weit verbreitete Variante der Selektion ist die Turnierselktion. Sie verzichtet komplett auf die Skalierung und Berechnung einer Wahrscheinlichkeitsverteilung und wählt stattdessen gleichverteilt k Individuen aus der Elternpopulation aus, von denen das beste (de-

terministisch) selektiert wird. Diese Variante kann somit über einen zusätzlichen exogenen Parameter k eingestellt werden und muss zu Selektion mehrere Individuen mehrfach durchgeführt werden, da pro Turnier nur ein Individuum selektiert wird.

Rekombination

Für rcGA existiert eine Vielzahl diverser Rekombinationsoperatoren, so dass hier nur eine Auswahl vorgestellt werden kann. Alle Operatoren haben gemeinsam, dass sie ausgehend von zwei Individuen ein Neues erzeugen. Beide Individuen werden dabei (ohne Zurücklegen) gleichverteilt aus der durch (Paarungs)selektion entstandenen Hilfspopulation P entnommen und mit einer Wahrscheinlichkeit p_{rek} rekombiniert, wodurch ein neues Individuum entsteht. Ansonsten wird (mit Wahrscheinlichkeit $1 - p_{rek}$) gleichverteilt einer der beiden Individuen unverändert zurückgegeben. In jedem Fall werden durch Rekombination μ Individuen erzeugt (dies setzt voraus, dass P aus 2μ Individuen bestand) und in einer weiteren Hilfspopulation P' vereinigt.

Die intermediäre Rekombination verhält sich ähnlich zur intermediären Rekombination der ES. Sie erzeugt jedoch nicht einen Nachkommen, der Schwerpunkt seiner Eltern ist, sondern gewichtet beide Individuen zufällig. Sind $I_1 = (x_{1,1}, \dots, x_{1,n})$ und $I_2 = (x_{2,1}, \dots, x_{2,n})$ zwei Individuen, die rekombiniert werden sollen und α eine gleichverteilte Zufallszahl aus $[0, 1]$, dann erzeugt die intermediäre Rekombination einen Nachkommen I'_1 wie folgt:

$$x'_{1,i} = x_{1,i} + \alpha \cdot (x_{2,i} - x_{1,i}) .$$

Betrachtet man den minimalen Hyperquader um beide Individuen (vergleiche auch Abbildung 3.6), so können aufgrund der Gewichtung aus dem Intervall $[0, 1]$ alle Punkte im minimalen, durch I_1 und I_2 definierten Hyperquader, erzeugt werden. Der Operator kann noch so erweitert werden, dass er eine weitere Suche auf dem Suchraum ausführt, in dem nicht nur im Hyperquader rekombiniert wird, sondern auch außerhalb. Dazu muss α als gleichverteilte Zufallszahl aus dem Intervall $[-0.25, 1.25]$ gewählt werden (Mühlenbein & Schlierkamp-Voosen, 1993).

Die diskrete Rekombination für rcGA gleicht der Variante für ES mit $\rho = 2$. Sind zwei Individuen I_1 und I_2 wie oben gegeben, so entsteht der Nachkomme

$$I'_1 = (x'_{1,1}, \dots, x'_{1,n}) \quad \text{mit}$$

$$x'_{1,i} = \begin{cases} x_{1,i} & \text{mit Wahrscheinlichkeit } \frac{1}{2} \\ x_{2,i} & \text{mit Wahrscheinlichkeit } \frac{1}{2} \end{cases} .$$

Die lineare Rekombination erzeugt entlang der Verbindungslinie beider Individuen I_1 und I_2 drei Punkte und übernimmt den Besten als Nachkommen I'_1 . Die drei Punkte ermitteln sich wie folgt:

$$I_1^1 = \frac{1}{2} \cdot I_1 + \frac{1}{2} \cdot I_2 ,$$

$$I_1^2 = \frac{3}{2} \cdot I_1 - \frac{1}{2} \cdot I_2 ,$$

$$I_1^3 = -\frac{1}{2} \cdot I_1 + \frac{3}{2} \cdot I_2 \quad \text{und}$$

liegen im ersten Fall in der Mitte und in den anderen beiden Fällen außerhalb des minimalen Hyperquaders, der durch I_1 und I_2 definiert wird. Diese Variante erfordert einen zusätzlichen Selektionsmechanismus, der deterministisch das beste Individuum auswählt.

Weitere Rekombinationsoperatoren können Herrera et al. (1998) entnommen werden. Erwähnenswert ist der Simulated Binary Crossover (SBX) Operator (Deb & Beyer, 1999) und (Beyer & Deb, 2001), der in rcGA ein selbstadaptativen Mechanismus etabliert. Ist eine hohe Diversität in der Population vorhanden, so werden sich i.d.R. die zwei zu rekombinierenden Individuen stark unterscheiden. In einem solchen Fall können Nachkommen erzeugt werden, die im Suchraum weit entfernt von ihren Eltern platziert werden. Ist hingegen die Diversität der Population gering, so erzeugt der SBX Operator Nachkommen, die im Suchraum nah an ihren Eltern liegen. Im ersten Fall ist anzunehmen, dass das Minimum noch weit entfernt ist und somit größere Schrittweiten erforderlich sind; im zweiten Fall ist anzunehmen, dass die Individuen der aktuellen Population nah am Minimum liegen, und durch kleine „Schrittweiten“ eine Konvergenz zum Minimum sichergestellt wird.

Mutation

Die Mutation wird zusätzlich auf den durch Rekombination entstandenen Individuen angewendet und erzeugt eine weitere Hilfspopulation P'' . Wie bereits die Rekombination wird der Mutationsoperator nicht zwingend auf den Individuen in P' angewendet, sondern mit einer Wahrscheinlichkeit p_{mut} ; ansonsten werden die Individuen unverändert in P'' aufgenommen. Ähnlich wie es eine Menge an diversen Skalierungs-, Rekombinations- und Selektionsoperatoren für rcGA gibt, sind auch diverse Realisierungen eines Mutationsoperators für rcGA vorhanden. Eine Zusammenstellung kann wiederum Herrera et al. (1998) entnommen werden. Michalewicz (1992) schlägt eine zufällige Mutation (random mutation) vor. Diese Variante wird auf allen in P' enthaltenen Individuen angewendet und weist einzelne Koordinaten eines Individuums mit Wahrscheinlichkeit p_{mut} einen neuen Wert aus dem Definitionsbereich der betreffenden Koordinate zu. Weicker (2002) stellt eine erweiterte Form dieser Mutationsvariante vor, die ein Individuum in einer Generation nur mit einer maximalen Schrittweite verschieben kann, um sicher zustellen, dass die Mutation in einer Nachbarschaft um ein Individuum sucht.

Beide Autoren führen aber an, dass die Mutation auch durch Addition einer Zufallszahl auf jede Koordinate des Individuum erfolgen kann, so dass man zu der Normalverteilung (vgl. auch Evolutionsstrategien) gelangt. Da rcGA keine Selbstadaptation der Schrittweiten kennen, können hier nur zeitabhängig abnehmende Schrittweiten verwendet werden (Michalewicz, 1992). Eine Realisierung der normalverteilten Mutation arbeitet auf zwei Parametern σ und s . Der Parameter σ gibt die Anfangsschrittweite für die Mutation in Form einer Standardabweichung an. Der zweite Parameter s gibt an, wie stark die Schrittweiten pro Generation abnehmen. Ist in Generation t die Schrittweite σ_t gegeben, kann die Schrittweite für die Folgegeneration ermittelt werden als

$$\sigma_{t+1} = \left(1 - s \cdot \frac{t+1}{T}\right) \cdot \sigma_t,$$

mit der Gesamtanzahl der Generationen T . Je nach Einstellung des Parameters s nehmen die Schrittweiten ab ($s > 0$), zu ($s < 0$) oder bleiben konstant ($s = 0$). Die praktisch relevante Einstellung für s ist aus dem Intervall $[0, 1]$, so dass die Schrittweiten konstant bleiben, oder mit zunehmenden Optimierungsfortschritt abnehmen und so zum Ende der Optimierung eine Feinabstimmung der Parameter ermöglichen. Die Mutation in der t -ten Generation selbst kann dann durch

$$I_1'' = I_1' + \mathbf{z} \quad \text{mit} \quad \mathbf{z} := \sigma_t \cdot \begin{pmatrix} N_1(0, 1) \\ \vdots \\ N_n(0, 1) \end{pmatrix}^T.$$

realisiert werden. Der Operator kann noch so erweitert werden, dass für jede Koordinate verschiedene Schrittweiten verwendet werden. Dann wird an Stelle eines Skalars s ein n -dimensionaler

Vektor s verwendet.

Behandlung von Nebenbedingungen

Die Behandlung von Nebenbedingungen erfolgt über die Zielfunktion, in dem die Fitness der Individuen, die eine Nebenbedingung nicht einhalten, mit einem Strafterm belegt wird (Michalewicz, 1992), (Gerdes et al., 2004). Dieser Strafterm kann beispielsweise über die Lagrange-funktion (vgl. Kapitel 2) mit nicht-negativen Lagrangemultiplikatoren realisiert werden. Dieses Verfahren findet nicht nur in GA Anwendung, sondern ist auch in ES üblich.

Zusammenfassung der reellwertigen genetischen Algorithmen

Reellwertig codierte genetische Algorithmen sind aus den weit verbreiteten bcGA hervorgegangen. Sie eignen sich vor allem für reellwertige Optimierungsprobleme und hier besser als bcGA, da sie weniger Vorkenntnisse vom Anwender über die Problemstellung erwarten und durch den Wegfall der Dekodierungsfunktion auch weniger Rechenzeit erfordern. So sehr rcGA den ES auch ähneln, sie unterscheiden sich an verschiedenen Stellen erheblich. Sie verfügen nicht über klar definierte genetische Operatoren, sondern stellen dem Anwender eine Vielzahl solcher Operatoren zu Verfügung. Neben der Zielfunktion f benötigen rcGA eine Skalierungsfunktion, die die Individuen auf eine positive Fitness abbildet, da nur so die fitnessproportionale Selektion angewendet werden kann. Die Paarungsselektion erfolgt fitnessproportional und die genetischen Operatoren nicht zwingend sondern basierend auf einer Rekombinations-, bzw. Mutationswahrscheinlichkeit. Während ES zwischen Populationsgröße und Anzahl der Nachkommen unterscheiden, sind in Fall der GA beide Größen stets identisch. Der Selektionsdruck ergibt sich dadurch, dass GA Individuen auch mehrfach selektieren können und somit besonders fitte Individuen in einer Generation auch mehrfach in die Folgepopulation aufgenommen werden können, während schlechtere Individuen mit hoher Wahrscheinlichkeit gar nicht selektiert werden. Ein rcGA kann allgemein durch das Tupel

$$rcGA = (P^{(0)}, \mu, rek, p_{rek}, mut, p_{mut}, \sigma_i, s, scal, sel, term)$$

definiert werden. Typische Werte und deren Bedeutung wird in der folgenden Tabelle 3.3 dargestellt.

Exkurs in binär codierte genetische Algorithmen

Binäre genetische Algorithmen sind rcGA sehr ähnlich und unterscheiden sich neben der anderen Repräsentation nur in den genetischen Operatoren Rekombination und Mutation. Der in Algorithmus 3 dargestellte Ablauf des GA kann beinahe unverändert übernommen werden; er muss nur so erweitert werden, dass in den Zeilen 3 und 10 vor Berechnung der Fitness die binär codierten Individuen in eine reellwertige Repräsentation transformiert werden, damit die Funktion f die Individuen auch abbilden kann.

Die bcGA codieren die Problemstellung in einem Individuum über einen Bitstring fester Länge, also über $I = \mathbb{B}^{n-l}$. Jeder Koordinate der Problemstellung stehen somit l Bits zu Verfügung. Für die hier betrachteten reellwertigen Optimierungsprobleme eignet sich diese Codierung noch nicht und muss interpretiert werden. Angenommen das Optimierungsproblem ist gegeben als

$$f : \prod_{i=1}^n [u_i, v_i] \rightarrow \mathbb{R} ,$$

dann ergeben sich die zwei bekanntesten Dekodierungsfunktionen aus dem Standard-Binär-code bzw. dem Gray-Code. Die Dekodierungsfunktion der i -ten Koordinate $\Gamma_i : \mathbb{B}^l \rightarrow [u, v]$ wird im

| SYMBOL | BEDEUTUNG | WERTEBEREICH | STANDARDWERT |
|------------|--|------------------------------|------------------------------------|
| $P^{(0)}$ | Startpopulation | $X^n \subseteq \mathbb{R}^n$ | — |
| μ | Populationsgröße | \mathbb{N} | 20 |
| rek | Rekombinationsoperator | — | diskrete Rekombination |
| p_{rek} | Rekombinationswahrscheinlichkeit | $[0, 1]$ | 0.8 |
| mut | Mutationsoperator | — | normalverteilte Mutation |
| p_{mut} | Mutationswahrscheinlichkeit | $[0, 1]$ | 0.2 |
| σ_i | Initialschrittweiten | \mathbb{R}_+ | 1 |
| s | Parameter zu Steuerung der Schrittweiten | $[0, 1]$ | 1 |
| scal | Skalierung | — | rang-basiert |
| sel | Selektion | — | stochastisch universelle Selektion |
| term | Abbruchkriterium | — | — |

Tabelle 3.3: Steuerparameter eines rcGA

Fall des Standard-Binärcode wie folgt realisiert:

$$\Gamma_i(b_{i+1}, \dots, b_{i+n}) = u_i + \frac{v_i - u_i}{2^l - 1} \cdot \left(\sum_{j=0}^l 2^j \cdot b_{i+j} \right)$$

Durch die binäre Repräsentation wird eine Art Gitter über den Suchraum gelegt, an dessen Gitterpunkten ausschließlich Lösungen erzeugt werden. So kann es einem bcGA im Allgemeinen nicht gelingen das globale Minimum exakt zu treffen. Tatsächlich verwenden aber alle Digitalrechner intern ein Binärcode und haben somit auch eine begrenzte Genauigkeit, die somit dann auch die rcGa, sowie die ES trifft. Ein wirklicher Nachteil des Standard-Binärcodes ist sein Verhalten bei geringen Veränderungen (Mutationen), bei denen je nach Position des mutierten Bits größere oder geringere Änderungen in der Interpretation eintreten. Zur Lösung wird der Gray-Code angeführt, bei dem jedoch ebenfalls geringe Veränderungen an der Repräsentation

| INTEGER | STANDARD-BINÄR | GRAY |
|---------|----------------|------|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |

Tabelle 3.4: Interpretation von Standardbinär- und Gray-Code

zu großen Veränderung an der Interpretation führen können, wie der Tabelle 3.4 zu entnehmen ist (z.B. bei Übergang von *0000* zu *0100*). Weitere Eigenschaften, sowie die Dekodierungsfunktion des Graycodes werden in (Bäck, 1994) und (Weicker, 2002) eingeführt.

Die genetischen (Standard-)Operatoren sind aufgrund des Binärcodes relativ einfach. Die Rekombination wählt gleichverteilt k Crossover-Punkte, an denen zwei Bitstrings getrennt werden, und vertauscht die so entstandenen Teilstrings. Für einen Crossover-Punkt j und zwei Individuen (a_1, \dots, a_n) sowie (b_1, \dots, b_n) entstehen somit die folgenden neuen Individuen:

$$(a_1, \dots, a_j, b_{j+1}, \dots, b_n)$$

und

$$(b_1, \dots, b_j, a_{j+1}, \dots, a_n),$$

von denen gleichverteilt einer der beiden verworfen wird. Die Rekombination wird nicht zwingend durchgeführt. Sie tritt zufällig, mit einer Wahrscheinlichkeit p_{rek} ein (ansonsten wird einer der beiden Individuen unverändert übernommen). Die anschließende Mutation durchläuft den durch Rekombination hervorgegangenen Bitstring und invertiert mit einer Wahrscheinlichkeit p_{mut} die einzelnen Bits. Die Empfehlung ist, die Rekombinationswahrscheinlichkeit eher hoch und die der Mutation eher gering zu wählen.

Ansonsten kann der Algorithmus 3 unverändert übernommen werden. Eine genauere Einführung in bcGA gibt (Bäck, 1994), (Gerdes et al., 2004) und (Weicker, 2002). Zum Schluss dieses Exkurses sollen noch die Vorteile eines bcGA erwähnt werden. Binär codierte Genetische Algorithmen zeichnen sich vor allem durch ihre einfachen genetischen Operatoren aus, die zudem von Digitalrechnern besonders gut unterstützt werden. Ein weiterer Vorteil ist, dass keine Lösungen außerhalb des Suchraums erzeugt werden können, da zuvor, über die Codierung, der Suchraum genau auf die Bitstrings gemappt wurde. Die Initialisierung eines bcGA gestaltet sich besonders einfach. Bei einer Repräsentation mit Bitlänge $m = n \cdot l$ kann durch m unabhängige Münzwürfe ein Individuum initialisiert werden.

3.4 Zusammenfassung

1. Für Optimierungsprobleme allgemein aber selbst für reellwertige Optimierungsprobleme speziell gibt es nicht das universelle Optimierungsverfahren.
2. Deterministische Hillclimbing Verfahren finden in der Regel nur ein lokales Minimum, wenn die Zielfunktion nicht konvex ist. Sie arbeiten (mit Ausnahme des Nelder-Mead-Simplex Verfahrens) auf einem Punkt, den sie anhand deterministischer Rechenvorschriften zum Minimum bewegen. Das wird i.d.R. das (lokale) Minimum sein, zu dem das Gefälle ausgehend vom Startpunkt führt.
3. Die hinter Hillclimbing Verfahren stehende deterministische Vorschrift wird je nach Funktion (separierbar, multimodal, differenzierbar, ...) besonders gute oder aber weniger gute Ergebnisse liefern (was mit dafür ein Grund sein kann, dass es eine so große Anzahl solcher Verfahren gibt).
4. Eine andere Lösung bieten stochastische Suchverfahren. Sie ersetzen die deterministische Rechenvorschrift durch eine stochastische.
5. Zu den stochastischen Suchverfahren gehören (unter anderem) die Evolutionären Algorithmen und die einfache stochastische Suche.
6. Die einfache stochastische Suche arbeitet auf einem Punkt, den sie ähnlich zu dem Hillclimbing Verfahren stetig zu einem Minimum bewegt. Ein zusätzlicher Mechanismus, der es diesem Verfahren ermöglicht auch schlechtere Lösungspunkte zu erzeugen, soll zu einer Konvergenz zum globalen Minimum beitragen.

7. Evolutionäre Algorithmen arbeiten auf einer i.d.R. größeren Menge an Lösungspunkten, die über genetische Operatoren durch den Suchraum verschoben werden. Dennoch sind EA kein random-walk. Sie sind universell einsetzbare Optimierverfahren (und nicht ausschließlich auf die Optimierung von Funktionen begrenzt sind), deren hohe Leistungsfähigkeit in vielen wissenschaftlichen Arbeiten nachgewiesen wurde.
8. Alle hier vorgestellten Optimierverfahren (mit Ausnahme der Suche entlang der Koordinatenachsen) können über diverse exogene Parameter eingestellt werden, die wesentlichen Einfluss auf die Optimierung haben. So gesehen ist vor der Optimierung der Problemstellung zusätzlich eine Optimierung (Tuning) des Optimierungsalgorithmus notwendig oder aber zumindest empfehlenswert.

4 Matlab

Für die experimentelle Untersuchung evolutionärer Algorithmen ist eine geeignete Softwareumgebung erforderlich, die es ermöglicht sowohl genetische Algorithmen als auch Evolutionsstrategien, sowie eine einfache stochastische Suche mit unterschiedlichen Parametereinstellungen auszuführen. Eine solche Umgebung wird teilweise über die *Genetic Algorithm and Direct Search Toolbox* von Matlab zu Verfügung gestellt.

Matlab selbst ist ein Werkzeug, welches in den 70er Jahren für numerische Problemstellungen aus der linearen Algebra und Analysis entwickelt wurde. Der Name *Matlab* leitet sich dementsprechend aus *Matrix Laboratory* ab und erinnert noch an diese Zeit. Heute ist Matlab weitaus mächtiger; es hat sich zu einer plattformunabhängigen Programmiersprache entwickelt, die über alle gängigen Sprachkonstrukte verfügt. Zudem ist aufgrund des stark mathematisch orientierten Syntax eine besonders einfache Programmierung möglich.

Einfache Programme können in Matlab in so genannten *m-Files* abgelegt werden, die in ihrer Gesamtheit ein komplexeres Programm bilden, das auch als Toolbox bezeichnet wird. Durch Einbindung des Maple Kerns verfügt Matlab mittlerweile auch über die Möglichkeit symbolische Berechnungen durchzuführen und kann (unabhängig vom Maple Kern) auch komplexe Grafiken erstellen, um Daten zu visualisieren. Zudem basieren auf Matlab auch die Erweiterungen Simulink und Stateflow, die zu Simulation komplexer zeit- (Simulink) oder ereignisgesteuerter (Stateflow) Systeme verwendet werden können. Mit der kommerziell vertriebenen Optimierungs-Toolbox liefert Matlab somit ein extrem vielseitiges Werkzeug zur Modellierung, Simulation und Optimierung unterschiedlichster Systeme.

4.1 Genetic Algorithm and Direct Search Toolbox

Die Genetic Algorithm and Direct Search (GADS) Toolbox ist eine Sammlung von Funktionen, die die Optimization Toolbox um Genetische Algorithmen und direkte Suchverfahren (Patternsearch) erweitert und so eine Optimierung von Problemen ermöglicht, auf denen die bisherige Optimization Toolbox nicht anwendbar war (vgl. die Diskussion im Kapitel 3). In dieser Arbeit wird ausschließlich der in der GADS Toolbox implementierte GA betrachtet, der sich in einigen Punkten von dem in Kapitel 3 eingeführten Standard-GA unterscheidet. Wegen der Unterscheidbarkeit beider Algorithmen wird er hier mit GA* bezeichnet. Die zwei Hauptunterschiede betreffen die Reihenfolge der genetischen Operatoren und die Einführung eines zusätzlichen neuen Operators.

Der GA* setzt das Prinzip des Standard-GA, eine neue Elternpopulation in der Reihenfolge „Selektion – Rekombination – Mutation“ zu erzeugen, außer Kraft. Stattdessen wählt er zuvor eine bestimmte, über den Parameter *EliteCount* einstellbare Anzahl an Individuen aus, die unverändert in die nächste Population übernommen werden. Bei diesen Individuen handelt es sich um die besten Individuen der aktuellen Population. Erst dann werden die Individuen fitnessproportional selektiert und, im Gegensatz zum Standard-GA, entsprechend der Rekombinationswahrscheinlichkeit unter dem Rekombinationsoperator und dem Mutationsoperator aufgeteilt. Der GA* verwendet zur Erzeugung eines Nachkommens somit nur maximal einen genetischen Operator.

Für die experimentelle Untersuchung stellt die GADS Toolbox einen Teil der benötigten Softwareumgebung zu Verfügung. Da die Toolbox es dem Anwender sogar erlaubt eine Erweiterung

durch eigene *m*-Files durchzuführen, kann sie zudem als Ausgangspunkt für die ES und die RII dienen. Vor der Erweiterung soll zuvor die GADS Toolbox und ihre Arbeitsweise hier kurz vorgestellt werden. Die ausführliche Beschreibung kann (The MathWork, Inc., 2005a) entnommen werden.

4.1.1 Kurze Einführung in die GADS Toolbox

Der GA* kann ausschließlich zum Minimieren einer Funktion verwendet werden, was jedoch keine wesentliche Einschränkung ist (vgl. Kapitel 2). Er kann sowohl über die Matlab Kommandozeile, als auch über eine graphische Benutzeroberfläche gestartet werden. In beiden Fällen können eine Vielzahl an Einstellungen vorgenommen werden, die in der Tabelle 4.1 zusammengestellt sind. Die Parameter und deren Werte werden in einer Struktur (Datentyp in Matlab, in

| PARAMETER | BEDEUTUNG |
|-------------------|--|
| PopInitRange | Bereich, in dem die Variablen gleichverteilt initialisiert werden |
| PopulationSize | Populationsgröße μ |
| EliteCount | Anzahl der qualitativ besten Individuen, die unverändert in die nächste Generation übernommen werden |
| CrossoverFraction | Rekombinationswahrscheinlichkeit p_{rek} |
| Generations | Maximale Anzahl der Generationen (Abbruchkriterium) |
| TimeLimit | Maximale Zeit in Sekunden (Abbruchkriterium) |
| FitnessLimit | Nimmt die Fitness diesen oder einen geringeren Wert an, so breche ab |
| StallGenLimit | Generationsintervall, in dem eine Verbesserung des besten Individuums eintreten muss (Abbruchkriterium) |
| StallTimeLimit | Zeitintervall, in dem eine Verbesserung des besten Individuums eintreten muss (Abbruchkriterium) |
| TolFun | Toleranz für StallGenLimit und StallTimeLimit |
| InitialPopulation | Anfangspopulation |
| InitialScores | Bewertung der Anfangspopulation |
| FitnessScalingFcn | Skalierungsfunktion |
| SelectionFcn | Selektionsfunktion |
| CrossoverFcn | Rekombinationsfunktion |
| MutationFcn | Mutationsfunktion |
| PlotFcns | Funktionen, die während der Optimierung den Verlauf visualisieren |
| Vectorized | Matlab kann besonders effizient auf Matrizen arbeiten. Ermöglicht die Fitnessfunktion die Übergabe einer Matrix aller Individuen und somit die Berechnung der Fitness der gesamten Population in einem Schritt kann die Optimierung zeitlich signifikant beschleunigt werden. Dann sollte Vectorized auf on gestellt werden. |

Tabelle 4.1: Einstellungsmöglichkeiten für den GA*

dem verschiedene Variablen verwaltet werden) abgelegt, auf die der GA während der Optimierung zugreift. Diese Struktur (die im folgenden Optionsstruktur genannt wird) hat die folgende Gestalt

```
defaultopt = struct('PopulationType', 'doubleVector', ...
```

```

'PopInitRange', [0;1], ...
'PopulationSize', 20, ...
'EliteCount', 2, ...
'CrossoverFraction', 0.8, ...
'MigrationDirection', 'forward', ...
'MigrationFraction', 0.2, ...
'Generations', 100, ...
'TimeLimit', inf, ...
'FitnessLimit', -inf, ...
'StallGenLimit', 50, ...
'StallTimeLimit', 20, ...
'TolFun', 1.0000e-006, ...
'InitialPopulation', [], ...
'InitialScores', [], ...
'PlotIntervall', 1, ...
'FitnessScalingFcn', @fitscalingrank, ...
'SelectionFcn', @selectionstochunif, ...
'CrossoverFcn', @crossoverscattered, ...
'MutationFcn', @mutationgaussian, ...
'PlotFcns', [], ...
'Display', 'final', ...
'OutputFcns', [], ...
'CreationFcn', @gacreationuniform, ...
'HybridFcn', [], ...
'Vectorized', 'off');

```

und ist hier mit den Standardwerten aufgeführt. Sie kann über die graphische Benutzeroberfläche aber auch über eine Funktion `gaoptimset.m` verändert werden, letzteres, wenn der GA* über die Matlab Kommandozeile aufgerufen wird.

Über den Befehl `gatoool` kann die graphische Benutzeroberfläche gestartet werden, die selbst-erklärend ist und hier nicht weiter betrachtet wird. Über den Aufruf

```
[X, FVAL, REASON, POPULATION, SCORES] = ga(FITNESSFCT, NVAR, OPTIONS);
```

kann der GA* über die Kommandozeile gestartet werden, in dem ihm die zu minimierende Funktion, die Anzahl der Variablen sowie die Optionsstruktur übergeben wird. Er gibt dann den Lösungspunkt, den dazugehörigen Funktions- (oder auch Fitness-)wert sowie weitere Informationen aus. Weitere Erläuterungen zum Aufruf können der Dokumentation entnommen werden.

Der GA* kann über eine Erweiterung der Argumente auch auf restringierten Optimierungsproblemen angewendet werden. Über einen weiteren Aufruf

```
ga(FITNESSFCT, NVAR, Aineq, Bineq, Aeq, Beq, LB, UB, NonLin, OPTIONS);
```

werden während der Optimierung die linearen Nebenbedingungen $A_{ineq} \cdot x \leq B_{ineq}$, $A_{eq} \cdot x = B_{eq}$, sowie untere Schranken LB und obere Schranken UB beachtet. Zusätzlich können über `NonLin` weitere nichtlineare Nebenbedingungen angegeben werden.

Beispiel 4.1.1 Soll eine Funktion f , abhängig von drei Variablen maximiert werden, unter den Nebenbedingungen $2 \cdot x_1 - x_3 \leq 5$ und $x_1, x_2 \geq 0$, so kann das Problem wie folgt gelöst werden: `[x, FVAL] = ga(-f, 3, [2, 0, -1], [5], [], [], [0;0;-inf], [], [])`;

■

Neben der oben beschriebenen Ausgabe können auch komplexe Grafiken den Verlauf der Optimierung visualisieren. Die GADS Toolbox liefert bereits vordefinierte Funktionen, die grafische Ausgaben erzeugen, erlaubt dem Anwender aber auch die Definition eigener Funktionen. Eine ausführliche Beschreibung liefert erneut die Dokumentation zur GADS Toolbox.

4.1.2 Arbeitsweise des GA*

Der GA* setzt die im Kapitel 3 beschriebenen GA mit einigen Abweichungen um und gibt einem Anwender einen maximalen Umfang an Freiheit. Das Gerüst des GA* wird durch die Funktion `ga.m` bereitgestellt, das weitere Funktionen aufruft, die in ihrer Gesamtheit die Funktionalität eines GA ermöglichen. Eine Übersicht der Funktionen und deren Relation ist in Abbildung 4.1 dargestellt. Die Funktion `ga.m` ruft die Funktionen `makeState.m`, `validate.m` und `stepGA.m`

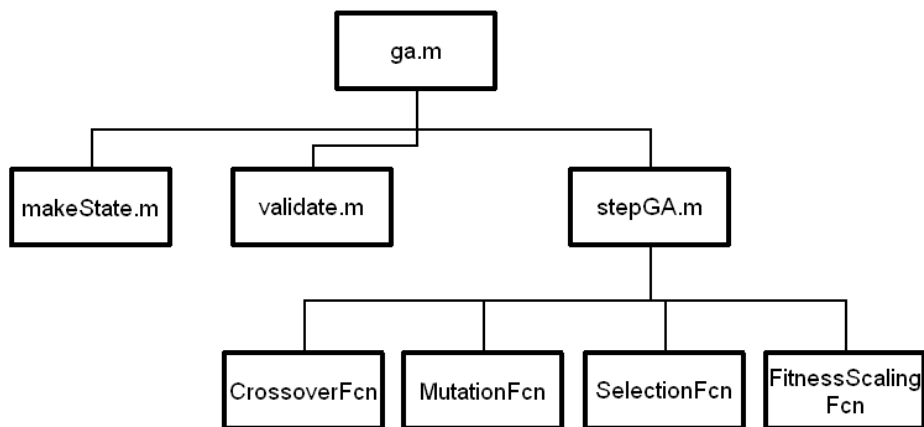


Abbildung 4.1: Beziehung der Funktionen des GA* zueinander

auf. Die erste Funktion erzeugt den Initialzustand. Dazu belegt sie Variablen, die während der Optimierung benötigt werden, mit Standardwerten. Dies sind die Variablen:

- Population: Die aktuelle Population.
- Score: Die Fitness der aktuellen Population.
- Generation: Die aktuelle Generation wird auf eins gesetzt.
- StartTime: Die Start-Zeit wird auf die aktuelle CPU-Zeit gesetzt.
- StopFlag: Die Abbruchbedingung ist zu Beginn leer.
- LastImprovement: Generation, in der die letzte Verbesserung eingetreten ist. Diese Variable ist notwendig, um nach `StallGenerations` (vgl. Tabelle 4.1) abbrechen zu können.
- LastImprovementTime: Letzter Zeitpunkt zu dem eine Verbesserung eingetreten ist. Diese Variable ist notwendig, um nach `StallTime` (vgl. Tabelle 4.1) abbrechen zu können.

Nach der Initialisierung durch `makeState.m` können alle Eingaben auf syntaktische aber auch auf semantische Korrektheit überprüft werden. Eine solche Funktionalität ist in der Funktion `validate.m` implementiert. Anschließend führt `ga.m` über eine Schleife eine weitere Funktion `stepGA.m` aus, bis eine Abbruchbedingung erfüllt ist. Diese Funktion führt genau einen Zyklus des GA durch, in dem ausgehend von einer aktuellen Population über genetische Operatoren

die Population für die Folgegeneration erzeugt wird. Dazu greift `stepGA.m` auf weitere Funktionen zu, die die Rekombination, Mutation, Skalierung und Selektion ausführen. Die Standardeinstellung, der in dieser Arbeit Anwendung findet, verwendet eine auf der Normalverteilung basierende Mutation (`mutationgaussian`), die diskrete Rekombination (`crossoverscattered`), die Rangskalierung (`fitscalingrank`) und die stochastisch universelle Selektion (`selectionstochunif`). Alle Operatoren wurden bereits im Kapitel 3 beschrieben.

Die GADS Toolbox erlaubt es einem Anwender aber auch, durch eigene Funktionen den GA* zu erweitern, um zusätzliche Funktionalität zu gewährleisten. Über die Options-Struktur können die genetischen Operatoren dann durch die neu implementierten ausgetauscht werden.

4.2 $(\mu, \kappa, \lambda, \rho)$ -Evolutionstrategie

Um eine vollständige Testumgebung für die experimentellen Untersuchungen zu erzeugen, muss die GADS Toolbox um die ES erweitert werden. Da beide im Kapitel 3 beschriebenen Varianten (klassische ES und Cauchy-ES) später einer experimentellen Untersuchung unterzogen werden, müssen beide Varianten hier implementiert werden.

Die ES soll, wegen der starken Ähnlichkeit, dem GA-Teil der GADS Toolbox hinzugefügt werden. Diese Erweiterung ist aber alleine durch den Austausch der genetischen Operatoren nicht möglich. Zum einen wegen der komplexeren Individuen, die neben der Objektkomponente auch über eine Strategiekomponente verfügen, zum anderen und vor allem aber wegen der starren Aufteilung der Individuen auf die unterschiedlichen genetischen Operatoren. Somit müssen zusätzlich zu den neuen Funktionen auch weitere Modifikationen am GA* vorgenommen werden.

4.2.1 Implementierung

Die Idee ist es, die starre Aufteilung der Individuen auf die unterschiedlichen genetischen Operatoren durch Modifikation der Funktion `stepGA` zu umgehen. Die zusätzlich im Individuum kodierten Schrittweiten sowie das Alter zwingen zu Modifikation der Funktion `gacreateuniform`, die die Startpopulation erzeugt. Die ES verfügt zudem über zusätzliche Parameter, die sich über die Funktion `gaoptimset` einstellen lassen müssen, so dass auch diese Funktion zu modifizieren ist. Anschließend kann eine neue Funktion implementiert werden, die einen Zyklus der ES ausführt.

So ist es auf eher einfache Art möglich, die Toolbox um ES zu erweitern. Tatsächlich werden aber weitere Funktionen modifiziert bzw. neu implementiert, um der Toolbox darüber hinaus gehende Funktionalität zu geben. Alle neu implementierten bzw. veränderten Funktionen werden nun im folgenden beschrieben:

Erweiterung der Funktion `stepGA.m`

Wie bereits beschrieben führt die Funktion `stepGA` einen Zyklus des GA durch und soll so erweitert werden, dass auch der Zyklus der ES durchgeführt werden kann. Dazu muss die starre Einteilung der Individuen an die Operatoren überwunden werden, so dass der ursprüngliche Code des GA* nicht angewendet werden kann. Über eine Fallunterscheidung muss der ursprüngliche Code der Funktion `stepGA` übersprungen werden, wenn eine ES angewendet werden soll. Dazu muss klar unterschieden werden, wann die ES und wann der GA* ausgeführt wird. Dies lässt sich am einfachsten über die Mutationsfunktion angeben. Der GA* erlaubt die Eingabe jeder Mutationsfunktion. Hier wird nun die Mutationsfunktion `stepES` exklusiv für die ES reserviert. Wird diese Funktion als Mutationsfunktion angegeben, so wird die ES ausgeführt

(`cauchyStepES` für die ES mit einer auf der Cauchyverteilung basierenden Mutationsfunktion), ansonsten der GA*. Die Realisierung der modifizierten `stepGA` Funktion ist im Folgenden dargestellt:

```
function [nextScore,nextPopulation,state] = stepGA(thisScore, ...
                                                thisPopulation, options, ...
                                                state,GenomeLength,FitnessFcn)

%Ist in Optionsstruktur als MutationsFcn nicht 'stepES' oder
%'cauchyStepES' angegeben führe GA durch
if (~strcmp(func2str(options.MutationFcn),'stepES') && ...
    ~strcmp(func2str(options.MutationFcn),'cauchyStepES'))

%unveränderter stepGA Code

%Ist in Optionsstruktur als MutationsFcn 'stepES' angegeben führe ES durch
if (strcmp(func2str(options.MutationFcn),'stepES') || ...
    strcmp(func2str(options.MutationFcn),'cauchyStepES'))
    parents = 1:length(thisPopulation);
    [nextPopulation nextScore] = feval(options.MutationFcn,parents, ...
                                      options, GenomeLength,FitnessFcn, ...
                                      state,thisScore, thisPopulation, ...
                                      options.MutationFcnArgs{:});
end
```

Wird als Mutationsfunktion eine von `stepES` und `cauchyStepES` verschiedene Funktion angegeben, so handelt es sich um eine Mutationsfunktion für den GA* und es kann der Originalcode der Funktion `stepGA` verwendet werden. Wird jedoch als Mutationsfunktion `stepES` oder `cauchyStepES` angegeben, so wird lediglich diese Funktion ausgewertet, wobei ihr dazu die gesamte aktuelle Population übergeben wird. Beide Funktionen führen dann einen vollständigen ES Zyklus durch (insbesondere also nicht nur die Mutation) und erzeugen dabei eine neue Population samt Bewertung. Da solche Funktionen in der GADS Toolbox noch nicht vorhanden sind, sind diese hier zu implementieren.

Erweiterung der Funktion `gaoptimset.m`

Die ES verfügt über zusätzliche Parameter, um die diese Funktion erweitert werden muss. Ihr müssen Felder für die folgenden Parameter hinzugefügt werden:

1. die Art der Rekombination (sowohl der Objekt- also auch der Strategiekomponente),
2. die Art der Selektion (bzw. der Parameter κ),
3. der Faktor für die Mutation der Strategiekomponente c_τ ,
4. der Selektionsdruck ν ,
5. den Rekombinationsparameter ρ ,
6. die Anzahl der Schrittweiten n_σ ,
7. die Mindestschrittweite, bei deren Unterschreitung der Algorithmus abgebrochen wird,
8. das Intervall in dem die Schrittweiten initialisiert werden,

9. sowie die Matrix A , der Vektor b und eine Funktion *nonLinFcn* für die Nebenbedingungen. Über das Produkt $A \cdot x \leq b$ können dann alle linearen Nebenbedingungen (samt unterer und oberer Grenzen) beschrieben werden. Nichtlineare Nebenbedingungen können über *nonLinFcn* angegeben werden.

Die Erweiterungen werden durch einfaches Hinzufügen der folgenden Einträge in die Funktion *gaoptimset* realisiert:

```
options = struct("Einträge der GA"  
    (...)  
    'RekSigma', 'int', ...  
    'RekX', 'dis', ...  
    'kappa', 1, ...  
    'c_tau', 1, ...  
    'nu', 7, ...  
    'rho', 2, ...  
    'NStepSizes', 'n', ...  
    'StepInitRange', [0,1], ...  
    'StepSizeLimit', 0, ...  
    'A', [], ...  
    'b', [], ...  
    'nonLinFcn', []  
);
```

Während der Erzeugung der Optionsstruktur prüft *gaoptimset* bereits, ob korrekte Datentypen übergeben werden. Somit bedarf es einer weiteren Modifikation, um auch eine Überprüfung der neuen Parameter sicherzustellen.

Diese Überprüfung ist ebenfalls denkbar einfach. Über *case*-Abfragen werden die einzelnen Parameter auf den erlaubten Datentyp hin überprüft. Die neuen Parameter werden nur an den richtigen Stellen in den *case*-Abfragen hinzugefügt. Lediglich für die Parameter *NStepSizes*, *RecX* und *RecSigma* sind neue Methoden zur Überprüfung zu implementieren:

```
case {'NStepSizes'}  
    if ~isa(value,'char') || ~any(strcmp(value,{'1','n'}))  
        valid = 0;  
        errmsg = sprintf('Invalid value for OPTIONS parameter %s: must ...  
            be ''1'' or ''n''.',field);  
    end  
case {'RecX','RecSigma'}  
    if ~isa(value,'char') || ~any(strcmp(value,{'dis','int'}))  
        valid = 0;  
        errmsg = sprintf('Invalid value for OPTIONS parameter %s: must ...  
            be ''dis'' or ''int''.',field);  
    end
```

Eine neue Fallunterscheidung für den Parameter *NStepSizes*, der nur einen der Werte '1' oder 'n' annehmen kann, prüft, ob einer dieser Werte tatsächlich vorliegt. Nur dann wird die Eingabe akzeptiert und ansonsten mit einer Fehlermeldung verworfen. Für die zwei Parameter *RecX* und *RecSigma* wird analog verfahren. Diese Parameter dürfen nur einen der Werte 'dis' oder 'int' annehmen.

Erweiterung der Funktion `validate.m`

Während die Funktion `gaoptimset` nur prüft, ob der für einen Parameter übergebene Datentyp mit dem verlangten übereinstimmt, prüft die Funktion `validate` die eingegebenen Werte auf Korrektheit. So sollen ungewollte Effekte in den Algorithmen vermieden werden, die beispielsweise in endlos-Schleifen enden könnten. Für die ES sind eine Menge an neuen Parametern hinzugekommen, so dass diese zusätzlich zu prüfen sind.

Erneut gestaltet sich die Erweiterung als eher einfach, weil die Funktion `validate` bereits über Unterfunktionen verfügt, die eine Überprüfung auf unterschiedlichste Wertebereiche (z.B. positiv und ganzzahlig) ermöglichen. Die Eingaben müssen dann nur an die entsprechenden Funktionen übergeben werden (die Wertebereiche können Tabelle 3.2 entnommen werden). Um endgültig zu überprüfen, ob alle Parameter korrekt übergeben worden sind, müssen die Parameter ρ und ν im Kontext zu den anderen Parametern betrachtet werden. Für den Parameter ν muss für $\kappa < \infty$ gelten: $\nu > 1$. Der Parameter ρ darf nicht größer gewählt werden, als die Population groß ist. Beide Anforderungen können über eine einfach bedingte Anweisung überprüft werden.

Zusätzlich werden in dieser Funktion die Eingaben für die uniforme Initialisierung überprüft. Weil Individuen der ES zusätzlich zur Objektkomponente auch noch über eine Strategiekomponente verfügen, ist eine weitere Eingabe erforderlich, die das Intervall angibt, in dem die Schrittweiten zu initialisieren sind. Erneut kann über bedingte Anweisungen überprüft werden, ob tatsächlich ein Intervall angegeben wurde. Das ist dann der Fall, wenn die Eingabe aus zwei Zeilen besteht, von denen der erste den linken und der zweite den rechten Endpunkt des Intervalls angibt. Dann muss die erste Zeile kleiner als die zweite sein und keine der Beiden darf $\pm \infty$ enthalten, damit ein endliches Intervall erzeugt wird.

Die Funktion, die die Strategiekomponente initialisiert, erfordert für jede zu initialisierende Schrittweite ein eigenes Intervall. Dennoch müssen nicht für alle Schrittweiten eigene Intervalle angegeben werden, so dass in Extremfall auch ein einzelnes Intervall ausreichend ist. Dazu wird die Funktion `validate` so modifiziert, dass ausreichend viele Intervalle erzeugt werden, in dem intern das letzte entnommen und so oft vervielfältigt wird, bis schließlich pro Schrittweite genau ein Intervall zu Verfügung steht.

Als letztes muss überprüft werden, ob die Intervalle positiv sind. Die Schrittweiten der ES dürfen nur positive Werte annehmen, so dass die Intervalle, in denen die Schrittweiten zu initialisieren sind, ebenfalls vollständig positiv sein müssen. Die Überprüfung kann erneut sehr einfach über die bedingte Anweisung

```
if (any(lb <= 0) || any(ub <= 0))
    error('gads:VALIDATE:not pos. stepsize: init interval not positiv');
end
```

erfolgen. Ein Intervall ist dann nicht vollständig positiv, wenn eines der Enden negativ ist. Über die Anweisung `any` wird überprüft, ob es ein solches Element gibt. In einem solchen Fall wird dann ein Fehler ausgegeben.

Implementierung der `stepES.m`

Die Funktion `stepES` wird so implementiert, dass sie die Ausführung der im Kapitel 3 beschriebenen ES ermöglicht. Anders als `stepGA` ruft sie dazu keine weiteren Funktionen auf, sondern implementiert die gesamte Funktionalität der ES. Diese Funktion überprüft im ersten Schritt ob eine oder aber n Schrittweiten verwendet werden. Über eine bedingte Anweisung kann dann der für den gegebenen Fall benötigte Code ausgeführt werden. So wird zwar der Code aufgebbläht, weil alle Operatoren zweimal implementiert werden, dies hat aber den Vorteil, dass in

den Operatoren selbst keine bedingten Anweisungen mehr zu beachten sind. So kann einerseits die Ausführung des Codes insgesamt beschleunigt und andererseits der Code besser lesbar gemacht werden. Die Implementierung wird hier im Folgenden für den Fall von n Schrittweiten beschrieben. Der Fall nur einer Schrittweite läuft analog ab.

Nach Feststellung, dass n Schrittweiten vorliegen, kommen die genetischen Operatoren zu Anwendung. Weil die Individuen als Vektoren codiert und in Matrizen zusammengefasst sind bietet Matlab besonders effiziente Operatoren an, auf denen diese Implementierung ansetzen wird.

Die Rekombination erzeugt die benötigten $\lambda = \nu \cdot \mu$ Nachkommen, in dem sie über eine Schleife mit λ Zyklen je ein Nachkommen erzeugt. Dazu wird eine zufällige Permutation der aktuellen Population erzeugt, aus der die ersten ρ Individuen entnommen werden und die Eltern bilden. Die so selektierten Eltern werden dann rekombiniert, wobei zwischen der diskreten und der intermediären Rekombination unterschieden werden kann. Beide Varianten werden sowohl für Objekt- als auch für Strategiekomponente implementiert, unterscheiden sich aber nur gering in der Art, dass der Operator für die Objektkomponente auf dem vorderen Teil (1 bis GenomeLength) des Individuums arbeitet, während der für die Strategiekomponente am hinteren Teil (GenomeLength bis $2 \cdot \text{GenomeLength}$) ansetzt. Die Implementierung beider Operatoren wird hier daher nur für die Objektkomponente eingeführt.

Der Matlab Befehl `mean` erzeugt einen Zeilen-Vektor, in dem er jede Spalte einer Matrix mittelt. Dieser Befehl ist optimal geeignet für die intermediäre Rekombination, da die Paarungsselektion eine Matrix zurückgibt, deren Zeilen einzelne Individuen sind. Für die Rekombination der Objektkomponente wird der vordere Teil des Individuums (Objektkomponente) betrachtet, aus der über den Befehl `mean` direkt der Nachkomme folgt. Die intermediäre Rekombination kann somit über die simple Anweisung

```
recoChildren(i, 1:GenomeLength) = mean(parent(:, 1:GenomeLength));
```

realisiert werden.

Die diskrete Rekombination entnimmt einzelne Werte aus den Eltern und erzeugt so ein Nachkommen. Matlab erlaubt es Matrizen nur über ein Argument anzusprechen. Die Nummerierung der einzelnen Elemente ergibt sich dann für das Beispiel einer (3×4) -Matrix als:

$$A_{Indices} = \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

Der Rekombinationsoperator muss aus jeder Spalte der Matrix, in der die Eltern zusammengefasst sind, genau ein Element entnehmen, um insgesamt einen Nachkommen zu erzeugen. Dazu wird zunächst ein Index `perm` erzeugt. Dieser Index gibt an Position i an, welchem Elter die i -te Koordinate entnommen werden soll. Dazu wird für jede Koordinate eine gleichverteilte Zufallszahl aus der Menge $\{0, \dots, \rho - 1\}$ erzeugt. Diese Zufallszahl bestimmt, aus welchem Individuum die betreffende Koordinate zu entnehmen ist. Um diese Zufallszahlen auch auf der oben beschriebenen Indizierung anzuwenden, muss für die erste Koordinate eine 1, für die zweite Koordinate $\rho + 1$, für die dritte Koordinate $2 \cdot \rho + 1$ und schließlich für die n -te Koordinate $n \cdot \rho + 1$ addiert werden. Anschließend werden mit Hilfe dieser Indizes die einzelnen Koordinaten aus den Eltern entnommen und bilden so einen Nachkommen. Der folgende Code stellt die Methode nochmals kompakt dar:

```
perm = fix(options.rho*rand(1, GenomeLength))+ ...  
      (1:options.rho:options.rho*GenomeLength);  
recoChildren(i,1:GenomeLength) = parent(perm); .
```

Nach Erzeugung der Nachkommen über Rekombination muss im nächsten Teil die Mutation durchgeführt werden. Dazu werden die Nachkommen über eine weitere Schleife einzeln entnommen und beginnend mit der Strategiekomponente mutiert. Die Mutation erfolgt dabei analog zur Beschreibung im Kapitel 3 und kann durch elementare Rechenoperatoren realisiert werden. Im letzten Schritt wird das Alter der neuen Individuen auf 0 gesetzt und die Nachkommen in einer Matrix zusammengefasst.

Falls Nebenbedingungen existieren, müssen die Nachkommen auf jene überprüft werden. Für die linearen Nebenbedingungen ist dies sehr einfach. Es können die linearen Nebenbedingungen $A \cdot x \leq 0$ sowie untere l und obere Schranken u angegeben werden. Alle drei Arten müssen dazu in einer Matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,(n-1)} & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,(n-1)} & a_{m,n} \\ -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & 0 \\ 0 & 0 & \dots & 0 & -1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

und einem Vektor

$$b = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -l_1 \\ \vdots \\ -l_n \\ u_1 \\ \vdots \\ u_n \end{pmatrix}$$

zusammengefasst werden, so dass anschließend nur noch der Ausdruck

$$A \cdot x \leq b$$

zu überprüfen ist. Hier ermöglicht Matlab erneut eine sehr einfache Berechnung durch den Ausdruck

```
lethal = sum(options.A * mutChildren(:, 1:GenomeLength)' ...
    > repmat(options.b,1,length(mutChildren(:,1)))); .
```

Die Matrix A wird mit den transponierten Objektkomponenten aller Nachkommen in einem Schritt multipliziert, so dass eine weitere Matrix entsteht, deren Element $a_{i,j}$ angibt, ob für das j -te Individuum die i -te Nebenbedingung verletzt ist. Das aufsummieren der Zeilen dieser Matrix ergibt dann einen Vektor, dessen i -ter Eintrag angibt, wie viele Nebenbedingungen für das i -te Individuum verletzt sind. Insgesamt müssen so alle Individuen entfernt werden, für

die eine oder mehr Nebenbedingungen verletzt sind. Ähnlich wird auch für die nicht-linearen Nebenbedingungen verfahren, wobei hier die nicht-lineare Funktion auszuwerten ist.

Wurden Individuen entfernt, so müssen diese erneut durch Paarungsselektion, Rekombination und Mutation erzeugt und anschließend auf Einhaltung der Nebenbedingungen überprüft werden. Aus diesem Grund wird die Prüfung der Nebenbedingung, Paarungsselektion, Rekombination und Mutation in einer `while` Schleife eingebettet, die solange abgearbeitet wird, bis die benötigten Individuen erzeugt sind.

Anschließend kann die Selektion durchgeführt werden, die die aktuelle Population und die Nachkommen in einer Matrix `mutChildren` zusammengefasst, und aus dieser die Individuen entfernt, deren Alter κ überschreitet. Dies kann in Matlab ebenfalls sehr einfach durch den Ausdruck

```
mutChildren = [thisPopulation; mutChildren];
delete = ( mutChildren(:, 2*GenomeLength+1) >= ...
          repmat(options.Kappa,length(mutChildren(:,1)),1) );
deleteInd = find(delete==1);
mutChildren(deleteInd, :) = [];
```

erreicht werden. Hier wird das Alter aller Individuen in einem Schritt überprüft. Das Ergebnis dieser Überprüfung ist ein Vektor `delete`, dessen Position i dann den Wert 1 annimmt, wenn das Individuum das Alter κ erreicht hat. Über die Anweisung `find` werden solche Positionen bestimmt, deren Werte 1 sind. Die Individuen in der Matrix `mutChildren` die sich an diesen Positionen (Zeilen) befinden werden dann entfernt.

Für die tatsächliche Selektion werden die Individuen über die Fitnessfunktion bewertet und über die Anweisung `sort` absteigend sortiert. Durch Übernahme der ersten μ Individuen werden so deterministisch die besten Individuen selektiert. Diese Individuen können dann samt Fitness an die Funktion `ga` zurückgegeben werden.

Implementierung der `cauchyStepES.m`

Die Implementierung der Funktion `cauchyStepES` kann hier sehr kurz beschrieben werden, da sie analog zu Implementierung der Funktion `stepES` erfolgt. Es muss lediglich der Zufallszahlengenerator für die Mutation der Objektkomponente ausgetauscht werden, da die Objektkomponente hier basierend auf cauchyverteilten Zufallszahlen mutiert wird.

Implementierung der `randc.m`

Weil Matlab keinen Zufallszahlengenerator zu Verfügung stellt, der cauchyverteilte Zufallszahlen erzeugt, ist ein solcher zu implementieren. Hildebrand (2001) beschreibt ausführlich, wie Generatoren für Φ -verteilte Zufallszahlen implementiert werden können. Er erzeugt die inverse Verteilungsfunktion $\bar{\Phi}$, die gleichverteilte Zufallszahlen aus dem Intervall $]0, 1[$ abbildet. Diese Bilder sind dann Φ -verteilt. Diese Methode wird als *Transformation der Inversen* bezeichnet und im Folgenden ausführlicher für die Cauchyverteilung erklärt.

Aus der Dichtefunktion $f(x) = \frac{1}{\pi \cdot (1+x^2)}$ der Cauchyverteilung (mit Zentrum 0 und Breitenparameter 1) kann durch Integration die Verteilung berechnet werden, die im letzten Schritt invertiert wird.

$$f(x) = \frac{1}{\pi \cdot (1 + x^2)}$$
$$P(X \leq b) = \int_{-\infty}^b f(x) dx = \frac{\arctan(x)}{\pi} + \frac{1}{2}$$
$$\bar{P} = -\cot(\pi \cdot x) .$$

Anschließend erzeugt die inverse Verteilungsfunktion \bar{P} ein Bild einer gleichverteilten Zufallszahl aus dem Intervall $]0, 1[$. Die so erzeugte Zahl ist dann cauchyverteilt. Die Idee dieses Verfahren ist in Abbildung 4.2 dargestellt. Die inverse Cauchyverteilung hat

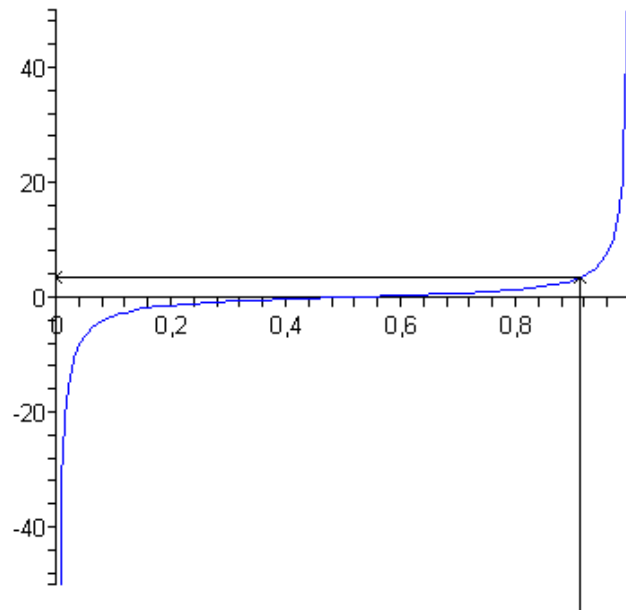


Abbildung 4.2: Inverse Cauchyverteilung

den Definitionsbereich $]0, 1[$ und den Wertebereich $] - \infty, \infty[$. An den Rändern des Definitionsbereichs verläuft sie eher steil, während sie in der Mitte eher einen flachen Verlauf aufweist. Werden nun gleichverteilte Zufallszahlen aus dem Intervall $]0, 1[$ erzeugt und von der inversen Verteilungsfunktion abgebildet, so verteilen sich die Bilder auf dem Wertebereich $] - \infty, \infty[$ entsprechend der Cauchyverteilung. Die Mehrzahl der so erzeugten Zufallszahlen verteilt sich dann um dem Erwartungswert 0, während die Wahrscheinlichkeit für betragsmäßig größere Zahlen eher gering ist.

Der Beweis (Hildebrand, 2001) basiert auf dem Satz, dass für beliebige $x \in \mathbb{R}$ gilt:

$$P(X \leq x) = \Phi .$$

In der Annahme, dass der oben beschriebene Generator korrekt arbeitet, gilt:

$$P(X \leq x) = P(\bar{\Phi}(G(0,1)) \leq x)$$

und, weil $\bar{\Phi}$ die inverse Funktion zu Φ ist, schließlich im letzten Schritt:

$$P(X \leq x) = P(G(0,1) \leq \Phi(x)) = \Phi(x)$$

■

Der Generator für cauchyverteilte Zufallszahlen kann somit sehr einfach über eine Funktion `randc(n,m)` realisiert werden, der eine $(n \times m)$ Matrix erzeugt, deren Einträgen einer Cauchyverteilung entnommen sind. Die Funktion wird wie folgt implementiert:

```
gv = rand(n,m);  
y = -cot(pi*gv);
```

Sie erzeugt zunächst die benötigten $n \times m$ gleichverteilten Zufallszahlen, die sie dann entsprechend der oben ermittelten Abbildung in cauchyverteilte Zufallszahlen transformiert.

An Generatoren für Zufallszahlen werden vor allem die Anforderungen der Effizienz und der Reproduzierbarkeit gestellt. Beide Anforderungen werden durch die oben beschriebene Implementierung erfüllt. Die Effizienz ergibt sich direkt aus dem einfachen geschlossenen arithmetischen Ausdruck, der durch Matlab sehr schnell ausgewertet werden kann (vorausgesetzt Matlab erzeugt effizient gleichverteilte Zufallszahlen). Die Reproduzierbarkeit verlangt von einem Generator, dass eine bestimmte Sequenz von Zufallszahlen erneut erzeugt werden kann. Da die hier vorliegende Implementierung auf dem Matlab-Generator für gleichverteilte Zufallszahlen ansetzt, der selbst reproduzierbar ist¹, folgt direkt, dass auch der Generator für cauchyverteilte Zufallszahlen reproduzierbar ist.

Erweiterung der Funktion `gacreateuniform.m`

Aufgrund der komplexeren Individuen muss die Methode zur Initialisierung erweitert werden. Sie muss zusätzlich zu der Objektkomponente noch die Schrittweiten und das Alter der Individuen initialisieren. Schrittweiten sind nur dann zu erzeugen, wenn eine ES durchgeführt wird. Dies kann über eine bedingte Anweisung überprüft werden. Zusätzlich muss auf gleiche Art festgestellt werden, wie viele Schrittweiten zu erzeugen sind. Die Funktion kann dann wie folgt realisiert werden:

```
if ( strcmp(options.NStepSizes,'n') && ...  
    ( strcmp(func2str(options.MutationFcn),'stepES') || ...  
      strcmp(func2str(options.MutationFcn),'cauchyStepES')  
    )  
    )  
    range = options.StepInitRange;  
    lowerBound = range(1,:);  
    span = range(2,:) - lowerBound;  
    StepSizes = repmat(lowerBound,totalstepsizes,1) + ...  
                repmat(span,totalstepsizes,1) .* ...  
                rand(totalstepsizes,GenomeLength);  
  
    Population = [Population StepSizes repmat(1,totalpopulation,1)];  
end
```

Für jede Schrittweite ist die untere Schranke gegeben durch den Zeilenvektor `lowerBound`. Dieser Vektor wird über die Anweisung `repmat` so reproduziert, dass eine Matrix erzeugt wird, die genau so viele Spalten hat, wie Individuen zu erzeugen sind. Ein weiterer Vektor `span` gibt für jede Schrittweite an, wie lang das Intervall ist, in dem sie zu initialisieren ist. Mit diesem Vektor wird analog verfahren. Mit diesen Informationen erlaubt es Matlab die Strategiekomponenten für die gesamte Population in einem Schritt zu berechnen. Die erste Matrix stellt bereits gültige Strategiekomponenten für die Startpopulation dar. Durch Gewichtung der zweiten Matrix mit Zufallszahlen aus dem Intervall $[0, 1]$ und Addition zur ersten werden so im Ergebnis Strategiekomponenten erzeugt, die sich im angegebenen Intervall verteilen. Schließlich werden die Objektkomponenten (hier: `Population`), die Schrittweiten und ein 1-Spaltenvektor zu einer

¹Über den Befehl `rand('state', S)` kann der Matlab-Generator auf einen beliebigen Zustand gesetzt werden, insbesondere auch auf den zuletzt verwendeten Zustand.

Matrix zusammengefasst, deren Zeilen dann die Individuen der ES darstellen.

Die Initialisierung bei nur einer Schrittweite läuft analog ab, mit dem Unterschied, dass hier nur noch eine Schrittweite zu initialisieren ist.

Implementierung der Funktion `creationAroundPoint.m`

Die Funktion `creationAroundPoint` implementiert die in Schwefel & Rudolph (1995) beschriebene Methode eine Startpopulation zu erzeugen. Diese Methode erwartet vom Benutzer die Eingabe einer Objektkomponente und einer Schrittweite, die angibt in welchem Radius sich die übrigen Objektkomponenten um die Erste verteilen sollen. Die restlichen Objektkomponenten für die Startpopulation werden dann über eine Art Mutation erzeugt:

```
Population = repmat(point,options.PopulationSize-1,1) + ...
             stepsize * randn(options.PopulationSize-1,GenomeLength);
```

Die Strategiekomponente wird hier analog zu Funktion `gacreationuniform` erzeugt.

Implementierung der Funktion `validInitPop.m`

Hat die Startpopulation eine große Distanz zum zulässigen Bereich gelingt es einer ES nur schwer über Rekombination und Mutation Nachkommen zu erzeugen, die im zulässigen Bereich liegen. Weil aber Nachkommen, die die Nebenbedingungen nicht einhalten, sofort verworfen werden, gelangt die ES so praktisch in eine endlos-Schleife.

Die Funktion `validInitPop` fängt diesen potenziellen Fehler ab, in dem sie überprüft, ob die Startpopulation die Nebenbedingungen einhält. Der Test läuft analog zu Behandlung der Nebenbedingungen in der Funktion `stepES` ab. Sollten die Startpopulation die Nebenbedingungen nicht einhalten gibt diese Funktion eine Fehlermeldung an den Anwender aus.

Erweiterung der Funktion `isItTimeToStop.m`

Die Schrittweiten der ES sind ein Indiz für den Verlauf der Optimierung, aus denen ein Abbruchkriterium hergeleitet werden kann. Sie nehmen mit zunehmenden Optimierungsfortschritt ab, so dass bei Unterschreitung einer gewissen Schrittweite, die über den Parameter `StepSizeLimit` eingestellt wird, abgebrochen werden kann. Weil die Population i.d.R. aus mehreren Individuen besteht, die ihrerseits über mehrere Schrittweiten verfügen, wird stets die größte Schrittweite auf Unterschreitung der Schranke überprüft. Anderenfalls könnte der Fall eintreten, dass viele (jedoch nicht alle) Entscheidungsvariablen optimal eingestellt sind und die zugehörigen Schrittweiten sehr klein werden. Würde man dann an Stelle der größten Schrittweite den Mittelwert oder Median betrachten, so könnte so eine vorzeitige Konvergenz verursacht werden.

Die Funktion `ga` bricht die Optimierung dann ab, wenn der String `exitFlag` nicht leer ist. Dieser String beinhaltet in einem solchen Fall den Grund für den Abbruch. Die Funktion `isItTimeToStop` verwendet wieder bedingte Anweisungen, um zu prüfen, ob eine Abbruchbedingung erfüllt ist. Falls dies der Fall ist, schreibt die Funktion dann den entsprechenden Grund in den String. Die Erweiterung kann durch eine zusätzliche bedingte Anweisung

```
elseif(largest <= options.StepSizeLimit)
    reasonToStop = sprintf(['Optimization terminated: ', ...
                           'minimum stepsize limit reached.']);
```

realisiert werden.

Um den Ausdruck auswerten zu können ist es notwendig die größte Schrittweite in der Population zu bestimmen. Dazu wird überprüft ob eine ES durchgeführt wird, weil nur dann Schrittweiten zu bestimmen sind. Im Fall eines GA wird `largest` auf ∞ gesetzt. Dann verursacht die

Erweiterung bei Verwendung des GA* keinen Fehler (der Wert wäre sonst nicht belegt), zum anderen wird diese Abbruchbedingung so im Fall des GA* deaktiviert. Exemplarisch wird die Bestimmung der größten Schrittweite für den komplexeren Fall von n Schrittweiten gezeigt. Der Codeausschnitt

```
indLength = length(state.Population(1,:));
gs = fix(indLength / 2);
stepSizes = state.Population(:, gs+1 : indLength-1);
largest = max(max(stepSizes));
```

bestimmt zunächst die Länge der Individuen `indLength`. Das ist notwendig, weil die Genomlänge (Anzahl der Variablen) der Funktion `isItTimeToStop` nicht bekannt ist. Für n Schrittweiten wird die Länge der Individuen ungerade sein. Das Individuum hat dann die folgende Gestalt:

$$I = \left(\overbrace{x_1, x_2, \dots, x_n}^{\text{Objektkomp.}}, \overbrace{s_1, s_2, \dots, s_n}^{\text{Strategiekomp.}}, \overbrace{K}^{\text{Alter}} \right)$$

mit $\text{length}(I) = 2 \cdot n + 1$. Die Variable `gs` bildet das Ende der Objektkomponente und somit `gs+1` bis `indLength-1` den Vektor der Strategiekomponente. Die größte Schrittweite in der Population wird dann durch den doppelten Aufruf der Funktion `max` geliefert.

Wird nur eine Schrittweite adaptiert, so befindet sich diese immer an der vorletzten Stelle des Individuums. Hier liefert ebenfalls die Funktion `max` die größte Schrittweite der Population.

Implementierung der Funktion `gaplotstepsize.m`

Weil die Schrittweiten ein starkes Indiz für den Fortschritt der Optimierung sind, kann es notwendig werden, diese über die Zeit graphisch auszugeben. Wird nur eine Schrittweite verwendet, gestaltet sich die Realisierung eher einfach. Dann kann in einer Generation t die Schrittweite des besten Individuums ausgegeben werden. Die Ausgabe gestaltet sich problematischer, wenn an Stelle nur einer n Schrittweiten verwendet werden. Dann werden in jeder Generation t zwei Punkte ausgegeben. Dazu wird aus den n Schrittweiten die größte und kleinste ermittelt, die dann beide Punkte ergeben.

Erneut müssen, auf gleiche Art wie bereits bei der Erweiterung der Funktion `isItTimeToStop`, zunächst die Schrittweiten im Individuum identifiziert werden. Danach wird über die Anweisung `min` zunächst das beste Individuum ermittelt und anschließend seine Schrittweite graphisch über eine weitere Anweisung `plot` ausgegeben.

Anstelle einfacher Punkte hätte Matlab auch die Möglichkeit eröffnet einen *Errorbar* auszugeben. Dieser hätte die minimale und maximale Schrittweite ausgegeben und zusätzlich noch den Mittelwert aller Schrittweiten. Diese Methode hätte aber verlangt, dass vorab eine maximale Anzahl der Generationen als Abbruchbedingung angegeben wird. So hätte der Graph aber in vielen Fällen nicht mehr ausgegeben werden können.

Erweiterung der Funktion `makeState.m`

Die Funktion `ga` verwaltet weitere Informationen in einer weiteren Struktur, die sie während der Optimierung aktualisiert. Diese Struktur benötigt zwei weitere Einträge, die die bisher beste Lösung und deren Fitness speichern. Die Funktion `makeState` initialisiert diese Struktur, in dem sie den dort verwalteten Variablen einen initialen Wert zuweist. Sie muss somit für die zwei zusätzlichen Variablen erweitert werden. Weil zum Zeitpunkt des Starts der Optimierung noch keine Lösungen bekannt sind, wird die optimale Lösung auf `[]` und deren Wert auf $+\infty$ gesetzt.

Erweiterung der Funktion `ga.m`

Die Fitness des besten Individuums des GA* über die Zeit ist monoton fallend. Das wird dadurch erreicht, dass die besten *EliteCount* Individuen stets in die nächste Generation übernommen werden. ES, für die $\kappa < \infty$ gilt, können während des Verlaufs der Optimierung die beste Lösung aber vergessen. Wie bereits beschrieben verwaltet die ES daher zwei Variablen, in denen die beste Lösung bzw. die Fitness der besten Lösung gespeichert wird. Nun muss die Funktion `ga` nur so verändert werden, dass bei Verwendung einer ES nicht mehr die beste Lösung der letzten Population ausgegeben wird, sondern die in den zwei zusätzlichen Variablen gespeicherte Lösung. Der aufmerksame Leser wird sich bereits vorstellen können, dass dies erneut über eine bedingte Anweisung bewerkstelligt wird. Diese ist erneut dann erfüllt, wenn eine ES ausgeführt wird. Dann wird die Lösung des GA überschrieben durch die beste Lösung, die im Laufe aller ES Zyklen gefunden wurde.

Implementierung der Funktion `EvolutionStrategy.m`

Die Funktion `EvolutionStrategy` stellt eine graphische Benutzeroberfläche zu Verfügung, die es dem Benutzer erlaubt alle Einstellungen auf komfortable Art durchzuführen. Graphische Benutzeroberflächen lassen sich in Matlab (wie in den meisten anderen Programmiersprachen auch) über spezielle Werkzeuge zusammenstellen. Anschließend braucht nur noch die Funktionalität implementiert zu werden. Das Werkzeug in Matlab, das die Erzeugung von graphischen Benutzeroberflächen (engl. Graphical User Interface (GUI)) ermöglicht heißt GUI Design Environment, oder kurz GUIDE. Über den gleichnamigen Aufruf kann dieses Werkzeug gestartet werden und erlaubt dann die schnelle Erzeugung der GUI.

Jede GUI setzt sich aus zwei Dateien zusammen. In einer `fig` Datei wird die Oberfläche abgelegt. Eine weitere `m` Datei implementiert die Funktionalität der GUI. Sie enthält für jede Komponente, die eine Benutzereingabe ermöglicht mindestens zwei Funktionen. Die erste Funktion wird aufgerufen, wenn die Komponente erzeugt wird, die zweite, wenn eine Eingabe über die Komponente getätigt wird. Benötigt wird nur die Funktion, die auf Benutzereingaben reagiert. Sie muss für die Radiobuttons, die die Auswahl der Schrittweiten ermöglichen, implementiert werden, um sicherzustellen, dass nur ein Radiobutton aktiviert werden kann. Das Popupmenu, aus dem die Methode zur Initialisierung selektiert werden kann, verlangt auch die Implementierung dieser Funktion. Je nach Methode zur Initialisierung werden unterschiedlich viele Eingabefelder benötigt, die von der Funktion erzeugt werden müssen. Als letztes muss diese Funktion noch für den Startbutton implementiert werden.

Hier müssen die Eingaben aller Komponenten ausgelesen werden. Diese Informationen werden dann über die Funktion `gaoptimset` an die Optionsstruktur übergeben. Anschließend kann die Funktion `ga` aufgerufen werden, deren Ergebnis über die entsprechende Listbox ausgegeben wird.

Auf die Beschreibung der genauen Bedienung der GUI soll hier verzichtet werden. Es ist hier ausreichend zu wissen, dass die GUI über den Aufruf `EvolutionStrategy` in der Matlab Kommandozeile gestartet wird. Über den Menüeintrag *Help* kann dann bei Bedarf eine genaue Anleitung abgerufen werden.

4.2.2 Verifikation

Bereits während der Implementierung und parallel dazu ablaufender Tests wurden einige Fehler in der Implementierung aufgedeckt und behoben. Nun soll die Implementierung der ES systematischer auf Korrektheit überprüft werden. Dazu wird in einem ersten Schritt die ES in ihre Komponenten *Rekombination*, *Mutation*, *Selektion* und die *Behandlung der Nebenbedingungen*

zerlegt, die einzeln einem Test unterzogen werden. Es werden Paare aus Eingabe und gewünschter Ausgabe betrachtet (Sollergebnis). Nach Spezifikation dieser Paare wird die Eingabe der zu testenden Komponente übergeben, die genau die gewünschte Ausgabe erzeugen muss. Es ist nicht möglich einen erschöpfenden Test durchzuführen. Für einen solchen Test müssten alle möglichen Eingaben der ES auf Korrektheit untersucht werden, von denen es im hier vorliegenden Fall unendlich viele gibt. Stattdessen wird der Test auf die Randbereiche beschränkt, da hier die Wahrscheinlichkeit für einen Fehler besonders hoch ist.

Problematisch an diesem Vorgehen ist das stochastische Verhalten der ES, die in jedem Lauf zu unterschiedlichen Werten führen wird. In den Tests werden daher die Zufallszahlengeneratoren durch Methoden ersetzt, die deterministisch vorgegebene Zahlen ausgeben. Die Zufallszahlen sind dann im vornherein bekannt, so dass das Ausgabeverhalten klar definiert werden kann.

Wenn nicht anders angegeben, werden die nun folgenden Tests mit den Standardeinstellungen aus Kapitel 3 durchgeführt.

Rekombination

Der Test der Rekombination unterteilt sich in einen Test für die diskreten und einen für die intermediäre Rekombination. Es muss zwischen Rekombination der Strategie- und die der Objektkomponente, sowie der Anzahl der Schrittweiten unterschieden werden. Die Tests werden für $\mu = \rho = 2$ und für $\rho = 2 < \mu = 4$ durchgeführt. Im ersten Fall nimmt die gesamte Population an der Rekombination teil, im zweiten nur eine Teilmenge davon.

Insgesamt sind so 16 Tests durchzuführen, die mögliche Fehler am Rekombinationsoperator aufdecken müssten. Es würde zu weit gehen alle Tests detailliert zu beschreiben, stattdessen wird hier nur die Vorgehensweise des ersten Tests beschrieben und anschließend die Ergebnisse aller Tests zusammengefasst.

Zum Testen der Rekombination werden zwei Individuen auf einem 2-dimensionalen Problem verwendet. Die Objektkomponente des ersten Individuums wird auf $[10, 5]$ und die des zweiten Individuums auf $[25, 15]$ initialisiert. Es wird intermediäre Rekombination verwendet, die als Ergebnis einen Nachkommen mit Objektkomponente $[17.5, 10]$ erzeugen muss.

Nach Ausführung der ES mit abgeschalteter Mutation und Ausführung nur einer Generation wird genau das gewünschte Ergebnis ausgegeben.

Die diskrete Rekombination wird auf der gleichen Testinstanz verwendet. Die benötigten zwei Zufallszahlen werden durch 0 und 1 ersetzt. Das Ergebnis der diskreten Rekombination für diese Instanz müsste $[10, 15]$ sein.

Unter Substituierung der Zufallszahlen durch 0 und 1 kann gerade das oben beschriebene Ergebnis beobachtet werden.

Die anderen 14 Tests werden analog durchgeführt und liefern stets die gewünschten Ergebnisse.

Mutation

Für den Tests der Mutation wird die Rekombination ausgeschaltet. Dazu bedarf es keiner Änderung im Code; die Rekombination kann durch $\rho = 1$ deaktiviert werden. Der Generator für normalverteilte Zufallszahlen wird durch eine deterministische Methode ersetzt, die konstant 1 ausgibt. Die Strategiekomponente wird auf $[1, 0.5]$ festgesetzt und als Testproblem wird $f = \sum_{i=1}^2 x_i^2$ verwendet. Ausgehend von einem Punkt $[10, 5]$ soll eine $(1 + 1)$ -ES durchgeführt werden, die nach 10 Generationen das Optimum $[0, 0]$ finden muss.

Das Resultat des Tests erfüllt nicht das erwartete Ergebnis. Bereits in der ersten Generation wird ein Wert zurückgegeben, der so nicht hätte eintreten dürfen.

Widererwartend war die Mutation nicht die Quelle dieses Fehlers. Der Fehler wird durch den

Rekombinationsoperator verursacht, der sich nicht ausschalten lässt. Wird $\rho = 1$ gewählt, so wird genau ein Elter selektiert. Die Matlab Anweisung `mean` (die die intermediäre Rekombination realisiert) kann auf dem so entstehenden Zeilenvektor nicht das gewünschte Ergebnis liefern. Über eine einfache Ergänzung

```
if(options.rho == 1)
recoChildren(i, 1:2*GenomeLength+1) = ...
    thisPopulation((recoParents), 1:2*GenomeLength+1);
else
%Code der Rekombination
```

wird für $\rho = 1$ der einzige Elter direkt als Nachkomme übernommen. Ansonsten kann die Rekombination, wie zuvor implementiert, durchgeführt werden. Nach dieser Korrektur liefert die ES das vorhergesagte Ergebnis. Die Abbildung 4.3 zeigt links den fehlerhaften und rechts den korrekten Verlauf der ES. Das bessere Resultat in den ersten Generationen der fehlerhaften

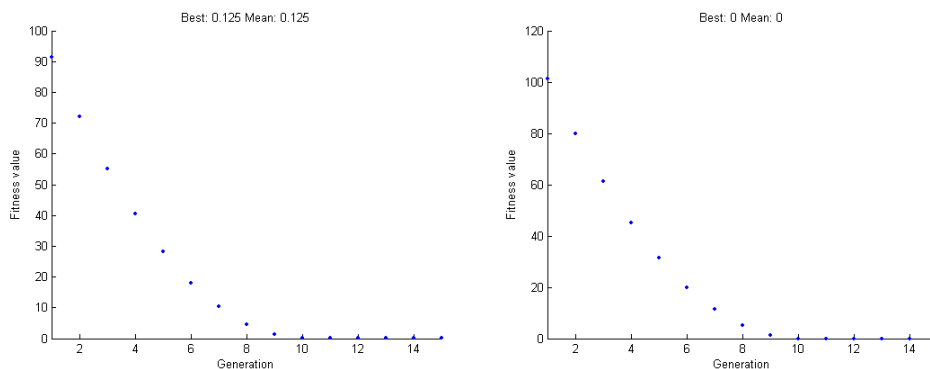


Abbildung 4.3: (links) fehlerhafter, (rechts) korrekter Verlauf der ES

ES ist hier rein zufällig und nicht typisch.

Nach Korrektur des Fehlers erzielt die ES das Soll-Ergebnis, sowohl für n Schrittweiten, als auch für eine Schrittweite. Ein analoger Test wurde auch für die Mutation der Schrittweiten durchgeführt und lieferte ebenfalls das gewünschte Ergebnis.

Weitere Untersuchungen bedarf es hier nicht. Die Mutation bekommt über eine `for` Schleife stets nur ein Individuum übergeben, so dass es unerheblich ist, ob $\lambda = \nu \cdot \mu = 1$ oder $\lambda = \nu \cdot \mu > 1$ gewählt wird. Der Parameter c_T muss ebenfalls nicht separat betrachtet werden, da auf diesem Parameter lediglich elementare arithmetische Operatoren angewendet werden, deren korrektes Arbeiten vorausgesetzt wird.

Selektion

Die Selektion wird für $\kappa = 1$ (Komma-Selektion) und $\kappa = \infty$ (Plus-Selektion) untersucht. Der Parameter κ hat den Definitionsbereich \mathbb{N} , so dass die beiden gewählten Einstellungen die minimale bzw. maximale Eingabe in der Matlab Implementierung darstellen. Wird $\kappa = \infty$ gewählt muss der zeitliche Verlauf der Fitnesswerte monoton fallend sein, anderenfalls ist die Selektion fehlerhaft. Um eine hohe Fehlerquelle zu induzieren, wird der Selektionsdruck besonders hoch gewählt, so dass eine hohe Anzahl an Individuen zu Verfügung steht, die in eine kleine Population selektiert werden muss. Der Test muss sowohl für n Schrittweiten als auch für eine Schrittweite durchgeführt werden, da sich die Position des Alters im Individuum je nach Art der Strategiekomponente variiert.

Der Graph der Fitness über die Zeit der ES mit einer solchen Einstellung verläuft sowohl für

eine als auch für n Schrittweiten monoton fallend und somit korrekt.

Die Überprüfung der Komma-Selektion gestaltet sich schwieriger, da sie sich nicht so einfach wie die Plus-Selektion über den Verlauf der Fitness charakterisieren lässt. Hier ist es nun notwendig, die Population in jeder Generation zu betrachten. Der Test soll auf einer Populationsgröße $\mu = 2$ und dem Selektionsdruck $\nu = 2$ erfolgen. Als Ergebnis muss in jeder Generation eine vollständig neue Population erzeugt werden, deren Individuen das Alter 1 haben.

Das Ergebnis bei Verwendung von n Schrittweiten ist das folgende:

```
P =  
 15.0589   18.1127   1.0000   1.0000   1.0000  
 16.4087   14.6658   1.0000   1.0000   1.0000
```

```
P =  
 11.9102   13.1963   3.2673   0.8271   1.0000  
 14.0163   14.0794   0.9909   0.5680   1.0000
```

```
P =  
  9.8228   13.3305   1.8242   0.3091   1.0000  
 13.8238   13.0680   1.7449   0.3608   1.0000
```

Es ist festzustellen, dass in jeder Generation eine vollständig neue Population erzeugt wird und das Alter der Individuen 1 nie überschreitet. Individuen der alten Population werden vollständig vergessen. Ein ähnliches Ergebnis ist auch bei Verwendung nur einer Schrittweite zu beobachten.

Behandlungen der Nebenbedingungen

Nach dem Test der genetischen Operatoren der ES kann nun überprüft werden, ob die ES auch bei Vorhandensein von Nebenbedingungen korrekt arbeitet. Dazu wird eine komplexere (20,140)-ES auf der 2-dimensionalen Sphere Funktion angewendet, die eine Reihe an Nebenbedingungen beachten muss. Die einzelnen Arten der Nebenbedingungen sollen wieder getrennt voneinander untersucht werden.

Der erste Test prüft das korrekte Verhalten der ES, falls obere und untere Schranken zu beachten sind. Dazu wird $u = [60, 45]$ als obere und $l = [2, 5]$ als untere Schranke definiert, sowie die Individuen im Intervall $[20, 30]$ zufällig initialisiert. Die ES muss dann die Lösung $[2, 5]$ zurückgeben.

Das Ergebnis des ES-Laufs nach 100 Generationen ist $[2.0013, 5.0002]$, wenn maximiert wird, wird sogar exakt das erwartete Ergebnis zurückgegeben. Die Abbildung 4.4 stellt den Verlauf dieser Optimierung ergänzend graphisch dar.

Der Test für die linearen Nebenbedingungen wird auf der gleichen Problemstellung durchgeführt. Als lineare Nebenbedingung wird $x_1 + x_2 \leq 0.2$ gewählt, so dass die optimale Lösung hier $[0.1, 0.1]$ ist.

Die ES liefert für diese Problemstellung kein Ergebnis, sondern verharrt nach zwei Generationen scheinbar in einer endlos-Schleife.

Der Grund konnte nach den Erkenntnissen aus dem Fehler in der Mutation ($\rho = 1$) schnell gefunden werden. Erneut wird hier auf Operatoren zugegriffen, die nur dann das gewünschte Verhalten gewährleisten, wenn auf Matrizen gearbeitet wird. Wird nur eine lineare Nebenbedingung angegeben (dann liegt erneut nur ein Zeilenvektor vor), verursachen diese Operatoren nicht das gewünschte Ergebnis. Die Lösung dieses Problems bildet wieder eine bedingte Anweisung, die je nach dem, ob nur eine oder aber mehrere linearer Nebenbedingungen vorliegen,

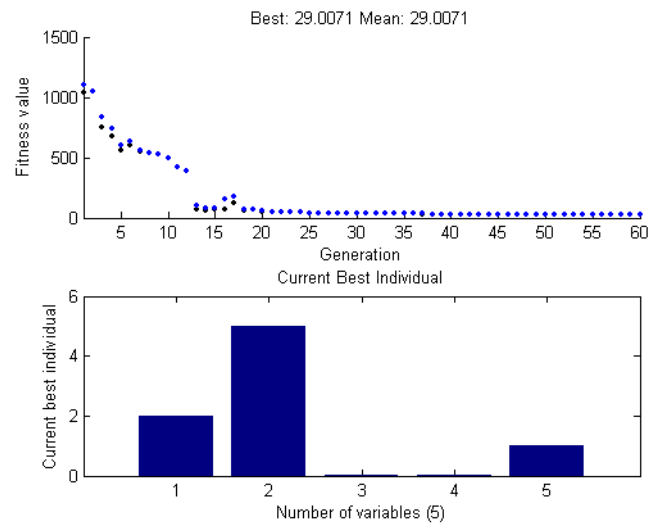


Abbildung 4.4: Verlauf der ES unter Beachtung von Nebenbedingungen

entsprechenden Code abarbeitet.

Nach dieser Modifikation werden die linearen Nebenbedingungen beachtet und das Problem korrekt gelöst. Für die betrachtete Problemstellung wird nach 100 Generationen die Lösung $[0.099808, 0.10019]$ gefunden. Zusätzlich wird der Test analog auf zwei linearen Nebenbedingungen durchgeführt. Auch dieser Test liefert das erwartete Ergebnis.

Als letztes muss noch die korrekte Behandlung nicht-linearer Nebenbedingungen untersucht werden. Als nichtlineare Nebenbedingung wird die Funktion $g = -x_1 - x_2 + 0.2$ gewählt. Tatsächlich ist diese Nebenbedingung zwar linear, für den Test ist dies aber unerheblich, da die Funktion durch Matlab ausgewertet wird und die Methode zur Behandlung der nicht-linearen Nebenbedingungen nur auf dieser Auswertung ansetzt. Die optimale Lösung, die von der ES ausgegeben werden muss, ist somit erneut $[0.1, 0.1]$.

Hier wird das Ergebnis annähernd erreicht. Der Test wurde erfolgreich auch für zwei nichtlineare Nebenbedingungen durchgeführt, um zu zeigen, dass auch mehrere nichtlineare Nebenbedingungen korrekt beachtet werden.

Die Verifikation hat für gegebene Eingaben gezeigt, dass (ggf. nach Korrekturen) die erwarteten Ergebnisse ausgegeben werden. Wie schon einleitend beschrieben, können solche Tests nicht alle Fehlerquellen aufdecken. Riedemann (1997) schreibt dazu: „Allgemeines Ziel des Programmtests ist es, die Qualität von Softwaresystemen (...) zu erhöhen.“ In der Regel wird ein Test nie zum Auffinden aller Fehler in einem System führen.

4.2.3 Validierung

Die Validierung überprüft nun abschließend, ob die Implementierung das Ergebnis liefert, das von ihr erwartet wird. Wie bereits bei der Verifikation gestaltet sich diese Überprüfung als schwierig, weil die ES eine stochastische Methode ist, die stets zu unterschiedlichen Ergebnissen führt.

Hier werden zwei Tests durchgeführt: Im ersten Teil wird eine Charakteristik der ES herangezogen. Der Graph der logarithmischen Fitness über die Zeit bei der Sphere-Funktion weist im Fall der ES einen linearen Verlauf auf, die auch unsere Implementierung liefern muss (Bartz-Beielstein, in einem Gespräch). Dazu wird die Implementierung auf der 50-, der 100- und der

500-dimensionalen Sphere-Funktion angewendet. Für die ersten beiden Problem-Instanzen werden 1000 Generationen gewählt, für die letzte Problem-Instanz schließlich 10000 Generationen. Für alle Tests werden die empfohlenen Standardeinstellungen der ES (vgl. Kapitel 3) verwendet.

Die Abbildung 4.5 zeigt den zeitlichen Verlauf dieser Optimierungen auf einer logarithmischen Skala, der in allen drei Fällen annähernd linear ist. In den ersten Generationen ist zudem in allen drei Fällen zu erkennen, dass eine Anpassung der Schrittweiten erfolgt. Der Graph verläuft

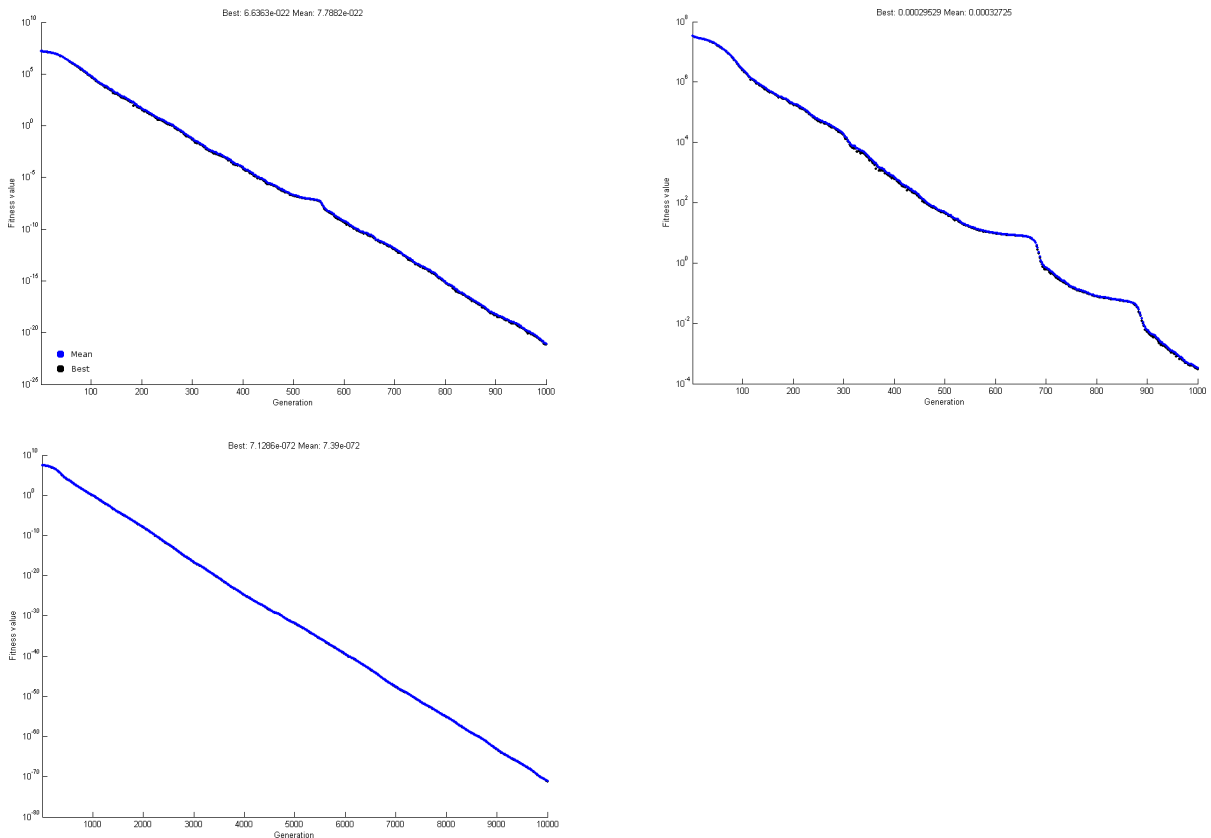


Abbildung 4.5: Verlauf der Optimierung für 50 (links), 100 (rechts) und 500 (unten) Dimensionen

zunächst in den ersten ca. 20 Generationen eher flach. Hier werden noch die durch Initialisierung erzeugten Schrittweiten verwendet, die nicht optimal an die Problemstellung angepasst sind. Während dieser ca. 20 Generationen werden die Schrittweiten entsprechend der Problemstellung eingestellt, so dass ab dieser Generation eine Beschleunigung zu erkennen ist.

In einem weiteren Test wird unsere ES mit einer anderen Implementierung verglichen. Diese Implementierung stammt von Hans-Georg Beyer (ehemals Lehrstuhl *Algorithm Engineering* an der Universität Dortmund) und wird im Folgenden mit Beyer-ES bezeichnet.

Die Beyer-ES codiert alle Eingaben im Code, in dem die Funktion $f = \sum_{i=1}^{20} i \cdot (x-i)^2$ minimiert wird und die Parameter wie folgt eingestellt sind:

- $\mu = 5$
- $\lambda = 20 \quad (\nu = \frac{\lambda}{\mu} = 4)$
- $\rho = \mu = 5$
- $n_{\sigma} = 1$

- $\tau = 0,3$ ($c_\tau = \tau \cdot \sqrt{n} = 0,3 \cdot \sqrt{20} \approx 1,34$)
- Kommaselektion
- Initial Schrittweite: 0.02
- Diskrete Rekombination für die Objekt- und intermediäre für die Strategiekomponente
- Maximale Anzahl der Generationen: 1000
- Minimale Schrittweite, bei deren Unterschreitung abgebrochen wird: 0.00001

Analog wird unsere ES eingestellt. Dennoch kann nicht erwartet werden, dass das exakt gleiche Ergebnis von beiden Implementierungen geliefert wird, da es sich um stochastische Verfahren handelt. Trotzdem sollten sich beide Verfahren ähnlich verhalten. Nach Durchführung beider Läufe wird tatsächlich ein ähnliches Ergebnis geliefert. Der Verlauf der Fitness sowie der Schrittweiten über die Zeit beider Implementierungen ist in Abbildung 4.6 dargestellt, wobei rechts die Ergebnisse der Beyer-ES abgetragen sind. Es fällt auf, dass die Optimierung bei beiden

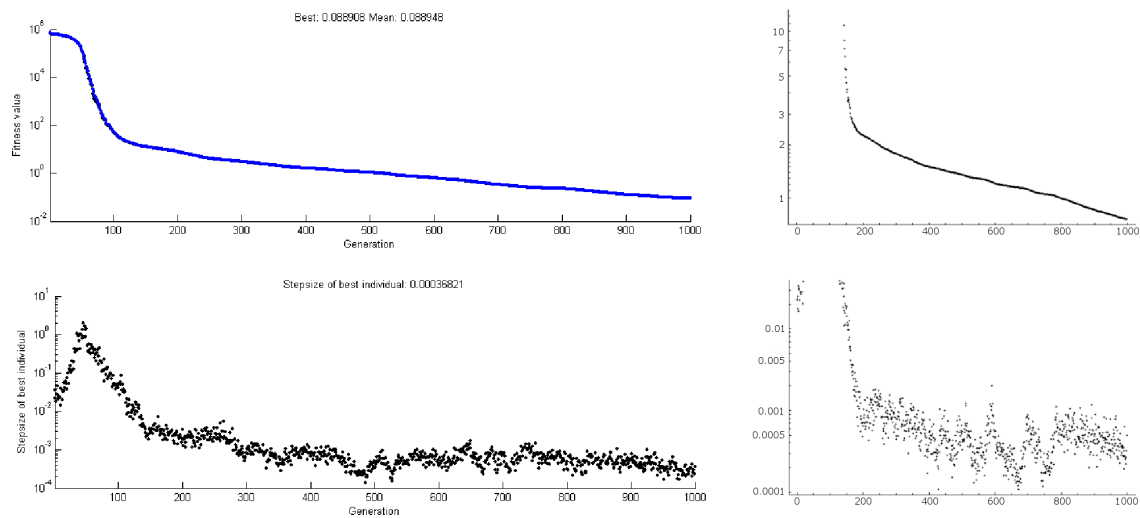


Abbildung 4.6: Vergleich zweier unterschiedlicher Implementierungen der ES

Implementierungen annähernd gleich verläuft. Die zu klein initialisierte Schrittweite werden zu Beginn schnell korrigiert und nehmen anschließend mit zunehmenden Fortschritt ab. Auch der Verlauf der Fitness über die Zeit ist bei Beiden sehr ähnlich. Zu Beginn der Optimierung nehmen die Fitnesswerte pro Generation stark ab, zum Ende hin verlangsamt sich dieser Effekt. Ein ähnlicher Test wird nochmals mit beiden Implementierungen durchgeführt, wobei nun anstelle der Kommaselektion die Plusselektion verwendet wird. Erneut werden die Ergebnisse graphisch in Abbildung 4.7 dargestellt. Auch hier ähnelt das Verhalten beider Implementierungen sehr.

Die Validierung der Funktion `randc` gestaltet sich eher einfach. Ist diese Funktion korrekt implementiert, so müssten sich die Zufallszahlen entsprechend der Cauchy-Dichte verteilen.

Für den Test werden fünf Millionen Zufallszahlen von der Funktion `randc` erzeugt, die anschließend in einem Histogramm abgetragen werden. Dem wird die tatsächliche Dichte der Cauchyverteilung gegenüber gestellt.

Der rot dargestellte Graph in der Abbildung 4.8 stellt die Dichte der Cauchyverteilung dar, die annähernd exakt das Histogramm umrandet. Die erzeugten Zufallszahlen sind somit cauchyverteilt.

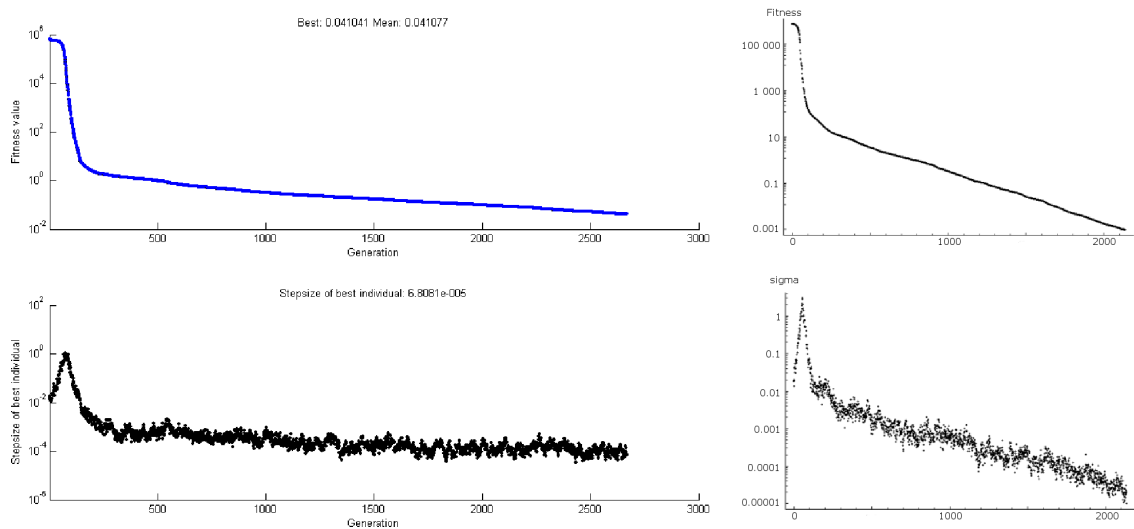


Abbildung 4.7: Vergleich zwei unterschiedlicher Implementierungen der ES

Eine andere Methode, die überprüft, ob eine Probe einer bestimmten Verteilung folgt, ist der Kolmogorov-Smirnov Test (KS-Test). Der KS-Test ist bereits in der Matlab Statistics Toolbox implementiert und prüft, ob eine Probe einer Normalverteilung folgt. Über einen erweiterten Aufruf kann auch überprüft werden, ob eine Probe einer beliebigen (hier: Cauchy) Verteilung entnommen ist. Dazu muss dem KS-Test neben der Probe noch ein Vektor übergeben werden, der Zahlen enthält, die aus einer Cauchy-Verteilung stammen. Dieser Vektor kann aber nicht ohne weiteres erzeugt werden, da Matlab keinen Zufallszahlengenerator für cauchyverteilte Zufallszahlen kennt und unser Generator vorab nicht als fehlerfrei vorausgesetzt werden kann. Deshalb ist der KS-Test hier nicht zum Einsatz gekommen.

4.3 Implementierung einer einfachen stochastischen Suche

Die Implementierung der einfachen stochastischen Suche erfolgt analog zur Implementierung der $(\mu, \kappa, \lambda, \rho)$ -Evolutionstrategie und wird hier daher nur kurz beschrieben. Einer Erweiterung der Funktion `gaoptimset.m` ist nicht mehr erforderlich, da die zusätzlichen Parameter für die random-walk Wahrscheinlichkeit, sowie der für die Schrittweitenanpassung als Funktionsargumente übergeben werden. Die Anpassung der Schrittweiten erfolgt analog zum GA*, in dem Initialschrittweiten vergeben werden, die stetig abnehmen.

Erneut wird über den Parameter `'MutationFcn'` in der `gaoptimset.m` eingestellt, dass die RII durchgeführt werden soll, und zwar in dem als Mutationsfunktion `@stepRII` angegeben wird. Die Populationsgröße wird auf eins gesetzt, so dass den Parametern ρ und `EliteCount` der Wert null zugewiesen werden muss, da sonst die Funktion `validate` einen Fehler ausgeben würde.

Der Funktion `stepRII` wird der einzige Lösungspunkt übergeben, die zufällig entsprechend der aktuellen Schrittweite durch den Suchraum bewegt wird. Ist die so erzeugte neue Lösung besser als die Alte, so wird sie beibehalten. Sie wird auch dann beibehalten, wenn in der aktuellen Iteration ein random-walk durchgeführt wird. Die Wahrscheinlichkeit für einen solchen random-walk wird vom Benutzer über ein Funktionsargument übergeben.

Zuletzt muss noch die Schrittweite angepasst werden. Dies erfolgt über den Ausdruck

```
scale = scale - shrink * scale * state.Generation / options.Generations;
```

Die Schrittweite `scale` wird linear reduziert, wobei die Steigung von Benutzer über das Funktionsargument `shrink` eingestellt werden kann. Eine Wert `shrink = 0` führt dazu, dass die

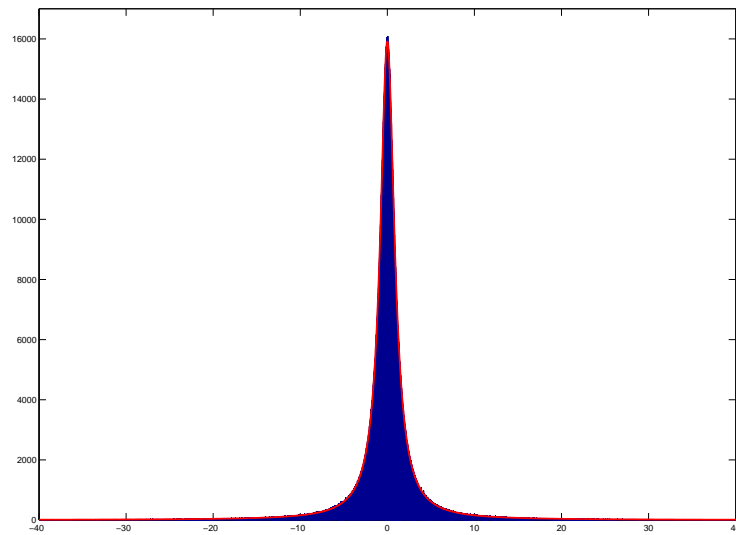


Abbildung 4.8: Histogramm (blau) und Dichte (rot) des Generators bzw. der Cauchyverteilung

Schrittweiten konstant bleiben; im Fall `shrink = 1` nehmen die Schrittweiten linear ab, bis in der letzten Iteration schließlich eine null Schrittweite vorliegt.

5 Experimente

Dieses Kapitel widmet sich den experimentellen Untersuchungen der EA. Theoretiker werden sich hier die Frage stellen, warum formale Systeme, wie es Algorithmen nun mal sind, experimentell untersucht werden.

Gründe für experimentelle Untersuchungen sind unter anderem die teils unvollständigen Theorien, die sogar zu irreführenden Ergebnissen führen können. Als Beispiel kann das Nelder-Mead Simplexverfahren (vgl. Kapitel 3) angeführt werden, dass zu den in der Praxis erfolgreich eingesetzten Verfahren zum Lösen unterschiedlicher (hochdimensionaler) Optimierprobleme gehört. Die theoretische Aussage ist aber eine ganz andere: Das Nelder-Mead Simplexverfahren konvergiert demnach nur dann sicher zum Minimum, wenn das Problem eindimensional ist. Für zweidimensionale Probleme kann gerade noch die Konvergenz zu einem nicht-stationären Punkt gezeigt werden. Für ein Optimierverfahren ist dies eine sicherlich katastrophale Aussage, die der tatsächlichen Leistung des Simplexverfahrens nicht gerecht wird.

Ein anderer Nachteil theoretischer Aussagen sind die Annahmen, unter denen sie Gültigkeit haben. Die Konvergenzaussagen im Kapitel 3 waren der Art: „*Ist das Problem streng konvex und stetig differenzierbar, so konvergiert das Verfahren zum Minimum.*“. Für praktische Problemstellung, in denen die Probleme meistens nicht konvex und nicht stetig differenzierbar sind, hat eine solche Aussage nur wenig Bedeutung.

Experimentelle Untersuchungen treten diesen Problemen entgegen. Sie können auf einfache Weise Aussagen für jede Probleminstanz (insbesondere auch nicht konvexe und nicht stetig differenzierbare Probleme) liefern und erlauben die Betrachtung des *average-case*-Falls, der in der Theorie, aufgrund der unbekanntenen Verteilung der Eingaben, meistens nur sehr schwer zu untersuchen ist. So kann sich auch die Theorie der Erkenntnisse aus den Experimenten bedienen, um neue Theorien aufzustellen, oder vorhandene zu ergänzen.

Ziel der hier durchgeführten Experimente wird es sein, unterschiedliche direkte Suchverfahren auf den Testfunktionen aus Kapitel 2 zu vergleichen. Es ist klar, dass mit unterschiedlichen Parametereinstellungen (Parametrisierungen) der Algorithmen praktisch jedes Ergebnis aufgezeigt werden kann. So wäre es sogar möglich, einen bestimmten Algorithmus gegenüber den Anderen besonders hervorzuheben. Dies soll aber nicht Zweck dieser Arbeit sein.

Um einen objektiven Vergleich zu ermöglichen, wird deshalb vorab eine gute Parametrisierung für alle Verfahren gesucht. Warum an dieser Stelle nur von einer guten, und nicht der optimalen Parametrisierung gesprochen wird, wird sich im weiteren Verlauf dieses Kapitels noch herauskristallisieren. Ist eine gute Parametrisierung für die Optimierverfahren ES, cauchy-ES, GA und RII ermittelt, kann anschließend ein objektiver Vergleich aller Verfahren durchgeführt werden. Zuvor wird aber ein Überblick über die experimentelle Untersuchung der EA gegeben.

5.1 Eine „Historie“ experimenteller Untersuchungen der EA

Bei Betrachtung unterschiedlicher experimenteller Untersuchungen fällt auf, dass mit der Zeit unterschiedliche Techniken angewendet wurden. In frühen Arbeiten wurden häufig nur der Mittelwert der Lösung oder die (mittlere) Anzahl der Funktionsauswertungen zum Erreichen einer bestimmten Lösung betrachtet. Aus heutiger Sicht ist aber klar, dass das Mittel nicht ausreichend ist, um die Leistung unterschiedlicher Algorithmen zu bewerten. Ausreißer in der

Messreihe können beispielsweise das Mittel verfälscht. Die in dieser Zeit betrachteten Algorithmen besaßen nur weniger Parameter (z.B. $(1 + 1)$ -ES), so dass die Parametrisierung hier nur ein kleines Problem darstellte.

In einer zweiten *Epoche* wurden statistische Methoden verwendet. Neben dem Mittelwert gaben hier nun Standardabweichung und Median weitere Anhaltspunkte über die Performance unterschiedlicher Algorithmen. Weiterhin wurden die Parametrisierung der Algorithmen sowie die Anzahl der Funktionsauswertungen scheinbar willkürlich gewählt. Einerseits hängt aber die Performance eines Algorithmus von seiner Parametrisierung ab, andererseits verhalten sich alle Algorithmen schlecht, wenn die Anzahl der Funktionsauswertungen besonders klein gewählt wird, bzw. besonders gut, wenn die Zahl extrem hoch ist.

Ein neuer Ansatz versucht die bisherigen Probleme weiter zu lösen, in dem gute Parametrisierungen gesucht und weitere statistische Methoden verwendet werden, um die Algorithmen sicher bewerten zu können. Um in der geschichtlichen Metapher zu bleiben, kann der im Folgenden vorgestellte Ansatz als *Revolution* in der experimentellen Untersuchung der EA angesehen werden. Dieser Ansatz wird im folgenden Abschnitt genauer erklärt, dessen Titel an die Arbeit von Bartz-Beielstein (2006) angelehnt ist.

5.2 Der neue Experimentalismus

Dieser Abschnitt wird sich, wie auch sein Titel, stark an (Bartz-Beielstein, 2006) orientieren und in drei Unterabschnitte gliedern. Beginnend mit einigen Vorüberlegungen, wird anschließend beschrieben, wie die Steuerparameter eines Optimierverfahrens eingestellt werden können und schließlich wie unterschiedliche Verfahren miteinander zu vergleichen sind.

5.2.1 Vorüberlegungen

Bevor ein Experiment durchgeführt werden kann, sind zuvor einige Vorüberlegungen notwendig. Soll der Ausgang eines Experiments bewertet werden, so ist ein geeignetes Bewertungsmaß zu definieren. Die sinnvolle Bewertung jedes Optimierverfahrens wird aber scheitern, wenn die maximale Anzahl der Funktionsauswertungen zu groß oder aber zu klein gewählt wird. Deshalb ist es vor Beginn der Experimente notwendig, eine geeignete Anzahl von Funktionsauswertungen zu bestimmen. Weil das Ergebnis eines Optimierverfahrens mit seiner Initialisierung und der Abbruchbedingung zusammenhängt, müssten diese auch in eine Vorüberlegung mit einbezogen werden. Der CEC 2005 Testfunktionensatz hat aber Initialisierung und Abbruchbedingung klar definiert (vgl. auch Kapitel 2), so dass es hier keiner weiteren Überlegungen bedarf. Die hier dargestellten Punkte betreffen nur die experimentelle Suche nach guten Parametrisierungen. Für den späteren Vergleich unterschiedlicher Optimierverfahren sind keine vorexperimentellen Planungen notwendig, weil der CEC 2005 Testfunktionensatz alle notwendigen Komponenten der Experimente klar definiert hat.

Bewertungsmaß

Sollen unterschiedliche Optimierverfahren oder Parametrisierungen miteinander verglichen werden, so ist es notwendig, ein geeignetes Bewertungsmaß zu definieren. Im Fall stochastischer Optimierverfahren ist die Bestimmung eines adäquaten Maßes nicht trivial. Solche Verfahren erzeugen bei mehrfacher Anwendung mit gleicher Parametrisierung stets unterschiedliche Lösungen (ausgenommen, es wird stets das gleiche random-seed verwendet), so dass solche Verfahren mit gleicher Parametrisierung mehrfach ausgeführt werden müssen, um aussagekräftige Ergebnisse zu erhalten.

In die Wahl des Bewertungsmaßes muss das Ziel der experimentellen Untersuchungen mit einbezogen werden. Soll die Effektivität eines Verfahrens gesteigert werden, so sind sicherlich andere Bewertungsmaßstäbe anzusetzen, als bei der Steigerung der Effizienz. Die Effektivität behandelt die Frage, wie gut sich ein Verfahren oder eine Parametrisierung auf unterschiedlichen Probleminstanzen verhält, die Effizienz, wie gut es oder sie auf einer bestimmten Probleminstanz ist. In dieser Arbeit wird für jede einzelne Probleminstanz eine optimale Parametrisierung gesucht, so dass hier die Effizienz behandelt wird.

Bartz-Beielstein (2006) stellt unterschiedliche Bewertungsmaßstäbe vor, von denen hier zwei zum Einsatz kommen werden. Zur Bestimmung guter Parameter werden die Optimierverfahren mit gleicher Parametrisierung r -fach wiederholt, so dass pro Parametrisierung r Werte zu Verfügung stehen. Der beste dieser r Werte gibt dann die Güte der betrachteten Parametrisierung an. Dieses Maß wird in den Tabellen im Abschnitt 5.3 mit *BEST* abgekürzt. Im Abschnitt über die Parameteroptimierung (SPO) wird zudem genauer motiviert, warum dieses Maß besonders adäquat ist.

Als ein weiterer Bewertungsmaßstab kann die *run-length distribution* (RLD) herangezogen werden. Sie ist insbesondere dann interessant, wenn die Anzahl der Funktionsauswertungen untersucht werden soll, die ein Optimierverfahren benötigt, um eine bestimmte Fitness zu erreichen. Aus Kapitel 2 ist bekannt, dass Auswertungen der Fitnessfunktion teuer sein können und man großes Interesse daran hat, die Anzahl zu minimieren.

Zur Berechnung der RLD wird ein Algorithmus r -mal wiederholt und es wird die Anzahl der Funktionsauswertungen $t_{run}(i)$ gemessen, die der i -te Lauf zum Auffinden einer bestimmten Fitness (gewöhnlich des Minimums) benötigt. Bei Anwendung der RLD muss zwangsläufig eine obere Grenze für die Anzahl der Funktionsauswertungen gezogen werden, damit nicht erfolgreiche Läufe nach einer gewissen Zeit abgebrochen werden. Dann wird ihnen die Rechenzeit ∞ zugeordnet. Mit Hilfe der ermittelten Rechenzeiten ist es möglich, eine Wahrscheinlichkeitsverteilung

$$Pr(t_{run}(j) \leq t) = \frac{|\{j \mid t_{run}(j) \leq t\}|}{r}$$

zu berechnen, die für eine Anzahl Funktionsauswertungen relativ angibt, wie viele Läufe des Optimierverfahrens das Minimum gefunden haben. Weil sich Verteilungen graphisch leicht darstellen lassen, kann die RLD zudem verwendet werden, um anschaulich einen Vergleich zwischen unterschiedlichen Parametrisierungen oder unterschiedlichen Verfahren zu ermöglichen.

Maximale Anzahl der Funktionsauswertungen

Die maximale Anzahl erlaubter Funktionsauswertungen ist maßgeblich für das Ergebnis eines Experiments entscheidend. Wird die Anzahl zu groß gewählt, so wird jedes Optimierverfahren bzw. jede Parametrisierung zu einem besonders guten Ergebnis kommen. Dann ist es aber nicht mehr möglich, klar zu entscheiden, welches Verfahren oder welche Parametrisierung besser ist. In einem solchen Fall spricht man von einem *ceiling-Effekt*.

Der Gegensatz dazu ist der *floor-Effekt*. Dieser tritt ein, wenn die Anzahl erlaubter Funktionsauswertungen zu gering gewählt wird. Dann werden erneut alle Verfahren bzw. Parametrisierungen ähnliche Werte liefern; diesmal aber ähnlich schlecht. Die Folge ist, dass aus den so gewonnenen Daten erneut nur wenig Information extrahiert werden kann, und die Experimente somit nutzlos sind.

Deshalb ist es für den Verlauf der Experimente entscheidend, eine geeignete Anzahl erlaubter Funktionsauswertungen zu bestimmen. Eine Entscheidungshilfe können zum einen die oben eingeführten RLD liefern, die die Auswirkung unterschiedlicher Rechenzeiten graphisch darstellen können. Im weiteren Verlauf dieser Arbeit wird eine andere Methode Anwendung finden, die in Schaffer et al. (1989) vorgestellt worden ist. Sie ermitteln die minimale Anzahl der Funkti-

onsauswertungen, so dass mindestens 10% der Konfigurationen, mindestens in jedem zweiten Lauf, die Lösung finden. Diese Anzahl ist dann gerade so groß, dass ceiling-Effekte verhindert werden.

5.2.2 Optimierung der Steuerparameter

Dieser Abschnitt beschäftigt sich mit der Optimierung der Steuerparameter eines Optimierverfahrens und wird einen kurzen Einblick in die Thematik geben. Ausführliche Beschreibungen finden sich in (Bartz-Beielstein, 2003, 2004, 2005, 2006).

Eine Optimierung der Steuerparameter ist immer empfehlenswert, weil gut eingestellte Optimierverfahren den Bedarf an Zeit und an Funktionsauswertungen stark senken können. Für den Vergleich unterschiedlicher Optimierverfahren ist sie sogar zwingend erforderlich, um einen fairen Vergleich unterschiedlicher Verfahren zu ermöglichen.

Im Folgenden wird ein Lauf des Optimierverfahrens als Experiment betrachtet, welches sich aus einem Problemdesign \mathcal{D}_P und einem Algorithmendesign \mathcal{D}_A zusammensetzt. \mathcal{D}_A beinhaltet eine Menge an Vektoren, von denen je einer eine Parametrisierung des Suchalgorithmus angibt. Das Algorithmendesign für die im Kapitel 3 definierten ES und GA kann offensichtlich unendlich viele Designpunkte beinhalten. \mathcal{D}_P enthält Informationen zum Problem. Dazu gehören das Optimierungsproblem, die Dimensionalität, die Nebenbedingungen, die Methode zu Initialisierung und die Abbruchbedingung.

Ziel wird es sein, die Effizienz der Optimierverfahren auf einer bestimmten Problem Instanz zu steigern. Formal wird deshalb für ein bestimmtes $x_p \in \mathcal{D}_P$ ein $x_a^* \in \mathcal{D}_A$ gesucht, welches auf x_p ein gutes Ergebnis liefert. Dazu werden r Experimente für ein x_p und ein x_a durchgeführt, die je eine stochastische Ausgabe $Y(x_p, x_a)$ liefern. Die Notwendigkeit der Wiederholung gleicher Experimente mit unterschiedlichen *random-seeds* wurde bereits motiviert.

Es wird nicht möglich sein, die Experimente für alle $x_a \in \mathcal{D}_A$ durchzuführen, weil ein aussagekräftiges Design \mathcal{D}_A zwangsläufig unendlich viele Designpunkte enthalten wird, beispielsweise wenn der Faktor ν der ES in einem Intervall $[0.5, 30]$ untersucht wird. Somit ist es notwendig Methoden zu verwenden, die in der Lage sind, mit wenigen Experimenten einen fast optimalen Designpunkt $x_a^* \in \mathcal{D}_A$ zu bestimmen. Solche Methoden sollen nun kurz vorgestellt werden.

Statistische Versuchsplanung

Die statistische Versuchsplanung (engl. Design of Experiments, DOE) betrachtet die Eckpunkte des Designraums \mathcal{D}_A . Jeder der k Faktoren wird in einem Intervall $[-1, +1]$ skaliert, so dass ein k -dimensionaler Hyperquader entsteht, dessen 2^k Ecken die Designpunkte $x_a \in \mathcal{D}_A$ darstellen, mit denen die Experimente durchgeführt werden. DOE verwendet ein Regressionsmodell

$$y = X \cdot \beta + \epsilon$$

besteht aus einem Spaltenvektor y mit den Ergebnissen aus den Experimenten, dem Vektor ϵ der Fehlerterme und dem Vektor β der Parameter. X ist eine Matrix, deren erste Spalte eine Dummyvariable mit dem Wert $\mathbf{1}$ ist. Die restlichen Einträge entsprechen dem gewählten Algorithmendesign. Somit hat X die folgende Gestalt:

$$X = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,(q-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{k,1} & \dots & x_{k,(q-1)} \end{pmatrix}.$$

Ziel ist es, ein $\hat{\beta}$ zu bestimmen, dass die Fehlerquadrate (engl. mean square error, MSE) minimiert. Gilt für das Matrixprodukt $X^T X = n \cdot \mathbf{I}$, so ist die Bestimmung besonders einfach, weil

sich dann die Inverse $(X^T X)^{-1}$ einfach berechnen lässt. Dann gilt:

$$\hat{\beta} = (X^T X)^{-1} X^T y .$$

Die Richtung des steilsten Abstiegs wird dann durch $-\hat{\beta}$ gegeben, entlang der, ein besseres Design gesucht werden kann. Eine tiefere Beschreibung dieses Verfahrens findet sich in (Bartz-Beielstein, 2003). Der Vollständigkeit halber soll hier nur noch erwähnt werden, dass sich das klassische DOE aus drei Schritten zusammensetzt. Das oben beschriebene Design wird als *faktorielles* Design bezeichnet, und hat den Nachteil, dass bei großem k eine hohe Anzahl an Experimenten durchzuführen ist. *Teilfaktorielle* Designs können deshalb in einer ersten Phase verwendet werden, um signifikante Faktoren zu ermitteln. Diese Designs benötigen wesentlich weniger Experimente, können aber nicht alle Abhängigkeiten zwischen Variablen aufzeigen.

Beide Designs können nur lineare Effekte aufdecken. Deshalb werden in der letzten Phase *zentral zusammengesetzte Pläne* (engl. central composite designs, CCD) verwendet werden, die das faktorielle Design um einen Punkt im Zentrum des Designraums und um $2 \cdot k$ weitere Punkte auf den Achsen erweitern. Anstelle eines linearen Regressionsmodells kann dann ein quadratisches Modell Anwendung finden.

Abschließend sollten an dieser Stelle noch die *one-factor-at-a-time* Methode erwähnt werden. Diese Methode variiert stets nur einen Faktor und hält alle anderen konstant. So kann ausschließlich der Einfluss nur einer Variable auf das Ergebnis betrachtet werden. Insbesondere können so keine Wechselwirkungen verschiedener Variablen auf die stochastische Ausgabe y aufgedeckt werden. Eine solche Herangehensweise sollte daher nur dann gewählt werden, wenn sich die Faktoren gegenseitig nicht beeinflussen und sich additiv verhalten. Das kann für die Parameter der Optimierungsalgorithmen jedoch nicht vorausgesetzt werden, so dass diese Methoden nicht angewendet werden sollten.

Design and Analysis of Computer Experiments

Die statistische Versuchsplanung geht von Annahmen aus, die in der Optimierung nicht erfüllt sind. Das Regressionsmodell $y = X\beta + \epsilon$ setzt voraus, dass y und ϵ die selbe Varianz haben. In der Optimierung hängt diese aber eher von der Parametrisierung x_a ab. Als ein weiterer Kritikpunkt kann aufgeführt werden, dass Designpunkte nur an den Rändern des Designraums platziert werden. So werden aber im vornherein interessante Punkte im Innern des Designraums ausgeschlossen.

Design and Analysis of Computer Experiments (DACE) ist ein anderer Ansatz, der das Regressionsmodell $y = X\beta + \epsilon$ durch die Summe aus Regressionsmodell \mathcal{F} und stochastischem Prozess Z ersetzt. Insgesamt wird so ein Modell

$$Y(x) = \mathcal{F}(\beta, x) + Z(x)$$

betrachtet. Das Regressionsmodell \mathcal{F} wird durch q Funktionen definiert und unterscheidet unter Polynomen der Ordnung 0, 1 und 2. Es erlaubt so den Einsatz von konstanten, linearen und quadratischen Modellen, die $q = 1$, $q = k + 1$ oder $q = \frac{k^2 + 3k + 2}{2}$ Funktionen benötigen, wenn k Faktoren untersucht werden. Im weiteren Verlauf der Arbeit werden Polynome der Ordnung 2 verwendet, so dass sich das Regressionsmodell

$$\mathcal{F}(\beta, x) = \sum_{j=1}^q \beta_j f_j(x) = f(x)^T \beta$$

aus $q = \frac{k^2 + 3k + 2}{2}$ Funktionen $f_1(x) = 1$, $f_2(x) = x_1, \dots, f_{k+1}(x) = x_k$, $f_{k+2}(x) = x_1 \cdot x_1, \dots, f_{2k+1}(x) = x_1 \cdot x_k, \dots, f_q(x) = x_k \cdot x_k$ zusammensetzt.

Der stochastische Prozess Z erlaubt die Modellierung der Messfehler. Anders als Zufallsvariablen, die sich durch ihre Verteilung charakterisieren lassen, hängt der stochastische Prozess von einem zusätzlichen Parameter t ab. Dieser Parameter erlaubt es nun, den Messfehler in Abhängigkeit von der Parametrisierung x zu betrachten.

DACE benötigt einen stochastischen Prozess mit Erwartungswert $E(Z(x)) = 0$. Die Kovarianz zwischen zwei Punkten $x^{(1)}$ und $x^{(2)}$ aus \mathcal{D}_A ist definiert als $V(x^{(1)}, x^{(2)}) = \sigma^2 \mathcal{R}(\theta, x^{(1)}, x^{(2)})$ mit der Prozessvarianz σ^2 und der Korrelationsfunktion

$$\mathcal{R}(\theta, x^{(1)}, x^{(2)}) = \prod_{j=1}^k \mathcal{R}_j(\theta_j, x_j^{(1)} - x_j^{(2)}),$$

die entsprechend den Daten eingestellt wird. Die θ_j geben an, wie lokal die Schätzung ist. Große Werte für θ_j zeigen an, dass nur benachbarte Punkte korreliert sind, wo hingegen kleine Werte auf eine Beziehung weit entfernter Punkte hindeuten.

Lophaven, et al. (2002) stellen sieben unterschiedliche Korrelationsfunktionen vor. Für die folgenden Untersuchungen wird die Gauss-Korrelationsfunktion

$$\mathcal{R}_j(\theta, (x_j^{(1)} - x_j^{(2)})) = e^{-\theta \cdot (x_j^{(1)} - x_j^{(2)})^2}.$$

verwendet.

Im Gegensatz zu DOE geht DACE davon aus, dass auch im Inneren des Designraums interessante Punkte gefunden werden können. Das *Latin Hypercube sampling (LHS)* ist eine Methode, die Punkte im gesamten Designraum platziert. Die so erzeugte Menge der Designpunkte wird als *Latin Hypercube Design (LHD)* bezeichnet.

Zur Erzeugung des LHD unterteilt LHS jede Achse des Designraums in N gleichgroße Teilintervalle, so dass N^k Hyperquader entstehen. Aus der Menge der so erzeugten Hyperquader werden gleichverteilt zufällig N Quader gewählt und aus je einem erneut gleichverteilt zufällig ein Punkt selektiert. So wird eine möglichst gleichmäßige Verteilung derer Punkte im Designraum angestrebt.

Sequentielle Parameter Optimierung

Es ist möglich DACE sequentiell einzusetzen. Die *Sequentielle Parameter Optimierung (SPO)* beginnt mit wenigen Designpunkten, die mit LHS erzeugt werden, um so eine möglichst breite Verteilung im Designraum zu ermöglichen. Nach Durchführung der ersten Experimente wird ein erstes (noch ungenaues) DACE-Modell erzeugt. Dieses Modell wird zur Bestimmung neuer Designpunkte verwendet, mit denen das DACE-Modell von Iteration zu Iteration weiter verbessert wird. Weil das Ziel dieses Prozesses das Auffinden eines guten Designpunktes ist, ist es zweckmäßig die Gleichung

$$I(x) = \begin{cases} f_{min}^n - f(x) & \text{falls } f(x) < f_{min}^n \\ 0 & \text{sonst} \end{cases}$$

mit dem besten Wert f_{min}^n in der n -ten Iteration des sequentiellen Optimierverfahrens zu betrachten. Diese Gleichung gibt dann die Verbesserung an, wenn der Punkt \mathbf{x} als ein neuer Designpunkt gewählt wird. Der exakte Wert $f(\mathbf{x})$ wird aber nicht bekannt sein und wird mit Hilfe des DACE-Modells ermittelt. Deshalb wird alternativ die erwartete Verbesserung betrachtet wird. Sie ist definiert als

$$E(I(x)) = \int_{-\infty}^{f_{min}^n} (f_{min}^n - f(x)) \cdot \phi(f(x)),$$

mit $\phi(\cdot)$ als Dichte der Ungewissheit bezüglich $f(x)$. DACE ermöglicht die Bestimmung des MSE und die Vorhersage $Y(x)$ eines noch nicht ausgewerteten Punktes. Mit beiden Werten ist es möglich, $E(I(x))$ ohne Auswertung des Integrals zu berechnen (Schonlau, 1997). Entsprechend $E(I(x))$ werden solche Punkte x bevorzugt ausgewählt, die in Bereiche des Designraums fallen, in denen entweder besonders gute Punkte erwarten werden, oder in denen eine hohe Ungewissheit herrscht. So wird die Genauigkeit des DACE-Modells in jeder Iteration weiter erhöht und zunehmend bessere Designpunkte gefunden. Ist das Modell und die gefundene Parametrisierung qualitativ ausreichend, oder ist eine maximale Anzahl an Iterationen durchgeführt, wird die Schleife abgebrochen. In ersten Fall ist die gesuchte Parametrisierung gefunden, so dass es keiner weiteren Experimente bedarf. Im zweiten Fall, kann das Algorithmendesign verändert werden. Sind beispielsweise in bestimmten Regionen hohe Fehler (hoher MSE) im Modell vorhanden, die nicht erklärbar sind, so können dort in einem weiteren Lauf zusätzliche Punkte platziert werden, um das Modell besser anzupassen. Hier wird nun auch klar, warum im Verlauf dieses Kapitels nicht von der Suche nach einem optimalen Design die Rede war. In der Praxis wird nicht ausreichend Zeit zu Verfügung stehen, um die optimale Parametrisierung zu finden. Hier muss ein Kompromiss aus Rechenzeit zu Optimierung der Parameter und Rechenzeit zu Optimierung der Problemstellung gefunden werden. Beispielsweise ist es nicht zu begründen, warum die Parametrisierung eines Optimerverfahrens mit großem Zeitaufwand optimiert wird, wenn das Problem auch mit Standardparametrisierung sehr schnell zu lösen ist. Letztendlich ist auch die Frage berechtigt, ob stochastische Suchverfahren *die* optimale Parametrisierung besitzen, oder es nicht vielmehr eine Menge guter Parametrisierungen gibt, die zu ähnlich guten Resultaten führen.

Bartz-Beielstein (2006) beschreibt einen Ablaufplan, der aus 12 Punkten besteht, und vorgibt, wie SPO durchzuführen ist. Hier werden einige Punkte zusammengefasst, so dass die folgende Liste zu Stande kommt:

Vorexperimentelle Planung: In einer vorexperimentelle Planung werden die Ziele der Experimente grob abgesteckt. Hier kann bereits ermittelt werden, wie viele Funktionsauswertungen benötigt werden, um floor- oder ceiling-Effekte zu vermeiden. Außerdem kann in der vorexperimentellen Phase abgeschätzt werden, wie zeitaufwendig die Experimente sind, so dass ggf. die Anzahl der Iterationen der SPO herabgesetzt werden kann.

Formulierung einer Behauptung: In einem zweiten Schritt wird das genaue Ziel der Experimente in einer wissenschaftlichen Behauptung formuliert.

Formulierung einer statistischen Hypothese: Um eine objektive Überprüfung der wissenschaftlichen Behauptung zu gewährleisten wird eine, mit der Behauptung verbundene statistische Hypothese aufgestellt, die mit statistischen Methoden bestätigt oder aber verworfen werden kann.

Spezifikation: Zur Durchführung der Experimente müssen Problem- und Algorithmendesign spezifiziert und ein geeignete Bewertungsmaß definiert werden. Alle drei Komponenten wurden bereits in den entsprechenden Abschnitten dieses Kapitels beschrieben.

Durchführung der Experimente: Die Experimente werden wie beschrieben sequentiell durchgeführt. In jeder Iteration werden neue Punkte ermittelt. Der beste bisher gefundene Punkt wird stets erneut ausgewertet und die Neuen mindestens so oft, wie der bisher Beste. So wird sichergestellt, dass keine Punkte bevorzugt werden. Hält man sich die ES vor Augen, so ist dieses Prinzip ähnlich zur Selektionsprozess einer $(1 + \lambda)$ ES.

Überprüfung der Hypothese: Hier wird die Hypothese aus Punkt 3 erneut betrachtet und entweder verworfen oder aber bestätigt werden. Graphische Methoden, wie Boxplots oder

Histogramme zeigen erste Anhaltspunkt über den Wahrheitsgehalt der aufgestellten Hypothese auf. Statistische Methoden liefern abschließend eine objektive Interpretation. Die Methoden werden im kommenden Abschnitt genauer beschrieben.

SPO-Toolbox

Die SPO Methode ist in einer Matlab Toolbox (Sequential Parameter Optimization Toolbox, SPOT) implementiert (Bartz-Beielstein, 2006). Der Benutzer muss dieser Toolbox lediglich die Faktoren und den Wertebereich, in dem sie untersucht werden sollen, übergeben. SPOT führt daraufhin ein LHS und die ersten Experimente durch. Nach Bestimmung des ersten DACE-Modells werden dann sequentiell neue Punkte erzeugt, bis eine Abbruchbedingung greift. SPOT benötigt eine geeignete Schnittstelle, um die Optimierverfahren mit den ermittelten Designpunkten (Parametrisierungen) aufzurufen. Diese Schnittstelle wird hier aber nicht explizit angegeben, weil sie, wie die gesamte Toolbox, in (Bartz-Beielstein, 2006) ausführlich beschrieben wird. Zudem sind alle hier untersuchten Verfahren in Matlab implementiert, so dass die Anbindung in besonders einfacher Art möglich ist.

5.2.3 Objektive Interpretierung der Ergebnisse

In Abschnitt über die Geschichte der experimentellen Untersuchungen der EA wurde bemängelt, dass oft nicht ausreichende Methoden angewendet wurden, um unterschiedliche Verfahren oder Parametrisierungen zu bewerten. Dies ist eigentlich erstaunlich, weil die Mathematik zahlreiche Methoden zu Verfügung stellt, die eine Bewertung erlauben. Drei solcher Verfahren werden nun vorgestellt. Boxplots und Histogramme sind zwei graphische Methoden. Der t-Test ist ein Verfahren, dass den Mittelwert einer Probe testen kann.

Boxplots

Boxplots sind Diagramme, die eine Darstellung numerischer Daten erlauben. Die Abbildung 5.1 stellt ein solches Diagramm dar, das auf Grundlage 100 Zahlen $z_i \in N_i(0, 1)$ und der Zahl 4 entstand. Wie dieser Abbildung zu entnehmen ist, besteht ein solches Diagramm aus

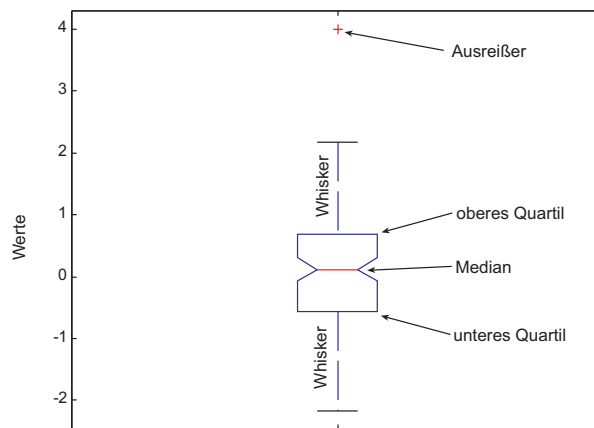


Abbildung 5.1: Boxplot einer Probe

vier Elementen: Dem oder den *Ausreißer(n)*, einem (unteren und oberen) *Whisker* und dem (unteren, mittleren und oberen) *Quartil*.

Die durch das untere und obere Quartil erzeugte Box, enthält 50% der Daten. Ihre Länge wird

durch den Abstand zwischen unteren und oberen Quartil definiert und als *Interquartilsabstand* (engl. interquartile range, IRQ) bezeichnet. Zusätzlich gibt das mittlere Quartil den Median der betrachteten Probe und die Einkerbung in der Box das Vertrauensintervall (die Definition wird im Abschnitt über den t-Test gegeben) für den Mittelwert an. Die Whisker enthalten die Daten, die sich außerhalb des Quartils befinden und definieren so zwei neue Intervalle, welche am oberen bzw. unteren Quartil beginnen und eine maximale Länge von $1,5 \cdot \text{IRQ}$ haben. Ggf. verbleibende Daten werden als Ausreißer bezeichnet.

Histogramme

Während Boxplots nicht die tatsächlichen Daten, sondern nur deren statistischen Eigenschaften angeben, stellen Histogramme die genauen Daten dar. Diese Diagramme tragen auf der Abszisse mögliche Werte einer Probe ab und geben dann über Balken die Anzahl der in der Probe tatsächlich vorhandenen Werte an. Die Abbildung 5.2 gibt ein mögliches Histogramm an. Wie dieser Abbildung entnommen werden kann, eignen sich solche Diagramme ebenfalls zu

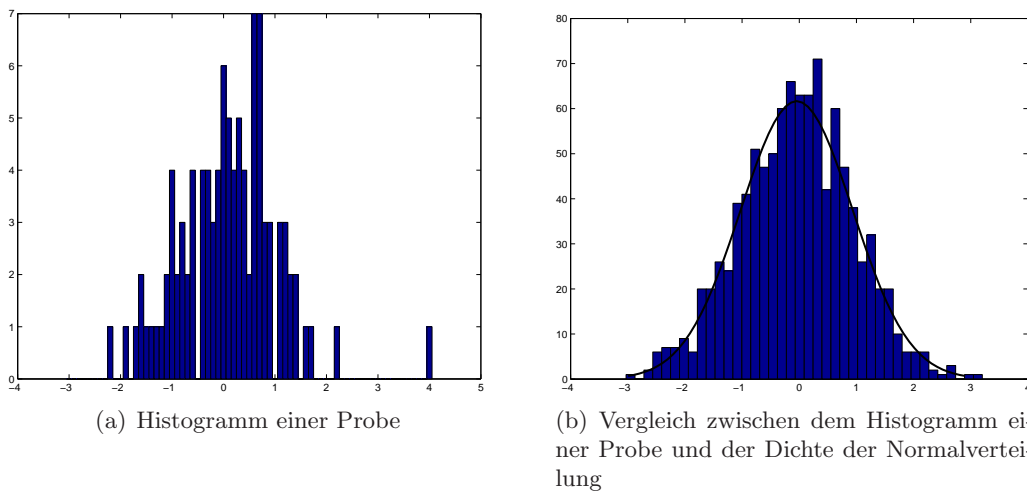


Abbildung 5.2: Histogramm einer Probe

Überprüfung, ob eine Probe einer bestimmten Verteilung folgt.

t-Test

Sollen zwei unterschiedliche Parametrisierungen bewertet werden, so müssen die Verfahren mit diesen Parametrisierungen mehrfach ausgeführt werden. Nach Berechnung der Mittelwerte, könnte der Schein entstehen, dass die Parametrisierung mit geringstem Mittelwert die restlichen Parametrisierungen übertrifft. Diese Folgerung ist aber nicht befriedigend, weil der Mittelwert verfälscht sein kann, wenn z.B. Ausreißer in der Probe vorhanden sind. Deshalb ist es interessanter, den Fehler abzuschätzen, um den der geschätzte Mittelwert von dem tatsächlichen Mittelwert abweicht. Eine Methode, die eine solche Abschätzung erlaubt ist der t-Test, der ausführlich in (Graef, 2004) beschrieben und hergeleitet wird. (Dörfler & Peschek, 1988) liefern einen ersten Einblick in die Thematik der schließenden Statistik.

Der t-Test wird hier verwendet, um zu Prüfen, ob sich zwei unterschiedliche Parametrisierungen in ihrer Leistung unterscheiden. Die mehrfache Ausführung beider Parametrisierungen x_1 und x_2 erlaubt es zwei Mittelwerte \bar{Y}_1 und \bar{Y}_2 zu berechnen. Die Differenz $\delta(\bar{Y}_1, \bar{Y}_2)$ beider Mittelwerte ermöglicht die Beantwortung der Frage, ob die Parametrisierung x_2 der Parametrisierung x_1 überlegen ist. Dazu werden zwei Hypothesen betrachtet. Die Nullhypothese H_0 nimmt an,

dass $\delta(\bar{Y}_1, \bar{Y}_2) = 0$ ist, und somit beide Parametrisierungen die gleiche Leistung erbringen. Die alternative Hypothese (oder kurz Alternative) H_1 geht davon aus, dass $\delta(\bar{Y}_1, \bar{Y}_2) > 0$ ist. Die Alternative sagt aus, dass der Mittelwert \bar{Y}_1 höher ist, als \bar{Y}_2 . Gilt die Alternative, so ist die Parametrisierung x_2 der Parametrisierung x_1 überlegen, weil minimiert wird.

Eine absolut sichere Aussage kann aber auch der t-Test nicht liefern. Dieser Test betrachtet deshalb ein Signifikanzniveau α für die Nullhypothese. Das Signifikanzniveau gibt die Irrtumswahrscheinlichkeit dafür an, dass die Alternative angenommen wird, obwohl die Nullhypothese gilt. Ein typischer Wert ist $\alpha = 0.05$, der auch in dieser Arbeit verwendet wird.

Der t-Test liefert zwei Werte, die die Überprüfung der Gültigkeit einer Hypothese ermöglichen. Das Konfidenzintervall gibt den Bereich an, in dem der Mittelwert mit einer Wahrscheinlichkeit $(1 - \alpha)$ liegt. Dieses Intervall ist natürlich stark vom Wert α abhängig. Deshalb gibt ein weiterer Wert p das kleinste Signifikanzniveau an, das bei der betrachteten Beobachtung die Alternative als signifikant nachweist.

Die dem t-Test zugrunde liegende Teststatistik ist t-Verteilt. Dörfler & Peschek (1988) zeigen, dass deshalb die Proben aus einer Normalverteilung stammen müssen. Die Statistik liefert zahlreiche Tests, mit denen überprüft werden kann, ob Beobachtungen aus einer Normalverteilung stammen. Histogramme erweisen sich hier als besonders hilfreich, weil sie diese Überprüfung graphisch ermöglichen.

5.3 Parameteroptimierung

In diesem Abschnitt werden die Probleminstanzen aus Kapitel 2 betrachtet. Zu jeder Probleminstanz wird eine gute Parametrisierung für die ES, cauchy-ES und GA gesucht. Dazu wird die SPOT verwendet. Für die RII ist muss keine gute Parametrisierung gesucht werden, weil sie in einer besonders einfachen Form mit $p_w = 1$, $\sigma = 1$ und $s = 0$ ausgeführt wird, und somit parameterlos ist.

Die Suche nach guten Parametrisierungen wird jeweils für ES, cauchy-ES und GA an einem ausführlichen Beispiel vorgeführt. Die restlichen Parametrisierungen werden in Tabellenform zusammengefasst.

5.3.1 Anpassung der ES auf die 10-dimensionale sphere-Funktion

Als erste Probleminstanz wird die 10-dimensionale **sphere**-Funktion herangezogen und die Evolutionsstrategie betrachtet, die diese Funktion minimieren soll. Um eine möglichst gute Performance zu erreichen, wird eine gute Parametrisierung der ES gesucht. Zunächst wird aber die genaue Probleminstanz nach Suganthan et al. (2005), die bereits im Kapitel 2 eingeführt wurde, in der Tabelle 5.1 zusammengefasst.

| t_{\max} | n | Init | Term | x_l | x_u | Perf |
|------------|-----|--------|------|-------|-------|------|
| 100000 | 10 | UNIRND | EXH | -100 | 100 | BEST |

Tabelle 5.1: Probleminstanz 10-dimensionale **sphere**-Funktion für die ES

Vorexperimentelle Planung

Das Ziel der Untersuchungen ist das Auffinden einer Parametrisierung, die die Leistung der ES auf der in Tabelle 5.1 dargestellten Probleminstanz verbessert. Es ist anzunehmen, dass so eine Parametrisierung existiert, weil die Standardparametrisierung so gewählt ist, dass sie möglichst auf einer breiten Menge von Problemen zu guten Ergebnissen führt. Sie ist in Bezug

auf die Effektivität gewählt worden. Hier soll nun die Effizienz gesteigert werden. Gesucht wird also eine spezielle Parametrisierung der ES, die auf der 10-dimensionalen **sphere**-Funktion eine besonders gute Leistung aufweist.

In einer vorexperimentellen Planung wird der Zeitbedarf für die Experimente abgeschätzt. Dazu werden einige Designpunkte erzeugt und genau eine Iteration der SPO durchgeführt. Dabei fällt auf, dass aufgrund der hohen Anzahl von Funktionsauswertungen ein extrem hoher Zeitbedarf entsteht, der zum einen ungewollt ist und zum anderen sogar unerwünschte Effekte liefert, wie der Tabelle 5.2 zu entnehmen ist. Alle Designpunkte finden die optimale Lösung, so dass praktisch keine Information erzeugt wird.

Deshalb muss das Problemdesign geändert werden. Dazu wird das Verfahren von Schaffer et al. (1989) verwendet. Mit einer nicht ganz exakten binären Suche wird die Anzahl der Funktionsauswertungen gesucht, die gerade ausreichend sind, damit mindestens 10% der Konfigurationen in mindestens jedem zweiten Lauf das Optimum finden. Diese Anzahl beträgt 2000 und führt neben der Vermeidung von ceiling-Effekten, auch zu wesentlichen Beschleunigung der Experimente. Das so korrigierte Problemdesign wird in Tabelle 5.3 dargestellt.

Wissenschaftliche Behauptung

Für die 10-dimensionale **sphere**-Funktion existiert eine Parametrisierung $x_p^* \in \mathcal{D}_A$ der ES, die bessere Ergebnisse gegenüber der Standardparametrisierung aus Tabelle 3.2 liefert.

Hypothese

Die ES mit Parametrisierung x_p^* findet bei Minimierung der 10-dimensionalen **sphere**-Funktion im Mittel bessere Lösungen, als die Standardparametrisierung. Es gilt also:

$$H_1 : \delta(ES_{x_p^{(0)}} - ES_{x_p^*}) > 0 .$$

Spezifikation des Algorithmendesigns \mathcal{D}_A und des Problemdesigns \mathcal{D}_P

Das Problemdesign ist in Tabelle 5.3 dargestellt. Bei der Definition des Algorithmendesigns wird die Standardparametrisierung als Anhaltspunkt verwendet, so dass der Designraum um diese Standardparametrisierung aufgespannt wird. Der Faktor κ muss gesondert betrachtet werden, weil das LHS, aufgrund der Intervallteilung offensichtlich nicht mit dem „Wert“ ∞ umgehen kann. Es ist aber nicht zwingend notwendig, diesen Wert zu benutzen. Weil die maximale Anzahl der Funktionsauswertungen bekannt ist, kann die obere Schranke des Faktors κ auf 2000 gesetzt werden. So ist sichergestellt, dass die Plus-Selektion für jedes Paar (μ, ν) angewendet werden kann, da selbst im Extremfall $\lambda = \mu \cdot \nu = 1$ bei 2000 Funktionsauswertungen nur 2000 Generationen möglich sind. Gleiches Problem tritt auch bei den qualitativen Faktoren rec_x , rec_σ und n_σ auf, die nur die Werte **dis** oder **int**, bzw. **1** oder **n** annehmen können. Deshalb werden diesen Faktoren über das LHS nur Punkte aus dem Intervall $[0, 1]$ zugewiesen. Über eine bedingte Anweisung wird dann, je nach dem, ob der Faktor einen Wert größer oder kleiner 0.5 zugewiesen bekommen hat, entweder der erste, oder aber der zweite diskrete (oder qualitative) Wert für den Faktor verwendet. Zuletzt müssen die Parameter μ , κ und ρ , gerundet werden, um sicherzustellen, dass die Eingaben für die ES zulässig sind. Die folgende Tabelle

| | μ | ν | c_τ | κ | ρ | $\sigma^{(0)}$ | rec_x | rec_σ | n_σ |
|----------------|-------|-------|----------|----------|--------|----------------|---------|--------------|------------|
| Designpunkt | 10.09 | 13.30 | 6.03 | 626.50 | 12.08 | 2.61 | 0.99 | 0.28 | 0.48 |
| ES-Einstellung | 10 | 13.30 | 6.03 | ∞ | 10 | 2.61 | int | dis | n |

gibt ein Beispiel, wie ein mit LHS erzeugter Designpunkt so transformiert wird, dass er eine korrekte Eingabe für die ES darstellt.

| Y | μ | ν | c_T | κ | ρ | $\sigma^{(0)}$ | ... | repeats | config | seed |
|------|-------|-------|-------|----------|--------|----------------|-----|---------|--------|------|
| -450 | 2.41 | 26.23 | 6.94 | 607.83 | 19.66 | 4.91 | ... | 3 | 1 | 0 |
| -450 | 11.32 | 3.87 | 3.85 | 476.75 | 3.48 | 1.30 | ... | 3 | 2 | 0 |
| -450 | 7.96 | 24.58 | 2.05 | 842.49 | 9.67 | 3.94 | ... | 3 | 3 | 0 |
| -450 | 8.60 | 29.19 | 1.93 | 266.99 | 5.40 | 3.72 | ... | 3 | 4 | 0 |
| -450 | 21.22 | 23.60 | 4.09 | 350.26 | 21.70 | 2.79 | ... | 3 | 5 | 0 |
| -450 | 10.25 | 28.54 | 5.21 | 114.70 | 15.71 | 4.39 | ... | 3 | 6 | 0 |
| -450 | 7.13 | 18.60 | 2.86 | 935.07 | 23.37 | 4.18 | ... | 3 | 7 | 0 |
| -450 | 23.49 | 19.31 | 5.14 | 555.42 | 14.29 | 2.06 | ... | 3 | 8 | 0 |
| -450 | 21.90 | 16.86 | 5.46 | 516.11 | 29.04 | 0.88 | ... | 3 | 9 | 0 |
| -450 | 20.17 | 8.78 | 3.20 | 319.02 | 19.89 | 3.83 | ... | 3 | 10 | 0 |
| -450 | 26.10 | 13.42 | 2.66 | 759.64 | 27.36 | 2.59 | ... | 3 | 11 | 0 |
| -450 | 13.17 | 1.73 | 1.37 | 226.02 | 15.14 | 0.04 | ... | 3 | 12 | 0 |
| -450 | 3.84 | 25.57 | 1.22 | 86.41 | 4.94 | 0.66 | ... | 3 | 13 | 0 |
| -450 | 6.61 | 9.84 | 4.95 | 44.80 | 28.23 | 1.84 | ... | 3 | 14 | 0 |
| -450 | 27.95 | 21.52 | 1.47 | 994.85 | 17.40 | 1.72 | ... | 3 | 15 | 0 |
| -450 | 15.79 | 26.85 | 1.70 | 548.45 | 29.36 | 3.25 | ... | 3 | 16 | 0 |
| -450 | 5.30 | 12.08 | 6.65 | 371.75 | 4.55 | 1.18 | ... | 3 | 17 | 0 |
| -450 | 22.69 | 1.34 | 5.61 | 634.56 | 11.70 | 3.61 | ... | 3 | 18 | 0 |
| -450 | 10.72 | 6.85 | 4.37 | 590.93 | 8.10 | 3.31 | ... | 3 | 19 | 0 |
| -450 | 27.75 | 14.77 | 1.08 | 918.36 | 18.27 | 4.73 | ... | 3 | 20 | 0 |
| -450 | 24.97 | 23.17 | 6.42 | 780.07 | 25.41 | 0.85 | ... | 3 | 21 | 0 |
| -450 | 15.03 | 7.83 | 6.32 | 298.32 | 16.37 | 2.65 | ... | 3 | 22 | 0 |
| -450 | 29.86 | 22.40 | 2.31 | 414.82 | 24.40 | 0.30 | ... | 3 | 23 | 0 |
| -450 | 17.68 | 20.09 | 6.23 | 141.65 | 26.10 | 4.32 | ... | 3 | 24 | 0 |
| -450 | 19.23 | 29.59 | 5.84 | 656.20 | 23.86 | 3.09 | ... | 3 | 25 | 0 |
| -450 | 2.60 | 17.84 | 1.79 | 214.51 | 20.62 | 4.07 | ... | 3 | 26 | 0 |
| -450 | 9.12 | 21.00 | 3.38 | 24.27 | 1.79 | 2.31 | ... | 3 | 27 | 0 |
| -450 | 28.99 | 9.53 | 3.58 | 733.65 | 18.60 | 0.18 | ... | 3 | 28 | 0 |
| -450 | 12.07 | 27.48 | 4.87 | 716.67 | 12.17 | 0.51 | ... | 3 | 29 | 0 |
| -450 | 26.52 | 15.96 | 4.74 | 60.74 | 22.37 | 1.43 | ... | 3 | 30 | 0 |
| -450 | 5.36 | 11.30 | 5.98 | 815.87 | 13.98 | 4.55 | ... | 3 | 31 | 0 |
| -450 | 23.29 | 12.88 | 4.19 | 193.29 | 2.88 | 4.79 | ... | 3 | 32 | 0 |
| -450 | 18.72 | 4.47 | 5.66 | 385.55 | 10.66 | 2.48 | ... | 3 | 33 | 0 |
| -450 | 17.63 | 6.57 | 2.36 | 960.38 | 1.35 | 2.97 | ... | 3 | 34 | 0 |
| -450 | 4.24 | 17.28 | 3.80 | 891.49 | 6.51 | 3.49 | ... | 3 | 35 | 0 |
| -450 | 1.30 | 2.86 | 2.98 | 871.07 | 6.92 | 1.57 | ... | 3 | 36 | 0 |
| -450 | 24.81 | 15.35 | 4.58 | 459.83 | 10.30 | 1.88 | ... | 3 | 37 | 0 |
| -450 | 14.43 | 10.47 | 6.73 | 685.95 | 27.07 | 0.45 | ... | 3 | 38 | 0 |
| -450 | 13.47 | 5.06 | 2.53 | 165.70 | 13.03 | 2.23 | ... | 3 | 39 | 0 |
| -450 | 16.71 | 5.39 | 3.55 | 439.70 | 8.27 | 1.12 | ... | 3 | 40 | 0 |

Tabelle 5.2: Ceiling Effekt bei 100000 Funktionsauswertungen

| t_{\max} | n | Init | Term | x_l | x_u | Perf |
|------------|-----|--------|------|-------|-------|------|
| 2000 | 10 | UNIRND | EXH | -100 | 100 | BEST |

 Tabelle 5.3: Korrigiertes Problemdesign: 10-dimensionale **sphere**-Funktion für die ES

Insgesamt entsteht so das in Tabelle 5.4 zusammengefasste Algorithmendesign, auf dem die nun folgenden Experimente durchgeführt werden.

| Schranke | μ | ν | c_τ | κ | ρ | $\sigma^{(0)}$ | rec_x | rec_σ | n_σ |
|----------|-------|-------|----------|----------|--------|----------------|----------------|---------------------|------------|
| untere | 1 | 0.5 | 1 | 1 | 1 | 0.001 | 0 | 0 | 0 |
| obere | 30 | 30 | 7 | 2000 | 30 | 5 | 1 | 1 | 1 |

Tabelle 5.4: Algorithmendesign für die ES

Experimente und Auswertung

Zur Suche nach einer guten Parametrisierung wird die SPOT eingesetzt. Wie bereits beschrieben beginnt diese Toolbox mit einer (zu Beginn kleinen) Menge von Designpunkten, die mit dem LHS erzeugt werden. Bartz-Beielstein (2006) gibt an, dass bei k Faktoren mindestens $\frac{1}{2} \cdot (k^2 + 3k + 2)$ Designpunkte benötigt werden, um das DACE-Modell (mit einem Regressionsmodell zweiter Ordnung) zu bestimmen. Das ist die minimale Anzahl, die es aber noch nicht erlaubt, den MSE zu ermitteln. Aus diesem Grund werden hier anstelle der mindestens 55 benötigten Designpunkte 60 Punkte benutzt, um eine Bestimmung des MSE zu ermöglichen. Nach Durchführung der Experimente mit den ersten 60 Designpunkten, wird das DACE-Modell bestimmt. Entsprechend der erwarteten Verbesserung werden sequentiell neue Punkte erzeugt und ausgewertet. SPOT erlaubt es, die Anzahl der neuen Punkte einzustellen. Wir werden in jeder Iteration nur 3 neue Punkte erzeugen und maximal 25 Iterationen durchlaufen. Dieser Ansatz erscheint sinnvoller, als der entgegengesetzte Fall, in dem pro Iteration viele neue Punkte erzeugt, aber nur wenige Iterationen durchlaufen werden. So kann ausgenutzt werden, dass das DACE-Modell in jeder Iteration genauer wird, und somit mit zunehmendem Fortschritt die Vorhersagen des Modells genauer werden.

Am Ende des Vorgangs wird ein verbesserter Designpunkt x_a^* ermittelt. Die verbesserte Parametrisierung x_a^* ist für den hier vorliegenden Fall in Tabelle 5.5 dargestellt. SPOT fasst die

| μ | ν | c_τ | κ | ρ | $\sigma^{(0)}$ | rec_x | rec_σ | n_σ |
|-------|-------|----------|----------|--------|----------------|----------------|---------------------|------------|
| 3 | 1.57 | 4.40 | ∞ | 3 | 1.79 | int | dis | 1 |

Tabelle 5.5: Verbesserter Designpunkt x_p^* für die ES und die 10-dimensionale `sphere`-Funktion

Ergebnisse aller Experimente in einer Result-File zusammen, die benutzt werden kann, um den Verlauf der Experimente nachzuverfolgen. So kann endgültig festgestellt werden, ob floor- oder ceiling Effekte eingetreten sind. Zusätzlich werden Graphen ausgegeben, die eine graphische Beurteilung des ermittelten Modells ermöglichen.

Überprüfung der Hypothese

Nach Ermittlung eines verbesserten Designpunktes x_a^* stellt sich die Frage, ob diese Parametrisierung tatsächlich besser ist, als die Standardparametrisierung. Entsprechend dem CEC Testfunktionensatz wird die ES mit beiden Parametrisierungen 25 mal wiederholt, mit dem Ergebnis, dass die mittlere Fitness von -449.9973 auf -450 , die Standardabweichung von $1.5e - 3$ auf 0 und der Median von -449.9978 auf -450 sanken. Insgesamt ist festzustellen, dass die verbesserte Parametrisierung in allen 25 Läufen die optimale Lösung -450 mit einer Genauigkeit 10^{-12} fand. Dieses Ergebnis wird zusätzlich in Abbildung 5.4 durch ein Boxplot und ein Histogramm visualisiert. Beide Diagramme geben weitere Hinweise, dass die verbesserte Parametrisierung der Standardparametrisierung überlegen ist.

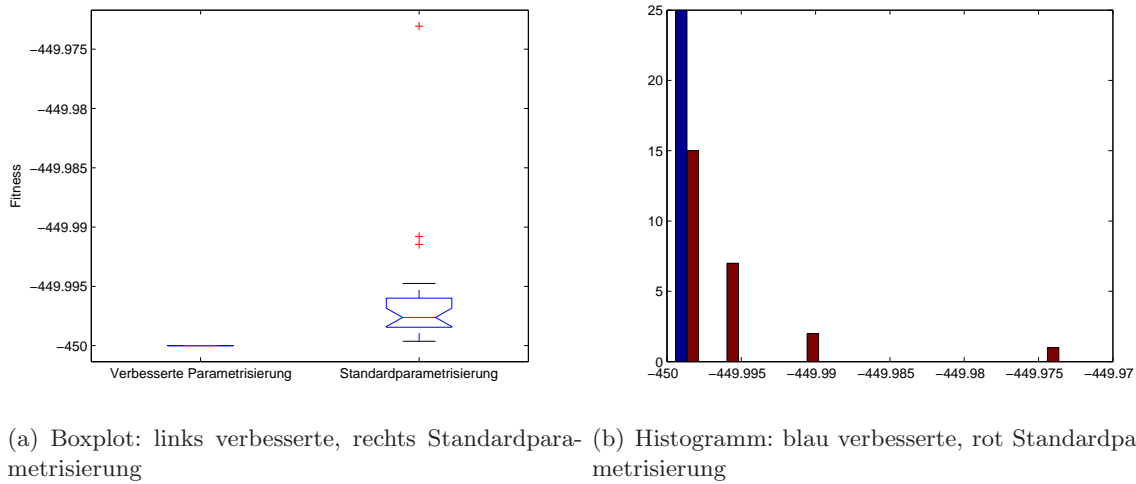


Abbildung 5.3: Visualisierung der Ergebnisse für die 10-dimensionale sphere-Funktion

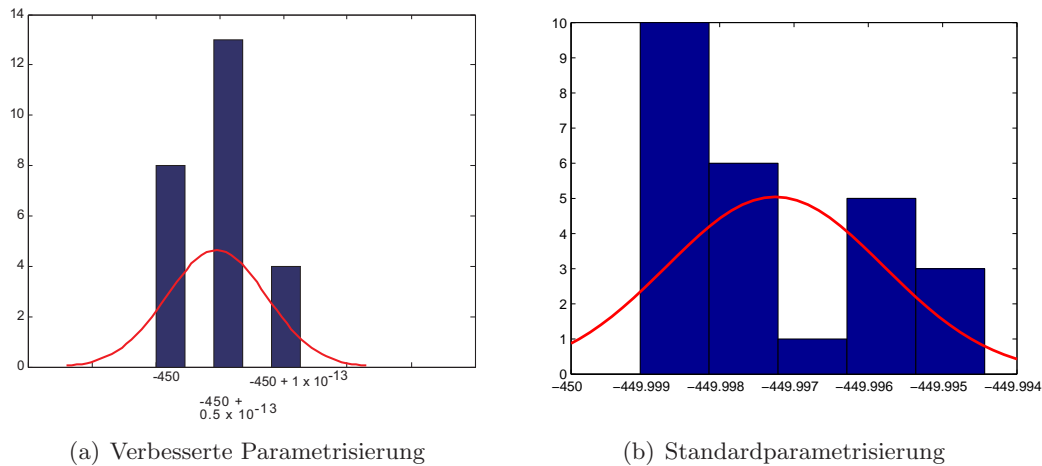


Abbildung 5.4: Histogramm der Proben mit passend überlagerter Dichte der Normalverteilung

Weil ES aber stochastische Suchverfahren sind, sind auch deren Ergebnisse Zufallsgrößen. Deshalb wird der t -Test verwendet, um den Wahrheitsgehalt der Hypothese H_1 zu überprüfen. Dieser Test setzt voraus, dass die Proben aus einer Normalverteilung entnommen sind. Die Histogramme aus Abbildung 5.4 zeigen auf, dass die Annahme der Normalverteilung berechtigt ist, so dass der t -Test angewendet werden kann. Das Ergebnis ist, dass das Konfidenzintervall für den Mittelwert mit einer Irrtumswahrscheinlichkeit $\alpha = 0.05$ in $[0.0021, 0.0034]$ liegt. Somit ist $\delta(ES_{x_p^{(0)}} - ES_{x_p^*}) > 0$ eine berechtigte Annahme. Die Irrtumswahrscheinlichkeit kann sogar bis auf $1.7256 \cdot 10^{-9}$ abgesenkt werden, ohne dass es Auswirkungen auf die Signifikanz von H_1 hat. Statistisch ist somit nachgewiesen, dass x_p^* der Standardparametrisierung überlegen ist. Einen letzten eindrucksvollen Vergleich beider Parametrisierungen liefert die *run length distribution*, die in Abbildung 5.5 dargestellt ist. Das Ergebnis ist mehr als eindeutig: Die verbesserte Parametrisierung erlaubt der ES das Auffinden des Minimums nach 2500 Funktionsauswertungen in 100% der Fälle. Die ES mit Standardparametrisierung hingegen benötigt über 15000 Funktionsauswertungen, damit mindestens einige Läufe zum Minimum gelangen. Erst nach ca. 20000 Funktionsauswertungen kann sicher davon ausgegangen werden, dass die ES das Mini-

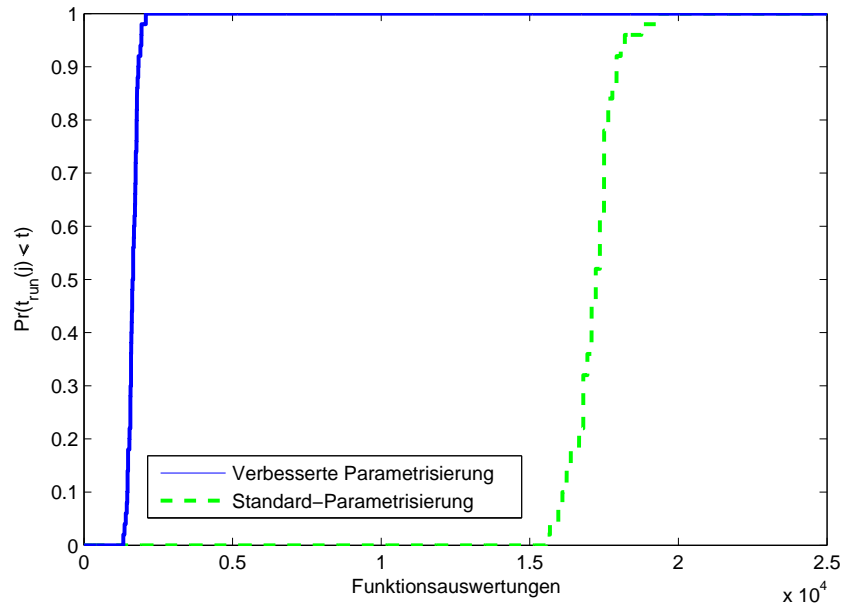


Abbildung 5.5: RLD der zwei unterschiedlich eingestellter ES

mum immer findet. Somit hat die Parameteroptimierung die Anzahl der Funktionsauswertungen um einen Faktor 10 reduziert.

Die Antwort auf die berechtigte Frage, warum die Parametrisierung x_a^* der Standardparametrisierung überlegen ist wird im Kapitel 6 gegeben.

5.3.2 Anpassung des GA auf die 10-dimensionale sphere-Funktion

Erneut wird die 10-dimensionale **sphere**-Funktion betrachtet, die nun mit dem GA minimiert werden soll. Erneut wird versucht die Effizienz des GA zu steigern, in dem eine gute Parametrisierung gefunden wird. Die Probleminstance ist analog zur der aus 5.3.1, mit dem Unterschied, dass der GA 6000 Funktionsauswertungen benötigt, um floor-Effekte zu vermeiden.

| t_{\max} | n | Init | Term | x_l | x_u | Perf |
|------------|-----|--------|------|-------|-------|------|
| 8000 | 10 | UNIRND | EXH | -100 | 100 | BEST |

 Tabelle 5.6: Probleminstance 10-dimensionale **sphere**-Funktion für den GA

Vorexperimentelle Planung

In einer vorexperimentellen Planung wird erneut das Ziel der Experimente abgesteckt. Es soll erneut eine Parametrisierung gefunden werden, die die Effizienz des GA auf der Probleminstance aus Tabelle 5.6 steigert. Es fällt auf, dass der Matlab-GA eine spezielle Mutationsfunktion implementiert hat, die auf restringierten Optimierungsproblemen angewendet wird. Dies ist in soweit enttäuschend, dass der GA so nur noch drei Parameter besitzt, die eine Einstellung erlauben. Die verwendete Mutationsfunktion `mutationadaptfeasible.m` besitzt einen Selbstadaptationsmechanismus, der die Schrittweiten einstellt. Auch eine Initialisierung der Schrittweiten ist nicht möglich, weil die Start-Schrittweiten in der Funktion fest auf 1 kodiert sind.

Wissenschaftliche Behauptung

Für die 10-dimensionale **sphere**-Funktion existiert eine Parametrisierung $x_p^* \in \mathcal{D}_A$ des GA, die bessere Ergebnisse gegenüber der, in der Toolbox eingestellten, Standardparametrisierung liefert.

Hypothese

Der GA mit Parametrisierung x_p^* findet bei Minimierung der 10-dimensionalen **sphere**-Funktion im Mittel bessere Lösungen, als die Standardparametrisierung. Es gilt also:

$$H_1 : \delta(GA_{x_p^{(0)}} - GA_{x_p^*}) > 0 .$$

Spezifikation des Algorithmendesigns \mathcal{D}_A und des Problemdesigns \mathcal{D}_P

Das Problemdesign wurde bereits einleitend dargestellt. Die Definition des Designraums gestaltet sich hier einfach. Die Rekombinationswahrscheinlichkeit p_{rek} kann nur Werte aus $[0, 1]$ annehmen, so dass $[0, 1]$ der Bereich für den Faktor p_{rek} wird. Die Populationsgröße μ wird im Intervall $[1, 40]$ untersucht, und damit verbunden auch der Parameter *EliteCount*. *EliteCount* muss kleiner sein als die Populationsgröße, so dass eine zusätzliche bedingte Anweisung in der Schnittstelle zu SPOT potentielle Fehler abfangen muss. Insgesamt entsteht so das in Tabelle 5.7 zusammengefasste Algorithmendesign, auf dem die nun folgenden Experimente durchgeführt werden.

| Schranke | μ | p_{rek} | EliteCount |
|----------|-------|-----------|------------|
| untere | 1 | 0 | 0 |
| obere | 40 | 1 | 39 |

Tabelle 5.7: Algorithmendesign für den GA

Experimente und Auswertung

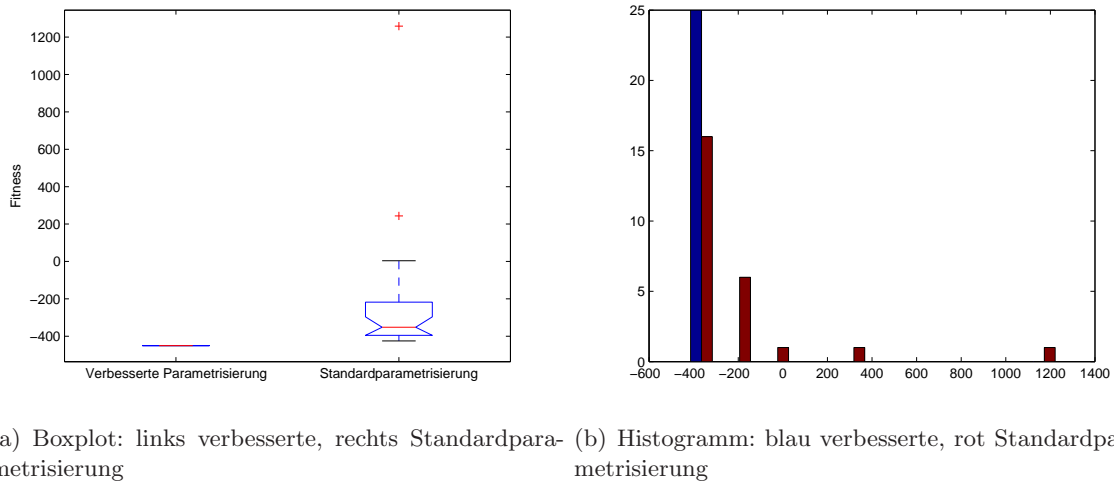
Erneut werden die Experimente mit SPOT durchgeführt. SPOT benötigt wesentlich weniger Designpunkte um ein erstes DACE-Modell aufzustellen, weil nur 3 Parameter untersucht werden. So wären bereits 10 Punkte ausreichend um das Modell aufzustellen, aber noch nicht, um den MSE zu ermitteln. Deshalb wird das erste Experiment mit 15 Designpunkten durchgeführt, und setzt sich dann iterativ weiter fort. Am Ende der Experimente steht wieder ein verbessertes Design, das in Tabelle 5.8 dargestellt ist.

| μ | p_{rek} | EliteCount |
|-------|-----------|------------|
| 2 | 0.06 | 1 |

Tabelle 5.8: Verbesserter Designpunkt x_p^* für den GA und die 10-dimensionale **sphere**-Funktion

Überprüfung der Hypothese

Zu Überprüfung der aufgestellten Hypothese wird der GA sowohl mit Standardparametrisierung, als auch mit der gefundenen Parametrisierung x_a^* 25-mal auf der Probleminstanz aus Tabelle 5.6 wiederholt. Die Ergebnisse werden erneut graphisch dargestellt. Die Abbildung 5.6 stellt den entsprechenden Boxplot und das entsprechende Histogramm dar. Beide Graphiken


 Abbildung 5.6: Visualisierung der Ergebnisse für die 10-dimensionale `sphere`-Funktion

zeigen eindeutig auf, dass die verbesserte Parametrisierung die Standardparametrisierung klar übertrifft, so dass der folgende t-Test eher unnötig erscheint. Er liefert das Konfidenzintervall $[75.42, 361.56]$ für den Mittelwert und den p-Wert 0.0043. Somit kann die Nullhypothese H_0 zugunsten unserer Alternative $H_1 : \delta(GA_{x_p^{(0)}} - GA_{x_p^*}) > 0$ verworfen werden.

Nach der ausführlicher Beschreibung der SPO am Beispiel der ES und des GA auf der 10-dimensionalen `sphere`-Funktion, werden die nun folgenden Parameteroptimierungen in kompakter Form dargestellt. Alle Experimente laufen nach dem oben beschriebenen Muster ab.

5.3.3 Optimierung der ES-Parametrisierung auf den übrigen Probleminstanzen

Die Ermittlung guter Parameter läuft für die übrigen Probleminstanzen nach dem gleichen Prinzip ab, so dass es nicht viel Sinn machen würde, hier alle Experimente ausführlich darzustellen, zumal bei der Durchführung keine besonderen Ereignisse eintraten. Deshalb werden die Ergebnisse in Tabellenform dargestellt. Die Tabelle 5.9 fasst die Probleminstanzen zusammen, mit denen die Experimente durchgeführt werden. Um insgesamt auch eine Zusammenfassung aller Untersuchungen zu ermöglichen, wird auch die bereits untersuchte Probleminstanz in die Tabellen aufgenommen. Das Algorithmendesign wurde bereits in Tabelle 5.4 zusammengefasst

| Nr. | Problem | t_{\max} | n | Init | Term | x_l | x_u | Perf |
|-----|-------------------------|------------|-----|--------|------|-------|-------|------|
| 1 | <code>sphere</code> | 2000 | 10 | UNIRND | EXH | -100 | 100 | BEST |
| 2 | <code>sphere</code> | 5000 | 30 | UNIRND | EXH | -100 | 100 | BEST |
| 3 | <code>sphere</code> | 9000 | 50 | UNIRND | EXH | -100 | 100 | BEST |
| 4 | <code>Rosenbrock</code> | 40000 | 10 | UNIRND | EXH | -100 | 100 | BEST |
| 5 | <code>Rosenbrock</code> | 105000 | 30 | UNIRND | EXH | -100 | 100 | BEST |
| 6 | <code>Rosenbrock</code> | 210000 | 50 | UNIRND | EXH | -100 | 100 | BEST |
| 7 | <code>Rastrigin</code> | 30000 | 10 | UNIRND | EXH | -5 | 5 | BEST |
| 8 | <code>Rastrigin</code> | 75000 | 30 | UNIRND | EXH | -5 | 5 | BEST |
| 9 | <code>Rastrigin</code> | 100000 | 50 | UNIRND | EXH | -5 | 5 | BEST |

Tabelle 5.9: Problemdesign der restlichen Testfunktionen für die ES

und wir unverändert übernommen. Nach Durchführung der Experimente liefern Histogramme

und t-Test stets das Ergebnis, dass die gefundene neue Parametrisierung der Standardparametrisierung überlegen ist. Die Parametrisierungen für die einzelnen Probleminstanzen werden in Tabelle 5.10 zusammengefasst. Eine Interpretation der Ergebnisse wird im Kapitel 6 gegeben.

| Nr. | μ | ν | c_τ | κ | ρ | $\sigma^{(0)}$ | rec _x | rec _{σ} | n_σ |
|-----|-------|-------|----------|----------|--------|----------------|------------------|------------------------------------|------------|
| 1 | 3 | 1.57 | 4.40 | ∞ | 3 | 1.79 | int | dis | 1 |
| 2 | 3 | 7.31 | 3.17 | ∞ | 2 | 2.16 | int | int | 1 |
| 3 | 2 | 1.53 | 2.93 | 365 | 2 | 4.79 | int | dis | 1 |
| 4 | 17 | 17.39 | 2.64 | 13 | 6 | 3.95 | dis | int | n |
| 5 | 16 | 13.6 | 1.18 | 288 | 8 | 4.64 | dis | int | n |
| 6 | 14 | 27.34 | 3.75 | 525 | 2 | 0.29 | dis | int | n |
| 7 | 21 | 12.20 | 3.91 | 789 | 11 | 0.37 | int | dis | n |
| 8 | 11 | 26.34 | 1.59 | 255 | 2 | 1.61 | int | int | n |
| 9 | 5 | 17.69 | 1.07 | 76 | 5 | 1.44 | int | int | n |

Tabelle 5.10: Verbesserte Parametrisierungen der ES für die Probleminstanzen 1 – 9

5.3.4 Optimierung der GA-Parametrisierung auf den übrigen Probleminstanzen

Analog fassen die Tabellen 5.9 und 5.12 die Probleminstanzen und die Resultate der Parametereoptimierung für den GA zusammen. Das Algorithmendesign wurde aus Tabelle 5.7 übernommen.

| Nr. | Problem | t_{\max} | n | Init | Term | x_l | x_u | Perf |
|-----|------------|------------|-----|--------|------|-------|-------|------|
| 1 | sphere | 8000 | 10 | UNIRND | EXH | -100 | 100 | BEST |
| 2 | sphere | 12000 | 30 | UNIRND | EXH | -100 | 100 | BEST |
| 3 | sphere | 15000 | 50 | UNIRND | EXH | -100 | 100 | BEST |
| 4 | Rosenbrock | 50000 | 10 | UNIRND | EXH | -100 | 100 | BEST |
| 5 | Rosenbrock | 150000 | 30 | UNIRND | EXH | -100 | 100 | BEST |
| 6 | Rosenbrock | 250000 | 50 | UNIRND | EXH | -100 | 100 | BEST |
| 7 | Rastrigin | 30000 | 10 | UNIRND | EXH | -5 | 5 | BEST |
| 8 | Rastrigin | 50000 | 30 | UNIRND | EXH | -5 | 5 | BEST |
| 9 | Rastrigin | 60000 | 50 | UNIRND | EXH | -5 | 5 | BEST |

Tabelle 5.11: Problemdesign der restlichen Testfunktionen für den GA

5.4 Bewertung unterschiedlicher Suchalgorithmen

Für die Bestimmung optimaler Parameter waren eine Reihe an Vorüberlegungen zu treffen. Für den Vergleich zwischen ES und GA ist dies nicht mehr notwendig, bzw. nicht mehr möglich. Der CEC 2005 Testfunktionensatz gibt ein starres Regelwerk vor, entsprechend dem die experimentellen Untersuchungen durchzuführen sind. Dieses Regelwerk besteht aus 7 Punkten, von denen die Punkte 6 und 7 entfallen. Der Punkt 6 behandelt die Parametereinstellung, die bereits im letzten Abschnitt ausführlich behandelt wurden. Punkt 7 ist nur dann relevant, wenn die Optimierverfahren auf einer speziellen Kodierung arbeiten. Dieser Fall trifft hier nicht zu, weil nur auf reellen Kodierungen gearbeitet wird. Jedes Optimierverfahren wird auf jeder Probleminstanz 25-mal wiederholt. Die so gewonnenen Ergebnisse sind in Tabellenform darzustellen.

| Nr. | μ | p_{rek} | EliteCount |
|-----|-------|-----------|------------|
| 1 | 2 | 0.06 | 1 |
| 2 | 5 | 0.09 | 2 |
| 3 | 3 | 0.37 | 1 |
| 4 | 3 | 0.26 | 1 |
| 5 | 24 | 0.55 | 6 |
| 6 | 34 | 0.37 | 6 |
| 7 | 4 | 0.16 | 2 |
| 8 | 4 | 0.21 | 3 |
| 9 | 2 | 0.06 | 1 |

Tabelle 5.12: Verbesserte Parametrisierungen des GA für die Probleminstanzen 1 – 9

5.4.1 Berechnung der Fehler

Ein Kriterium zur Bewertung unterschiedlicher Verfahren ist der Wert, um die deren Lösung vom tatsächlichen Minimum abweicht. Dazu wird den Verfahren eine bestimmte Anzahl an Funktionsauswertungen zu Verfügung gestellt und die Fehler dokumentiert. So entstehen die Tabellen 5.13, 5.14 und 5.15, die die Fehler für ES, GA und RII nach 1000, 10000 und 100000 Funktionsauswertungen angeben.

Die Zeilen der Tabellen werden unter der ES, dem GA und der RII aufgeteilt. Die Spalten unterteilen sich in die Dimension $n = 10$, $n = 30$ und $n = 50$ und dann weiter in die **sphere**-, die **Rosenbrock**- und die **Rastrigin**-Funktion. Für jede Probleminstanz und jedes Optimierverfahren werden die Fehlerwerte dokumentiert. Zusätzlich werden Mittelwert und Standardabweichung angegeben.

| Dimension | | $n = 10$ | | | $n = 30$ | | | $n = 50$ | | |
|-----------|---------|----------|---------|--------|----------|---------|--------|----------|---------|--------|
| | Prob | 1 | 6 | 10 | 1 | 6 | 10 | 1 | 6 | 10 |
| ES | Min | 6.70e-3 | 3.33e6 | 5.12e1 | 1.75e3 | 1.00e9 | 3.99e2 | 1.13e4 | 8.44e9 | 5.94e2 |
| | 7. | 5.25e-2 | 8.33e6 | 6.31e1 | 3.45e3 | 2.74e9 | 4.66e2 | 1.68e4 | 1.47e10 | 6.81e2 |
| | Median | 2.02e-1 | 1.35e7 | 6.69e1 | 4.54e3 | 3.09e9 | 4.94e2 | 2.76e4 | 1.82e10 | 7.10e2 |
| | 19. | 7.54e-1 | 2.27e7 | 7.42e1 | 7.18e3 | 4.66e9 | 5.22e2 | 3.34e4 | 2.09e10 | 7.53e2 |
| | Max | 3.48 | 1.07e8 | 8.82e1 | 1.69e4 | 5.31e10 | 6.61e9 | 7.80e4 | 3.21e10 | 8.07e2 |
| | Mittel | 6.02e-1 | 1.87e7 | 6.91e1 | 5.86e3 | 3.57e9 | 4.93e2 | 2.72e4 | 1.84e10 | 7.07e2 |
| Std | 9.54e-1 | 2.05e7 | 9.49 | 3.62e3 | 1.48e9 | 4.08e1 | 1.53e4 | 5.73e9 | 5.58e1 | |
| GA | Min | 5.01e2 | 5.22e6 | 1.78e2 | 2.94e3 | 7.51e9 | 4.62e2 | 9.62e4 | 4.27e10 | 8.92e2 |
| | 7. | 5.50e2 | 2.15e7 | 1.90e2 | 5.06e3 | 1.26e10 | 4.81e2 | 1.82e5 | 4.90e10 | 1.03e3 |
| | Median | 5.53e2 | 5.47e7 | 2.07e2 | 6.45e3 | 1.67e10 | 5.47e2 | 3.21e5 | 5.06e10 | 1.08e3 |
| | 19. | 5.62e2 | 1.12e8 | 2.23e2 | 7.59e3 | 1.98e10 | 6.00e2 | 3.97e5 | 5.37e10 | 1.22e3 |
| | Max | 6.68e2 | 5.24e8 | 2.67e2 | 1.58e4 | 2.48e10 | 6.67e2 | 4.95e5 | 5.98e10 | 1.33e3 |
| | Mittel | 5.53e2 | 1.09e8 | 2.07e2 | 6.72e3 | 1.62e10 | 5.42e2 | 2.99e5 | 5.13e10 | 1.12e3 |
| Std | 1.43e1 | 1.38e8 | 2.17e1 | 2.72e3 | 5.02e9 | 7.58e1 | 1.29e5 | 4.33e9 | 1.30e2 | |
| RII | Min | 1.35e4 | 2.56e6 | 2.38e2 | 8.65e4 | 1.53e11 | 6.93e2 | 2.06e5 | 2.33e11 | 1.53e3 |
| | 7. | 3.77e4 | 2.90e10 | 3.03e2 | 1.44e5 | 2.51e11 | 1.03e3 | 2.61e5 | 3.70e11 | 1.99e3 |
| | Median | 5.26e4 | 5.33e10 | 3.11e2 | 1.62e5 | 2.78e11 | 1.20e3 | 3.06e5 | 4.25e11 | 2.30e3 |
| | 19. | 7.21e4 | 9.39e10 | 4.19e2 | 2.02e5 | 3.55e11 | 1.27e3 | 3.50e5 | 4.89e11 | 2.38e3 |
| | Max | 1.15e5 | 2.74e11 | 9.39e2 | 2.56e5 | 5.28e11 | 1.88e3 | 3.98e5 | 6.81e11 | 2.61e3 |
| | Mittel | 5.62e4 | 7.10e10 | 6.77e2 | 1.70e5 | 3.10e11 | 1.19e3 | 3.06e5 | 4.27e11 | 2.19e3 |
| Std | 2.52e4 | 6.28e10 | 1.50e2 | 3.89e4 | 1.04e11 | 2.83e2 | 4.85e4 | 9.91e10 | 3.15e2 | |

Tabelle 5.13: Fehler unterschiedlicher Optimierverfahren nach 1000 Funktionsauswertungen

| Dimension | | n = 10 | | | n = 30 | | | n = 50 | | |
|-----------|--------|--------|---------|--------|----------|---------|--------|----------|---------|--------|
| | Prob | 1 | 6 | 10 | 1 | 6 | 10 | 1 | 6 | 10 |
| ES | Min | 0 | 2.01e1 | 3.98 | 5.68e-14 | 2.65e2 | 1.40e2 | 1.71e-13 | 9.88e5 | 4.01e2 |
| | 7. | 0 | 8.94e1 | 6.97 | 1.14e-13 | 7.32e2 | 2.21e2 | 2.84e-13 | 4.51e6 | 4.45e2 |
| | Median | 0 | 1.91e2 | 1.05e1 | 1.14e-13 | 8.62e2 | 2.30e2 | 3.98e-13 | 6.16e6 | 4.83e2 |
| | 19. | 0 | 4.94e2 | 2.47e1 | 1.14e-13 | 1.41e3 | 2.40e2 | 7.39e-13 | 1.01e7 | 4.93e2 |
| | Max | 0 | 2.84e4 | 4.30e1 | 1.71e-13 | 1.14e4 | 2.70e2 | 9.10e-12 | 2.02e7 | 5.25e2 |
| | Mittel | 0 | 2.05e3 | 1.53e1 | 1.09e-13 | 1.74e3 | 2.25e2 | 9.75e-13 | 7.95e6 | 4.71e2 |
| | Std | 0 | 5.84e3 | 1.21e1 | 3.64e-14 | 2.56e3 | 2.48e1 | 1.83e-12 | 4.83e6 | 3.35e1 |
| GA | Min | 5.69e1 | 8.10 | 9.02 | 9.98e1 | 1.36e5 | 1.47e2 | 1.15e2 | 1.11e9 | 4.69e2 |
| | 7. | 8.74e1 | 9.31e1 | 2.39e1 | 1.13e2 | 3.64e5 | 1.96e2 | 1.18e2 | 1.62e9 | 5.55e2 |
| | Median | 1.07e2 | 2.25e2 | 4.09e1 | 1.15e2 | 7.19e5 | 2.45e2 | 1.21e2 | 2.68e9 | 6.02e2 |
| | 19. | 1.14e2 | 6.66e2 | 5.48e1 | 1.18e2 | 1.52e6 | 2.77e2 | 1.25e2 | 3.48e9 | 6.61e2 |
| | Max | 1.20e2 | 1.11e4 | 8.06e1 | 1.50e2 | 3.08e6 | 4.69e2 | 1.28e2 | 5.08e9 | 7.73e2 |
| | Mittel | 9.96e1 | 1.30e3 | 3.98e1 | 1.15e2 | 1.06e6 | 2.54e2 | 1.22e2 | 2.75e9 | 6.06e2 |
| | Std | 1.83e1 | 2.69e3 | 1.98e1 | 4.55 | 9.22e5 | 8.07 | 1.02e2 | 1.14e9 | 8.80e1 |
| RII | Min | 1.56e4 | 2.01e10 | 1.93e2 | 1.26e5 | 1.31e11 | 6.45e2 | 2.20e5 | 2.19e11 | 1.58e3 |
| | 7. | 3.32e4 | 4.12e10 | 2.97e2 | 1.64e5 | 2.31e11 | 9.95e2 | 2.90e5 | 3.88e11 | 2.07e3 |
| | Median | 4.57e4 | 9.56e10 | 4.07e2 | 1.89e5 | 3.41e11 | 1.23e3 | 3.04e5 | 4.32e11 | 2.23e3 |
| | 19. | 5.12e4 | 1.21e11 | 4.77e2 | 2.17e5 | 3.55e11 | 1.36e3 | 3.40e5 | 5.25e11 | 2.45e3 |
| | Max | 8.71e4 | 1.65e11 | 7.70e2 | 2.52e5 | 4.24e11 | 2.06e3 | 4.01e5 | 7.28e11 | 2.95e3 |
| | Mittel | 4.61e4 | 8.62e10 | 4.35e2 | 1.91e5 | 3.06e11 | 1.19e3 | 3.08e5 | 4.61e11 | 2.22e3 |
| | Std | 1.80e4 | 4.23e10 | 1.41e2 | 3.22e4 | 8.31e10 | 3.05e2 | 4.57e4 | 1.32e11 | 3.34e2 |

Tabelle 5.14: Fehler unterschiedlicher Optimierverfahren nach 10000 Funktionsauswertungen

| Dimension | | n = 10 | | | n = 30 | | | n = 50 | | |
|-----------|--------|---------|---------|--------|----------|---------|--------|----------|---------|--------|
| | Prob | 1 | 6 | 10 | 1 | 6 | 10 | 1 | 6 | 10 |
| ES | Min | 0 | 6.20e-1 | 2.98 | 0 | 2.40e1 | 4.26e1 | 2.27e-13 | 4.70e1 | 8.26e1 |
| | 7. | 0 | 8.77e-1 | 5.97 | 5.68e-14 | 2.51e2 | 1.67e2 | 3.41e-13 | 1.13e2 | 9.65e1 |
| | Median | 0 | 2.56 | 7.96 | 5.68e-14 | 1.19e2 | 1.84e2 | 4.55e-13 | 1.52e2 | 1.23e2 |
| | 19. | 0 | 3.26e1 | 9.95 | 5.68e-14 | 1.64e2 | 1.94e2 | 4.55e-13 | 3.59e2 | 1.37e2 |
| | Max | 0 | 9.36e1 | 1.29e1 | 5.68e-14 | 4.82e2 | 2.00e2 | 5.23e-12 | 1.31e4 | 3.53e2 |
| | Mittel | 0 | 1.71e1 | 7.64 | 5.46e-14 | 1.20e2 | 1.54e2 | 6.68e-13 | 1.47e3 | 1.28e2 |
| | Std | 0 | 2.43e2 | 2.75 | 1.14e-14 | 1.03e2 | 6.03e1 | 9.79e-13 | 3.56e3 | 5.33e1 |
| GA | Min | 7.46e-1 | 1.23e-1 | 9.96 | 9.54e1 | 7.48 | 9.05e1 | 2.42e5 | 2.05e2 | 2.59e2 |
| | 7. | 1.07 | 3.53 | 2.68e1 | 1.09e2 | 2.19e2 | 1.52e2 | 2.92e5 | 3.36e2 | 3.84e2 |
| | Median | 1.15e1 | 2.92e1 | 3.18e1 | 1.15e2 | 3.30e2 | 1.92e2 | 3.35e5 | 6.70e2 | 4.27e2 |
| | 19. | 1.16e1 | 2.93e2 | 4.87e1 | 1.18e2 | 1.39e3 | 2.23e2 | 3.61e5 | 1.36e2 | 4.61e2 |
| | Max | 1.19e1 | 1.09e3 | 6.96e1 | 1.19e2 | 1.10e4 | 3.45e2 | 4.08e5 | 6.92e2 | 8.48e2 |
| | Mittel | 1.09e1 | 2.15e2 | 3.71e1 | 1.13e2 | 2.02e3 | 1.92e2 | 3.27e5 | 1.52e3 | 4.46e2 |
| | Std | 3.25 | 3.34e2 | 1.83e1 | 6.72 | 3.50e3 | 5.8e1 | 4.64e4 | 2.06e3 | 1.23e2 |
| RII | Min | 1.80e4 | 1.74e10 | 2.04e2 | 1.17e5 | 1.59e11 | 6.89e2 | 2.42e5 | 2.57e11 | 1.49e3 |
| | 7. | 4.98e4 | 6.67e10 | 3.08e2 | 1.72e5 | 2.42e11 | 1.00e3 | 2.92e5 | 3.56e11 | 1.82e3 |
| | Median | 5.96e4 | 1.01e11 | 3.56e2 | 1.94e5 | 3.19e11 | 1.18e3 | 3.35e5 | 4.94e11 | 2.15e3 |
| | 19. | 7.27e4 | 1.39e11 | 3.73e2 | 2.33e5 | 3.87e11 | 1.31e3 | 3.61e5 | 5.89e11 | 2.59e3 |
| | Max | 1.09e5 | 2.52e11 | 8.88e2 | 2.67e5 | 5.08e11 | 1.81e3 | 4.08e5 | 7.41e11 | 2.94e3 |
| | Mittel | 5.95e4 | 1.02e11 | 3.87e2 | 1.98e5 | 3.21e11 | 1.17e3 | 3.27e5 | 4.88e11 | 2.22e3 |
| | Std | 1.96e4 | 5.52e10 | 1.05e2 | 4.18e4 | 9.39e10 | 2.84e2 | 4.64e4 | 1.36e11 | 4.20e2 |

Tabelle 5.15: Fehler unterschiedlicher Optimierverfahren nach 100000 Funktionsauswertungen

5.4.2 Benötigte Funktionsauswertungen

Bisher wurde gemessen, wie hoch der Fehler ist, wenn einem Optimierverfahren eine begrenzte Anzahl an Funktionsauswertungen zu Verfügung gestellt wird. Jetzt wird das Problem von einer anderen Seite aus betrachtet. Die Verfahren müssen die optimale Lösung mit einer gewissen Toleranz finden und es wird die dazu benötigte Anzahl an Funktionsauswertungen gemessen. Um einen gescheiterten Lauf nach einer gewissen Zeit abbrechen zu können, wird die maximale Anzahl der Funktionsauswertungen auf $10000 \cdot n$ gesetzt. Wird diese Anzahl überschritten, so wird der Lauf als erfolglos gekennzeichnet. Dazu wird in den Tabellen ein „–“ verwendet. Neben den benötigten Funktionsauswertungen wird hier auch die Erfolgswahrscheinlichkeit und die Erfolgsperformance berechnet. Die *Erfolgsrate* gibt für jedes Optimierverfahren und jede Problem Instanz relativ an, wie viele der 25 Läufe das Minimum gefunden haben. Die Erfolgsperformance betrachtet

$$\text{mean}(\text{FE für erfolgreichen Lauf}) \cdot \frac{25}{\# \text{ erfolgreiche Läufe}}$$

mit der Anzahl der Funktionsauswertungen FE. Sie gibt an, wie viele Funktionsauswertungen im Schnitt benötigt werden, um das Minimum einmal zu finden.

Leider war es in der gegebenen Zeit nicht mehr möglich alle Berechnungen durchzuführen, so dass in den Tabellen auch der Eintrag „k.W.“ auftaucht, der angibt, dass kein Wert berechnet werden konnte. Betrachtet man die Tabelle 5.17 und die Zeit, die der GA bei 200000 Funktionsauswertungen auf einer 50 dimensionalen Funktion benötigt, so erkennt man schnell, dass nur *eine* Berechnung $2319.171s = 38.65min$ benötigt und so ein enormer Zeitbedarf entstand.

| Dim | Prob | Tol | Min | 7. | Median | 19. | Max | Mittel | Std | p_{succ} | SP |
|-----|------|------|--------|--------|--------|--------|--------|--------|--------|------------|--------|
| 10 | 1 | 1e-6 | 7.05e2 | 7.68e2 | 8.28e2 | 8.88e2 | 1.04e3 | 8.29e2 | 8.74e1 | 1 | 8.29e2 |
| | 6 | 1e-2 | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. |
| | 10 | 1e-2 | – | – | – | – | – | – | – | 0 | – |
| 30 | 1 | 1e-6 | 5.09e3 | 5.18e3 | 5.29e3 | 5.57e3 | 5.81e3 | 5.37e3 | 2.23e2 | 1 | 5.37e3 |
| | 6 | 1e-2 | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. |
| | 10 | 1e-2 | – | – | – | – | – | – | – | 0 | – |
| 50 | 1 | 1e-6 | 5.50e3 | 5.92e3 | 6.10e3 | 6.50e3 | 6.95e3 | 6.17e3 | 3.85e2 | 1 | 6.17e3 |
| | 6 | 1e-2 | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. | k.W. |
| | 10 | 1e-2 | – | – | – | – | – | – | – | 0 | – |

Tabelle 5.16: Von der ES benötigte Funktionsauswertungen zum Erreichen einer bestimmten Genauigkeit

Für die RII wird keine Tabelle angegeben, weil dieses Verfahren auf allen betrachteten Problem Instanzen erfolglos war, Werte für den GA konnte, aus den bereits aufgeführten Gründen, in der vorgegebenen Zeit gar nicht berechnet werden.

5.4.3 Run-length distribution

5.4.4 Komplexität der Algorithmen

Die Komplexität der Algorithmen ergibt sich aus den Berechnungen, die das Verfahren neben der Auswertung der Fitnessfunktion durchführt. Weil die hier verwendeten Verfahren in Matlab implementiert sind, gestaltet sich die Berechnung der Komplexität, so wie sie im CEC 2005 Testfunktionensatz beschrieben ist, als eher ungeeignet.

Matlab wertet mehrere Punkte auf einer Funktion sehr schnell in einem Schritt aus, während es die gleiche Anzahl von Punkten über eine Schleife nur schleppend auswertet. Ein kleines Beispiel

```
a = rand(100000,1);
tic; sqrt(a); toc
Elapsed time is 0.032000 seconds.
```

```
tic;
for i = 1: 100000
sqrt(a(i));
end
toc
Elapsed time is 0.172000 seconds.
```

verdeutlicht die Unterschiede. Wird die Wurzel aller 100000 Zahlen in einem Schritt berechnet, so benötigt Matlab auf einem Intel Celeron 3.08 GHz mit 1024 MB Arbeitsspeicher und dem Betriebssystem Windows XP SP2 0.032 Sekunden, während eine iterative Berechnung auf dem gleichen Rechner 0.172 Sekunden benötigt. So wird aber eine $(\mu, 1)$ -ES wesentlich mehr Zeit benötigen, als eine $(\mu, 20)$ -ES, obwohl beide Verfahren gleich komplex sind.

Eine erste Idee zu Lösung des Problems war, die optimale Parametrisierung zu bestimmen und mit dieser Parametrisierung die Laufzeiten zu berechnen. Dieser erste Ansatz ist aber nicht besonders brauchbar. Schließlich ist es hier nicht gewollt, die Effizienz der Verfahren zu steigern, sondern abzuschätzen, wie viel Zeit die Algorithmen neben der Auswertung der Zielfunktion benötigen. Es ist deshalb sinnvoller die Algorithmen so einzustellen, dass in jeder Generation die gleiche Anzahl an Nachkommen zu bewerten ist. Entsprechend dem GA (mit Standardparametrisierung), der in jeder Generation 20 neue Individuen erzeugt und bewertet, wird eine $(20, 20)$ -ES verwendet. Die verbleibenden Parameter werden mit den Standardwerten belegt. Es soll hier nochmals explizit hingewiesen werden, dass das Ziel des Tests die Berechnung der Rechenzeit der Verfahren nach 200000 Funktionsauswertungen ist. Wir sind nicht an der gefundenen Lösung interessiert. Deshalb kann die oben angegebene, für die ES *simlose* Einstellung

| Dim | | ES | GA | RII |
|-----|-------------------------------|----------|-----------|---------|
| 10 | T_0 | 4.453s | 4.453s | 4.453s |
| | T_1 | 0.015s | 0.015s | 0.015s |
| | \hat{T}_2 | 109.719s | 1260.984s | 50.843s |
| | $\frac{\hat{T}_2 - T_1}{T_0}$ | 24.636 | 283.173 | 11.414 |
| | | | | |
| 30 | T_0 | 4.453s | 4.453s | 4.453s |
| | T_1 | 0.046s | 0.046s | 0.046s |
| | \hat{T}_2 | 117.688s | 1800.062s | 52.547s |
| | $\frac{\hat{T}_2 - T_1}{T_0}$ | 26.4191 | 404.225 | 11.790 |
| | | | | |
| 50 | T_0 | 4.453s | 4.453s | 4.453s |
| | T_1 | 0.078s | 0.078s | 0.078s |
| | \hat{T}_2 | 127.813s | 2319.171s | 58.39s |
| | $\frac{\hat{T}_2 - T_1}{T_0}$ | 28.685 | 520.791 | 13.095 |
| | | | | |

Tabelle 5.17: Komplexität der Algorithmen für 10, 30 und 50 Dimensionen

verwendet werden. Auf gleiche Weise wird auch die RII eingestellt. Aufgrund der Fairness, arbeitet sie auf 20 Punkten, die sie zufällig über den Suchraum bewegt.

Die Ergebnisse werden in Tabelle 5.17 zusammengefasst. Das genaue Testprogramm kann (Su-

ganthan et al., 2005) entnommen werden. Der Wert T_0 gibt die Dauer an, die für $7 \cdot 1000000$ elementare Rechenoperationen benötigt werden. T_1 gibt die benötigte Zeit für 200000 Auswertungen der Zielfunktion an, und T_2 die Zeit, die das Optimierverfahren bei gleicher Anzahl von Funktionsauswertungen benötigt. Die Subtraktion von T_2 mit T_1 ergibt dann die Zeit, die ein Optimierverfahren benötigt, um alle Berechnungen mit Ausnahme der Auswertung der Fitnessfunktion durchzuführen. Der Quotient aus dieser Zeit und T_0 gibt dann eine relative Abschätzung der elementaren Operationen an (im Hinblick auf die $7 \cdot 1000000$ Rechenoperationen, die zum Wert T_0 geführt haben).

Es fällt auf, dass der GA wesentlich mehr Zeit benötigt, als die ES. Der relativ niedrige Wert der RII ist nicht besonders eindrucksvoll, wenn die Ergebnisse aus den vorherigen Untersuchungen mit in die Betrachtung gezogen werden. Der erschreckenden Unterschied in der Rechenzeit zwischen ES und GA, der die rechtzeitige Fertigstellung der Diplomarbeit stark behindert hat, lässt sich z.B. bei Betrachtung der Rekombination einfach erklären. MathWorks erzeugen einen Nachkommen, in dem iterativ für jede Koordinate ein Gen entsprechend einer gleichverteilten Zufallszahl aus einem der beiden Eltern übernommen wird. Der Code

```
(...)  
for j = 1:GenomeLength  
    if(rand > 0.5)  
        xoverKids(i,j) = thisPopulation(r1,j);  
    else  
        xoverKids(i,j) = thisPopulation(r2,j);  
    end  
end  
(...)
```

realisiert dies über eine `for` Schleife. Diese Implementierung ist bemerkenswert, zumal in (The MathWorks, 2005a) die Benutzer der GADS-Toolbox explizit darauf hingewiesen werden, dass Matlab auf Matrizen schneller rechnet, als auf Schleifen. Die Implementierung der diskreten Rekombination der ES, die äquivalent zu der beschriebenen Rekombination des GA ist, berechnet den Nachkommen in einem Schritt. Der entsprechende Code,

```
(...)  
perm = fix(options.rho*rand(1, GenomeLength))+ ...  
        (1:options.rho:options.rho*GenomeLength);  
recoChildren(i,1:GenomeLength) = parent(perm);  
(...)
```

ist zwar schlechter lesbar, dafür aber wesentlich effizienter, wie folgende Messung zeigt: Mit beiden Implementierungen werden 1000 Nachkommen mit Dimension 50 aus zwei Individuen $I_1 = (1, \dots, 1)$ und $I_2 = (0, \dots, 0)$ erzeugt. Die Implementierung der GA-Rekombination benötigt dazu 0.187 Sekunden, die ES-Rekombination hingegen nur 0.031 Sekunden. Wir haben also die Rechenzeit der (diskreten) Rekombination um mehr als 80% reduziert. Weil der GA wesentlich mehr Funktionsauswertungen und somit auch mehr Generationen benötigt, wird schnell klar, dass ein eklatanter Unterschied der Rechenzeiten zwischen GA und ES entsteht.

6 Interpretation der Experimente

Im Kapitel 5 wurden zu einer Parametrisierung gefunden, die den Standardparametrisierungen nachweislich überlegen ist und zum anderen ein Vergleich zwischen ES und GA durchgeführt, mit dem Ergebnis, dass die ES den GA in vielen hier betrachteten Probleminstanzen übertrifft. Dieses Kapitel wird eine Erklärung für diese Erkenntnisse und Beobachtungen liefern.

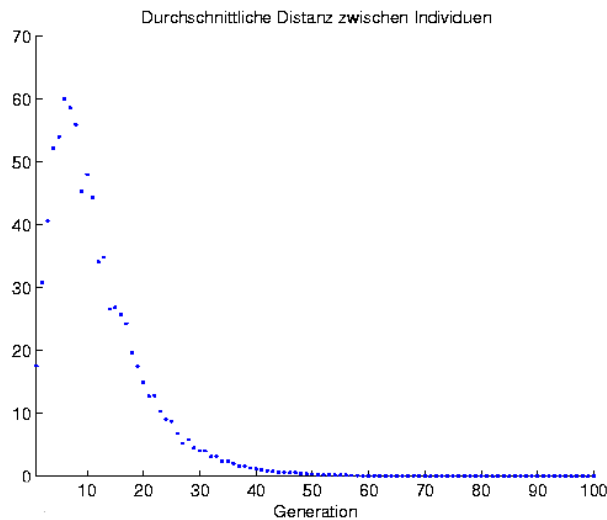
6.1 ES-Parametrisierung

Bei der Betrachtung der gefundenen Parametrisierungen fällt auf, dass die Parametrisierungen nicht von der Dimension abhängen, sondern vielmehr von der Problemstellung. Die Werte unterscheiden sich zwar (was nicht viel erstaunen auslösen sollte, weil SPO eine Heuristik ist), aber eine Grundtendenz ist zu erkennen. Es kann bereits vorweggenommen werden, dass der Parameter κ relativ hoch gewählt ist. Dies ist erstaunlich, weil zurzeit die (μ, ν) -ES *state-of-the-art* ist und propagiert wird, dass das Vergessen eine wichtige Eigenschaft in der Optimierung ist. Die Untersuchungen können dies nicht bestätigen. Die Normalverteilung verursacht mit einer gewissen Wahrscheinlichkeit größere Sprünge, so dass auch bei Anwendung der Plus-Selektion ein lokales Minimum übersprungen werden kann.

6.1.1 Parametrisierung der sphere-Funktion

An der Parametrisierung für die **sphere**-Funktion fällt auf, dass eine sehr kleine Population, ein geringer Selektionsdruck und nur eine Schrittweite benötigt werden. Wie ist das zu erklären? Die **sphere**-Funktion ist unimodal, so dass Punkte schrittweise entlang des Gefälles verschoben werden können, und so mit der Zeit das Minimum erreichen. Mit einer kleinen Population werden pro Generation nur wenige Funktionsauswertungen verbraucht, so dass sich die wenigen Punkte langsam aber stetig dem Minimum nähern können. Eine große Population würde pro Generation wesentlich mehr Funktionsauswertungen benötigen, die nutzlos wären, weil die Divergenz in der Population schnell zusammenbricht, wie der Abbildung 6.1 entnommen werden kann. Hier häufen sich dann viele Individuen in einem Bereich, so dass das Konzept der großen Population keinen Sinn mehr macht. Dennoch wird die Population größer gewählt, vor allem um die Rekombination zu ermöglichen. Dies deckt sich mit dem frühen Erkenntnis Schwefels, dass die Rekombination die Optimierung beschleunigt.

Die ES benötigt hier nur eine Schrittweite. Schrittweiten werden über die Selbstadaptation eingestellt. Das bedeutet, dass zuerst die Schrittweiten und dann die Objektkomponenten manipuliert werden. Die Güte der Objektkomponente ergibt sich dann aus der Güte der Schrittweiten. So wird bei Verwendung von n Schrittweiten eine hohe Fehlerquelle induziert, so dass der Selektionsdruck erhöht werden muss, um die Schaffung von Nachkommen zu erzeugen, deren Schrittweiten und Objektkomponenten gut eingestellt sind. Bei Minimierung der **sphere**-Funktion sind n Schrittweiten aber nicht erforderlich, so dass der Selektionsdruck gemindert werden kann, und der ES so mehr Generationen ermöglicht werden, in denen sie sich Schritt für Schritt dem Minimum nähern kann.

Abbildung 6.1: Distanz der Individuen bei Minimierung der `sphere`-Funktion

6.1.2 Parametrisierung der Rosenbrock-Funktion

Die `Rosenbrock`-Funktion besitzt ein schmales Tal, durch das sich die Individuen bewegen müssen, um zum Minimum zu gelangen. Hier werden n Schrittweiten eingesetzt, so dass sich die Strategiekomponente diesem Tal anpassen kann. Entsprechend muss der Selektionsdruck erhöht werden.

Die Funktion ist multimodal, so dass nicht davon ausgegangen werden kann, dass ein oder wenige Individuen sich iterativ dem globalen Minimum nähern können. Deshalb wird eine große Population verwendet, um Individuen möglichst breit im Suchraum zu verteilen.

6.1.3 Parametrisierung der Rastrigin-Funktion

Im Gegensatz zu den anderen Funktionen benötigt die `Rastrigin`-Funktion geringere Schrittweiten und, obwohl sie stark multimodal ist, eine kleinere Population. Die Begründung liefert der kleine Suchraum, auf dem große Schrittweiten sogar störend wären, weil die Nachkommen dann außerhalb des zulässigen Bereichs erzeugt würden. Die im Vergleich zur `Rosenbrock`-Funktion kleinere Population lässt sich analog begründen: Der kleine Suchraum wird auch mit im Durchschnitt 12 Individuen ausreichend abgedeckt.

Erneut werden n Schrittweiten verwendet, so dass der Selektionsdruck ebenfalls hoch gewählt werden muss. Ein hoher Selektionsdruck hat hier den weiteren Vorteil, dass die Wahrscheinlichkeit erhöht wird, dass Nachkommen in der Nähe der globalen Minimums erzeugt werden.

6.2 GA-Parametrisierung

Bemerkenswert an der gefundenen verbesserten GA-Parametrisierung ist, dass sie in allen Fällen von der Standardparametrisierung des Matlab-GA abweicht. Dies liegt daran, dass die Standardparametrisierung für den rcGA unüberlegt vom bcGA übernommen wurde, obwohl sich die genetischen Operatoren des rcGA und des bcGA stark unterscheiden (vgl. Kapitel 3). So wird die Rekombinationswahrscheinlichkeit wesentlich geringer gewählt, weil die Mutation der vorherrschende Operator des rcGA sein muss. Die Rekombination ist nicht in der Lage, neue Ausprägungen einer Koordinate zu erzeugen, sondern kann nur vorhandene austauschen. EliteCount wird immer größer null gewählt, weil nur so die Fitness monoton fallend über die Zeit

ist. Anderenfalls würde der GA ggf. die beste gefundene Lösung vergessen, was nicht gewollt sein kann. Die Population wird ebenfalls eher klein gewählt. Hier zeigt sich der hohe Bedarf an Generationen des rcGA. Er nähert sich von einer Generation zu nächsten nur schleppend dem Minimum, so dass er auf eine große Population zugunsten einer höheren Anzahl an Generationen verzichtet.

6.3 Vergleich zwischen ES und GA

Beim Vergleich zwischen ES und GA fällt auf, dass der GA der ES nur auf einem Problem überlegen war (der **Rosenbrock-Funktion**) und nur dann, wenn ihm ausreichend viele Funktionsauswertungen zu Verfügung standen. Ansonsten hat stets die ES die kleineren Fehler verursacht und war bei der Berechnung zudem viel schneller. Ein Grund ist darin zu finden, dass die ES von Beginn an auf der reellen Kodierung arbeitet und in über 40 Jahren stetig weiterentwickelt wurde. Der rcGA ist eine neuerer Entwicklung, die noch Weiterentwicklungsbedarf hat. So verwendet der rcGA eine Anfangsschrittweite, die er entweder konstant hält oder linear reduziert. So bleibt er aber schnell in schmalen Tälern stecken. Mit der Matlab Release 14 SP 3 ist der GADS Toolbox eine neue Mutationsfunktion hinzugefügt worden, die nun die Schrittweiten ähnlich zu $\frac{1}{5}$ -Erfolgsregel (Beyer & Schwefel, 2002) verändert. So bleibt zu hoffen, dass der rcGA durch Weiterentwicklung noch bessere Performance erreicht.

Beide Verfahren schneiden sehr schlecht bei Minimierung der **Rosenbrock-Funktion** ab, weil sie stets in einem der vielen lokalen Minima stecken bleiben. Deshalb wäre die Untersuchung des Verhaltens cauchy-ES auf dieser Funktion sehr interessant gewesen, war aber aufgrund der langen Rechendauer nicht mehr möglich.

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse und Beobachtung der Arbeit kurz zusammen und schließt mit einem Ausblick ab.

7.1 Zusammenfassung

In dieser Arbeit wurden unterschiedliche Optimierverfahren kurz vorgestellt. Der Schwerpunkt lag auf den stochastischen Suchverfahren und insbesondere den evolutionären Algorithmen. Extrem einfache stochastische Suchverfahren versprechen keinen Erfolg, so dass sie nicht angewendet werden sollten. Evolutionsstrategie und genetischer Algorithmus sind zwei leistungsfähige Verfahren, die sich über viele Parameter einstellen lassen. Die Arbeit hat gezeigt, dass die Optimierverfahren auf unterschiedlichen Problemen unterschiedliche Einstellungen benötigen, so dass eine Einstellung der Parametrisierung empfehlenswert ist. In praktischen Anwendungen sollte dann aber immer die Ungleichung:

$$\text{Zeitbedarf der Standardparametrisierung} > \text{Zeitbedarf der verbesserten Parametrisierung} + \text{Zeitbedarf der Parameteroptimierung}$$

gelten. Ansonsten ist die Parameteroptimierung nutzlos (es sei denn, sie wird in ähnlichen Anwendungen weiterverwendet), weil die Standardparametrisierung die Lösung insgesamt schneller findet.

Die Vergleiche zwischen ES und rcGA lassen den Schluss zu, dass die Erweiterung der GA von bcGA hin zum rcGA nicht in der Lage ist, die ES auch nur auf einer Probleminstanz zu übertreffen. Sie war in allen hier durchgeführten Vergleichen der ES unterlegen. Erstaunlich ist, dass der rcGA mit der Standardeinstellung $p_{rek} = 0.8$ ausgeliefert wird. Die Empfehlung, dass die Rekombinationswahrscheinlichkeit hoch gewählt werden soll, kann sicherlich nicht auf die rcGA übertragen werden, was auch die Ergebnisse der Parameteroptimierung zeigen. Die Rekombination kann im Fall der bcGA neue Ausprägungen einer Koordinate erzeugen, im Fall der rcGA können aber nur bereits bestehende Belegungen einer Koordinate ausgetauscht werden. So wird aber mit $p_{rek} = 0.8$ die Suche im zulässigen Bereich stark eingeschränkt.

Es stellt sich auch die Frage, in wie weit eine Weiterentwicklung der GA auf die reelle Kodierung Sinn macht. Die ES basieren auf dem gleichen Prinzip (der biologischen Evolution) und arbeiten auf der reellen Kodierung. Sie existieren bereits seit über 40 Jahren und wurden stetig weiterentwickelt, so dass die Frage erlaubt sein muss, ob am Ende der Weiterentwicklung von bcGA hin zu einem leistungsstarken rcGA nicht gilt:

$$\text{rcGA} = \text{ES} ?$$

7.2 Ausblick

Diese Arbeit betrachtet nur eine Auswahl der Probleminstanzen aus dem CEC 2005 Testfunktionensatz und die Optimierverfahren ES und GA. Im Laufe der Bearbeitung dieser Arbeit ist die weitere Idee entstanden, auch die cauchy-ES in die experimentellen Untersuchungen mit einzubeziehen. Leider war es aufgrund des sehr hohen Zeitbedarfs nicht mehr möglich, diese

Untersuchung durchzuführen.

Dieser hohe Zeitbedarf hätte im Nachhinein betrachtet etwas abgesenkt werden können. Der Aufwand zur Ermittlung einer guten Parametrisierung steigt exponentiell mit der Anzahl der Parameter (Bartz-Beielstein, in einem Gespräch). Für die ES wurden bei der Parameteroptimierung 9 Parameter betrachtet, davon drei qualitative Parameter (vgl. Abschnitt 5.3), die nur zwei Werte annehmen können. Es wäre effizienter gewesen, wenn 2^3 Experimente mit nur sechs quantitativen Parametern und drei fest eingestellten qualitativen Parametern durchgeführt worden wären. Tatsächlich wurden die qualitativen Faktoren aber in einem Intervall $[0, 1]$ betrachtet und somit wurde viel Information verschenkt.

Aufgrund des Zeitmangels war auch die Untersuchung der Cauchy-ES nicht mehr möglich, obwohl es sicherlich sehr interessant gewesen wäre, diese, von den Autoren hoch gelobte ES intensiver zu betrachten. Das im Rahmen dieser Arbeit entstandene Software-Paket *ESGA-Toolbox* kann sowohl die Standard-ES als auch die Cauchy-ES ausführen, so dass mit den ESGA und SPO Toolboxen die Untersuchungen leicht durchzuführen sind.

Literaturverzeichnis

- [Auger & Hansen, 2005a] Auger, A. & Hansen, N. (2005a). Performance evaluation of an advanced local search evolutionary algorithm. *Proceedings of the IEEE Congress on Evolutionary Computation*.
- [Auger & Hansen, 2005b] Auger, A. & Hansen, N. (2005b). A restart cma evolution strategy with increasing population size. *Proceedings of the IEEE Congress on Evolutionary Computation*.
- [Bartz-Beielstein, 2003] Bartz-Beielstein, T. (2003). *Experimental Analysis of Evolution Strategies: Overview and Comprehensive Introduction*. Technical Report Reihe CI 157/03, SFB 531, Universität Dortmund, Dortmund, Germany.
- [Bartz-Beielstein, 2005] Bartz-Beielstein, T. (2005). *New Experimentalism Applied to Evolutionary Computation*. Dissertation am Fachbereich Informatik, Universität Dortmund.
- [Bartz-Beielstein & Markon, 2004] Bartz-Beielstein, T. & Markon, S. (2004). Tuning search algorithms for real-world applications: A regression tree approach. *Congress on Evolutionary Computation*.
- [Bäck, 1994] Bäck, T. (1994). *Evolutionary Algorithms in Theory and Practise*. Dissertation am Fachbereich Informatik, Universität Dortmund.
- [Bäck et al., 1997] Bäck, T., Hammel, U., & Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state.
- [Bäck & Schwefel, 1996] Bäck, T. & Schwefel, H.-P. (1996). Evolutionary computation: An overview.
- [Beyer & Deb, 2001] Beyer, H. & Deb, K. (2001). On self-adaptive features in real-parameter evolutionary algorithms.
- [Beyer, 2001] Beyer, H.-G. (2001). *The theory of evolution strategies*. Springer, Heidelberg.
- [Beyer & Schwefel, 2002] Beyer, H.-G. & Schwefel, H.-P. (2002). *Evolution strategies – A comprehensive introduction*. Kluwer Academic Publishers, Bosten.
- [Czarn et al., 2003] Czarn, A., MacNish, C., Vijayan, K., Turlach, B., & Gupta, R. (2003). *Statistical Exploratory Analysis of Genetic Algorithms*. Technical Report UWA-CSSE-03-001.
- [Dean & Voss, 1999] Dean, A. & Voss, D. (1999). *Design and analysis of experiments*. Springer, New York.
- [Deb & Beyer, 1999] Deb, K. & Beyer, H. (1999). Self-adaptation in real-parameter genetic algorithms with simulated binary crossover.
- [Gerdes et al., 2004] Gerdes, I., Klawonn, R., & Kruse, R. (2004). *Evolutionäre Algorithmen*. Vieweg, Wiesbaden.

-
- [Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley Publishing Company, Inc., New York.
- [Graef, 2004] Graef, F. (2004). *Mathematische statistik für informatiker und ingenieure*.
- [Herrera et al., 1998] Herrera, F., Lozano, M., & Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4), 265–319.
- [Hoffmeister & Bäck, 1992] Hoffmeister, F. & Bäck, T. (1992). *Genetic Algorithms and Evolution Strategies: Similarities and Differences*. Technical Report SYS-1/92.
- [Hoss & Stützle, 2005] Hoss, H. & Stützle, T. (2005). *Stochastic Local Search – Foundations and Applications*. Morgan Kaufmann Publishers, New York.
- [Liang et al., 2006] Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello, C. A. C., & Deb, K. (2006). *Problem Definitions and Evaluation Criteria for the CEC 2006 – Special Session on Constrained Real-Parameter Optimization*. Technical Report, Nanyang Technological University, Singapore.
- [McKinnon, 1996] McKinnon, K. (1996). Convergence of the nelder-mead simplex method to a non-stationary point.
- [Mühlenbein & Schlierkamp-Voosen, 1993] Mühlenbein, H. & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1(1), 25–49.
- [Michalewicz, 1992] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin.
- [Nelder & Mead, 1965] Nelder, J. & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7, 4 (Jan. 65), 308–313.
- [Neumann & Morlock, 1993] Neumann, K. & Morlock, M. (1993). *Operations Research*. Hanser, München.
- [Nocedal & Wright, 1999] Nocedal, J. & Wright, S. J. (1999). *Numerical Optimization*. Springer, New York.
- [Ono et al., 2003] Ono, I., Kita, H., & Kobayashi, S. (2003). A real-coded genetic algorithm using the unimodal normal distribution crossover. (pp. 213–237).
- [Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie*. Friedrich Frommann Verlag, Stuttgart.
- [Rudolph, 1992] Rudolph, G. (1992). On correlated mutations in evolution strategies. In R. Männer & B. Manderick (Eds.), *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)* (pp. 105–114). Amsterdam: Elsevier.
- [Rudolph, 1998] Rudolph, G. (1998). Asymptotical convergence rates of simple evolutionary algorithms under factorizing mutation distributions. *Lecture Notes in Computer Science*, 1363, 223 ff.
- [Schöneburg et al., 1994] Schöneburg, E., Heinzmann, F., & Feddersen, S. (1994). *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, Bonn.

- [Schonlau, 1997] Schonlau, M. (1997). *Computer Experiments and Global Optimization*. PhD Thesis, University Waterloo, Canada.
- [Schwefel, 1977] Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser Verlag, Basel.
- [Schwefel, 1994] Schwefel, H.-P. (1994). *Evolution and Optimum Seeking*. Wiley, New York.
- [Schwefel & Rudolph, 1995] Schwefel, H.-P. & Rudolph, G. (1995). Contemporary evolution strategies. In *European Conference on Artificial Life* (pp. 893–907).
- [Suganthan et al., 2005] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., & Tiwari, S. (2005). *Problem Definitions and Evaluation Criteria for the CEC 2005 – Special Session on Real-Parameter Optimization*. Technical Report, Nanyang Technological University, Singapore.
- [The MathWork, Inc., 2001] The MathWork, Inc. (2001). *Statistics Toolbox – User’s Guide Version 3*. The MathWork Inc., Natick.
- [The MathWork, Inc., 2005a] The MathWork, Inc. (2005a). *Genetic Algorithm and Direct Search Toolbox – User’s Guide Version 2*. The MathWork Inc., Natick.
- [The MathWork, Inc., 2005b] The MathWork, Inc. (2005b). *MATLAB – Getting Started with MATLAB, Version 7*. The MathWork Inc., Natick.
- [Wegener, 2003] Wegener, I. (2003). *Komplexitätstheorie*. Springer, Berlin.
- [Weicker, 2002] Weicker, K. (2002). *Evolutionäre Algorithmen*. B. G. Teubner, Stuttgart.
- [Whitley et al., 1995] Whitley, D., Mathias, K., Rana, S., & Dzubera, J. (1995). Building better test functions.
- [Whitley et al., 1996] Whitley, D., Rana, D., Dzubera, J., & Mathias, E. (1996). Evaluating evolutionary algorithms.
- [Wright, 1991] Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In G. J. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 205–218). San Mateo, CA: Morgan Kaufmann.
- [Yao & Liu, 1997] Yao, X. & Liu, Y. (1997). Fast evolution strategies. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, & R. Eberhart (Eds.), *Evolutionary Programming VI* (pp. 151–161). Berlin: Springer.

Abbildungsverzeichnis

| | | |
|------|---|----|
| 2.1 | Minima einer Funktion | 7 |
| 3.1 | Exemplarischer Ablauf der Koordinatenstrategie | 18 |
| 3.2 | Ablauf (links) und Operatoren (rechts) des Simplexverfahrens | 20 |
| 3.3 | Zyklus eines EA | 24 |
| 3.4 | Mögliche Verteilung der Nachkommen bei einer Schrittweite | 25 |
| 3.5 | Mögliche Verteilung der Nachkommen bei n Schrittweiten | 25 |
| 3.6 | Visualisierung der (a) diskreten Rekombination, (b) intermediären Rekombination | 29 |
| 3.7 | Dichte einer logarithmischen Normalverteilung | 31 |
| 3.8 | Dichte der Normalverteilung | 32 |
| 3.9 | Gegenüberstellung der Dichten von Cauchy- und Normalverteilung | 35 |
| 3.10 | Roulettrad Selektion (links), uniform stochastische Selektion (rechts) | 41 |
| 4.1 | Beziehung der Funktionen des GA* zueinander | 51 |
| 4.2 | Inverse Cauchyverteilung | 59 |
| 4.3 | (links) fehlerhafter, (rechts) korrekter Verlauf der ES | 65 |
| 4.4 | Verlauf der ES unter Beachtung von Nebenbedingungen | 67 |
| 4.5 | Verlauf der Optimierung für 50 (links), 100 (rechts) und 500 (unten) Dimensionen | 68 |
| 4.6 | Vergleich zweier unterschiedlicher Implementierungen der ES | 69 |
| 4.7 | Vergleich zwei unterschiedlicher Implementierungen der ES | 70 |
| 4.8 | Histogramm (blau) und Dichte (rot) des Generators bzw. der Cauchyverteilung | 71 |
| 5.1 | Boxplot einer Probe | 79 |
| 5.2 | Histogramm einer Probe | 80 |
| 5.3 | Visualisierung der Ergebnisse für die 10-dimensionale sphere -Funktion | 85 |
| 5.4 | Histogramm der Proben mit passend überlagerter Dichte der Normalverteilung | 85 |
| 5.5 | RLD der zwei unterschiedlich eingestellter ES | 86 |
| 5.6 | Visualisierung der Ergebnisse für die 10-dimensionale sphere -Funktion | 88 |
| 6.1 | Distanz der Individuen bei Minimierung der sphere -Funktion | 96 |

Tabellenverzeichnis

| | | |
|------|---|----|
| 3.1 | Bedeutung der Biologischen Begriffe in der Optimierung mit EA | 22 |
| 3.2 | Steuerparameter einer ES | 37 |
| 3.3 | Steuerparameter eines rcGA | 45 |
| 3.4 | Interpretation von Standardbinär- und Gray-Code | 45 |
| 4.1 | Einstellungsmöglichkeiten für den GA* | 49 |
| 5.1 | Probleminstanz 10-dimensionale sphere -Funktion für die ES | 81 |
| 5.2 | Ceiling Effekt bei 100000 Funktionsauswertungen | 83 |
| 5.3 | Korrigiertes Problemdesign: 10-dimensionale sphere -Funktion für die ES | 83 |
| 5.4 | Algorithmendesign für die ES | 84 |
| 5.5 | Verbesserter Designpunkt x_p^* für die ES und die 10-dimensionale sphere -Funktion | 84 |
| 5.6 | Probleminstanz 10-dimensionale sphere -Funktion für den GA | 86 |
| 5.7 | Algorithmendesign für den GA | 87 |
| 5.8 | Verbesserter Designpunkt x_p^* für den GA und die 10-dimensionale sphere -Funktion | 87 |
| 5.9 | Problemdesign der restlichen Testfunktionen für die ES | 88 |
| 5.10 | Verbesserte Parametrisierungen der ES für die Probleminstanzen 1 – 9 | 89 |
| 5.11 | Problemdesign der restlichen Testfunktionen für den GA | 89 |
| 5.12 | Verbesserte Parametrisierungen des GA für die Probleminstanzen 1 – 9 | 90 |
| 5.13 | Fehler unterschiedlicher Optimierverfahren nach 1000 Funktionsauswertungen | 90 |
| 5.14 | Fehler unterschiedlicher Optimierverfahren nach 10000 Funktionsauswertungen | 91 |
| 5.15 | Fehler unterschiedlicher Optimierverfahren nach 100000 Funktionsauswertungen | 91 |
| 5.16 | Von der ES benötigte Funktionsauswertungen zum Erreichen einer bestimmten Genauigkeit | 92 |
| 5.17 | Komplexität der Algorithmen für 10, 30 und 50 Dimensionen | 93 |

Liste der Algorithmen

| | | |
|-----|---|----|
| 3.1 | Einfache stochastische Suche | 21 |
| 3.2 | $(\mu, \kappa, \lambda, \rho)$ -Evolutionsstrategie | 27 |
| 3.3 | Genetischer Algorithmus | 38 |
| 3.4 | Roulettrad Selektion | 40 |
| 3.5 | Stochastisches universelles sampling | 41 |

Abkürzungsverzeichnis

| | |
|--------------|--|
| bcGA | Binär codierter genetischer Algorithmus |
| CEC | Congress on Evolutionary Computation |
| DACE | Design and Analysis of Computer Experiments |
| DOE | Statistische Versuchsplanung (engl. Design of Experiments) |
| EA | Evolutionäre Algorithmen |
| ES | Evolutionstrategie |
| GA | Genetischer Algorithmus |
| GADS Toolbox | Genetic Algorithm and Direct Search Toolbox |
| GA* | In der GADS Toolbox implementierter GA |
| KS-Test | Kolmogorov-Smirnov Test |
| LHD | Latin Hypercube Design |
| LHS | Latin Hypercube Sampling |
| <i>NP</i> | Nondeterministic polynomial time |
| rcGA | Reellwertig codierter genetischer Algorithmus |
| RII | Einfaches stochastisches Suchverfahren (Randomised Iterative Improvement) |
| RLD | Run length distribution |
| SPO | Sequentielle Parameter Optimierung (engl. Sequential Parameter Optimization) |
| SPOT | Sequential Parameter Optimization Toolbox |
| TSP | Traveling Salesperson Problem |

Verzeichnis der verwendeten Symbole

| | |
|---------------------|--|
| \mathbb{B} | Menge der binären Zahlen |
| $\#$ | Anzahl der Elemente in einer Menge |
| $C(0, a)$ | Cauchyverteilte Zufallszahl mit Zentrum 0 und Breitenparameter a |
| $e^{N(0, \tau)}$ | Logarithmisch normalverteilte Zufallszahl |
| $H_f(x)$ | Hesse-Matrix der Funktion f an dem Punkt x |
| I | Einheitsmatrix |
| $\mathcal{L}(x, u)$ | Lagrange Funktion |
| \mathbb{N} | Menge der natürlichen Zahlen ohne die Null |
| \mathbb{N}_0 | Menge der natürlichen Zahlen mit der Null |
| $N(0, \sigma)$ | Normalverteilte Zufallszahl mit Erwartungswert 0 und Standardabweichung σ |
| $\nabla f(x)$ | Gradient der Funktion f an dem Punkt x |
| \mathbb{R} | Menge der reellen Zahlen |
| \mathbb{R}_+ | Menge der echt positiven reellen Zahlen |
| U_n | Gleichverteilte, diskrete Zufallszahl aus der Menge $\{1, n\}$ |
| $U[a, b]$ | Gleichverteilte, kontinuierliche Zufallszahl aus dem Intervall $[a, b]$ |
| $U_\epsilon(x)$ | ϵ -Umgebung des Punktes x |