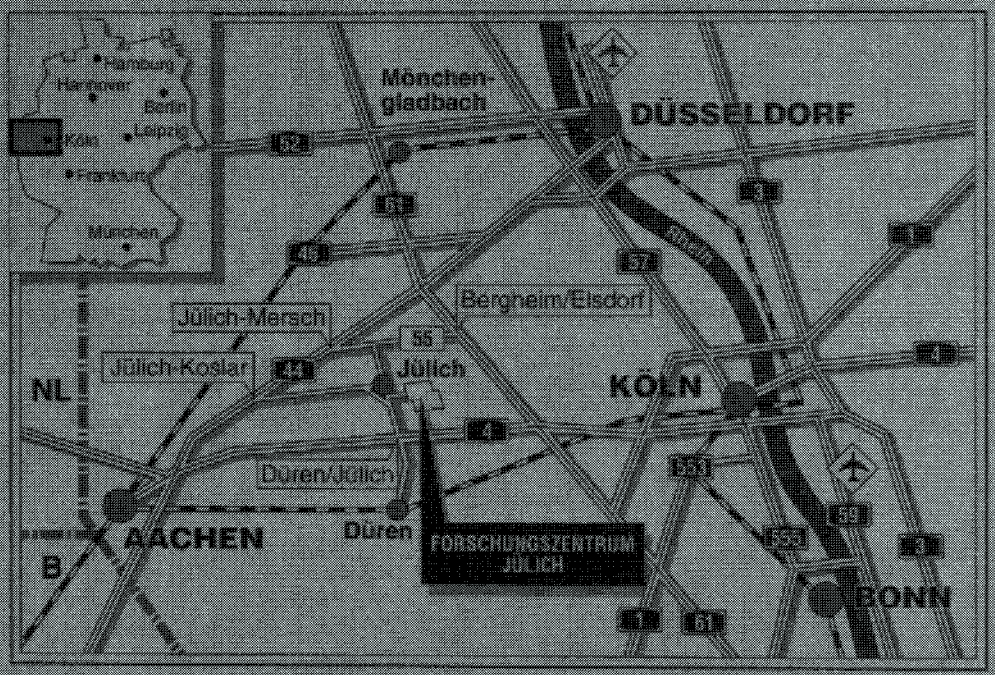
FORSCHUNGSZENTRUM JÜLICH GMBH
KFA

Zentralinstitut für Angewandte Mathematik

Parallelisierung nichtlinearer Optimierungsverfahren am Beispiel eines Raffineriemodells

Johannes Faassen



Berichte des Forschungszentrums Jülich ; 2969
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2969

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Parallelisierung nichtlinearer Optimierungsverfahren am Beispiel eines Raffineriemodells

Johannes Faassen

Kurzfassung

Im Rahmen der vorliegenden Arbeit werden Parallelisierungsansätze für Quasi-Newton-Verfahren zur nichtlinearen Optimierung diskutiert und die Implementationen einiger Verfahrensvarianten für den Intel Paragon XP/S 10 verglichen. Der Vergleich erfolgt am Beispiel eines ökonomischen Modells einer Erdölraffinerie mit linearen Nebenbedingungen und nichtlinearer Zielfunktion. Dabei werden die beschränkten Probleme mit Hilfe der "Method of Multipliers" in Folgen unbeschränkter Probleme transformiert. Die daraus resultierenden unbeschränkten Probleme werden mit dem BFGS-Verfahren und einigen Varianten des Straeter-Verfahrens jeweils in Kombination mit verschiedenen Methoden zur eindimensionalen Optimierung gelöst. Gegenstand der Betrachtungen ist insbesondere der Einfluß der eindimensionalen Optimierung auf die Iterationszahlen und die Rechenzeiten. Bei den untersuchten Problemen und den gewählten Implementationen erweist sich das BFGS-Verfahren sowohl im Sequentiellen als auch im Parallelen gegenüber dem Straeter-Verfahren als überlegen.

Abstract

This report discusses parallelizations of quasi-Newton methods for nonlinear optimization and compares implementations of some methods on the Intel Paragon XP/S 10. The comparision is done with an economic model of an oil refinery with linear constraints and nonlinear objective function. Using the method of multipliers, the constrained problems are transformed into sequences of unconstrained problems. The resulting unconstrained problems are solved by the BFGS method and some variants of Straeter's method, each in combination with different methods for the one-dimensional subproblems. Especially, the influence of one-dimensional optimization on the number of iterations and the performance is analysed. The computational experiments show that the BFGS method is superior to Straeter's method in serial as well as in parallel computation for the considered problems and the chosen implementations.

Inhalt

Kapitel 1

Einleitung

Die Optimierung umfaßt ein sehr weites Gebiet der Mathematik, in dem nach "besten" Lösungen für mathematisch formulierte Probleme gesucht wird. Optimierungsanwendungen finden sich in vielen Bereichen der Ingenieur- und Naturwissenschaften, aber auch in ökonomischen Entscheidungsprozessen und in der Mathematik selbst. Beispiele solcher Anwendungen sind Regelungs- und Steuerungsaufgaben, die Bestimmung chemischer Gleichgewichte, Layout von VLSI-Chips oder die Planung einer kostengünstigsten Lagerhaltung.

Die Suche nach effektiven numerischen Methoden zur Lösung von Optimierungsproblemen wurde wie auch andere numerische Methoden maßgeblich durch die Entwicklung der Digitalrechner beeinflußt. Vor 1940 war das Wissen um Optimierungsverfahren noch recht gering, und es wurden nur vereinzelt in der Physik und der theoretischen Chemie sehr einfache Verfahren eingesetzt. Das mehrdimensionale Newton-Verfahren war zwar schon bekannt, konnte aber wegen des hohen numerischen Aufwands praktisch nicht verwendet werden. In den vierziger und fünfziger Jahren richteten sich die Hauptbemühungen auf die Entwicklung von Verfahren zur linearen Optimierung. Erst 1959 gelang durch eine Veröffentlichung von Davidon [Da59] der Durchbruch auf dem Gebiet der nichtlinearen Optimierung, welche die Minimierung einer nichtlinearen Funktion von mehreren Veränderlichen zum Ziel hat. Das von Davidon beschriebene Verfahren der variablen Metrik legte den Grundstein zu einer großen Klasse von Optimierungsverfahren, den sogenannten Quasi-Newton-Verfahren, die in dieser Arbeit im Mittelpunkt der Untersuchungen stehen. Der Wunsch nach schnelleren Lösungsverfahren und die Verfügbarkeit von Parallelrechnern führten in jüngster Zeit zu Bemühungen, Verfahren zu entwickeln, die speziell für den Einsatz auf Parallelrechnern zugeschnitten sind. Im Gegensatz zu den sequentiellen Verfahren, die schon relativ ausgereift sind, befindet sich die Entwicklung paralleler Optimierungsverfahren noch in den Anfängen. Deshalb lohnt ein Vergleich zwischen für Parallelrechner entwickelten Verfahren und den Parallelisierungen sequentieller Verfahren.
Ein umfassender Vergleich von Optimierungsverfahren und ihrer Implementationen

auf Parallelrechnern ist jedoch im Rahmen einer Diplomarbeit nicht möglich, da die Leistungsfähigkeit solcher Verfahren stark vom vorgegebenen Problem, dem zugrundeliegenden Parallelrechner und der gewählten Implementation abhängt. Deshalb erfolgt hier der Vergleich einiger Verfahren exemplarisch am Beispiel eines konkreten Problems aus der nichtlinearen Optimierung und jeweils einer speziellen Implementationsvariante auf dem Intel Paragon XP/S 10.

In Kapitel 2 wird das zugrundeliegende Optimierungsproblem, die Prozeßoptimierung einer Raffinerie, dargestellt. Für die durch dieses Problem definierte Problemklasse wird in den Kapiteln 3 und 4 die benötigte Theorie entwickelt. Speziell in Kapitel 3 soll ein detaillierter Einblick in die Eigenschaften der benutzten Verfahren gegeben werden, der für die Interpretation der Meßergebnisse und das Verständnis der Verfahren notwendig ist. Kapitel 5 behandelt die Anwendung der entwickelten Methoden auf das Raffineriemodell und die auftretenden numerischen Probleme. Neben einem Nachweis für die Konvexität des Problems werden auch die ersten sequentiellen Meßergebnisse präsentiert. Kapitel 6 beschäftigt sich ausführlich mit der Parallelisierung der Optimierungsverfahren. Es werden allgemeine Konzepte und spezielle Ansätze für die Implementation auf dem Intel Paragon XP/S 10 vorgestellt. Weiterhin erfolgt die Diskussion der parallelen Meßergebnisse und eine Bewertung der benutzten Verfahren. Die Arbeit schließt mit Kapitel 7, wo neben einer Zusammenfassung der Ergebnisse ein Ausblick auf Verbesserungsmöglichkeiten der Verfahren und ihrer Implementationen gegeben wird.

Kapitel 2

Das Raffineriemodell

Das im folgenden beschriebene Modell einer Raffinerie ist ein Beispiel aus einer Klasse von Optimierungsproblemen, auf die sich die Hauptaufmerksamkeit in dieser Arbeit richtet. Es wurde deswegen ausgewählt, weil es für ökonomische Modellierungen typische Merkmale aufweist.

Als Import in die Raffinerie werden Rohöl und Fremdwasserstoff berücksichtigt, die mit einer Reihe von Verarbeitungstechniken zu den Endprodukten Benzin, Diesel, Heizöl und Petrolkoks umgesetzt werden sollen. Die Endprodukte gehen als Export aus der Raffinerie heraus und bringen einen gewissen Ertrag.
In Abb. 2.1 ist die Verknüpfung der insgesamt 18 Verarbeitungstechniken (Verfahren) mit den internen Mengenströmen schematisch dargestellt. In jedem Verfahren werden gewisse Produkte zu wiederum anderen Produkten weiterverarbeitet. Ausnahmen davon sind die Verfahren REIMPR und REIMHG sowie PBOUT, PLDOUT, PLOOUT und CPOUT, die nur die Menge der importierten bzw. exportierten Produkte bestimmen. Die Produktionsraten, mit denen die einzelnen Verfahren ablaufen, können vom Betreiber der Raffinerie festgelegt werden, wohingegen die Verhältnisse zwischen den Inputs und den Outputs eines Verfahrens invariant sind. Die Produktionsraten der Verfahren dürfen allerdings nicht in beliebigen Relationen zueinander eingestellt werden, da für jedes Produkt die Summe der Outputs aus allen Verfahren mindestens so groß sein muß wie die Summe der Inputs. Ebenso können die Verfahren nicht rückwärts laufen, was bedeutet, daß keine negativen Produktionsraten erlaubt sind.
Wenn man das berücksichtigt und die Produktionsraten x_i als die Problemvariablen wählt, dann ergeben sich in natürlicher Weise zwei Sorten von Restriktionen. Zum einen sogenannte Bounds $x_i \geq 0$, die besagen, daß die Verfahren nicht rückwärts laufen können, und lineare Ungleichungs-Restriktionen $a_j^T x \geq 0$, die sicherstellen, daß von keinem Produkt mehr in Verfahren eingebracht wird als produziert wurde.

Unter Berücksichtigung dieser Nebenbedingungen soll nun der Gesamtnutzen der Raffinerie maximiert werden, was hier eine Gewinnmaximierung bedeutet. Dazu

führt man eine Zielfunktion ein, deren Werte das gewünschte Kostenmaß modellieren. Die Zielfunktion $Z(x)$ ergibt sich aus dem Erlös $N(x)$ der verkauften Produkte abzüglich der Kosten $K(x)$, die bei der Produktion entstehen, d.h.

$$Z(x) = N(x) - K(x).$$

Die Kosten für die einzelnen Verfahren sollen proportional zu den Produktionsraten sein, was insgesamt bedeutet, daß $K(x) = c^T x$ ist. Über die Verfahren REIMHG und REIMPR, die die Menge des importierten Wasserstoffs und Rohöls festlegen, werden die Kosten für die Ausgangsstoffe berücksichtigt. Da hier alle Verfahrenskosten als linear angenommen wurden, sind also auch die Kosten für die Rohstoffe linear.
Für die verkauften Produkteinheiten hingegen sollen Marktsättigungseffekte berücksichtigt werden, so daß der Erlös nicht proportional zur abgesetzten Menge ist, sondern relativ gesehen geringer wird, je mehr von einem Produkt auf den Markt kommt.

In diesem Modell wird diese Abhängigkeit mit Wurzelfunktionen der Gestalt

$$N(x) = \frac{1}{1-\alpha} x^{1-\alpha} \qquad \text{mit} \quad \alpha \in (1,0) \tag{2.1}$$

modelliert. Der Parameter α ist dabei ein Maß für die Preisreagiblität, die um so größer ist, je größer α ist. Hier wurde speziell $\alpha = 0.2$ für Benzin und Diesel bzw. $\alpha = 0.1$ für Heizöl und Petrolkoks gewählt, wodurch für alle Produkte die Marktsättigungseffekte als gering angenommen werden.

Das gesamte Modell läßt sich formal beschreiben durch das nichtlineare Optimierungsproblem

$$\begin{aligned}
&\text{maximiere} \qquad Z(x) = \sum_{i=1}^{4} \tfrac{1}{1-\alpha_i} x_i^{1-\alpha_i} - c^T x \qquad x \in \mathbb{R}^{18} \\
&\text{mit den Nebenbedingungen} \\
&x_i \geq 0 \qquad\quad i = 1, 2, \ldots, 18 \\
&Ax \geq 0 \qquad\quad A \in \mathbb{R}^{12 \times 18}
\end{aligned} \tag{2.2}$$

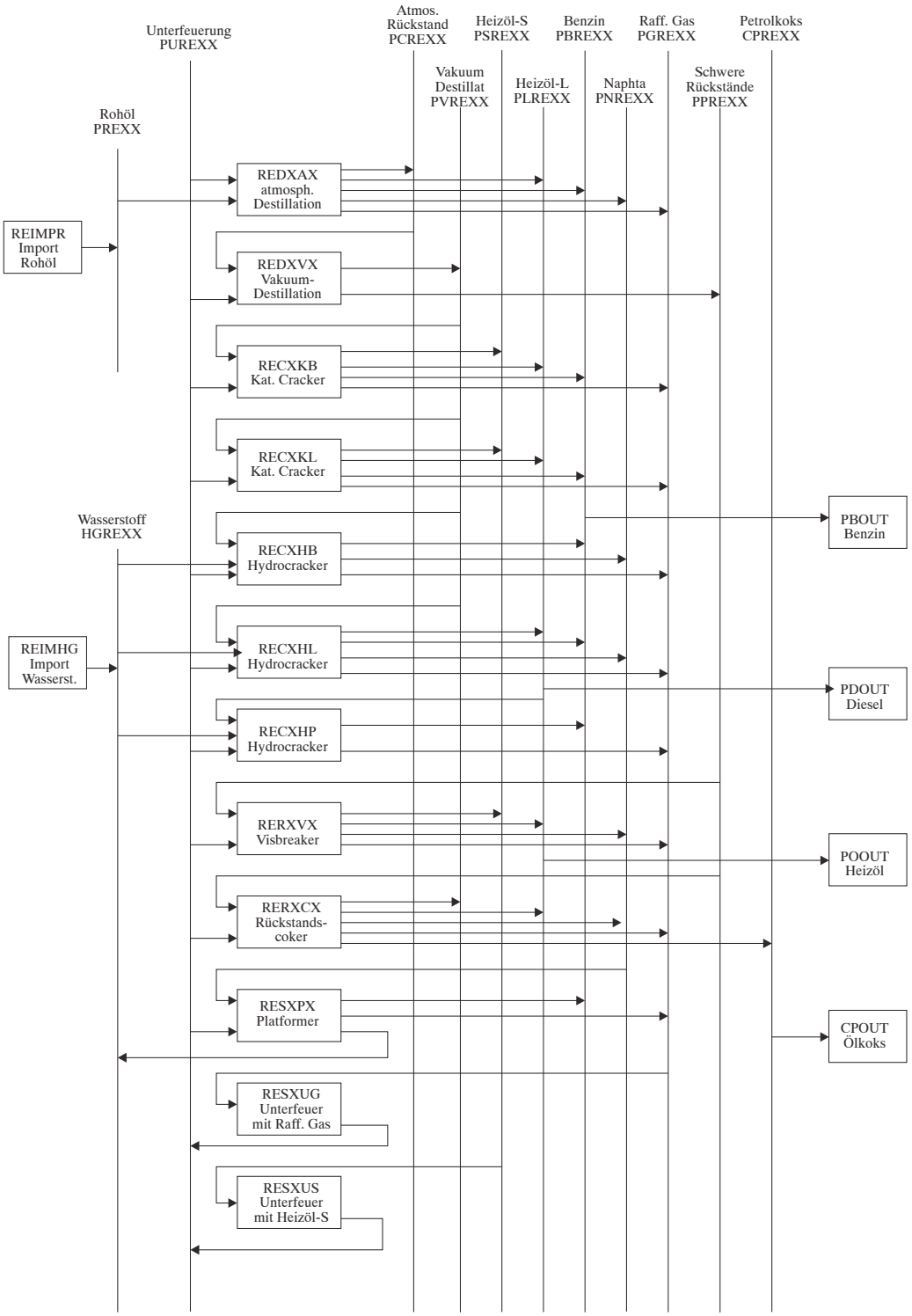


Abbildung 2.1: Schemabild der Raffinerie

Kapitel 3

Optimierung ohne Nebenbedingungen

Ziel dieses Kapitels ist es, die mathematischen Grundlagen der Optimierung so weit darzustellen, wie es für das Verständnis der in dieser Arbeit benutzten Verfahren notwendig ist. Für ausführlichere Darstellungen und Beweise zu den angegebenen Sätzen seien die Bücher [Fl87, GrTe93, Lue73, JoTr88] empfohlen.

Obwohl das zu lösende Ausgangsproblem (2.2) ein Optimierungsproblem mit Nebenbedingungen ist, werden die Nebenbedingungen zunächst vernachlässigt und es wird erst in Kapitel 4 beschrieben, wie sich die Theorie auf Probleme mit Nebenbedingungen übertragen läßt. Ferner werden im weiteren nur noch Minimierungsprobleme behandelt, obwohl das Ausgangsproblem als Maximierungsaufgabe formuliert war. Jedes Maximierungsproblem kann jedoch durch einen Vorzeichenwechsel der Zielfunktion in ein äquivalentes Minimierungsproblem umgewandelt werden.

3.1 Mathematische Grundlagen

Es sei $f : \mathbb{R}^n \to \mathbb{R}$ zweimal stetig differenzierbar. Ausgangsproblem des gesamten Kapitels 3 ist nun:

$$f(x) \to \min \qquad \text{mit} \quad x \in \mathbb{R}^n \tag{3.1}$$

Es ist also das Minimum der Funktion f auf ihrem Definitionsbereich $\mathbb{R}^n$ gesucht.

Bevor man Methoden entwickelt, um solche Probleme zu lösen, ist es naheliegend, zunächst geeignete Charakterisierungen für Minima bzw. optimale Lösungen zu entwickeln.

3.1.1 Optimalitätskriterien

Definition 3.1 Sei f wie in Ausgangsproblem (3.1) gegeben. Ein Punkt x^* heißt *lokales Minimum* von f, wenn es eine offene Umgebung U von x^* gibt, mit:

$$f(x^*) \leq f(x) \qquad \text{für alle } x \in U.$$

x^* heißt *globales Minimum* von f, wenn

$$f(x^*) \leq f(x) \qquad \text{für alle } x \in I\!R^n.$$

x^* wird dann auch *Lösung* des Problems (3.1) genannt.

Der folgende Satz liefert notwendige und hinreichende Bedingungen für lokale Minima.

Satz 3.1 Sei x^* ein lokales Minimum von f, dann gilt

$$\nabla f(x^*) = 0. \tag{3.2}$$

Gilt umgekehrt $\nabla f(x^*) = 0$ und zusätzlich

$$\nabla^2 f(x^*) \quad \text{ist positiv definit,} \tag{3.3}$$

dann ist x^* lokales Minimum von f.

Die Bedingung (3.2) besagt lediglich, daß an der betreffenden Stelle der Gradient der Funktion null ist. Solche Punkte werden auch als stationäre Punkte bezeichnet. Es muß sich dann aber nicht um ein Minimum handeln, sondern es kann auch ein Sattelpunkt vorliegen. Die Bedingung (3.3) schließt die Möglichkeit eines Sattelpunktes aus.

Im Satz 3.1 werden allerdings nur Aussagen über *lokale* Minima gemacht. Da man aber an einer Lösung für das Ausgangsproblem (3.1) interessiert ist, also *globale* Minima sucht, kann man zusätzliche Forderungen an die Zielfunktion stellen, um sicherzustellen, daß jedes lokale Minimum auch ein globales Minimum ist. Eine Eigenschaft, die das sicherstellt, ist die Konvexität.

Definition 3.2 Eine Funktion $f : I\!R^n \to I\!R$ heißt *konvex*, wenn für alle $x, y \in I\!R^n$ gilt:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \qquad \text{für alle} \quad \lambda \in (0,1) \tag{3.4}$$

Gilt die Ungleichung (3.4) für alle $x \neq y$ strikt, dann heißt f *strikt konvex*.

Satz 3.2 $f : \mathbb{R}^n \to \mathbb{R}$ ist genau dann konvex, wenn $\nabla^2 f(x)$ positiv-semi-definit ist für alle $x \in \mathbb{R}^n$. Falls $\nabla^2 f(x)$ positiv definit ist für alle $x \in \mathbb{R}^n$, so ist f strikt konvex.

Mit Satz 3.2 erhält man das gewünschte Ergebnis:

Satz 3.3 Ist f in dem Ausgangsproblem (3.1) konvex, dann ist jedes lokale Minimum von f auch ein globales Minimum. Falls f strikt konvex ist, ist die Lösung für (3.1) sogar eindeutig.

Dieser Satz stellt zwar sicher, daß jeder stationäre Punkt eine Lösung des Optimierungsproblems ist, nicht aber die Existenz einer Lösung überhaupt. Es wäre nämlich denkbar, daß die Funktion f auf $\mathbb{R}^n$ nicht nach unten beschränkt ist. Die Existenz einer Lösung ergibt sich allerdings in den meisten Fällen in natürlicher Weise aus der Modellierung, da es z.B. unmittelbar einleuchtet, daß Gewinne nicht unendlich groß werden können.

Die Konvexität der Zielfunktion ist eine sehr wünschenswerte Eigenschaft. Bei Problemen mit mehreren lokalen Minima kann es nämlich nötig sein, daß man alle lokalen Minima auffinden und miteinander vergleichen muß, bevor man entscheiden kann, welches nun die optimale Lösung darstellt. Die systematische Suche nach allen lokalen Minima ist in der Regel aber sehr aufwendig oder gar nicht möglich, so daß in solchen Fällen vielfach stochastische Suchverfahren benutzt werden.
In Kapitel 5 wird gezeigt werden, daß das mathematische Modell der Raffinerie die Voraussetzungen der Konvexität erfüllt. Deshalb werden sich alle im weiteren dargestellten Ansätze darauf beschränken, einen stationären Punkt zu finden.

3.1.2 Abstiegsverfahren und Konvergenz

Nachdem nun Kriterien zur Verfügung stehen, mit denen man von einem Punkt entscheiden kann, ob er ein Minimum ist, soll jetzt das grundsätzliche Vorgehen zum Auffinden eines Minimums dargestellt werden. Da es i. allg. nicht möglich ist, das Problem (3.1) analytisch zu lösen, werden numerische Verfahren benutzt, welche eine Folge von Iterationspunkten x^k erzeugen, die gegen die Lösung konvergieren.

Die Art und Weise, wie die Folge x^k erzeugt wird, ist von Verfahren zu Verfahren verschieden. Aber fast alle Verfahren lassen sich als sogenannte Abstiegsverfahren interpretieren. Abstiegsverfahren machen nichts anderes als das, was man intuitiv auch machen würde, wenn man von einem Berg ins Tal finden will: "Man macht sukzessive Schritte talwärts, bis man unten im Tal angelangt ist."

Etwas formaler läßt sich diese zunächst heuristische Vorgehensweise im folgenden Iterationsschema wiederfinden:

1. Bestimme eine Abstiegsrichtung d^k.
2. Finde das α^k, das $f(x^k + \alpha d^k)$ minimiert.
3. Setze $x^{k+1} = x^k + \alpha^k d^k$.
4. Prüfe, ob x^{k+1} ein Minimum ist, wenn nicht, dann gehe zu 1., sonst stoppe.

Die eindimensionale Minimierung in Schritt 2 wird dabei als Strahlminimierung bezeichnet. Hauptunterscheidungsmerkmal der verschiedenen Verfahren ist allerdings nicht die Technik der Strahlminimierung, sondern primär die Wahl der Abstiegsrichtung. Die Bestimmung geeigneter Abstiegsrichtungen ist Thema eines späteren Abschnitts. Hier sollen zunächst einige grundlegende Eigenschaften solcher Abstiegsverfahren zusammengestellt werden.

Eine interessante Frage ist, unter welchen Bedingungen ein Abstiegsverfahren überhaupt konvergiert bzw. welche Eigenschaften die Strahlminimierung und die Suchrichtungen dazu haben müssen. Die Forderung, in jedem Schritt die Zielfunktion zu erniedrigen, d.h. $f(x^{k+1}) < f(x^x)$, ist sicherlich nicht hinreichend, um die Konvergenz gegen eine optimale Lösung zu gewährleisten, da es denkbar wäre, daß für große k die Verbesserungen beliebig klein werden, d.h. $|f(x^{k+1}) - f(x^k)| \to 0$. Deshalb muß man sowohl an die Suchrichtung als auch an die Strahlminimierung zusätzliche Forderungen stellen.

Sei nun zunächst die Strahlminimierung Gegenstand der Betrachtung. Ziel der Strahlminimierung ist es, eine Schrittweite $\alpha^k \in \mathbb{R}_+$ zu finden, die $f(x)$ auf dem Strahl $x^k + \alpha d^k$ minimiert. Da man i. allg. das Minimum nicht exakt bestimmen kann, wird man sich mit einer Approximation begnügen. Hier soll jetzt diskutiert werden, wie genau diese Approximation sein muß, damit das Abstiegsverfahren konvergiert.

Wenn $\bar{\alpha}^k$ der kleinste positive Wert für α mit $f(x^k + \bar{\alpha}^k d^k) = f(x^k)$ ist, dann kann der oben beschriebene Effekt auftreten, wenn entweder $\alpha^k \to 0$ oder $\alpha^k \to \bar{\alpha}^k$ (siehe dazu auch Abb. 3.1).

Zwei Bedingungen, die diese Fälle ausschließen, hat Goldstein [Gol65] angegeben. Um die Schreibweise zu vereinfachen, wurde $f(x^k + \alpha d^k)$ durch $f(\alpha)$ ersetzt. Damit ergibt sich

$$f(\alpha^k) \leq f(0) + \alpha^k \rho f'(0), \tag{3.5}$$

um $\alpha^k \to \bar{\alpha}^k$ zu vermeiden, und

$$f(\alpha^k) \geq f(0) + \alpha^k (1 - \rho) f'(0), \tag{3.6}$$

um $\alpha^k \to 0$ auszuschließen, wobei $\rho \in (0, \frac{1}{2})$ ein fester Parameter ist. Die Bedingung (3.5) stellt z.B. sicher, daß

$$f(x^k) - f(x^{k+1}) \geq -\rho\, \alpha^k (\nabla f(x^k))^T d^k, \tag{3.7}$$

wodurch eine vernachlässigbar kleine Reduktion von f durch ungünstige Wahl von α^k verhindert wird. Ein ähnliches Resultat folgt für (3.6).

Die Goldstein-Bedingungen sichern zwar die hinreichende Reduktion der Zielfunktion, haben aber den Nachteil, daß sie für allgemeine Funktionen ausschließen, daß α^k das Minimum für $f(\alpha)$ liefert. Aus diesem Grund ersetzt man (3.6) durch eine Bedingung der Form

$$|f'(\alpha^k)| \leq -\sigma f'(0) \qquad \text{mit} \quad \sigma \in (\rho, 1), \tag{3.8}$$

die das Minimum für $f(\alpha)$ nicht ausschließt. Abbildung 3.1 veranschaulicht die Bedingungen (3.5) und (3.8).

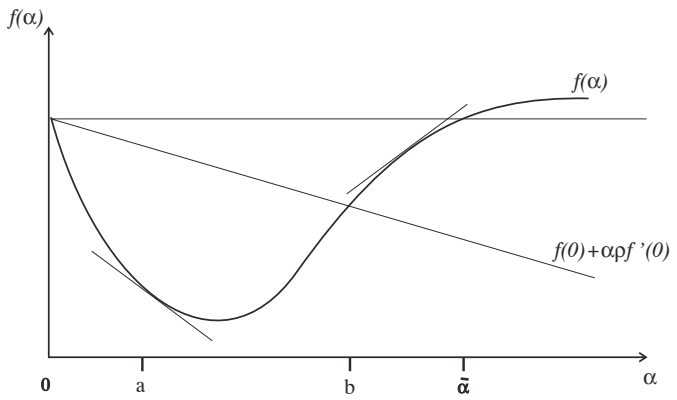


Abbildung 3.1: Die zulässigen Werte für α^k liegen im Intervall [a,b].

Es reicht allerdings nicht aus, nur Forderungen an die Strahlminimierung zu stellen, um die Konvergenz eines Abstiegsverfahrens zu sichern. Wie aus Gleichung (3.7) zu erkennen ist, kann $|f(x^{k+1}) - f(x^k)| \to 0$ auch eintreten, wenn die Suchrichtung d^k ungünstig gewählt wird, d.h. wenn

$$(-\nabla f(x^k))^T d^k \to 0. \tag{3.9}$$

Anschaulich bedeutet diese Gleichung, daß die Suchrichtung und der negative Gradient nahezu senkrecht aufeinander stehen und somit d^k quasi parallel zu einer Höhenlinie ist. Um das zu verhindern, reicht ein einfaches Winkelkriterium.

Man fordert, daß

$$\theta^k \leq \frac{\pi}{2} - \mu \qquad \text{für alle } k \in I\!N \tag{3.10}$$

mit $\mu > 0$ unabhängig von k und $\theta^k \in \left[0, \frac{\pi}{2}\right)$, wobei

$$\cos\theta^k = \frac{(-\nabla f(x^k))^T d^k}{|\nabla f(x^k)|_2 \, |d^k|_2} \tag{3.11}$$

Mit dieser Winkelbedingung und den Forderungen (3.5) und (3.8) an die Strahlminimierung erhält man für Lipschitz-stetig differenzierbare Funktionen die globale Konvergenz der Abstiegsverfahren.

In den in dieser Arbeit benutzten Verfahren werden die Suchrichtungen d^k im Iterationspunkt x^k durch

$$d^k = -H_k^{-1} \nabla f(x^k) \tag{3.12}$$

bestimmt, wobei die H_k gleichmäßig positiv definite und beschränkte Matrizen sind, d.h. für die mit Konstanten $0 < m \leq M$ die Abschätzung

$$m|z|^2 \leq z^T H z \leq M|z|^2 \qquad \text{für alle} \quad z \in \mathbb{R}^n \tag{3.13}$$

gilt. Zur Konvergenz derartiger Verfahren gilt:

Satz 3.4 Sei f eine nach unten beschränkte, Lipschitz-stetig differenzierbare Funktion. Für eine Abstiegsmethode, in der die Suchrichtungen gemäß (3.12) aus einer Folge H_k von Matrizen erzeugt werden, die der Bedingung (3.13) genügen und deren Strahlminimierung die Bedingungen (3.5) und (3.8) erfüllt, gilt $\nabla f(x^k) = 0$ für ein $k \in \mathbb{N}$ oder $\nabla f(x^k) \to 0$ für $k \to \infty$.

Ein weiterer wesentlicher Aspekt numerischer Verfahren ist die Konvergenzgeschwindigkeit, d.h. die Geschwindigkeit, mit der sich die Folge der Iterationspunkte x^k an die optimale Lösung x^* annähert.

Definition 3.3 Sei $\{x^k\}_{k \in \mathbb{N}}$ eine Folge reeller Zahlen mit $\lim_{k\to\infty} x^k = x^*$ und $x^k \neq x^*$ für alle k. Die *Konvergenzordnung* p der Folge ist das Supremum aller positiven Zahlen l, für die gilt

$$\limsup_{k\to\infty} \frac{|x^{k+1} - x^*|}{|x^k - x^*|^l} = \beta < \infty. \tag{3.14}$$

Wenn $p = 1$ und $\beta < 1$ (β wird dann auch als Kontraktionsfaktor bezeichnet), spricht man von *linearer Konvergenz*. Für $p > 1$ oder $p = 1$ und $\beta = 0$ erhält man *superlineare Konvergenz* und im Fall $p = 2$ und $\beta < \infty$ *quadratische Konvergenz*. Lineare Konvergenz ist in der Praxis nur dann akzeptabel, wenn der Kontraktionsfaktor β hinreichend klein ist. Andernfalls ist quadratische oder mindestens superlineare Konvergenz anzustreben.

Als letztes in diesem Abschnitt sollen Abbruchkriterien behandelt werden. Aus der vorherigen Diskussion ist schon klar geworden, daß die Iterationspunkte x^k in der Regel das Minimum x^* niemals erreichen werden. Daher braucht man Kriterien, die Aufschluß darüber geben, wie gut die momentane Approximation des Minimums ist. Die theoretisch günstigsten Kriterien

$$\begin{aligned} |x^k - x^*| &< \epsilon \\ |f(x^k) - f(x^*)| &< \epsilon \end{aligned}$$

scheiden natürlich aus, da man das Minimum x^* vorher nicht kennt. Deshalb wird man alternativ den Fortschritt des Verfahrens oder die Einhaltung notwendiger bzw. hinreichender Kriterien für lokale Minima überwachen. Das führt dann zu den Kriterien

$$|x^k - x^{k+1}| < \epsilon \tag{3.15}$$

$$|f(x^k) - f(x^{k+1})| < \epsilon \tag{3.16}$$

und

$$|\nabla f(x^k)| < \epsilon. \tag{3.17}$$

Die Kriterien (3.15) und (3.16) eignen sich allerdings nur in den Fällen, wo die Verfahren relativ schnell konvergieren. Aber auch bei Benutzung von (3.17) können Probleme der Art auftreten, daß durch numerische Ungenauigkeiten dieses Kriterium niemals erreicht wird. Solche Effekte hat man insbesondere bei schlecht konditionierten Problemen, wie bei den hier benutzten Penalty-Verfahren (siehe dazu auch Kapitel 4).
Deshalb bietet sich eine Kombination aus mehreren Abbruchkriterien an, z.B. (3.15) und (3.17) zusammen mit einer Beschränkung der Iterationszahl für den Fall, daß das Verfahren nicht konvergiert.

Nachdem einige allgemeine Eigenschaften von Abstiegsverfahren vorgestellt wurden, sollen nun konkrete Algorithmen für die Strahlminimierung und die Richtungsbestimmung entwickelt werden.

3.2 Strahlminimierung

In jedem Iterationsschritt der Abstiegsverfahren ist eine eindimensionale Minimierung der Form

$$f(\alpha) \rightarrow \min \qquad \text{mit } \alpha \in \mathbb{R}_+ \tag{3.18}$$

zu lösen. Das Ergebnis dieser Minimierung bestimmt die Schrittweite im Abstiegsverfahren.

Je exakter man dieses Problem lösen will, desto höher ist der damit verbundene Rechenaufwand. Da man für die Konvergenz der Abstiegsverfahren aber keine exakte Lösung von (3.18) braucht, begnügt man sich mit mehr oder weniger guten Approximationen des Minimums. Es ist allerdings zu erwarten, daß die Konvergenz des Abstiegsverfahrens um so schlechter wird, je schlechter die eindimensionale Minimierung ist, so daß man einen vernünftigen Mittelweg zwischen Geschwindigkeit und Genauigkeit der Strahlminimierung finden muß, um die Gesamtrechenzeit des Abstiegsverfahrens zu minimieren.

Die Verfahren für die eindimensionale Optimierung lassen sich unterteilen in solche, die Ableitungen benutzen, und solche, die ableitungsfrei sind. Verfahren, die Ableitungen benutzen, haben in der Regel bessere Konvergenzeigenschaften als ableitungsfreie Methoden. Aber gerade wenn die Berechnung der Ableitungen, die sich real als Skalarprodukt des Gradienten mit der Suchrichtung ergeben, im Vergleich zur Funktionsauswertung teuer ist, kann es lohnend sein, ableitungsfreie Verfahren zu benutzen.

Dementsprechend werden hier mehrere Verfahren vorgestellt, die bezüglich der Genauigkeit und der Benutzung von Ableitungen verschiedene Charakteristika aufweisen, wobei der Aspekt der Parallelisierbarkeit erst im Kapitel 6 behandelt wird.

Alle im weiteren betrachteten Algorithmen bestehen im Prinzip aus zwei Phasen. In der ersten Phase, die man auch als Einschachtelung bezeichnet, wird ein Intervall $[a, b]$ gesucht, in dem das Minimum von f liegt, und in der zweiten Phase wird das Intervall so lange verfeinert, bis man das Minimum in gewünschter Genauigkeit bestimmt hat. Die erste Phase kann man auch auslassen, wenn man eine maximale Schrittweite vorgibt und die zweite Phase dann auf dem Intervall $[0, maxstep]$ startet.

Für alle weiteren Betrachtungen sei angenommen, daß die Funktion f auf $\mathbb{R}_+$ *unimodal* ist.

Definition 3.4 Eine Funktion f ist auf dem Intervall $[a, b]$ *unimodal*, wenn sie dort genau ein relatives Minimum α^* hat.

Wenn eine Funktion $f : \mathbb{R} \to \mathbb{R}$ strikt konvex ist, so ist sie auch unimodal. Weiterhin gilt für eine strikt konvexe Funktion $F : \mathbb{R}^n \to \mathbb{R}$, daß sie auch strikt konvex auf jedem Strahl $\{x + \alpha d \mid \alpha \in \mathbb{R}\}$ ist. Das bedeutet, daß bei strikt konvexen Zielfunktionen die eindimensionalen Probleme die Eigenschaft der Unimodalität haben.

Das Problem der Einschachtelung löst man typischerweise dadurch, daß man eine Folge γ^k von Schrittweiten mit $\gamma^k < \gamma^{k+1}$ erzeugt. Sobald man ein k findet, für das $f(\gamma^k) < f(\gamma^{k+1})$ ist, weiß man, daß das Minimum von f wegen der Unimodalität im Intervall $[0, \gamma^{k+1}]$ liegen muß. Dabei wählt man für die Schrittweiten z.B. $\gamma^k = 2^k$. Natürlich gibt es eine Reihe von Strategien, die das Einschachtelungs-

Problem günstiger lösen. Da der Einschachtelungs-Schritt aber bei der Strahlminimierung nur einen geringen Anteil an der Rechenzeit hat, soll sich die Aufmerksamkeit ausschließlich auf die zweite Phase, die Intervallverfeinerung, richten. Es sei an dieser Stelle nur erwähnt, daß bei Newton- und Quasi-Newton-Verfahren, die weiter unten beschrieben werden, die Schrittweiten zur 1 hin tendieren und somit die Einschachtelung meist kein Problem darstellt.

Im weiteren wird also davon ausgegangen, daß ein Intervall $[a, b]$ vorgegeben ist, in dem das Minimum von f liegt.

3.2.1 Strahlminimierung nach Fletcher

Der erste Algorithmus für die Strahlminimierung ist [Fl87] entnommen, wo dieses Verfahren ausführlich mit einem Korrektheitsbeweis dargestellt ist. Ziel ist es, eine Schrittweite $\bar{\alpha}$ zu finden, die den Bedingungen (3.5), (3.8) genügt, d.h. für die

$$f(\bar{\alpha}) \leq f(0) + \bar{\alpha}\rho f'(0)$$

$$|f'(\bar{\alpha})| \leq -\sigma f'(0)$$

gilt mit frei wählbaren Parametern $\sigma \in (0, 1)$, ρ und $\sigma \geq \rho$. σ ist dabei ein Maß für die Genauigkeit des Verfahrens, das um so genauer wird, je kleiner σ ist.

Die Intervallverfeinerung läuft nach folgendem Schema ab:

Im vorgegebenen Intervall $[a^k, b^k]$ wird ein $\alpha^k \in [a^k, b^k]$ ausgewählt. Wenn das α^k den obigen Bedingungen genügt, bricht man das Verfahren ab. Ansonsten berechnet man $f'(\alpha^k)$ und bestimmt, in welchem der Teilintervalle $[a^k, \alpha^k]$ oder $[\alpha^k, b^k]$ das Minimum liegt. Wenn $f'(\alpha^k) < 0$, liegt das Minimum in $[\alpha^k, b^k]$, und falls $f'(\alpha^k) > 0$, liegt es in $[a^k, \alpha^k]$. Auf dieses neu gewonnene Intervall wendet man das Verfahren dann erneut an, bis schließlich ein α^k gefunden wird, das die Abbruchkriterien erfüllt.

Wie schnell das Verfahren abbricht, hängt entscheidend davon ab, wie man das $\alpha^k \in [a^k, b^k]$ auswählt. Eine günstige Wahl ist das Minimum der quadratischen Approximation von f auf dem Intervall $[a^k, b^k]$. Die quadratische Approximation berechnet man aus $f(a^k)$, $f(b^k)$ und $f'(a^k)$ bzw. $f'(b^k)$, die sowieso notwendigerweise in jeder Iteration berechnet werden müssen. Um in jedem Schritt eine echte Intervallverkleinerung zu sichern, muß man verhindern, daß $\alpha^k \to a^k$ oder $\alpha^k \to b^k$. Deshalb schränkt man die Wahl des α^k auf das Intervall $[a^k + \tau_1(b^k - a^k), b^k - \tau_2(b^k - a^k)]$ mit $0 < \tau_1 < \tau_2 < \frac{1}{2}$ ein, wodurch dieses Problem offensichtlich vermieden wird.

Algorithmisch formuliert ergibt sich

for k:=1,2,... do
begin bestimme $\alpha^k \in [a^k + \tau_1(b^k - a^k), b - \tau_2(b^k - a^k)]$
berechne $f(\alpha^k)$
if $f(\alpha^k) > f(0) + \rho\alpha^k f'(0)$ or $f(\alpha^k) \geq f(a^k)$
then $a^{k+1} := a^k$; $b^{k+1} := \alpha^k$
else
berechne $f'(\alpha^k)$
if $|f'(\alpha^k)| \leq -\sigma f'(0)$ then stop
$a^{k+1} := \alpha^k$
if $(b^k - a^k)f'(\alpha^k) \geq 0$ then $b^{k+1} := a^k$ else $b^{k+1} := b^k$
end

Die Vorteile dieser Methode liegen sicherlich darin, daß unabhängig vom speziellen Problem Bedingungen eingehalten werden, die die Konvergenz des Abstiegsverfahrens sicherstellen. Auch erweist sich der Ansatz der iterativen quadratischen Interpolation als äußerst effizient. Problematisch ist nur die Benutzung der Ableitungen von f, wodurch das Verfahren für spezielle Zielfunktionen sehr aufwendig wird, wie später zu sehen ist.
Deswegen sollen noch zwei Methoden zur Intervallverfeinerung vorgestellt werden, die ohne Ableitungen auskommen.

3.2.2 Verfahren des goldenen Schnitts

Sei wieder ein Intervall $[a^k, b^k]$ vorgegeben, in dem das Minimum α^* von f liegt. Weiterhin seien ω_1^k, ω_2^k zwei Punkte mit $a^k < \omega_1^k < \omega_2^k < b^k$.

Wegen der Unimodalität von f gilt:

$$\begin{aligned} \alpha^* &\in [a^k, \omega_2^k] \qquad \text{falls } f(\omega_1^k) \leq f(\omega_2^k) \\ \alpha^* &\in [\omega_1^k, b^k] \qquad \text{falls } f(\omega_1^k) \geq f(\omega_2^k) \end{aligned} \tag{3.19}$$

Auf das so neu gewonnene Intervall $[a^{k+1}, b^{k+1}]$ wendet man das Verfeinerungsschema so lange an, bis $|b^k - a^k| \leq \epsilon$ ist.

Es verbleibt aber noch die Entscheidung, wie die ω_1^k, ω_2^k zu wählen sind, um eine maximale Intervallverkürzung mit möglichst wenigen Funktionsauswertungen zu erreichen. Theoretisch optimal ist hierbei die Fibonacci-Suche, die asymptotisch mit dem Verfahren des goldenen Schnitts identisch ist (siehe dazu z.B. [BaSS93]). Beim Verfahren des goldenen Schnitts werden die ω_1^k, ω_2^k so gewählt, daß für die Teilungsverhältnisse der Teilintervalle

$$\frac{|\omega_1^k - a^k|}{|b^k - \omega_1^k|} = \frac{|b^k - \omega_1^k|}{|b^k - a^k|}$$

und

$$\frac{|\omega_2^k - b^k|}{|a^k - \omega_2^k|} = \frac{|a^k - \omega_2^k|}{|b^k - a^k|}$$

gilt. Weil diese Teilungsverhältnisse genau den goldenen Schnitt definieren, heißt das Verfahren dementsprechend. Durch die Einhaltung dieser Verhältnisse können drei der vier Punkte a^k, ω_1^k, ω_2^k und b^k auch in der nächsten Iteration benutzt werden, so daß in jeder Iteration nur ein neuer Punkt mit dem zugehörigen Funktionswert berechnet werden muß.

Der Algorithmus läuft dann wie folgt ab:

gegeben sei $[a^0, b^0]$
berechne ω_1^0 und ω_1^0 gemäß (3.2.2) und (3.2.2)
berechne $f(\omega_1^0)$ und $f(\omega_2^0)$
while $|a^k - b^k| \geq \epsilon$ do
begin
    if $f(\omega_1^k) \leq f(\omega_2^k)$
    then
        $a^{k+1} := a^k$
        $b^{k+1} := \omega_2^k$
        $\omega_2^{k+1} := \omega_1^k$
        berechne ω_1^{k+1} gemäß (3.2.2) und (3.2.2)
        berechne $f(\omega_1^{k+1})$
    else
        $a^{k+1} := \omega_1^k$
        $b^{k+1} := b^k$
        $\omega_1^{k+1} := \omega_2^k$
        berechne ω_2^{k+1} gemäß (3.2.2) und (3.2.2)
        berechne $f(\omega_2^{k+1})$
end

Das Verfahren des goldenen Schnitts ist ein sehr robustes Verfahren, in dem nur das absolute Abbruchkriterium $|a^k - b^k| \leq \epsilon$ Schwierigkeiten machen kann, wenn f sehr "steil" ist. Deshalb muß das ϵ ggf. dem jeweiligen Problem angepaßt werden, um die Konvergenz des Abstiegsverfahrens zu sichern. Aus dem gleichen Grund eignet sich diese Methode nicht für eine unexakte Strahlminimierung, da sich dann Bedingungen wie (3.5) und (3.8) nicht einhalten lassen. Obwohl sich dieses Problem durch ein anderes Abbruchkriterium vermeiden ließe, soll das Verfahren des goldenen Schnitts hier nur für eine exakte eindimensionale Minimierung benutzt werden.

3.2.3 Armijo-Schrittweite

Das letzte Verfahren versucht nicht, das Minimierungsproblem (3.18) zu lösen, sondern liefert nur eine Schrittweite, die unter gewissen Voraussetzungen aber dennoch die Konvergenz des Abstiegsverfahrens sicherstellt. Aus diesem Grund ist die Schrittweitenwahl nach dem Armijo-Prinzip ein Prototyp für eine unexakte Strahlminimierung.

Es bezeichne $S = \{2^{-j}\}_{j=0}^{\infty} \subset \mathbb{R}_{>0}$. Mit einem festen Parameter $\delta \in (0,1)$ wird die *Armijo-Schrittweite* $\bar{\alpha}$ bestimmt durch

$$\bar{\alpha} := \max\{\alpha \in S \mid f(\alpha) \leq f(0) + \alpha\delta f'(0)\}. \tag{3.20}$$

Die obige Bedingung an das $\bar{\alpha}$ entspricht bis auf die Wahl des Parameters δ genau der ersten Goldsteinschen Bedingung (3.5). Die Abbildung 3.2 verdeutlicht, daß das $\bar{\alpha}$ so gewählt wird, daß $f(\bar{\alpha})$ kleiner ist als eine lineare "Unterschätzung" von f.

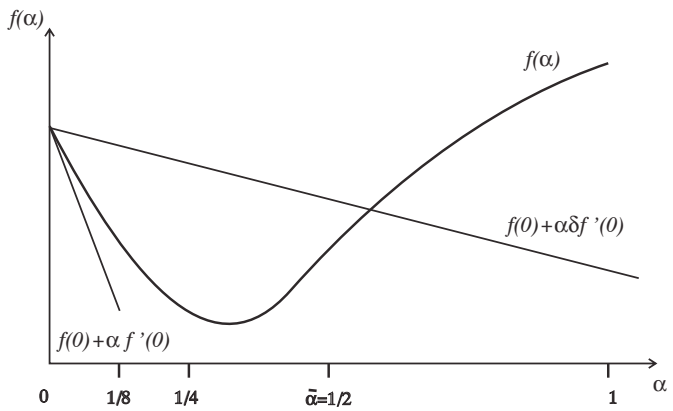


Abbildung 3.2: Armijo-Schrittweite

Obwohl dieses Verfahren im Nullpunkt die Ableitung von f benötigt, kann man diese Methode insofern als ableitungsfrei bezeichnen, als daß die Ableitung von f im Nullpunkt sowieso bei gradientenbasierten Verfahren schon vorliegt und somit nicht extra berechnet werden muß. Außerdem werden meistens nur sehr wenige Funktionsauswertungen benötigt, da die α^j exponentiell kleiner werden und somit die Bestimmung der Armijo-Schrittweite im Vergleich zu anderen Strahlminimierungen sehr schnell geht.
Interessanterweise kann man für Lipschitz-stetig differenzierbare Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ dennoch zeigen, daß ein Abstiegsverfahren, das die Winkelbedingung (3.10) für die Suchrichtungen einhält und die Armijo-Schrittweite benutzt, konvergiert, obwohl es sich um ein sehr ungenaues Suchverfahren handelt. (Auf eine weitere Bedingung wie die 2. Bedingung von Goldstein (3.6) kann man an dieser

Stelle verzichten, weil man $\bar{\alpha}$ als maximales $\alpha \in S$ wählt, weswegen $\bar{\alpha} \to 0$ ebenfalls ausgeschlossen wird.)

Wie später zu sehen sein wird, kann man mit dieser Methode in der Tat brauchbare Ergebnisse erzielen. Dies liegt wohl auch daran, daß bei Newton-ähnlichen Verfahren die optimale Schrittweite gegen 1 tendiert und die 1 als Schrittweite auch im Armijo-Verfahren bevorzugt wird.

3.3 Bestimmung geeigneter Abstiegsrichtungen

Wie bereits erwähnt, unterscheiden sich die verschiedenen Abstiegsverfahren weniger durch die Methoden der Strahlminimierung als vielmehr durch die Wahl der Abstiegsrichtungen d^k. Der nun folgende Abschnitt beschäftigt sich damit, wie man die Suchrichtungen d^k wählt, damit das Abstiegsverfahren gute Konvergenzeigenschaften bekommt. Dabei stehen zunächst nur die grundlegenden mathematischen Aspekte im Vordergrund. Algorithmisch orientierte Probleme und die Parallelisierung dieser Verfahren werden in Kapitel 5 und Kapitel 6 ausführlich behandelt.

3.3.1 Steepest Descent

Eine naheliegende Wahl für d^k im Punkte x^k ist sicherlich $d^k = -\nabla f(x^k)$, womit sich zusammen mit einer exakten Strahlminimierung das Verfahren des steilsten Abstiegs (steepest descent) ergibt. Dieses Verfahren hat allerdings nur lineare Konvergenz, und das mit einem Kontraktionsfaktor, der beliebig nahe an 1 kommt, d.h. man kann zeigen, daß ein solches Verfahren beliebig langsam gegen ein Minimum konvergiert. Da die Ursachen für diesen Effekt auch später häufiger in die Argumentationen eingehen, sollen sie hier kurz beleuchtet werden (für eine ausführliche Darstellung siehe [BaSS93]).

Wenn man das Verhalten von Steepest Descent geometrisch betrachtet, so gilt für aufeinanderfolgende Suchrichtungen $(d^k)^T(d^{k+1}) = 0$, d.h. d^k und d^{k+1} stehen senkrecht aufeinander, und die Annäherung an das Minimum x^* von f geschieht entlang einer Zickzack-Kurve.
Wie stark der Effekt, der im Englischen auch als Zigzagging bezeichnet wird, ins Gewicht fällt, hängt davon ab, wie f konditioniert ist. Die Kondition einer Funktion f ist dabei gleich der Kondition ihrer Hesse-Matrizen.

Beispielsweise ist für $f(x_1, x_2) = \frac{1}{2}(x_1^2 + \alpha x_2^2)$ die Kondition von f gleich α, und man kann die Kondition von f durch Erhöhen von α beliebig schlecht machen. Für einen Startpunkt $x = (x_1, x_2)^T$ mit $x_1 \neq 0$ und $x_2 \neq 0$ ergibt sich dann folgende

Reduktion der Funktionswerte:

$$\frac{f(x^{k+1})}{f(x^k)} = \frac{(\alpha-1)^2}{(\alpha+1)^2} \tag{3.21}$$

Für $\alpha \to \infty$ folgt daraus die beliebig langsame lineare Konvergenz.

Der negative Gradient ist also keine sehr günstige Suchrichtung für ein Abstiegsverfahren. Anstatt in die Richtung des steilsten Abstiegs zu gehen, d.h. ausschließlich Steigungsinformationen zu verwenden, scheint es vernünftig zu sein, zusätzlich die Krümmung von f zu berücksichtigen und so Richtungen zu erzeugen, die "weitsichtiger" sind und dadurch eher zum Mimimum von f tendieren.

Eine Reihe von Ideen dazu lassen sich aus quadratischen Modellen gewinnen. Man bestimmt dann eine optimale Abstiegsrichtung unter der Annahme, die Zielfunktion sei quadratisch. Obwohl die Zielfunktionen i. allg. nicht quadratisch sind, lassen sich mit dieser Annahme die Konvergenzeigenschaften der Abstiegsverfahren erheblich verbessern. Die meisten heute verwendeten Methoden wie die Newton-, Quasi-Newton- und CG-Verfahren basieren auf diesem Ansatz.

3.3.2 Newton-Verfahren

Die Grundidee des Newton-Verfahrens ist die, daß man in jedem Schritt in die Richtung des Minimums der quadratischen Approximation von f geht. Als Approximation nimmt man die Taylor-Entwicklung von f im Punkte x^k bis einschließlich zum quadratischen Glied.

$$F_k(x) = f(x^k) + \nabla f(x^k)^T(x-x^k) + \frac{1}{2}(x-x^k)^T\nabla^2 f(x^k)(x-x^k) \tag{3.22}$$

Die Suchrichtung d^k ergibt sich aus der Lösung des Minimierungsproblems

$$F_k(x) \to \min \qquad \text{mit} \quad x \in I\!R^n. \tag{3.23}$$

Das Problem hat genau dann eine Lösung, wenn die Hesse-Matrix $\nabla^2 f(x^k)$ positiv semi-definit ist. In diesem Fall lautet das notwendige und hinreichende Kriterium für die Lösung $\bar{x}$

$$\nabla f(x^k) + \nabla^2 f(x^k)(\bar{x}-x^k) = 0 \tag{3.24}$$

und es folgt, falls $\nabla^2 f(x^k)$ nicht singulär ist,

$$\bar{x} = x^k - (\nabla^2 f(x^k))^{-1}\nabla f(x^k). \tag{3.25}$$

Als Suchrichtung wählt man nun $d^k = -(\nabla^2 f(x^k))^{-1}\nabla f(x^k)$, woraus sich ein Iterationsschritt des Newton-Verfahrens ergibt zu

$$x^{k+1} = x^k - \alpha^k(\nabla^2 f(x^k))^{-1}\nabla f(x^k). \tag{3.26}$$

Beim klassischen Newton-Verfahren gilt zusätzlich in jedem Schritt $\alpha^k = 1$.

Für das Newton-Verfahren erhält man auf einer hinreichend kleinen Umgebung um ein lokales Minimum superlineare Konvergenz. Wenn man zusätzlich annimmt, daß die Hesse-Matrizen von f Lipschitz-stetig sind, dann erhält man sogar quadratische Konvergenz der Folge $\{x^k\}_{k \in N}$ gegen x^*.

Obwohl die formalen Konvergenzeigenschaften das Newton-Verfahren sehr attraktiv machen, hat es doch einige schwerwiegende Nachteile:

- Die Hesse-Matrizen sind nicht unbedingt regulär, und somit ist die Matrixinvertierung nicht durchführbar, oder die Hesse-Matrizen sind so schlecht konditioniert, daß die Invertierung problematisch ist. Weiterhin muß $(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$ keine Abstiegsrichtung sein, wenn die Hesse-Matrix nicht positiv definit ist. Aus diesem Grund wurden viele Modifiktionen und Regularisierungstechniken des Newton-Verfahrens entwickelt (siehe z.B. [Mar63, Mu72, FlFr77, Fl87]).

- Bei großen Problemen wird der Aufwand der Matrixinvertierung sowohl für direkte als auch für iterative Methoden zu groß. Als Ausweg bietet sich hier an, die inverse Hesse-Matrix mit iterativen Verfahren nur sehr grob zu approximieren, wie es z.B. in einem Ansatz von Nash und Sofer [NaSo89, NaSo92b] gemacht wurde.

- Die Verwendung der Hesse-Matrix setzt natürlich voraus, daß die zweiten Ableitungen überhaupt analytisch vorliegen, was gerade bei praktischen Anwendungen einen erheblichen Aufwand von seiten des Anwenders bedeuten kann. Die Alternative, die Hesse-Matrix über Gradienten zu approximieren, führt zwangsläufig verstärkt zu Problemen, wie sie im vorigen Punkt erwähnt wurden.

Eine Klasse von Verfahren, welche die meisten der oben genannten Nachteile vermeidet, aber trotzdem gute Konvergenzeigenschaften hat, wird durch die sogenannten Quasi-Newton-Verfahren gebildet.

3.3.3 Quasi-Newton-Verfahren

Ausgangspunkt der Quasi-Newton-Verfahren ist die Idee, die inverse Hesse-Matrix nur durch eine symmetrische positiv definite Matrix H zu approximieren und diese Approximation nach jedem Iterationsschritt zu verbessern. Die k-te Iteration des

Abstiegsverfahrens hat dann die Struktur

1. Berechne die Suchrichtung als $d^k = H^k \nabla f(x^k)$.
2. Bestimme ein α^k und setze $x^{k+1} = x^k + \alpha^k d^k$.
3. Führe das Update für H^{k+1} durch.

Als H^0 wählt man eine beliebige symmetrische positiv definite Matrix, meistens die Einheitsmatrix.

Die potentiellen Vorteile dieser Verfahren gegenüber den Newton-Verfahren sind:

- Es wird nur die 1. Ableitung benötigt (anstatt der 2. Ableitung).
- $O(n^2)$ Multiplikationen pro Iteration (anstatt $O(n^3)$).
- Die H^k sind positiv definit (gegenüber möglicherweise indefiniten $\nabla^2 f(x^k)$).

Die Approximationen H^k der Hesse-Matrizen lassen sich wie folgt aus den Differenzen $s^k = x^{k+1} - x^k$ und $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$ gewinnen. Nach der Taylorschen Formel gilt

$$x^{k+1} - x^k = (\nabla^2 f(x^k))^{-1}(\nabla f(x^{k+1}) - \nabla f(x^k)) + o(|x^{k+1} - x^k|). \tag{3.27}$$

Unter Vernachlässigung des Restgliedes ergibt sich dann

$$s^k \approx (\nabla^2 f(x^k))^{-1} y^k \tag{3.28}$$

Für die Approximation H^k gilt i. allg. nicht, daß $s^k = H^k y^k$, und deswegen wird die Matrix H^{k+1} so gewählt, daß die Gleichung

$$s^k = H^{k+1} y^k \tag{3.29}$$

erfüllt ist. Gleichung (3.29) wird auch als Quasi-Newton-Gleichung bezeichnet und die zugehörigen Verfahren dementsprechend als Quasi-Newton-Verfahren. Fordert man, daß H^{k+1} ebenso wie $(\nabla^2 f(x^{k+1}))^{-1}$ symmetrisch ist, so liefert (3.29) für festes k nur n Bedingungen zur Bestimmung der $n(n+1)/2$ freien Parameter. Daher gibt es eine große Zahl unterschiedlicher Konstruktionsvarianten für Matrizen H^{k+1}, die der Quasi-Newton-Gleichung (3.29) genügen.
Mit dem Ziel, bereits gewonnene Informationen weitgehend zu erhalten, wird H^{k+1} aus der vorhergehenden Matrix H^k durch Modifikationen der Form

$$H^{k+1} = H^k + U^k \tag{3.30}$$

bestimmt. Dieser Vorgang wird als Update bezeichnet.

Im weiteren sollen die benutzten Update-Formeln und die Eigenschaften der daraus resultierenden Verfahren dargestellt werden.

Als einfachste Möglichkeit für das Update ergibt sich ein sogenanntes Rang-1-Update der Form

$$H^{k+1} = H^k + auu^T \tag{3.31}$$

mit einem Vektor $u \in \mathbb{R}^n$. Unter Berücksichtigung der Quasi-Newton-Gleichung erhält man das Update

$$H^{k+1}_{BROYDEN} = H + \frac{(s - Hy)(s - Hy)^T}{(s - Hy)^T y}, \tag{3.32}$$

wobei hier und im weiteren die Indizes k an den y, s und H weggelassen wurden. Weil Broyden 1967 diese Formel erstmals vorschlug, wird sie meistens als Broyden-Update bezeichnet und das zugehörige Quasi-Newton-Verfahren entsprechend als Broyden-Verfahren.

Satz 3.5 Sei $f : \mathbb{R}^n \to \mathbb{R}$ eine strikt konvexe quadratische Funktion. Sind die $s^1, s^2, \ldots, s^n$ bei Anwendung des Broyden-Verfahrens auf f linear unabhängig, dann terminiert es nach höchstens $n + 1$ Schritten mit $H^{k+1} = (\nabla^2 f)^{-1}$.

Wenn man das Verfahren auf quadratische Probleme anwendet, wird nach n Schritten unter der Voraussetzung, daß die $s^1, s^2, \ldots, s^n$ linear unabhängig sind, die inverse Hesse-Matrix exakt approximiert, so daß der $(n + 1)$-te Schritt mit einem Newton-Schritt identisch ist.
Von der Möglichkeit, mit n linear unabhängigen Richtungen die Hesse-Matrix vollständig zu approximieren, wird im Straeter-Verfahren Gebrauch gemacht, das weiter unten beschrieben ist.

Ein wesentlicher Nachteil des Broyden-Verfahrens ist, daß die so erzeugten Matrizen H^k nicht positiv definit sind, wodurch die resultierenden Suchrichtungen nicht notwendigerweise Abstiegsrichtungen sind. Weiterhin kann der Nenner in (3.32) null werden, womit das Verfahren nicht immer wohldefiniert ist. Ferner sind die sukzessive erzeugten Richtungen möglicherweise nicht linear unabhängig, wodurch eine vollständige Approximation der inversen Hesse-Matrix verhindert werden kann.
Da das Broyden-Update das einzige symmetrische Rang-1-Update ist, das die Quasi-Newton-Bedingung erfüllt, ist es naheliegend, die Möglichkeiten für Rang-2-Versionen zu untersuchen, d.h. Updates der Form

$$H^{k+1} = H^k + auu^T + bvv^T. \tag{3.33}$$

Eine Formel, die auf Davidon [Da59] zurückgeht und später von Fletcher und Powell [FlPo63] in der hier beschriebenen Form präsentiert wurde, ist das nach ihnen benannte DFP-Update.

$$H^{k+1}_{DFP} = H + \frac{ss^T}{s^T y} - \frac{Hyy^T H}{y^T Hy}. \tag{3.34}$$

Quasi-Newton-Verfahren, die dieses Update zusammen mit einer exakten Strahlminimierung benutzen, haben eine Reihe interessanter Eigenschaften:

- Wenn f quadratisch ist, dann sind die sukzessive erzeugten Richtungen zueinander konjugiert bzgl. $(\nabla^2 f)^{-1}$, und mit $H^0 = I$ erhält man konjugierte Gradienten.
- Für quadratische Funktionen gilt nach spätestens n Iterationen $H^{n+1} = (\nabla^2 f)^{-1}$.
- Die Matrizen H^k sind positiv definit.
- Die Verfahren haben lokal superlineare Konvergenz.
- Die Verfahren sind global konvergent für strikt konvexe Funktionen.

Die erste Eigenschaft bewirkt, daß nach spätestens n Schritten der gesamte Raum mit den Vektoren $s^1, \ldots, s^n$ aufgespannt wird und somit für quadratische Funktionen die Konvergenz nach spätestens $(n+1)$ Schritten folgt. Außerdem sichert die positive Definitheit der H^k, daß die hiermit erzeugten Richtungen wirklich Abstiegsrichtungen sind.
Offensichtlich hat das DFP-Verfahren mehr nützliche Eigenschaften als durch die Quasi-Newton-Gleichung gefordert wird, und für quadratische Zielfunktionen hat es sogar zusätzlich die Eigenschaften eines Konjugierten-Gradienten-Verfahrens. Leider zeigt sich das DFP-Verfahren als sehr empfindlich gegenüber einer ungenauen Strahlminimierung, was es für die praktische Anwendung weniger attraktiv macht. Eine wesentliche Verbesserung diesbezüglich bringt das BFGS-Update (Broyden [Br70], Fletcher [Fl70], Goldfarb [Go70], Shanno [Sh70]), das zwar die gleichen oben aufgeführten Vorteile des DFP-Update hat, aber wesentlich unempfindlicher gegenüber der ungenauen eindimensionalen Optimierung ist.

Die BFGS-Formel ist die zum DFP-Update duale Formel. Sie ergibt sich durch formale Vertauschung der y und s und anschließende Invertierung der so gewonnenen Matrix mit der Sherman-Morrison Formel:

$$H_{BFGS}^{k+1} = H + \left(1 + \frac{y^T H y}{s^T y}\right) \frac{s s^T}{s^T y} - \left(\frac{s y^T H + H y s^T}{s^T y}\right). \tag{3.35}$$

Die auf dieser Formel basierenden Quasi-Newton-Verfahren haben sich in der Praxis außerordentlich bewährt. Es bleibt die Frage, ob die in der Praxis herausragende Rolle des BFGS-Verfahrens auch theoretisch begründet werden kann oder ob es möglicherweise noch leistungfähigere Quasi-Newton-Verfahren gibt. Es gibt theoretische Hinweise darauf, daß dem BFGS-Verfahren unter den Newton-Verfahren eine besondere Rolle zukommt. Zwei dieser Aspekte sollen kurz erwähnt werden.

Weiter oben wurde der Einfluß der Kondition der Hesse-Matrix bei Anwendung auf quadratische Funktionen geschildert. Beim Verfahren des steilsten Abstiegs wird die Konvergenz um so schlechter, je schlechter die Kondition der Hesse-Matrix ist. Die Newton-Verfahren sind gegenüber diesem Effekt zumindest theoretisch immun, aber es bleibt zu klären, wie sich die Quasi-Newton-Verfahren diesbezüglich verhalten.
Sei eine quadratische Funktion der Form

$$f(x) = \frac{1}{2}x^T Gx + b^T x \tag{3.36}$$

mit einer symmetrischen positiv definiten Matrix G gegeben, d.h. G ist die Hesse-Matrix von f. Wenn man ein Verfahren auf diese Funktion anwendet und G schlecht konditioniert ist, so wäre es zumindest theoretisch denkbar, daß man das Problem mit einer affin linearen Transformation

$$T(x) = Ax + a \qquad A \text{ regulär} \tag{3.37}$$

auf ein besser konditioniertes Problem abbildet, das transformierte Problem löst und die Lösung wieder zurücktransformiert.
Vom BFGS-Verfahren kann man zeigen, daß es gegenüber einer solchen Transformation bei geeigneter Wahl der Matrix H^0 invariant ist, d.h. wenn $\tilde{x}^0 = T(x^0)$, dann gilt auch $\tilde{x}^k = T(x^k)$ für $k = 1, 2, \ldots, n$. Dabei sei $\{\tilde{x}^k\}_{k \in \mathbb{N}}$ die Folge der mit dem transformierten Problem erzeugten Iterationspunkte.
Diese Invarianzeigenschaft besitzt allerdings nicht nur die BFGS-Formel, sondern z.B. auch die gesamte Broyden-Familie

$$H_\Phi^{k+1} = (1 - \Phi) H_{DFP}^{k+1} + \Phi H_{BFGS}^{k+1}.$$

Es gibt aber eine Eigenschaft, die die BFGS-Formel unter den anderen auszeichnet. Es scheint sinnvoll zu sein, daß die Matrix H^{k+1} nicht nur symmetrisch ist und der Quasi-Newton-Gleichung genügt, sondern zusätzlich der "Abstand" von H^k zu H^{k+1} möglichst klein ist, um die vorher gesammelten Informationen soweit wie möglich zu erhalten.
Als Maß für den Abstand zweier Matrizen nimmt man die gewichtete Euklidische Norm

$$\|A\|_W^2 := \|W^{1/2} A W^{1/2}\|_2^2 = tr(WA^T WA), \tag{3.38}$$

wobei W eine positiv definite symmetrische Matrix ist, die der Quasi-Newton-Gleichung genügt. Die BFGS-Formel ist die einzige Formel, die symmetrische Matrizen erzeugt, die die Quasi-Newton-Gleichung erfüllen und zusätzlich den Abstand von H^{k+1} und H^k bzgl. der oben definierten Norm minimiert (siehe dazu auch [Fl87] und [Gre70]).

Diese und andere Ergebnisse legen nahe, daß man wahrscheinlich keine wesentliche Verbesserung des Verfahrens durch andere Updates mit Hilfe der Differenzen y^k und s^k erzielen kann. Der Satz 3.5 motiviert jedoch einen Ansatz, der von Straeter [Stra73] speziell in Hinblick auf die parallele Lösung nichtlinearer Optimierungsprobleme vorgeschlagen wurde.

Anstatt in jedem Schritt die Differenzen y^k und s^k für das Update zu benutzen, kann man auch direkt in jeder Iteration n Broyden-Updates entlang linear unabhängiger Richtungen machen und somit die inverse Hesse-Matrix in jedem Schritt noch besser approximieren.

Um die Darstellungen zu vereinfachen, sei das Broyden-Update jetzt als

$$H^{k+1} = H^k + \kappa^k r^k (r^k)^T \tag{3.39}$$

geschrieben, mit dem Vektor

$$r^k = s^k - H^k y^k, \tag{3.40}$$

so daß

$$\kappa^k = ((r^k)^T y^k)^{-1}. \tag{3.41}$$

Seien nun in der k-ten Iteration der Iterationspunkt x^k, eine Approximation der Hesse-Matrix H^k und n linear unabhängige Vektoren $s^{k_1}, \ldots, s^{k_n}$ (z.B. die Einheitsvektoren) vorgegeben. Zunächst berechnet man die Punkte

$$x^{k_j} = x^k + s^{k_j} \qquad j = 1, \ldots, n$$

und die zugehörigen Gradienten $\nabla f(x^{k_j})$ nebst

$$y^{k_j} = \nabla f(x^{k_j}) - \nabla f(x^k).$$

Mit diesen s^{k_j} und y^{k_j} führt man n aufeinanderfolgende Broyden-Updates durch, die dann auch partielle Updates genannt werden.

$$\begin{aligned}
V^{k_0} &= H^k \\
r^{k_j} &= s^{k_j} - V^{k_{j-1}} y^{k_j} && j = 1, \ldots, n \\
\kappa^{k_j} &= ((r^{k_j})^T y^{k_j})^{-1} && j = 1, \ldots, n \\
V^{k_j} &= V^{k_{j-1}} + \kappa^{k_j} (r^{k_j})(r^{k_j})^T && j = 1, \ldots, n \\
H^{k+1} &= V^{k_n}
\end{aligned} \tag{3.42}$$

Für quadratische Zielfunktionen gilt nach Satz 3.5, daß $H^1 = (\nabla^2 f(x))^{-1}$, wodurch das Verfahren nach nur einem Schritt terminiert. Durch die vollständige Approximation der inversen Hesse-Matrix bekommt man dann auch ähnliche Konvergenzeigenschaften wie bei approximierten Newton-Verfahren, handelt sich anderseits aber auch Nachteile ein:

- Die H^k sind nicht notwendigerweise positiv definit.
- Es werden $O(n^3)$ Operationen für ein vollständiges Update benötigt.

- Im Gegensatz zu den Newton-Verfahren erhält das Straeter-Update nicht die Struktur der Hesse-Matrix.

Unklar ist an dieser Stelle, wie die n linear unabhängigen Vektoren s^{k_j} gewählt werden sollten. Eine naheliegende Möglichkeit ist sicherlich $s^{jk} = \epsilon\, e^j$, wobei e^j der j-te Einheitsvektor ist und ϵ eine hinreichend kleine Konstante mit $\epsilon > 0$. Die Ergebnisse von Kapitel 5 deuten jedoch darauf hin, daß diese einfache Wahl zu Problemen führen kann, wenn z.B. einige Variablen nur linear in die Zielfunktion eingehen und dadurch $y^{k_j} = 0$ bzw. κ^{k_j} undefiniert wird. Auch ist es nicht einzusehen, warum die Quasi-Newton-Matrix H^{k+1} bei einem vollständigen Update entlang aller Raumrichtungen von der Matrix H^k des Vorpunktes abhängen soll, anstatt das Update jedesmal mit der Einheitsmatrix zu beginnen. Der Einfluß eines Neustarts des Verfahrens in jeder Iteration ist den Meßergebnissen in Kapitel 5 zu entnehmen.
Für große Probleme ist es sehr zeitaufwendig, in jedem Schritt alle partiellen Updates durchzuführen. Deshalb ist eine weitere Variationsmöglichkeit, in jeder Iteration nur einen Teil der partiellen Updates zu berechnen. Dazu kann man entweder reihum eine feste Menge von Vektoren s^{k_j} benutzen, wie es z.B. von Lootsma [Lo91] vorgeschlagen wird, oder man bestimmt nach geeigneten Strategien in jedem Iterationspunkt eine neue Menge von Vektoren.
Durch die Vermeidung vollständiger Updates läßt sich die Update-Zeit in jedem Schritt reduzieren oder sogar gezielt einstellen, was z.B. bei Parallelisierungen zur Lastbalancierung nützlich sein kann (siehe Kapitel 6). Andererseits wird man erwarten, daß durch diese Technik auch die Konvergenz der Verfahren schlechter wird. Wie man die Vektoren bei unvollständigen Updates wählt und wieviele partielle Updates in jedem Schritt optimal sind, um die Gesamtrechenzeit zu minimieren, ist ein bislang kaum behandeltes Problem. Die wenigen numerischen Erfahrungen, die über das Straeter-Verfahren veröffentlicht wurden, behandeln nur peripher die Aspekte unvollständiger Updates. So zeigen z.B. Dyde et al. [DyLT89], daß das Straeter-Verfahren mit vollständigen Updates für kleine und mittelgroße Probleme ($n \approx 50$) in seiner Leistungfähigkeit (auch auf sequentiellen Rechnern) durchaus mit anderen Quasi-Newton-Verfahren vergleichbar ist. Van Laarhoven [La85] entwickelte die Idee von Straeter sogar weiter und benutzt für die partiellen Updates statt des Broyden-Updates ein Update aus der von Huang [Hu70] angegebenen Klasse. Der Einfluß unvollständiger Updates wird allerdings nicht untersucht, und alle numerischen Experimente werden nur mit kleinen Problemen durchgeführt.
Lootsma [Lo91] diskutiert zwar den Einfluß unvollständiger Updates, die präsentierten Ergebnisse erlauben aber nur einen geringen Rückschluß auf die Qualität solcher Methoden.

Die zuverlässige Bewertung von Optimierungsverfahren ist schon im Sequentiellen ein schwerwiegendes Problem, da alle Verfahren in gewissen Fällen ihre spezifischen Vorteile haben. So wird man neben den theoretischen Erklärungen auch ausführliche Testreihen mit realen Problemen durchführen müssen, um die Qualität eines Verfahrens beurteilen zu können. Für sequentielle Verfahren existieren einige Studien (z.B. [Sch80, SaRa80]), die mit einer Vielzahl von Testproblemen genau diese

Frage klären wollen. Beim Vergleich paralleler Verfahren treten zusätzlich Aspekte wie die Implementation und die zugrundeliegende Rechnerarchitektur in den Vordergrund, so daß ein umfassender Vergleich zum momentanen Zeitpunkt fast unmöglich scheint.
Aber selbst eine systematische Untersuchung, die auf das Straeter-Verfahren und einen Rechnertyp beschränkt ist, war im Rahmen dieser Diplomarbeit nicht möglich, da es zu viele Variationsmöglichkeiten sowohl im Verfahren als auch in der Implementation gibt.
Stattdessen werden einige wenige Ansätze für das Straeter-Verfahren exemplarisch am Modell der Raffinerie getestet und mit dem BFGS-Verfahren verglichen. Dazu ist noch ein Ansatz zur Behandlung der Nebenbedingungen notwendig, der im nächsten Kapitel dargestellt wird.

Kapitel 4

Optimierung mit Nebenbedingungen

Das zu lösende Raffineriemodell (2.2) hat neben einer nichtlinearen Zielfunktion eine Reihe von Ungleichungs-Nebenbedingungen und Bounds, die sich aber ebenfalls als Ungleichungs-Nebenbedingungen interpretieren lassen. Für eine allgemeinere Darstellung der Theorie seien jetzt sowohl Ungleichungs- als auch Gleichungs-Restriktionen zugelassen. Das Problem hat damit formal die Gestalt

$$\begin{aligned} & f(x) \rightarrow \quad \min \qquad \text{mit } x \in I\!R^n \\ & \text{mit den Nebenbedingungen} \\ & g_j(x) \geq 0 \qquad j = 1, 2, \ldots, J \\ & h_k(x) = 0 \qquad k = 1, 2, \ldots, K \end{aligned} \tag{4.1}$$

Dabei sollen f und die g_j, h_k stetig differenzierbar sein. Der durch die Nebenbedingungen festgelegte zulässige Bereich wird im weiteren mit G bezeichnet.

4.1 Mathematische Grundlagen

Definition 4.1 Sei f wie in (4.1) gegeben. Ein Punkt x^* heißt *lokales Minimum* von f, wenn es eine offene Umbgebung U von x^* gibt, mit:

$$f(x^*) \leq f(x) \qquad \text{für alle } x \in U \cap G.$$

x^* heißt *globales Minimum* von f, wenn

$$f(x^*) \leq f(x) \qquad \text{für alle } x \in G.$$

x^* wird dann auch *Lösung* des Problems (4.1) genannt.

Es ist intuitiv sofort klar, daß sich die Optimalitätskriterien aus dem vorherigen Abschnitt nicht ohne weiteres auf die beschränkte Optimierung übertragen lassen, wenn das Minimum auf dem Rand des Restriktionsbereichs liegt. Deswegen müssen einige Formalisierungen durchgeführt werden, um analoge Kriterien zu erhalten.

Definition 4.2 Eine Nebenbedingung $g_j(x)$ heißt *aktiv* in einem Punkt $\bar{x}$, wenn $g_j(\bar{x}) = 0$. Dementsprechend ist $J_0(\bar{x}) = \{j \mid g_j(\bar{x}) = 0\}$ die Menge der *aktiven Indizes.*

Da ohne weitere Forderungen an die Nebenbedingungen nur sehr schwache Kriterien für lokale Minima hergeleitet werden können (wie etwa die John-Bedingungen, siehe z.B. [JoTr88]), werden in der Regel weitere Struktureigenschaften der Nebenbedingungen, sogenannte *constraint qualifications* , vorausgesetzt. Eine sehr starke und anschauliche *constraint qualification* ist die sogenannte lineare Unabhängigkeitsbedingung:

Definition 4.3 Die Nebenbedingungen h_k, g_j erfüllen die *lineare Unabhängigkeitsbedingung* (LUB) im Punkt $\bar{x}$, falls die Vektoren $\nabla h_k(\bar{x})$, $k = 1, \ldots, K$, und $\nabla g_j(\bar{x})$, $j \in J_0(\bar{x})$, linear unabhängig sind.

Schwächere und deswegen allgemeinere Möglichkeiten für die *constraint qualification* sind die Mangasarian-Fromovitz- und die Slater-Bedingungen (siehe [JoTr88, BaSS93]). Mit einer beliebigen dieser Bedingungen ergibt sich das wohl bekannteste notwendige Kriterium für lokale Minima.

Satz 4.1 (Karush-Kuhn-Tucker-Bedingungen) Seien f, g_j, h_k stetig differenzierbar und x^* ein lokales Minimum von f auf G. Ferner gelte für die Nebenbedingungen im Punkt x^* eine *constraint qualification*, dann gibt es λ_j, $\mu_k \in \mathbb{R}$, $j = 1, 2, \ldots, J$ und $k = 1, \ldots, K$, mit

$$\nabla f(x^*) = \sum_{k=1,\ldots,K} \mu_k \nabla h_k(x^*) + \sum_{j \in J_0(x^*)} \lambda_k \nabla g_j(x^*) \tag{4.2}$$

$$\lambda_j \, g_j(x^*) = 0 \qquad j = 1, 2, \ldots, J \tag{4.3}$$

$$\lambda_j \geq 0 \qquad j = 1, 2, \ldots, J \tag{4.4}$$

Jeder Punkt, der die Bedingungen (4.2), (4.3) und(4.4) erfüllt, ist ein Karush-Kuhn-Tucker-Punkt oder kurz KKT-Punkt.

Die λ_j, μ_k sind die Lagrange-Multiplikatoren, die wegen ihrer anschaulichen Interpretation in wirtschaftlichen Modellierungen auch Schattenpreise genannt werden. Sie geben darüber Aufschluß, wie sensibel die Lösung gegenüber Störungen der Nebenbedingungen ist, d.h. wie stark sich der Wert der Zielfunktion noch reduzieren

läßt, wenn man die Nebenbedingungen geringfügig verschiebt. Wenn der Lagrange-Multiplikator zu einer Nebenbedingung groß ist, dann bedeutet dies, daß die Lösung sehr sensibel auf die Störung dieser Nebenbedingung reagiert. Die Lagrange-Multiplikatoren sind somit ein wichtiges Hilfsmittel, um die Qualität einer Lösung zu bewerten.

Die Karush-Kuhn-Tucker-Bedingungen (meistens auch nur Kuhn-Tucker-Bedingungen genannt) liefern nur ein notwendiges Kriterium für lokale Minima. Aber wie im unbeschränkten Fall lassen sich diese Bedingungen zu einem hinreichenden Kriterium erweitern, wenn man weitere Bedingungen an die Zielfunktion und die Nebenbedingungen stellt.

Definition 4.4 Das Problem (4.1) heißt *konvex*, wenn sowohl die Restriktionsmenge G als auch f auf G konvex ist.

Mit Hilfe der Konvexität erhält man ein zu Satz 3.3 analoges Ergebnis.

Satz 4.2 Seien f und G im Ausgangsproblem (4.1) konvex. Dann ist jeder KKT-Punkt ein globales Minimum für f auf G.

Nachdem man nun eine geeignete Charakterisierung für eine optimale Lösung von (4.1) hat, muß eine Strategie zum Auffinden dieser Lösung gewählt werden. Von der großen Zahl der Verfahren, die für solche Probleme entwickelt wurden, werden hier nur die Penalty-Verfahren behandelt.

4.2 Penalty-Verfahren

Die Grundidee der Penalty- oder Strafkostenverfahren ist es, die Einhaltung der Nebenbedingungen mit Mehrkosten in der Zielfunktion zu bestrafen und die Strafen sukzessive zu erhöhen, bis die Nebenbedingungen in hinreichender Weise eingehalten werden. Dadurch wird ein beschränktes Optimierungsproblem in eine Folge unbeschränkter Probleme transformiert, die dann mit Methoden des vorherigen Kapitels gelöst werden können. Wegen der Einfachheit dieser Idee und der Tatsache, daß die Struktur der Nebenbedingungen in diesem Ansatz nicht sonderlich berücksichtigt werden muß, haben diese Methoden in der Praxis eine weite Verbreitung, obwohl es in den meisten Fällen sicherlich sinnvoller wäre, die spezielle Struktur der Nebenbedingungen auszunutzen.

4.2.1 Das Grundkonzept der Penalty-Verfahren

Statt des Ausgangsproblems (4.1) soll jetzt ein Ersatzproblem der Form

$$P(x,R) = f(x) + R\sum_{j=1}^{J}\Omega(g_j(x)) + R\sum_{k=1}^{K}\Omega(h_k(x)) \to \min \qquad \text{mit} \quad x \in I\!R^n \quad (4.5)$$

gelöst werden, wobei die Ω-Terme die Strafterme sind, welche die Nichteinhaltung der Nebenbedingungen mit einem positiven Beitrag zur Zielfunktion bestrafen. Mit Hilfe des Parameters R kann man das Gewicht der Strafterme so weit erhöhen, wie es notwendig ist, um die Nebenbedingungen im Rahmen der gewünschten Genauigkeit einzuhalten.

Für Ω kann man z.B. *quadratische Straffunktionen*

$$\begin{aligned} \Omega(g(x)) &= \min\{0, g(x)\}^2 \\ \Omega(h(x)) &= |h(x)|^2 \end{aligned}$$

wählen. Abbildung 4.1 verdeutlicht dies sowohl für Gleichungs- als auch für Ungleichungs-Nebenbedingungen.

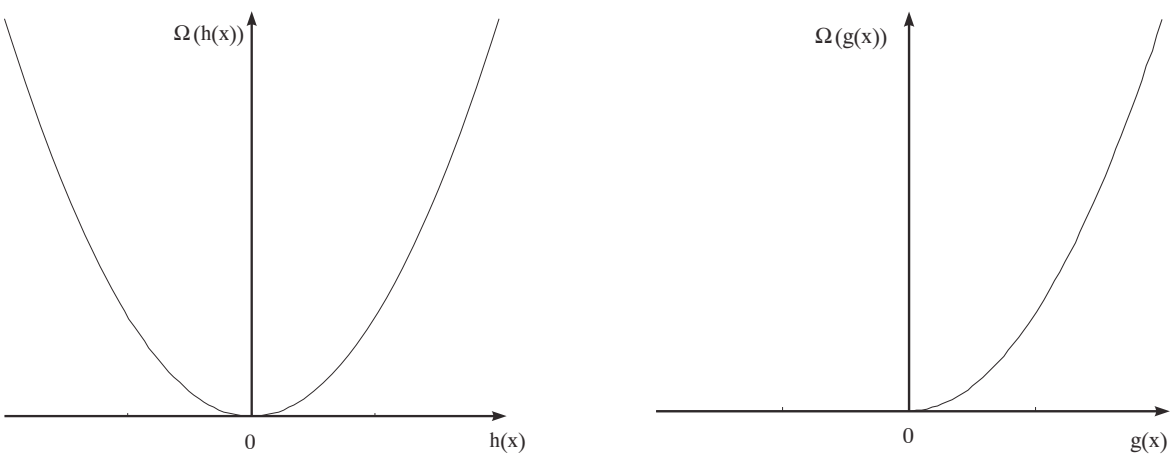


Abbildung 4.1: Quadratische Strafterme für Gleichungs- und Ungleichungs-Nebenbedingungen

Da sich die Folge $\{\bar{x}^k\}$ der Lösungen der Ersatzprobleme sukzessive der Lösung des Ausgangsproblems x^* annähert, ist es sinnvoll, die letzte Lösung als Startpunkt für die nächste Iteration zu verwenden.

Das führt zu folgendem Iterationsschema:

1. Wähle eine feste Folge $\{R^k\}_{k\in\mathbb{N}}$, z.B. $\{1, 10, 10^2, 10^3, \ldots\}$ und einen beliebigen Startpunkt $\bar{x}^0$.

2. Bestimme das Minimum $\bar{x}^k$ von $P(x, R^k)$ mit dem Startpunkt $\bar{x}^{k-1}$.

3. Terminiere, wenn die Nebenbedingungen hinreichend eingehalten werden, sonst gehe zu 2.

Daß man den Parameter R sukzessive erhöht und nicht gleich mit einem hinreichend großen R beginnt, liegt daran, daß die resultierenden Ersatzprobleme zunehmend schwieriger zu lösen sind. Man hat also die Wahl, entweder wenige schwierige oder mehrere leichte Probleme zu lösen, und die Erfahrung zeigt, daß wohl ein Mittelweg anstrebenswert scheint.

Mehrere Effekte führen zur schlechten Lösbarkeit der Ersatzprobleme:

- Für große R sind die Hesse-Matrizen der erweiterten Zielfunktion schlecht konditioniert. Das läßt sich sofort an dem Beispiel $P(x, R) = x^2 + Ry^2$ erkennen. Obwohl die Newton-ähnlichen Verfahren (zumindest bei quadratischen Funktionen) theoretisch nicht sensibel gegenüber schlechter Konditionierung sind, werden sie in der Praxis durch Rundungsfehler dennoch merklich schlechter.

- Ein weiterer Effekt ist, daß bei allgemeinen nichtlinearen Nebenbedinungen gekrümmte, steil umrandete Täler entstehen können. Bei solchen Problemen haben Abstiegsverfahren allgemein eine schlechte Konvergenz, weil viele Richtungswechsel notwendig werden.

- Als letztes soll hier noch ein numerisches Problem erwähnt werden. Sei ein Problem nur mit Gleichungs-Nebenbedingungen gegeben, d.h. $J = 0$ in Problem (4.1), und sei $\bar{x}$ eine Lösung, in der die *constraint qualification* erfüllt ist. Für $l \to \infty$ und $R_l \to \infty$ konvergieren die Lösungen der Ersatzprobleme $\bar{x}^l$ gegen $\bar{x}$, und es gilt

$$\nabla f(x^l) = \sum_{k=1}^{K} 2R^l h_k(x^l) \nabla h_k(x^l) \tag{4.6}$$

  Daraus folgt

$$\lim_{l\to\infty} 2R^l h_k(x^l) = -\mu_k \tag{4.7}$$

  d.h., das Produkt $2R^l h_k(x^l)$ konvergiert gegen den negativen Lagrange-Multiplikator. Da $R^l \to \infty$ und $h_k(x^l) \to 0$ für $k \to \infty$, ist das Produkt numerisch nicht stabil.

Alle diese Probleme sind darin begründet, daß die Gewichte der Strafterme zu groß werden. Deswegen wäre es wünschenswert, wenn man die Strafterme so gestalten könnte, daß der Parameter R nicht beliebig erhöht werden muß, um die Konvergenz zu gewährleisten. Das wird in den sogenannten Multiplier-Verfahren realisiert, von denen nun ein spezielles vorgestellt wird, das auch in den Implementationen benutzt wurde.

4.2.2 Method of Multipliers (MOM)

Folgende Idee führt zum Ansatz der "Method of Multipliers". Anstatt die Gewichte der Strafterme sukzessive zu erhöhen, kann man die Strafterme auch in "geeigneter Weise" verschieben. Die Abbildung 4.2 zeigt sowohl die vorher beschriebene Technik, durch Erhöhen des Gewichtes R als auch die, durch Verschieben der Strafterme die Einhaltung der Nebenbedingungen zu erzwingen. Dabei wird davon ausgegangen, daß die Gleichungs-Nebenbedingung nach oben verletzt wird, d.h. $h(x) > 0$ ist.

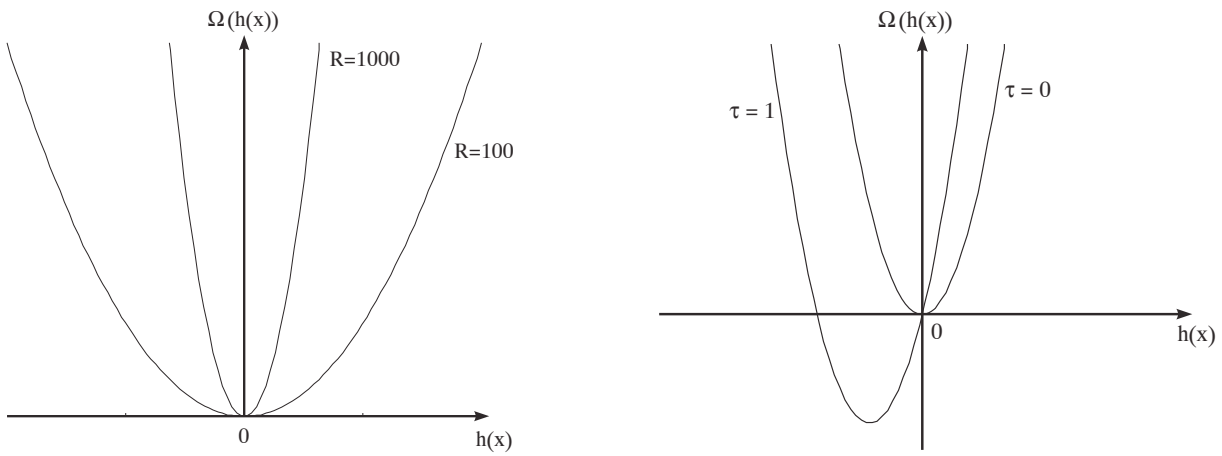


Abbildung 4.2: links: Erhöhung des Gewichts R
rechts: Verschiebung der Straffunktion um τ

Wie weit die Strafterme verschoben werden müssen, hängt direkt von den Lagrange-Multiplikatoren ab, deren Bedeutung im Zusammenhang mit Penalty-Verfahren schon durch (4.6) und (4.7) angedeutet wurde. Die genauen Zusammenhänge werden in den folgenden Darstellungen klarer.

Zur Verkürzung der Schreibweise soll im weiteren $\langle\alpha\rangle := \min\{0, \alpha\}$ sein.

Die modifizierte Straffunktion hat die Gestalt

$$P(x, \sigma, \tau) = f(x) + R \sum_{j=1}^{J} \left[\langle g_j(x) + \sigma_j \rangle^2 - \sigma_j^2 \right] + R \sum_{k=1}^{K} \left[(h_k(x) + \tau_k)^2 - \tau_k^2 \right], \quad (4.8)$$

wobei jetzt R ein konstanter Faktor ist und sich nur die Parameter-Vektoren σ und τ nach jeder unbeschränkten Optimierung verändern.

Die Aufdatierung wird so durchgeführt, daß die σ_j und τ_k die Lagrange-Multiplikatoren approximieren. Man startet mit $\sigma = \tau = 0$, wodurch das erste Ersatzproblem identisch ist mit dem zuerst vorgestellten Ansatz.

Wenn der Vektor $\bar{x}^t$ die Penalty-Funktion $P(x, \sigma^t, \tau^t)$ der t-ten Stufe minimiert, dann wählt man die σ^{t+1} und τ^{t+1} wie folgt:

$$\begin{aligned} \sigma_j^{t+1} &= \left\langle g_j(\bar{x}^t) + \sigma_j^t \right\rangle \qquad & j = 1, 2, \ldots, J \\ \tau_k^{t+1} &= h_k(\bar{x}^t) + \tau_k^t \qquad & k = 1, 2, \ldots, K \end{aligned} \tag{4.9}$$

Je stärker die Verletzung der Nebenbedingungen ist, desto größer wird der Betrag der σ_j und τ_k und somit auch die Bestrafung der verletzten Nebenbedingung. Dieser Effekt stellt die Konvergenz gegen einen zulässigen Punkt sicher.

Iterationsschema für das MOM-Verfahren

1. Starte mit $\sigma_j^0 = \tau_k^0 = 0$ und $\bar{x}^0$ beliebig.
2. Bestimme das Minimum $\bar{x}^t$ der Straffunktion $P(x, \sigma, \tau)$ mit dem Startpunkt $\bar{x}^{t-1}$.
3. Berechne die σ_j^{t+1} und τ_k^{t+1} mit den Update-Formeln (4.9).
4. Wenn die Nebenbedingungen hinreichend eingehalten werden, breche ab, sonst gehe zu 2.

Unter welchen Bedingungen dieses Verfahren konvergiert, soll hier nicht behandelt werden (siehe dazu [Sc75]). Es wird aber gezeigt, daß, wenn das Verfahren einen Fixpunkt hat, dieser auch ein KKT-Punkt für das Ausgangsproblem ist und daß die σ_j und τ_k eine Approximation der Lagrange-Multiplikatoren darstellen.

Sei $\bar{x}^*$ der Grenzwert der Folge $\{\bar{x}^t\}_{t \in N}$ und σ^*,τ^* die zugehörigen Verschiebungsvektoren. Aus der Konvergenz der Folge $\tau^{t+1} = h_k(\bar{x}^t) + \tau_k^t$ folgt sofort, daß $h_k(\bar{x}^t) \to 0$. Analog folgt aus der Konvergenz von $\sigma_j^{t+1} = \left\langle g_j(\bar{x}^t) + \sigma_j^t \right\rangle$, daß entweder $g_j(\bar{x}^*) > 0$ und $\sigma_j^* = 0$, oder $g_j(\bar{x}^*) = 0$ und $\sigma_j^* \leq 0$.

Für den Gradienten der Penalty-Funktion gilt

$$\nabla P(\bar{x}^*) = \nabla f(\bar{x}^*) + 2R \sum_{j=1}^{J} \sigma_j^* \nabla g_j(\bar{x}^*) + 2R \sum_{k=1}^{K} \tau_k^* \nabla h_k(\bar{x}^*) = 0 \tag{4.10}$$

mit

$$\begin{aligned} g_j(\bar{x}^*) &\geq 0 \qquad j = 1, 2, \ldots, J \\ \sigma_j^* g_j(\bar{x}^*) &= 0 \qquad j = 1, 2, \ldots, J \\ \sigma_j^* &\leq 0 \qquad j = 1, 2, \ldots, J \\ h_k(\bar{x}^*) &= 0 \qquad k = 1, 2, \ldots, K \end{aligned}$$

was direkt äquivalent zu den Karush-Kuhn-Tucker-Bedingungen ist. Das zeigt also, daß jeder Grenzwert $\bar{x}^*$ der Folge der Ersatzprobleme auch ein KKT-Punkt für das Ausgangsproblem ist. Aus (4.10) ersieht man auch sofort, daß

$$\lambda_j = -2R\sigma_j^t \tag{4.11}$$

$$\mu_k = -2R\tau_k^t \tag{4.12}$$

wirklich Approximationen der Lagrange-Multiplikatoren darstellen.

Es verbleiben nun noch einige Bemerkungen zu den Eigenschaften der durch das MOM-Verfahren definierten Penalty-Funktionen.

Speziell bei linearen Restriktionen, auf die dieses Verfahren hier auch angewendet wird, ergibt sich für die Hesse-Matrix der Penalty-Funktion

$$\nabla^2 P(x) = \nabla^2 f(x) + 2R \sum_{j=1}^{J} [\nabla g_j(x)]^2 + 2R \sum_{k=1}^{K} [\nabla h_k(x)]^2, \tag{4.13}$$

d.h. die Konditionierung der Ersatzprobleme ist nicht von σ und τ abhängig, und für eine vernünftige Wahl von R kann man erwarten, daß die Ersatzprobleme ähnlich gut zu lösen sind wie die Ausgangsprobleme.

Aber gerade eine sinnvolle Wahl von R ist nicht ganz einfach und kann meistens nur experimentell durchgeführt werden. Wählt man R zu klein, so kann es sein, daß das Verfahren nicht konvergiert, und für sehr große R sind die Ersatzprobleme schlecht konditioniert.

Ein weiteres Problem bei transformativen Ansätzen allgemein ist die Skalierung der Nebenbedingungen. Wenn die Werte der Variablen und der Nebenbedingungen nicht in der gleichen Größenordung liegen, kann es passieren, daß die Strafterme durch einzelne Nebenbedingungen dominiert werden und somit die Konvergenz der Verfahren verschlechtert wird. Dieser Effekt kann jedoch durch eine Skalierung der Nebenbedingungen verhindert werden.

Nachdem nun mit der Method of Multipliers ein Verfahren zur Verfügung steht, um Probleme mit Nebenbedingungen zu lösen, wird im nächsten Kapitel beschrieben, wie dieses zusammen mit den Quasi-Newton-Verfahren aus Kapitel 3 auf das Raffineriemodell angewendet werden kann.

Kapitel 5

Die sequentielle Anwendung der Verfahren

In diesem Kapitel wird demonstriert, wie sich die bisher dargestellten Verfahren auf das mathematische Modell der Raffinerie anwenden lassen. Bevor man die Verfahren jedoch anwendet, müssen noch einige Modifikationen an der mathematischen Formulierung durchführt werden, um numerische Probleme zu vermeiden. Außerdem wird die Konvexität des Problems nachgewiesen, was sicherstellt, daß die Minima, die mit den Quasi-Newton-Verfahren gefunden werden, wirklich eine globale Lösung für das Problem sind. Schließlich werden die vorgestellten Optimierungsalgorithmen in verschiedenen Kombinationen auf das Raffineriemodell und einige weitere Testprobleme angewendet, um einen Eindruck von den Eigenschaften der Verfahren zu gewinnen.

5.1 Das Raffineriemodell

5.1.1 Konvexität

Das zum Maximierungsproblem (2.2) äquivalente Minimierungsproblem lautet:

$$\begin{aligned} &\text{minimiere} \qquad f(x) = \sum_{i=1}^{4} -\tfrac{1}{1-\alpha_i} x_i^{1-\alpha_i} + c^T x \qquad x \in I\!R^n \\ &\text{mit den Nebenbedingungen} \\ &x_i \geq 0 \qquad i = 1, \ldots, n \\ &Ax \geq 0 \qquad A \in I\!R^{m \times n} \qquad \text{mit } n = 18,\ m = 12 \end{aligned} \tag{5.1}$$

Der Satz 4.2 besagt, daß jedes lokale Minimum von (5.1) auch ein globales Minimum ist, wenn sowohl die Zielfunktion als auch der Restriktionsbereich konvex ist.

Die Konvexität des Restriktionsbereichs folgt aus der Linearität der Nebenbedingungen. Jede lineare Ungleichungs-Nebenbedingung definiert einen Halbraum des $\mathbb{R}^n$ und ist damit offensichtlich konvex. Die Gesamtheit der linearen Ungleichungs-Nebenbedingungen definiert dann den Durchschnitt einer Menge von Halbräumen. Da der Durchschnitt konvexer Mengen wieder eine konvexe Menge ist, folgt daraus die Behauptung.
Die Konvexität der Zielfunktion auf dem Restriktionsbereich ist ebenfalls relativ einfach zu zeigen. Dazu bedient man sich des Satzes (3.2), der den Zusammenhang zwischen der Konvexität von f und der positiven Semi-Definitheit der Hesse-Matrix von f aufzeigt. Auf $\mathbb{R}^n_{>0}$ ist die Zielfunktion f aus Problem (5.1) zweimal differenzierbar, und für $\nabla^2 f(x)$ mit $x \in \mathbb{R}^n_{>0}$ ergibt sich die Diagonalmatrix

$$\nabla^2 f(x) = \begin{pmatrix} \alpha_1 x_1^{-(\alpha_1+1)} & & & & & \\ & \ddots & & & 0 & \\ & & \alpha_4 x_4^{-(\alpha_4+1)} & & & \\ & & & 0 & & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{pmatrix}. \tag{5.2}$$

Eine Diagonalmatrix D ist genau dann positiv semi-definit, wenn ihre Diagonalelemente nicht negativ sind, d.h. $d_{ii} \geq 0$. Da $\alpha_i x_i^{-(\alpha_i+1)} \geq 0$ für $x_i > 0$, folgt daraus die Konvexität von f auf $\mathbb{R}^n_{>0}$ und zusammen mit der Stetigkeit von f auch für den Quadranten $\mathbb{R}^n_{\geq 0}$, der eine Obermenge der Restriktionsmenge ist.
Es bleibt noch zu untersuchen, ob die Strafterme der Method of Multipliers die Konvexität der Zielfunktion erhalten und so die Ersatzprobleme ebenfalls konvex sind. Dazu sei bemerkt, daß eine Summe von konvexen Funktionen wieder eine konvexe Funktion ergibt. Das ist leicht einzusehen, da die Summe zweier positiv definiter Matrizen ebenfalls positiv definit ist. Es ist also noch zu zeigen, daß die Strafterme

$$p_h(x) = (h(x) + \tau)^2 - \tau^2 \tag{5.3}$$
$$p_g(x) = \langle g(x) + \sigma \rangle^2 - \sigma^2 \tag{5.4}$$

für lineare Funktionen g und h ebenfalls konvex sind. Der Nachweis wird für die Gleichungs-Nebenbedingungen erbracht, der für die Ungleichungs-Nebenbedingungen erfolgt mit Fallunterscheidung analog. Am einfachsten ist der Beweis wieder über die Hesse-Matrix der Straffunktionen zu führen. Sei jetzt $h(x) = c^T x + b$ mit $c \in \mathbb{R}^n$ und $b \in \mathbb{R}$, dann gilt

$$\begin{aligned} (h(x) + \tau)^2 - \tau^2 &= h(x)^2 + 2h(x)\tau \\ &= (c^T x + b)^2 + 2(c^T x + b)\tau \\ &= (c^T x)^2 + 2c^T x b + b^2 + 2(c^T x + b)\tau \end{aligned}$$

Bis auf $(c^T x)^2$ sind alle Terme linear, so daß sie keinen Einfluß auf die Hesse-Matrix haben. Es ist

$$(c^T x)^2 = (c^T x)(c^T x) \tag{5.5}$$

$$= x^T(cc^T)x \tag{5.6}$$

$$= x^T \begin{pmatrix} c_1c_1 & \cdots & c_1c_n \\ \vdots & & \vdots \\ c_nc_1 & \cdots & c_nc_n \end{pmatrix} x \tag{5.7}$$

Daraus folgt für die Hesse-Matrix

$$\nabla^2 p_h(x) = 2 \begin{pmatrix} c_1c_1 & \cdots & c_1c_n \\ \vdots & & \vdots \\ c_nc_1 & \cdots & c_nc_n \end{pmatrix}. \tag{5.8}$$

Die Matrix $\nabla^2 p_h(x)$ ist genau dann positiv semi-definit, wenn für alle $x \in \mathbb{R}^n$ $x^T(\nabla^2 p_h(x))x \geq 0$.

$$x^T(\nabla^2 p_h(x))x = 2x^T \begin{pmatrix} c_1c_1 & \cdots & c_1c_n \\ \vdots & & \vdots \\ c_nc_1 & \cdots & c_nc_n \end{pmatrix} x \tag{5.9}$$

$$= 2(c^Tx)^2 \tag{5.10}$$

$$\geq 0 \tag{5.11}$$

Damit ist bewiesen, daß sowohl das Raffineriemodell als auch die Ersatzprobleme, die mit der Method of Multipliers erzeugt werden, konvex sind, wodurch sich die Suche nach möglichen Nebenminima erübrigt.

Die Zielfunktion wirft aber dennoch ein Problem auf, da sie nicht definiert ist, wenn $x_i < 0$ für ein $i \in \{1, 2, 3, 4\}$. Die Punkte, auf denen f nicht definiert ist, gehören zwar nicht zum zulässigen Bereich, aber beim Ansatz der Penalty-Verfahren sind die erzeugten Iterationspunkte i. allg. keine zulässigen Punkte, so daß man nicht ausschließen kann, daß im Verlauf des Verfahrens eine der kritischen Komponenten negativ wird. Außerdem ist der Gradient von f, der von allen Quasi-Newton-Verfahren benutzt wird, im Nullpunkt, der offensichtlich ein zulässiger Punkt ist, ebenfalls nicht definiert.
Es bieten sich zwei Möglichkeiten an, dieses Problem zu vermeiden. Entweder wählt man so lange Startpunkte $x^s > 0$, bis man mit dem Verfahren eine stabile Punktfolge erzeugen konnte, die gegen das Minimum konvergiert, oder man modifiziert die Zielfunktion so, daß sowohl $f(x)$ als auch $\nabla f(x)$ auf ganz $\mathbb{R}^n$ definiert sind. Da der erste Ansatz zeitaufwendig sein kann und nicht zu gewährleisten ist, daß überhaupt jemals ein geeigneter Startpunkt gefunden wird, wurde hier der zweite Weg, die Modifikation der Zielfunktion, gewählt.

5.1.2 Modifikationen der Zielfunktion

Die Zielfunktion soll so verändert werden, daß sowohl $f(x)$ als auch $\nabla f(x)$ auf ganz $\mathbb{R}^n$ definiert sind. Das geschieht dadurch, daß die kritischen Terme

$$f_i(x_i) = \frac{1}{1-\alpha_i} x_i^{1-\alpha_i} \qquad i = 1, \ldots, 4$$

für $x_i < \epsilon$ einfach durch lineare Terme ersetzt werden, die die Potenzfunktion und ihre erste Ableitung stetig fortsetzen:

$$f_i^{mod}(x_i) = \begin{cases} \frac{1}{1-\alpha_i} x_i^{1-\alpha_i} & x_i \geq \epsilon \\ a_i x_i + b_i & x_i < \epsilon \end{cases} \tag{5.12}$$

Dabei sind $a_i = f_i'(\epsilon)$ und $b_i = f_i(\epsilon) - a\epsilon$ so gewählt, daß $f_i^{mod}(x_i)$ und ihre erste Ableitung im Punkt ϵ stetig sind. Nach dieser Modifikation ist $f^{mod}(x)$ und $\nabla f^{mod}(x)$ offensichtlich auf ganz $\mathbb{R}^n$ definiert und auch konvex. Es verbleibt noch zu prüfen, ob durch diese Veränderung der Zielfunktion möglicherweise auch die Lösung des Problems verschoben wurde.
Eine Beeinflussung ist nur dann möglich, wenn bei der ursprünglichen Lösung x^* eine der kritischen Komponenten kleiner als ϵ ist, d.h. wenn $x_i^* < \epsilon$ für ein $i \in \{1,2,3,4\}$. A priori läßt sich nicht sagen, ob das zutrifft oder nicht. Die Bedeutung der Komponenten im Modell legt jedoch nahe, daß dieser Fall wahrscheinlich nicht eintritt, da diese Komponenten gerade die Menge der verkauften Produkteinheiten angeben und daher im Optimum nicht beliebig klein sein werden. Auf jeden Fall kann man sicher sein, daß die Lösung nicht verändert wurde, wenn für die Lösung $\tilde{x}^*$ des modifizierten Problems $\tilde{x}_i^* > \epsilon$ für $i = 1,2,3,4$ gilt.

5.1.3 Skalierung der Problemvariablen und Nebenbedingungen

Wie in Kapitel 4 schon erwähnt, bleibt eines der Hauptprobleme bei der Anwendung von Penalty-Verfahren die Skalierung der Problemvariablen und der Nebenbedingungen. Es soll kurz erläutert werden, was genau man unter der Skalierung versteht und wie man sie grundsätzlich durchführt. Die Darstellungen orientieren sich dabei im wesentlichen an [RRR83], wo auch ein Algorithmus zur Skalierung nachgelesen werden kann.
Die Skalierung der Problemvariablen hat zum Ziel, daß die Problemvariablen für realistische Werte alle in der gleichen Größenordung liegen. So wird man anstreben, daß z.B. alle Variablen im Intervall $[0.1, 10]$ liegen. Um dies zu erreichen, werden die Variablen ggf. durch andere Variablen substituiert, die mit einer geeigneten Konstante multipliziert werden. Die Variable x_i kann z.B. durch cx_i' ersetzt werden, wobei c so zu wählen ist, daß x' im gewünschten Bereich liegt.
Die Skalierung der Nebenbedingungen erfolgt idealerweise so, daß die Gradienten

der Nebenbedingungen in der Nähe der Lösung etwa gleiche Beträge haben. Dadurch ist sichergestellt, daß die Werte der Nebenbedingungen ähnlich sensibel auf die Veränderung der Problemvariablen reagieren und so die Nebenbedingungen in den Straftermen etwa gleich gewichtet sind. Bei einer ungleichen Gewichtung können die Suchrichtungen durch einzelne Nebenbedingungen dominiert werden, wodurch die Konvergenz der Verfahren beeinträchtigt wird. Die Skalierung einer Nebenbedingung erfolgt durch Multiplikation mit einer geeigneten Konstante. I. allg. ist es nicht einfach, diese Konstante zu bestimmen, da man eine gute Schätzung der Lösung braucht. Bei linearen Nebenbedingungen sind die Gradienten allerdings auf dem ganzen Definitionsbereich konstant, so daß sich eine Schätzung der Lösung erübrigt.
Hier wurden die Gradienten aller Nebenbedingungen auf 1 normiert, und auf eine Skalierung der Variablen wurde verzichtet, was durch die Lösung des Problems gerechtfertigt scheint.

Jetzt ist das mathematische Modell endlich so weit, daß man die Verfahren darauf anwenden kann, und es verbleibt nur noch die Wahl einiger Parameter, wie z.B. die Gewichtung der Strafterme und die durch die Abbruchkriterien gewährleistete Genauigkeit. Doch bevor die ersten Meßergebnisse präsentiert werden, soll noch ein weiteres verfahrensspezifisches Problem angesprochen werden.

5.1.4 Regularisierung der Quasi-Newton-Verfahren

Wenn man die vorgestellten Quasi-Newton-Verfahren auf das Raffineriemodell anwendet, so konvergiert das BFGS-Verfahren bei sinnvoller Parameterwahl gegen die korrekte Lösung, während das Straeter-Verfahren nach einigen Iterationen mit undefinierter Quasi-Newton-Matrix abbricht.
Bei näherer Betrachtung sieht man, daß der Effekt dadurch zustande kommt, daß die Differenzen der Gradienten, die man für die partiellen Updates benutzt, 0 werden, d.h. $y^{k_j} = 0$ und damit $\tau^{k_j} = ((r^{k_j})^T y^{k_j})^{-1}$ undefiniert ist.
Wieso dieser Effekt beim Straeter-Verfahren auftaucht, aber beim BFGS-Verfahren nicht zu beobachten ist, läßt sich leicht einsehen.
Wie den Ausführungen über das Straeter-Verfahren zu entnehmen ist, werden in jedem Update-Schritt n partielle Broyden-Upates entlang linear unabhängiger Richtungen s^{k_j}, $j = 1, \ldots, n$ durchgeführt. Hier wurden die $s^{k_j} = \epsilon\, e^j$ gewählt, mit e^j als dem j-ten Einheitsvektor und $\epsilon = 10^{-10}$. Die Zielfunktion in (5.1) ist aber so beschaffen, daß

$$\nabla f(x + \epsilon\, e^j) - \nabla f(x) = 0 \qquad \text{für } j = 5, \ldots, n$$

und damit ist sofort $y^{k_j} = 0$, wenn nicht die Strafterme einen Beitrag zum Gradienten leisten, damit $\nabla P(x + \epsilon\, e^j) - \nabla P(x) \neq 0$ wird ($P(x)$ ist hier die Straffunktion bzw. die Zielfunktion des Ersatzproblems). Im Problem (5.1) kommen aber nur Ungleichungs-Nebenbedingungen vor, die bei Einhaltung keinen Beitrag zum Funktionswert oder zum Gradienten leisten. Wenn also alle Nebenbedinungen eingehalten

werden, in die x_i, $i \in \{5, \ldots, n\}$, eingeht, dann folgt daraus sofort, daß

$$\nabla P(x + \epsilon\, e^j) - \nabla P(x) = 0$$

und damit auch der oben beschriebene Effekt. Die wohl einfachste Maßnahme, dieses Problem zu vermeiden, ist, die partiellen Updates nur durchzuführen, wenn der Wert $\tau^{k_j} = ((r^{k_j})^T y^{k_j})^{-1}$ definiert ist.
Beim BFGS-Verfahren treten solche Probleme wesentlich seltener auf, da die Update-Richtungen nicht so "regelmäßig" sind. Dennoch empfiehlt es sich auch hier, präventiv Abfragen vorzunehmen, die verhindern, daß einer der Quotienten undefiniert wird. Für den Fall, daß das Update undefiniert ist, wird ebenfalls einfach auf dieses Update verzichtet und mit der alten Quasi-Newton-Matrix weitergerechnet.

Nachdem nun die wesentlichen Probleme bzgl. der Modellierung und der Quasi-Newton-Verfahren beseitigt sind, können die Verfahren zur Lösung des Problems eingesetzt werden.

5.2 Testprobleme

Neben dem Bemühen, das Konvergenzverhalten von Straeter- und BFGS-Verfahren besser miteinander vergleichen zu können, ist ein wesentlicher Grund für die Hinzunahme weiterer Testprobleme die Notwendigkeit, für die Analyse der parallelen Verfahren Probleme variabler Größe zu haben. Da keine realen Optimierungsprobleme geeigneter Größe und Struktur verfügbar waren und auch in der Literatur (z.B. [HS81]) nur relativ kleine Probleme genannt werden, wurden hier kleinere Probleme iteriert, d.h. mehrere identische Probleme werden zu einem System von Problemen zusammengefaßt und über eine gemeinsame Zielfunktion gekoppelt. Da die Quasi-Newton-Verfahren nicht strukturerhaltend sind, scheint diese Vorgehensweise durchaus gerechtfertigt, wenngleich immer die Gefahr besteht, daß sich doch gewisse Symmetrien ausbilden.

Das iterierte Raffineriemodell

Da das Raffineriemodell für die Analyse paralleler Verfahren nicht ausreicht, wird ein in der Größe skalierbares Modell konstruiert, das die gleichen Struktureigenschaften hat. Wie oben schon erwähnt, erfolgt dies durch Iterieren des einfachen Raffineriemodells.
Wenn das einfache Modell, hier zusätzlich mit dem Parameter p bezeichnet, die Form

$$\text{minimiere} \qquad f^{(p)}(x^{(p)}) = \sum_{i=1}^{4} -\tfrac{1}{1-\alpha_i}(x_i^{(p)})^{1-\alpha_i} + c^T x^{(p)} \qquad x^{(p)} \in I\!R^n$$

mit den Nebenbedingungen

$$x_i^{(p)} \geq 0 \qquad i = 1, \ldots, n$$
$$Ax^{(p)} \geq 0 \qquad A \in I\!R^{m \times n}$$

hat, dann soll das N-fach iterierte Modell definiert sein als

$$\begin{aligned} &\text{minimiere} \quad R^{(N)}(y) = \sum_{p=1}^{N} f^{(p)}(x^{(p)}) \qquad y = (x^{(1)}, \ldots, x^{(N)})^T \in I\!R^{N \times n} \\ &\text{mit den Nebenbedingungen} \qquad (5.13) \\ &x_i^{(p)} \geq 0 \qquad\qquad i = 1, \ldots, n \quad p = 1, 2, \ldots, N \\ &Ax^{(p)} \geq 0 \qquad\qquad A \in I\!R^{m \times n} \qquad p = 1, 2, \ldots, N \end{aligned}$$

Da für das einfache Modell $n = 18$ und $m = 12$ gilt, hat das iterierte Problem die Dimension $n' = 18N$ mit $m' = 12N$ Nebenbedingungen und n' Bounds. Als Startpunkt für das einfache wie für das iterierte Raffineriemodell wird der Nullpunkt gewählt, d.h. $x^0 = 0$.

Iterierte Rosenbrock-Funktion

Die Rosenbrock-Funktion ist wohl die bekannteste Testfunktion aus dem Bereich der nichtlinearen Optimierung. Im zweidimensionalen Fall ist sie durch

$$f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$$

definiert und beschreibt ein parabolisch gekrümmtes, steil umrandetes Tal, wodurch Abstiegsverfahren zu vielen Richtungswechseln gezwungen werden. Obwohl für konvexe Zielfunktionen und lineare Nebenbedingungen bei Anwendung der Method of Multipliers solche Geometrien nicht auftreten können, gibt die Rosenbrock-Funktion einen guten Aufschluß über das Konvergenzverhalten von Verfahren. Die iterierte Rosenbrock-Funktion ist definiert als

$$f(x) = \sum_{i=1}^{n/2} \left[100(x_{2i-1}^2 - x_{2i})^2 + (1 - x_{2i-1})^2 \right]. \qquad (5.14)$$

Startpunkt für alle Testläufe ist $x^0 = 0$. Da dieses Problem keine Nebenbedingungen hat, ist die Berechnung der Funktionswerte und Gradienten im Vergleich zum (iterierten) Raffineriemodell recht schnell durchführbar.

5.3 Meßergebnisse für die sequentiellen Verfahren

Ziel der Messungen ist es, die numerischen Eigenschaften der Quasi-Newton-Verfahren und ihr Zusammenspiel mit den verschiedenen Algorithmen zur Strahlminimierung zu beleuchten, um die Verfahrensvarianten zu bestimmen, die später parallelisiert werden sollen.

Verfahren	Abbruchkriterium bzw. Parameter
BFGS- und Straeter-Verfahren	$\max_i(\nabla_i f(x))^2 \leq 10^{-9}$ oder $\max_i(x_i^{k+1} - x_i^k)^2 \leq 10^{-30}$
Method of Multipliers	$R = 10^5 \quad \max_i \min\{0, g_i(x)\} \leq 10^{-8}$
Strahlminimierung nach Fletcher	$\sigma = 0.1 \quad \rho = 0.01$
Verfahren des golden Schnitts	$\epsilon = 10^{-8}$
Armijo-Schrittweite	$\delta = 0.5$

Tabelle 5.1: *Verfahrensparameter*

Folgende Quasi-Newton-Verfahren wurden getestet:

1. Das klassische BFGS-Verfahren
2. Das Straeter-Verfahren mit einem vollständigen Update in jeder Iteration
3. Das Straeter-Verfahren mit einem halben Update in jeder Iteration, d.h. in jedem Update werden zyklisch nur die Hälfte der partiellen Updates durchgeführt
4. Das "gedächtinslose" Straeter-Verfahren mit einem vollständigen Update, bei dem nach jeder Iteration die Quasi-Newton-Matrix auf die Einheitsmatrix zurückgesetzt wird

Alle Quasi-Newton-Verfahren wurden mit den drei hier vorgestellten Methoden zur Strahlminimierung kombiniert. Dabei wurden die Verfahrensparameter so gewählt, daß die Strahlminimierung nach Fletcher und die des goldenen Schnitts relativ exakte Approximationen der optimalen Schrittweite liefern und die Armijo-Schrittweite naturgemäß nur sehr grobe Approximationen. Die anderen Abbruchkriterien und Parameter lassen sich der Tabelle 5.1 entnehmen. Alle im weiteren dargestellten Ergebnisse wurden mit diesem Parametersatz durchgeführt, der sich beim einfachen Raffineriemodell als günstig erwiesen hat. Auf eine individuelle Anpassung der Parameter auf die anderen Probleme wurde verzichtet.

Folgende Aspekte sollen im Mittelpunkt der Betrachtungen stehen:

- Der Einfluß der Strahlminimierung auf die Konvergenz und Rechenzeit
- Gedächtnisloses Straeter-Verfahren vs. Standard-Straeter-Verfahren
- Der Einfluß unvollständiger Updates

5.3.1 Meßergebnisse für das Raffineriemodell

Da das Raffineriemodell hier das zentrale Testproblem für die Verfahren ist, werden zunächst alle Verfahren am einfachen und an zwei iterierten Modellen ausprobiert. Im zweiten Schritt werden dann auftretende Trends am anderen Testproblem verifiziert werden.

Die Tabellen 5.2, 5.3 und 5.4 zeigen die Meßergebnisse für das BFGS-Verfahren und die drei Varianten des Straeter-Verfahrens in Kombination mit den verschiedenen Algorithmen zur Strahlminimierung.

Von jeder Variante sind die insgesamt benötigten Iterationen der Quasi-Newton-Verfahren und der Strahlminimierung, die Rechenzeit insgesamt und pro Iteration, der Rechenzeitanteil der Strahlminimierung und ein Performance-Index angegeben. Die Iterationszahlen sind jeweils die Summen der zur Lösung aller durch die Method of Multipliers erzeugten Ersatzprobleme benötigten Iterationen. Nicht angegeben ist die Anzahl der benötigten Ersatzprobleme, da sie bei allen Verfahren fast identisch waren und nur einen vernachlässigbaren Anteil an den Kosten der Verfahren haben (Beim einfachen Modell waren es 3, beim zweifach iterierten Modell 4 und beim dreifach iterierten Modell 5 bis 8 Ersatzprobleme). Die Rechenzeiten wurden bei sequentieller Abarbeitung der Programme auf einem Knoten des Intel Paragon XP/S 10 erzielt. Auf diese Rechenzeiten bezieht sich auch der Performance-Index, der angibt, wieviel mal langsamer ein Verfahren im Vergleich zum schnellsten Verfahren war. Das schnellste Verfahren bekommt also den Index 1.0, und der Faktor 3.0 bedeutet, daß dieses Verfahren die dreifache Laufzeit des schnellsten Verfahrens hatte.

Man erkennt, daß die Iterationszahlen für das Straeter-Verfahren mit vollständigem Update deutlich niedriger liegen und dieser Unterschied mit der Problemgröße zunimmt, weil die Iterationszahlen beim BFGS-Verfahren kontinuierlich mit der Problemgröße ansteigen, während sie beim Straeter-Verfahren mit vollständigem Update im Rahmen gewisser Schwankungen konstant bleiben[1].
Dieser Effekt entspricht etwa den Erwartungen. Wenn man nämlich beide Verfahren auf quadratische Zielfunktionen der Dimension n anwendet, so wird das BFGS-Verfahren etwa n Schritte benötigen, das Straeter-Verfahren unabhängig von der Problemdimension dagegen nur genau einen Schritt. Anderseits ist der Aufwand für ein Straeter-Update auch ca. n-fach so groß wie für ein BFGS-Update, da im Straeter-Update n partielle Updates durchgeführt werden.
Deswegen ist das BFGS-Verfahren, außer bei Benutzung der Fletcherschen Strahlminimierung, auch meist schneller als die Straeter-Verfahren mit vollständigem Update. Dieser Unterschied nimmt ebenfalls mit der Problemgröße zu, d.h. je größer das Problem, desto größer wird der Zeitvorteil des BFGS-Verfahrens, wenn eine ableitungsfreie Strahlminimierung benutzt wird.

[1] Die Werte für das dreifach iterierte Problem beim Straeter-Verfahren mit vollständigem Update und der Strahlminimierung des goldenen Schnitts fallen extrem aus dem Rahmen und werden deswegen von den Betrachtungen ausgeklammert.

Die Unterschiede zwischen dem gedächtnislosen Straeter-Verfahren, bei dem nach jeder Iteration die Quasi-Newton-Matrix auf die Einheitsmatrix zurückgesetzt wird, und dem Standard-Straeter-Verfahren sind gering, wenngleich ein leichter Vorteil zugunsten der gedächtnislosen Variante erkennbar ist.

Der Einfluß unvollständiger Updates ist relativ unklar. Das Straeter-Verfahren, das in jedem Schritt nur die Hälfte der partiellen Updates durchführt, benötigt zwar weniger Zeit pro Iteration (die Zeit für das Update halbiert sich), aber dafür wird die Konvergenz der Verfahren schlechter. Beim einfachen Raffineriemodell erzielt man durch die halben Updates, zumindest in Kombination mit den ableitungsfreien Strahlminimierungen, einen Zeitvorteil gegenüber den Versionen mit vollständigem Update. (Bei Anwendung der Strahlminimierung nach Fletcher ist das nicht der Fall, weil diese einen erheblichen Anteil an der Rechenzeit hat und es günstiger ist, die Anzahl der Iterationen zu reduzieren, als die Zeit für das Update zu verkürzen.) Bei dem zweifach und dreifach iterierten Modell hingegen verhält sich die Variante mit halbem Update nicht mehr stabil. Es treten extreme Schwankungen in den Iterationszahlen auf, und in zwei Fällen mußte das Verfahren nach 1000 Iterationen erfolglos abgebrochen werden. Ein solch labiles Verhalten macht das Verfahren für praktische Anwendungen unbrauchbar.
Insgesamt scheint also von den Straeter-Verfahren die gedächtnislose Variante die erfolgversprechendste zu sein.

Interessant ist auch der Einfluß der Strahlminimierung auf die Iterationszahlen und die Rechenzeiten des gesamten Verfahrens. Die Strahlminimierung nach Fletcher ist durch die Benutzung von Ableitungen ganz klar das teuerste Verfahren. Sie dominiert im BFGS-Verfahren mit 87 % der Rechenzeit sogar die Kosten des gesamten Verfahrens. Das Verfahren des goldenen Schnitts ist zwar schon schneller, aber im Vergleich zur Armijo-Schrittweite immer noch erheblich aufwendiger. Mit der Armijo-Schrittweite läßt sich der Anteil der Strahlminimierung im BFGS-Verfahren je nach Problemgröße auf 11–21% senken. So war die Kombination des BFGS-Verfahrens mit der Armijo-Schrittweite bei allen drei Problemen das schnellste Verfahren, obwohl hier die meisten Iterationen benötigt werden.

Bei den Straeter-Verfahren (mit vollem Update) fallen die Kosten der Strahlminimierung nicht ganz so sehr ins Gewicht, weil die Updates wesentlich zeitaufwendiger sind. Trotzdem macht der Anteil der Strahlminimierung nach Fletcher noch 20–40% aus. Das Verfahren des goldenen Schnitts hat beim einfachen Problem nur einen Anteil von 11%, der beim zweifach und dreifach iterierten Problem auf 3 % bzw. 2% sinkt. Die Armijo-Schrittweite hat beim einfachen Problem einen Anteil von 4%, und bei den iterierten Varianten liegt er sogar unter 1%.

Mit der Armijo-Schrittweite läßt sich offensichtlich der Anteil der Strahlminimierung am gesamten Verfahren effektiv verringern. Beim BFGS-Verfahren steigt zwar dadurch ganz klar die Iterationszahl für das gesamte Verfahren, führt aber insgesamt dennoch zu einer Rechenzeitverkürzung.
Beim Straeter-Verfahren ist ein Vorteil der Armijo-Schrittweite nicht zu erkennen.

Bei Anwendung auf das einfache und das zweifach iterierte Problem ist die Kombination mit dem Verfahren des goldenen Schnitts am schnellsten. Obwohl sich dieser Trend im dreifach iterierten Modell nicht fortsetzt, scheint es plausibel zu sein, daß man beim Straeter-Verfahren eher die Anzahl der Quasi-Newton-Iterationen verringern sollte, als den Anteil der Strahlminimierung auf Kosten ihrer Genauigkeit zu reduzieren.

Tabelle 5.2: **Meßergebnisse für das einfache Raffineriemodell auf einem Knoten des Intel Paragon XP/S 10. 18 Variablen, 30 Nebenbedingungen**

BFGS-Verfahren			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	255	218	366
Strahlminimierung Iterationen insgesamt	1494	9592	1553
Rechenzeit in Sekunden	23.5	9.0	6.0
Rechenzeit pro Iteration in Sekunden	0.092	0.041	0.016
Anteil Rechenzeit Strahlminimierung	87 %	70 %	21 %
Performance-Index	3.89	1.58	1.0
Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	26	47	56
Strahlminimierung Iterationen insgesamt	345	2068	751
Rechenzeit in Sekunden	11.9	13.5	15.3
Rechenzeit pro Iteration in Sekunden	0.45	0.28	0.27
Anteil Rechenzeit Strahlminimierung	43 %	11 %	4 %
Performance-Index	1.97	2.23	2.53
gedächtnisloses Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	30	33	39
Strahlminimierung Iterationen insgesamt	425	1452	366
Rechenzeit in Sekunden	13.9	9.33	10.4
Rechenzeit pro Iteration in Sekunden	0.46	0.28	0.27
Anteil Rechenzeit Strahlminimierung	44 %	10 %	3%
Performance-Index	2.29	1.54	1.72
Straeter-Verfahren mit halbem Update pro Iteration			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	65	39	60
Strahlminimierung Iterationen insgesamt	1007	1716	677
Rechenzeit in Sekunden	23.0	6.8	8.95
Rechenzeit pro Iteration in Sekunden	0.35	0.17	0.15
Anteil Rechenzeit Strahlminimierung	62 %	18 %	6 %
Performance-Index	3.83	1.12	1.47

Tabelle 5.3: **Meßergebnisse für das zweifach iterierte Raffineriemodell auf einem Knoten des Intel Paragon XP/S 10. 36 Variablen, 60 Nebenbedingungen**

BFGS-Verfahren			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	319	331	500
Strahlminimierung Iterationen insgesamt	2235	14564	2602
Rechenzeit in Sekunden	127.2	35.47	29.1
Rechenzeit pro Iteration in Sekunden	0.40	0.11	0.05
Anteil Rechenzeit Strahlminimierung	87 %	54 %	15 %
Performance-Index	4.37	1.22	1.0
Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	28	37	43
Strahlminimierung Iterationen insgesamt	218	1628	352
Rechenzeit in Sekunden	66.3	73.2	83.6
Rechenzeit pro Iteration in Sekunden	2.36	1.98	1.94
Anteil Rechenzeit Strahlminimierung	18 %	3 %	< 1 %
Performance-Index	2.27	2.51	2.86
gedächtnisloses Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	28	30	46
Strahlminimierung Iterationen insgesamt	240	1320	444
Rechenzeit in Sekunden	67.3	57.22	85.4
Rechenzeit pro Iteration in Sekunden	2.40	1.90	1.85
Anteil Rechenzeit Strahlminimierung	20 %	3 %	< 1 %
Performance-Index	2.30	1.96	2.93
Straeter-Verfahren mit halbem Update pro Iteration			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	202	> 1000	91
Strahlminimierung Iterationen insgesamt	3103	-	1175
Rechenzeit in Sekunden	356.6	-	91.95
Rechenzeit pro Iteration in Sekunden	1.77	-	1.01
Anteil Rechenzeit Strahlminimierung	45 %	-	2 %
Performance-Index	12.23	-	3.15

Tabelle 5.4: **Meßergebnisse für das dreifach iterierte Raffineriemodell auf einem Knoten des Intel Paragon XP/S 10. 54 Variablen, 90 Nebenbedingungen**

BFGS-Verfahren			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	404	495	580
Strahlminimierung Iterationen insgesamt	2871	21780	3349
Rechenzeit in Sekunden	354.7	97.9	72.1
Rechenzeit pro Iteration in Sekunden	0.88	0.20	0.12
Anteil Rechenzeit Strahlminimierung	87 %	45 %	11 %
Performance-Index	4.92	1.35	1.0
Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	43	95	30
Strahlminimierung Iterationen insgesamt	541	4180	262
Rechenzeit in Sekunden	320.9	594.2	185.4
Rechenzeit pro Iteration in Sekunden	7.46	6.25	6.18
Anteil Rechenzeit Strahlminimierung	19 %	2 %	< 1 %
Performance-Index	4.45	8.24	2.57
gedächtnisloses Straeter-Verfahren mit vollständigem Update			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	41	35	37
Strahlminimierung Iterationen insgesamt	533	1540	244
Rechenzeit in Sekunden	313.3	214.4	223.6
Rechenzeit pro Iteration in Sekunden	7.64	6.13	6.04
Anteil Rechenzeit Strahlminimierung	19 %	2 %	< 1 %
Performance-Index	4.34	3.25	3.20
Straeter-Verfahren mit halbem Update pro Iteration			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	> 1000	229	127
Strahlminimierung Iterationen insgesamt	–	10076	1777
Rechenzeit in Sekunden	–	762.2	409.7
Rechenzeit pro Iteration in Sekunden	–	3.32	3.23
Anteil Rechenzeit Strahlminimierung	–	3 %	1 %
Performance-Index	–	9.85	5.68

5.3.2 Meßergebnisse für die Rosenbrock-Funktion

Die bisher beobachteten Trends sollen nun anhand der iterierten Rosenbrock-Funktion verifiziert werden. Der Vergleich der Rechenzeiten zwischen den einzelnen Verfahren wird wahrscheinlich anders ausfallen als bei dem Raffineriemodell, weil die Zielfunktion und der Gradient der iterierten Rosenbrock-Funktion wegen fehlender Nebenbedingungen nicht durch Strafterme verteuert werden. Die Tabelle 5.5 zeigt die Meßergebnisse für die iterierte Rosenbrock-Funktion mit 128 Variablen.
Wie beim Raffinerieproblem benötigt das BFGS-Verfahren wieder ein Vielfaches der Iterationen des Straeter-Verfahrens und ist dennoch deutlich schneller. Der Zeitunterschied zwischen BFGS- und Staeter-Verfahren ist sogar noch extremer als beim Raffineriemodell, da bei der Rosenbrock-Funktion die Verfahrenskosten primär durch die Updates bestimmt werden. Selbst beim BFGS-Verfahren liegt der Zeitanteil der Fletcherschen Strahlminierung nur bei 14 %. Das Verfahren des goldenen Schnitts hat einen Anteil von 17 % und die Armijo-Schrittweite sogar nur 5 %. Beim Straeter-Verfahren liegt der Kostenanteil der Strahlminimierung generell unter 1 %.

Die Einschätzung der Straeter-Verfahren untereinander findet sich hier bestätigt. Das klassische Straeter-Verfahren im Vergleich zur gedächtnislosen Version zeigt bei Anwendung auf die Rosenbrock-Funktion keine Unterschiede. Sowohl die Iterationszahlen als auch die Rechenzeiten sind paktisch identisch.
Die Version mit halben Updates pro Iteration ist gegenüber denen mit vollständigem Update unterlegen, wenngleich keine extremen Schwankungen in Abhängigkeit von der benutzten Strahlminimierung auftreten.

Der Einfluß der Strahlminimierung ist hier durch die geringen Kosten für die Funktionsauswertung und Gradienten etwas verschoben. Die Armijo-Schrittweite verliert ihren Vorteil vollständig durch die erhöhten Iterationszahlen. (Nur beim Straeter-Verfahren mit halbem Update zeigen sich die Iterationszahlen erstaunlicherweise invariant gegenüber der Strahlminimierung). Daß die Fletchersche Strahlminimierung schlechter ist als die Methode des goldenen Schnitts, was speziell beim BFGS-Verfahren auffällt, wird an den Abbruchkriterien der beiden Strahlminimierungen liegen, die sich schlecht vergleichen lassen. Deswegen ist zu vermuten, daß das Verfahren des goldenen Schnitts bei der iterierten Rosenbrock-Funktion exaktere Werte liefert.
Bei einem Problem wie der Rosenbrock-Funktion, wo die Kosten für die Berechnung der Funktionswerte gering sind, wird man demnach eine exaktere Strahlminimierung bevorzugen, um die Anzahl der Updates zu reduzieren.

Offensichtlich hängt die optimale Wahl der Strahlminimierung von den Kostenanteilen der Strahlminimierung und des Update ab.
Im Falle des Raffineriemodells wird man beim BFGS-Verfahren mehr Updates in Kauf nehmen, um die Kosten für die Strahlminimierung zu senken. Beim Straeter-Verfahren wird man dagegen darauf bedacht sein, durch eine genauere, aber auch zeitaufwendigere Strahlminimierung die Anzahl der Updates klein zu halten. Im

Sequentiellen waren dementsprechend die Kombinationen BFGS-Update/Armijo-Schrittweite und Straeter-Update/goldener Schnitt zu bevorzugen. Diese Konfiguration läßt sich dann aber nicht ohne weiteres auf die parallelen Verfahren übertragen, da sich die Kostenanteile verschieben können. So wäre es z.B. möglich, daß sich das Straeter-Update so gut parallelisieren läßt, daß sich die Kosten im Straeter-Verfahren auf die Seite der Strahlminimierung verlagern. In einem solchen Fall würde man dann bemüht sein, den Zeitanteil der Strahlminimierung am gesamten Verfahren zu reduzieren. Für die weitere Diskussion dieser Problematik wird auf das nächste Kapitel verwiesen, das sich ausführlich mit der Paralleliserung der Verfahren beschäftigt.

Tabelle 5.5: **Meßergebnisse für die iterierte Rosenbrock-Funktion auf einem Knoten des Intel Paragon XP/S 10. 128 Variablen**

BFGS-Verfahren			
	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	95	38	110
Strahlminimierung Iterationen insgesamt	476	1672	656
Rechenzeit in Sekunden	2.05	0.90	2.15
Rechenzeit pro Iteration in Sekunden	0.022	0.024	0.020
Anteil Rechenzeit Strahlminimierung	14 %	17 %	5 %
Performance-Index	2.29	1.0	2.40

Straeter-Verfahren mit vollständigem Update

	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	9	8	14
Strahlminimierung Iterationen insgesamt	30	352	20
Rechenzeit in Sekunden	18.99	16.91	29.44
Rechenzeit pro Iteration in Sekunden	2.11	2.11	2.10
Anteil Rechenzeit Strahlminimierung	< 1 %	< 1 %	< 1 %
Performance-Index	21.25	18.92	32.71

gedächtnisloses Straeter-Verfahren mit vollständigem Update

	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	9	8	14
Strahlminimierung Iterationen insgesamt	30	352	20
Rechenzeit in Sekunden	19.03	16.91	29.28
Rechenzeit pro Iteration in Sekunden	2.11	2.11	2.09
Anteil Rechenzeit Strahlminimierung	< 1 %	< 1 %	< 1 %
Performance-Index	21.29	18.92	32.76

Straeter-Verfahren mit halbem Update pro Iteration

	Fletcher	gold. Schnitt	Armijo
Quasi-Newton-Iterationen insgesamt	21	22	22
Strahlminimierung Iterationen insgesamt	60	968	42
Rechenzeit in Sekunden	21.92	22.98	22.98
Rechenzeit pro Iteration in Sekunden	1.04	1.05	1.05
Anteil Rechenzeit Strahlminimierung	< 1 %	< 1	< 1 %
Performance-Index	24.53	25.72	25.72

Kapitel 6

Parallelisierung der Optimierungsverfahren

6.1 Parallelisierungskonzepte in der nichtlinearen Optimierung

Es wird davon ausgegangen, daß der Leser mit den grundlegenden Prinzipien der Parallelverarbeitung vertraut ist, weshalb an dieser Stelle keine Einführung zu diesem Thema gegeben wird. Vielmehr sollen die in der Problemstruktur und ihren Lösungsverfahren vorhandenen Parallelismen und ihr Potential für die parallele Problemlösung aufgezeigt werden.
Dabei ist es meist nicht möglich, probleminhärente Parallelismen aufzudecken, da sich das Denken um parallele Lösungsansätze meist auf dem Niveau konkreter Verfahren abspielt. So werden sich auch hier alle dargestellten Ansätze, außer dem der Dekomposition, auf der algorithmischen Ebene der vorgestellten Optimierungsverfahren bewegen. Dies ist wohl auch einer der bestimmenden Gründe, warum die Entwicklung eines parallelen Algorithmus in der Regel von einem sequentiellen Verfahren ausgeht, dessen vorhandene Parallelismen erst einmal ausgenutzt werden. Erst im nächsten Schritt werden Modifikationen am Verfahren vorgenommen, die möglicherweise bei sequentieller Abarbeitung sinnlos sind, aber das Parallelisierungspotential erhöhen. In den meisten Fällen bleibt jedoch trotz dieser Modifikationen die ursprüngliche sequentielle Struktur erhalten, wodurch dann die Möglichkeiten für Parallelisierungen beschränkt bleiben.
Zwei Ansätze, die über die klassischen sequentiellen Strukturen hinausgehen, sogenannte asynchrone Verfahren, sollen kurz vorgestellt werden, obwohl ihr praktischer Nutzen für die Optimierung bislang fraglich ist. Einer der beiden Ansätze liefert aber nützliche Ideen, die auch in diese Arbeit Eingang gefunden haben.
Grundsätzlich sollen hier drei Sorten von Parallelismus unterschieden werden, die hierarchisch übereinander angeordnet sind:

- Dekomposition, d.h. Ansätze, mit denen das Ausgangsproblem in unabhängige Teilprobleme zerlegt wird
- Funktions-Parallelismus
- Daten-Parallelismus

Dabei soll die konkrete Realisierung auf Parallelrechnern zunächst eine untergeordnete Rolle spielen.

Dekomposition

Auch wenn es in der Regel nicht möglich ist, die Modellierungen so vorzunehmen, daß eine Menge von kleineren, unabhängigen Problemen entsteht, die sich trivial parallelisieren lassen, so kommt es doch häufiger vor, daß nicht alle Variablen in direkter Weise durch die Nebenbedingungen bzw. die Zielfunktion gekoppelt sind. Vielmehr treten bestimmte Gruppen von Variablen auf, die vorrangig untereinander verbunden sind und nur in begrenztem Maße über diese Blöcke hinauswirken.
Solche Strukturen sind z.B. in großen Betrieben, die aus mehreren Abteilungen bestehen, oder bei Lagerhaltungsproblemen über lange Zeiträume in natürlicher Weise gegeben. Eine ganze Reihe von angepaßten Lösungsverfahren, die man als Dekompositionsverfahren bezeichnet, sind für so strukturierte Probleme entwickelt worden. Dabei zerfällt das Ausgangsproblem in ein sogenanntes Master-Problem und mehrere kleinere, voneinander unabhängige Teilprobleme. Auch wenn diese Techniken hier nicht behandelt werden, so haben sie doch – gerade für sehr große Probleme – eine tragende Bedeutung und können sogar bei sequentieller Bearbeitung der Teilprobleme zu Geschwindigkeitsvorteilen führen. Für eine ausführlichere Darstellung dieser Ansätze sei z.B. auf [GrTe93] verwiesen.

Funktions-Parallelismus

Unter Funktions-Parallelismus wird hier die parallele Abarbeitung komplexer Teilaufgaben des Optimierungsverfahrens durch verschiedene Prozesse oder Prozessoren verstanden. Für die benutzten Verfahren betrifft das die folgenden Teilaufgaben:

- Richtungsbestimmung, hier als Matrix-Vektor-Produkt $d^k = -H^k \nabla f(x^k)$
- Strahlminimierung
- Berechnung des neuen Iterationspunktes $x^{k+1} = x^k + \alpha^k d^k$
- Update der Quasi-Newton-Matrix H^{k+1}
- Abbruchstest

Ein funktionaler Parallelismus scheint jedoch auf den ersten Blick nicht möglich zu sein, da die aufgezählten Schritte strikt nacheinander erfolgen müssen. Daß diese Form der Parallelisierung, z.T. auch ohne Modifikation der Verfahren, dennoch möglich ist, wird später gezeigt werden.
Aber selbst wenn es gelingen sollte, in jeder Iteration alle oben aufgeführten Teilaufgaben parallel zu lösen, so bleibt der potentielle Speedup[1] durch diese Maßnahme gering. Das liegt zum einen an der geringen Anzahl der Teilprobleme und zum anderen daran, daß ihre Lösung nicht vergleichbar viel Zeit in Anspruch nimmt, wodurch am Ende einer jeden Iteration Wartezeiten für die schnelleren Prozesse entstehen. (Ein Ansatz, der das Problem vermeidet, wird weiter unten vorgestellt.) Außerdem ist der Parallelismus invariant gegenüber der Problemgröße, d.h. es können nicht in natürlicher Weise für größere Probleme auch mehr Prozessoren benutzt werden.

Daten-Parallelismus

Von Daten-Parallelismus spricht man, wenn die gleichen Operationen parallel auf einer Menge von Teildaten ausgeführt werden. Typische Beispiele dafür sind Vektor- und Matrixoperationen, bei denen die einzelnen Komponenten des Vektors bzw. der Matrix unabhängig voneinander berechnet werden können.
Auch bei den Quasi-Newton-Verfahren sind eine Reihe von Operationen, wie z.B. die Berechnung der Gradienten oder die Matrix-Vektor-Operationen, in dieser Weise parallelisierbar. Großer Vorteil der Datenparallelität ist, daß der mögliche Parallelisierungsgrad mit der Problemgröße wächst. Aber auch bei diesem Vorgehen bekommt man einen Synchronisations-Overhead. Zwar ist in den meisten Fällen gewährleistet, daß die einzelnen Prozesse gleich schnell sind, dafür verfügt aber jeder Prozeß nur über einen Teil der Daten. Weil i. allg. aber Abhängigkeiten zwischen den einzelnen Datensegmenten bestehen, müssen die Teilergebnisse zwischen den Prozessen an gewissen Punkten der Berechnung, sogenannten Synchronisationspunkten, ausgetauscht werden.
Die Kommunikations-Overheads verhindern in der Regel die Effizienz solcher Parallelisierungen. Häufig übersteigen die Kommunikations- bzw. Synchronisationskosten für steigende Prozessorzahlen schnell die numerischen Kosten, so daß der Speedup solcher Verfahren nicht nur durch die Anzahl der Datensegmente begrenzt ist.

Sowohl beim Funktions- als auch beim Daten-Parallelismus stellt die Synchronisation der Prozesse den maßgeblich hemmenden Faktor dar. Deshalb gibt es Bemühungen, auf die Synchronisation der Prozesse vollständig oder wenigstens teilweise zu verzichten.

[1] Sofern es nicht explizit anders gesagt wird, bezieht sich der Speedup auf die sequentielle Laufzeit des gleichen Algorithmus.

Asynchroner Funktions-Parallelismus

Fisher und Ritter [FiRi88] entwickelten ein modifiziertes Newton-Verfahren, bei dem die einzelnen Prozesse ohne Synchronisation nach jeder Iteration durchlaufen. Die funktionale Aufteilung des Algorithmus erfolgt in direkter Anlehnung an die Iterationsschritte:

- Berechnung des Gradienten g^k
- Berechnung der Hesse-Matrix G^k
- Berechnung der Cholesky-Faktorisierung von G^k
- Berechnung der Suchrichtung d^k aus der Cholesky-Faktorisierung
- Berechnung des nächsten Iterationspunktes x^{k+1}

Es ist klar, daß z.B. die Cholesky-Faktorisierung eigentlich nicht durchgeführt werden kann, bevor nicht die Hesse-Matrix vollständig berechnet wurde. Aber gerade diese Tatsache wird ignoriert, und jeder Prozeß rechnet mit den Daten, die bis dahin vorliegen, d.h. es vermischen sich alte und neue Daten, z.B. in der Hesse-Matrix. Die Pointe dieses Ansatzes ist also nicht nur, daß man Synchronisationen spart, sondern auch, daß man Teilprobleme parallel löst, die eigentlich sequentiell abgearbeitet werden müßten.
Obwohl dies einen schwerwiegenden Eingriff in den Algorithmus darstellt, konnten Fisher und Ritter zeigen, daß dieses Verfahren unter gewissen Voraussetzungen trotzdem konvergiert. Conforti, Grandinetti und Musmanno [CGM92] zeigten weiterhin, daß sich mit diesem Ansatz auf Rechnern mit gemeinsamem Speicher sogar reale Zeitgewinne erzielen lassen.
Natürlich kann man die Idee noch variieren, indem man partiell doch Synchronisationen durchführt und nur bestimmte Prozesse asynchron durchlaufen läßt. Ideen, die in diese Richtung gehen, werden z.B von Lootsma [Lo91] im Kontext der Quasi-Newton-Verfahren aufgegriffen. Insgesamt sind die Erfahrungen mit Algorithmen dieses Typs für die nichtlineare Optimierung recht gering, und in den wenigen Veröffentlichungen auf diesem Gebiet fehlen meist Messungen auf realen Parallelrechnern.

Asynchroner Daten-Parallelismus

Analog zum asynchronen Funktions-Parallelismus kann man auch beim Daten-Parallelismus auf die Synchronisation verzichten. Dies führt zu einem Ansatz, der erstmalig systematisch von Bertsekas und Tsitsiklis [BeTs89] in dieser Form dargestellt wurde. Sei ein allgemeines Iterationsschema der Form

$$x^{k+1} = T(x^k) \qquad x \in \mathbb{R}^n \tag{6.1}$$

vorgegeben und angenommen, daß jede Komponente x_i von einem eigenen Prozessor i berechnet wird. Wenn x_i^{k+1}, $i = 1, \ldots, n$, von allen Komponenten x_j^k, $j = 1, \ldots, n$, abhängt, dann kann der Prozessor i die Komponente x_i^{k+1} erst berechnen, wenn der gesamte Vektor x^k vorliegt. Das bedeutet, daß am Ende einer jeden Iteration alle Prozessoren warten müssen, bis alle anderen fertig sind und ihre Teilergebnisse mitgeteilt haben. Gerade diese Synchronisation verhindert in der Praxis eine perfekte Skalierbarkeit solcher Algorithmen.
Deshalb scheint ein immenses Potential für massiv parallele Algorithmen darin zu liegen, auf die Synchronisation vollständig oder teilweise zu verzichten. Dabei wartet kein Prozessor, bis er die anderen Teilergebnisse erhalten hat, sondern alle rechnen mit den Informationen weiter, die bis dahin vorliegen. Beim total asynchronen Ansatz setzt man nur voraus, daß jeder Prozessor nach endlicher Zeit die Teilergebnisse der anderen Prozessoren erhält, und beim partiell asynchronen Ansatz wird eine Zeitgrenze vorgegeben, innerhalb derer die anderen Teilergebnisse vorliegen müssen. Bertsekas und Tsitsiklis untersuchen auch die Auswirkungen dieser Idee auf die Konvergenz gradientenbasierter Verfahren zur Optimierung. Beim total aynchronen Ansatz läßt sich die Konvergenz allerdings nur unter sehr restriktiven Annahmen zeigen, die in der Praxis sehr selten vorliegen. Für den partiell asynchronen Fall hingegen erhält man eine Reihe von Konvergenzresultaten, die eine praktische Anwendung solcher Methoden durchaus möglich erscheinen lassen. Dennoch liegen bisher keinerlei Ergebnisse vor, die einen gewinnbringenden Einsatz partiell asynchroner Verfahren in der nichtlinearen Optimierung dokumentieren. Ob das probleminhärent oder verfahrensbedingt ist, bleibt unklar.

Für die hier benutzten Verfahren wurde meist der Daten-Parallelismus bevorzugt, da nur dieser Ansatz zu einem mit der Problemgröße skalierbaren Parallelismus führt. Im zweiten Schritt wird dann der Daten-Parallelismus mit einer funktionalen Parallelisierung kombiniert. Asynchrone Konzepte kommen in der oben beschriebenen reinen Form nicht vor, wenngleich das starr sequentielle Schema auf der funktionalen Ebene aufgeweicht wurde.

6.2 Der Intel Paragon XP/S 10

Da die Entwurfsentscheindungen für parallele Algorithmen von der Architektur des zugrundeliegenden Parallelrechners abhängen, werden im folgenden die diesbezüglich relevanten Aspekte des Intel Paragon XP/S 10 beschrieben. Für eine ausführliche Beschreibung von Hardware und Software-Umgebung sei z.B. auf [Be94] verwiesen.

Der Intel Paragon XP/S 10 ist ein Parallelrechner mit verteiltem Speicher, dessen Knoten in einem zweidimensionalen Gitter angeordnet sind. Die momentane Konfiguration im Forschungszentrum Jülich besteht aus 140 Knoten, auf denen das Release 1.2 des Paragon OSF/1 Betriebssystems installiert ist.

Jeder Knoten verfügt über einen Speicher von 32 MB und zwei RISC-Prozessoren des Typs Intel i860 XP, von denen der eine ausschließlich für Kommunikationszwecke vorgesehen ist. Nominal hat der Intel i860 XP bei 50 Mhz eine Peak-Performance von 75 MFLOPS, von denen real 10 bis maximal 40 MFLOPS bei opitimierten Codes erreicht werden.
Für das Verbindungsnetzwerk ist jedem Knoten ein sogenannter Mesh Routing Chip (iMRC) zugeordnet, der Nachrichten unabhängig vom zugehörigen Knoten routen kann. Dadurch können Nachrichten über längere Distanzen verschickt werden, ohne daß die Daten im Hauptspeicher der dazwischenliegenden Knoten gepuffert werden müssen. Zusammen mit der Technik des Wormhole-Routing [NM93] wird so die Bandbreite zwischen zwei Knoten, zumindest für längere Nachrichten, fast unabhängig von ihrem Abstand. Die Bandbreite zwischen zwei Knoten beträgt in jede Richtung 75 MB/s, die durch die verhältnismäßig hohen Startup-Zeiten aber bei kurzen Nachrichten nicht annähernd erreicht wird.

Die Programmierung des Intel Paragon XP/S 10 erfolgte im Message-Passing-Programmiermodell in der Programmiersprache C zusammen mit der NX Message-Passing Bibliothek. Zu Beginn wird auf die gewünschte Anzahl von Knoten jeweils ein Programm geladen. Üblicherweise sind diese Programme identisch, so daß man eine SPMD-Struktur (SPMD=Single Program Multiple Data) erhält. Es sind aber auch alternative Strukturen wie z.B. Master-Worker möglich.
Das Mapping der Programme auf die physikalischen Knoten geschieht automatisch, wenngleich der Benutzer dies im Einzelfall auch selber steuern kann. Durch die speziellen Eigenschaften des Verbindungsnetzwerks kann der Programmierer aber praktisch davon ausgehen, daß seine Knoten vollständig vernetzt sind, so daß das Mapping-Problem in der Regel in den Hintergrund tritt. Dieser topologieunabhängige Programmierstil ermöglicht eine komfortablere Programmierung und eine einfache Portierbarkeit solcher Programme.
Für die Kommunikation zwischen den Knoten werden drei Kommunikationsarten angeboten, synchrone, asynchrone und interrupt-gesteuerte. Zusätzlich gibt es eine Reihe von globalen Operationen. Da die Begriffe synchron und asynchron in diesem Zusammenhang anders als üblich benutzt werden, sollen sie kurz erklärt werden.

Bei einer synchronen Sendeoperation wird der Message-Passing-Coprozessor aktiviert, der dann die Daten aus dem vorgegebenen Speicherbereich liest und verschickt. Der Hauptprozessor wartet so lange, bis die Nachricht komplett verschickt ist. Dies bedeutet allerdings nicht, daß die Nachricht schon vom Zielknoten empfangen wurde. Bei einer asynchronen Sendeoperation dagegen wartet der Hauptprozessor nicht, sondern rechnet weiter, während im Hintergrund der Coprozessor die Daten aus dem Speicher liest und versendet.
Der Empfang von Daten kann analog ebenfalls synchron oder asynchron erfolgen, unabhängig davon, in welchem Modus sie verschickt wurden. Bei der synchronen Variante wartet der Hauptprozessor wieder, bis die Daten vollständig empfangen und im gewünschten Speicherbereich abgelegt wurden. Im asynchronen Fall wird der Coprozessor nur angewiesen, in welchen Speicherbereich er die Daten schreiben

soll, und der Hauptprozessor rechnet ohne zu warten weiter.
Sowohl beim asynchronen Senden als auch beim asynchronen Empfangen kann die Beendigung der Operation vom Hauptprozessor erfragt werden oder alternativ durch Auslösung eines Interrupts signalisiert werden, was dann zur interrupt-gesteuerten Kommunikation führt.

Zusätzlich zu den einfachen Send- und Receive-Anweisungen werden eine Reihe von globalen Kommunikationsoperationen wie globale Summation, total Exchange, Broadcast und Synchronisation zur Verfügung gestellt. Leider können solche Operationen (außer das Broadcast) nicht auf Teilmengen der Knoten angewendet werden, so daß im Falle heterogener Prozeßstrukturen die Programmierung dieser Operationen selber durchgeführt werden muß.

6.3 Parallelisierung der Quasi-Newton-Verfahren

Bevor man mit der Parallelisierung beginnt, sollte geklärt sein, welche Teilprobleme den bestimmenden Anteil an den Kosten der Verfahren haben. Die Anteile von Strahlminimierung und Update an der gesamten Rechenzeit sind der Tabelle 6.1 zu entnehmen, wo für das BFGS-Verfahren und das gedächtnislose Straeter-Verfahren in Kombination mit den drei Techniken zur Strahlminimierung die prozentualen Rechenzeitanteile bei Anwendung auf das (iterierte) Raffineriemodell dargestellt sind. Dabei wurde die Richtungsbestimmung $-H^k \nabla f(x^k)$ mit zum Update gerechnet, weil sie sich, wie später zu sehen ist, in die Berechnung des Update integrieren läßt. Offenbar werden die Kosten der Verfahren ausschließlich durch das Update und die Strahlminimierung bestimmt, d.h. daß z.B. Abbruchtests und Verwaltungsoperationen nicht ins Gewicht fallen und sich die Parallelisierung voll auf die Strahlminimierung und das Update konzentieren kann.

Je nach Kombination der Verfahren lassen sich die Zeitanteile von Update und Strahlminimierung erheblich variieren. Beim BFGS-Verfahren liegt der Zeitanteil der Strahlminimierung selbst bei der Armijo-Schrittweite noch über 10 %, so daß man hier darauf angewiesen ist, sowohl das Update als auch die Strahlminimierung zu parallelisieren, um befriedigende Speedups zu erzielen. Falls sich z.B. die Strahlminimierung als schlecht parallelisierbar erweisen sollte, so bestünde beim Straeter-Verfahren die Möglichkeit, den Anteil der Strahlminimierung an der Rechenzeit durch entsprechende Auswahl des Verfahrens unter 2 % zu senken und so das Parallelisierungspotential zu erhöhen.
Diese Vorgehensweise birgt jedoch die Gefahr, daß man ein Verfahren auswählt, das zwar gut parallelisierbar, aber im Vergleich zu anderen Verfahren trotzdem langsamer ist. Deshalb ist es notwendig, nach der Parallelisierung nicht nur Speedups, sondern auch die realen Rechenzeiten miteinander zu vergleichen, da die Parallelisierung nicht zum Selbstzweck werden soll.
Bei den sequentiellen Testläufen war das BFGS-Verfahren dem Straeter-Verfahren

Tabelle 6.1: **Zeitanteile von Strahlminimierung und Update in den Quasi-Newton-Verfahren bei Anwendung auf das iterierte Raffineriemodell**

BFGS-Verfahren	Fletcher			gold. Schnitt			Armijo		
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Update gesamt	13.8	12.3	12.4	31.2	45.6	55.1	78.1	84.9	88.3
Strahlminimierung gesamt	86.1	87.6	87.6	68.4	54.1	44.6	21.1	14.7	11.3

Straeter-Verfahren	Fletcher			gold. Schnitt			Armijo		
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Update gesamt	55.4	80.2	80.6	88.0	96.5	98.4	96.6	98.9	99.6
Strahlminimierung gesamt	44.4	19.6	19.4	10.8	3.2	1.5	2.9	0.8	0.3

Angaben in Prozent, $R^{(N)} = N$-fach iteriertes Raffineriemodell

bzgl. der Rechenzeit überlegen, und so könnte es sein, daß das BFGS-Verfahren auch im Parallelen schneller bleibt, obwohl sich mit dem Straeter-Verfahren höhere Speedups erzielen lassen.

Für die Untersuchung der Parallelisierbarkeit werden zunächst Strahlminimierung und Update getrennt betrachtet, und erst im nächsten Schritt wird auf ihr Zusammenspiel eingegangen.

6.3.1 Strahlminimierung

Alle hier beschriebenen Ansätze zur Strahlminimierung bestehen im wesentlichen aus der sukzessiven Berechnung von Funktionswerten bzw. von Gradienten. Die Parallelisierung dieser Aufgabe kann in zwei Stoßrichtungen erfolgen. Entweder parallelisiert man die Funktions- bzw. die Gradientenauswertung, oder man berechnet parallel verschiedene Funktionswerte. Natürlich ist auch eine Kombination beider Ansätze denkbar.
Die parallele Auswertung verschiedener Funktionswerte bzw. Gradienten erscheint jedoch problematisch, da bei den oben beschriebenen Verfahren zur Strahlminimierung in jeder Iteration nur ein Funktionswert ausgerechnet werden muß. So wäre es im Prinzip nur möglich, spekulativ Funktionswerte auszurechnen, von denen man noch nicht weiß, ob man sie überhaupt braucht.
Bei der Armijo-Schrittweite ist ein solches Vorgehen direkt klar. Wenn p Prozessoren zur Verfügung stehen, rechnet man in jedem Schritt parallel eine Serie von p Funktionswerten aus und wählt darunter den mit dem kleinsten Funktionswert aus, der die Armijo-Bedingung erfüllt. Wenn keiner der Punkte die Bedingung erfüllt, berechnet man parallel eine neue Serie von p Punkten. Ein solcher Ansatz wird von Nash und Sofer [NaSo89] beschrieben und auch in ihrer Implementation eines

approximierten Newton-Verfahrens [NaSo92b] benutzt. In diesem Kontext betonen sie den Nutzen dieses Vorgehens insbesondere für nicht-konvexe Probleme, da durch die zusätzlich berechneten Punkte möglicherweise günstigere lokale Minima gefunden werden können.
Bei konvexen Problemen in Verbindung mit Newton-ähnlichen Verfahren wird ein Zeitgewinn allerdings höchstens dann zu erwarten sein, wenn die Kosten für die Funktionsauswertungen im Vergleich zu den Kommunikationszeiten hoch sind. Das liegt daran, daß die Schrittweiten von Newton- und Quasi-Newton-Verfahren gegen 1 tendieren und somit in vielen Fällen direkt einer der ersten Punkte die Armijo-Bedingung erfüllt. In diesen Fällen wären die restlichen der p Punkte umsonst berechnet worden, und die anschließende Ermittlung des Minimums der Funktionswerte verursacht nur einen zusätzlichen Kommunikations-Overhead. Dementsprechend ist der Speedup, der sich mit dieser Methode erzielen läßt, durch die im sequentiellen Verfahren benötigten Funktionsauswertungen beschränkt. Im günstigsten Fall kann man also erwarten, daß man mit der parallelen Berechnung *einer* Punktserie auskommt, was sich dann nicht mehr steigern läßt.

Für das Verfahren des goldenen Schnitts sieht die Parallelisierbarkeit schlechter aus, da hier nicht die Funktionswerte einer vorher feststehenden Folge von Punkten berechnet werden, sondern die Punktfolge von den Entscheidungen, die im Algorithmus getroffen werden, abhängt. Deshalb müßten hier noch mehr Funktionswerte berechnet werden, die schließlich nicht benötigt werden. Wenn man die Funktionswerte der nächsten p Iterationen im voraus berechnen wollte, so wären dies $2^p - 1$ Funktionswerte, die man berechnen müßte.
Noch schwerwiegender sind die Probleme wohl bei der Strahlminimierung nach Fletcher, da hier die nächsten neu zu berechnenden Punkte nicht einmal auf einem festen Gitter liegen.

Deshalb kommt als alternativer Ansatz noch die parallele Berechnung der einzelnen Funktionswerte bzw. Gradienten in Frage. Ob sich eine solche Parallelisierung lohnt, hängt natürlich wieder davon ab, wie teuer diese Berechnungen sind. Die Parallelisierung der Funktionsauswertung ist ungünstig, da der Anwender, der die Optimierungssoftware benutzt, die Parallelisierung selber vornehmen müßte. Außerdem lassen sich nicht alle Zielfunktionen gleichermaßen gut parallel auswerten. Im speziellen Kontext der Penalty-Verfahren ist eine automatische Parallelisierung allerdings dennoch möglich, da die Zielfunktion aus einer Summe von Straftermen besteht, die unabhängig voneinander berechnet werden können. Aber auch dieses Vorgehen gewinnt erst an Bedeutung, wenn man eine hinreichend große Anzahl von Nebenbedingungen hat, da die Berechnung der einzelnen Strafterme relativ billig ist. Deshalb erscheint dieser Ansatz beim Raffineriemodell nicht als sonderlich lohnend. Bei der Gradientenberechnung sieht das hingegen anders aus, da diese im Vergleich zur Funktionsauswertung um ein Vielfaches zeitaufwendiger ist. Theoretisch benötigt eine Gradientenberechnung beim Raffineriemodell $O(n^3)$ Operationen und die Funktionswerte nur $O(n^2)$, wobei n die Anzahl der Variablen ist. Wie Tabelle 6.2 zu entnehmen ist, dominiert die Gradientenberechnung bei Verwendung der Strahl-

Tabelle 6.2: Zeitanteile der Strahlminimierung im Quasi-Newton-Verfahren bei Anwendung auf das iterierte Raffineriemodell

BFGS-Verfahren	Fletcher			gold. Schnitt			Armijo		
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Strahlminimierung gesamt	86	88	87	70	55	46	21	14	11
Gradientenberechnung	79	84	85	0	0	0	0	0	0
Funktionsauswertung	6	3	2	69	55	46	21	14	11

Straeter-Verfahren	Fletcher			gold. Schnitt			Armijo		
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Strahlminimierung gesamt	39	31	16	11	3	2	2	1	< 1
Gradientenberechnung	36	30	15	0	0	0	0	0	0
Funktionsauswertung	2	1	< 1	11	3	2	2	1	< 1

Angaben in Prozent, $R^{(N)} = N$-fach iteriertes Raffineriemodell

minimierung nach Fletcher die Kosten des gesamten BFGS-Verfahrens.

Naheliegenderweise berechnet man parallel mehrere Komponenten des Gradienten, da die parallele Berechnung einer einzelnen Komponente die gleichen Probleme aufwirft wie die Parallelisierung der Funktionsauswertung. In der Strahlminimierung nach Fletcher werden jedoch nicht die Gradienten, sondern Richtungsableitungen benötigt, die sich als Skalarprodukt des Gradienten mit der jeweiligen Richtung ergeben. Wenn also ein Punkt x und eine Richtung d vorgegeben sind mit $x, d \in \mathbb{R}^n$, dann können n Prozessoren parallel die Produkte $d_i(\nabla f(x))_i$ berechnen. Das gewünschte Ergebnis entsteht dann durch eine globale Summation der Teilergebnisse. Da man den Gradienten eigentlich nur benutzt, um die Richtungsableitung zu berechnen, könnte man statt der exakten Ableitung auch eine Approximation durch Differenzenquotienten verwenden, die nur zwei Funktionswerte benötigt. Dadurch ließe sich möglicherweise die Geschwindigkeit der Strahlminimierung nach Fletcher auch im Sequentiellen erheblich steigern. Experimente, die Ableitung mit zentralen Differenzen zu approximieren, d.h. $f'(x) \approx \frac{f(x+hd)-f(x-hd)}{2h}$ mit einem geeigneten h und der Suchrichtung d, hatten jedoch keinen Erfolg. Die Strahlminimierung arbeitete mit der Approximation nicht mehr zuverlässig, was vermutlich daran liegt, daß durch die Strafterme die Steigungen sehr groß werden können. In solchen Fällen muß man das h extrem klein wählen, wodurch wieder Probleme in der Rechnerarithmetik auftreten. Doch selbst wenn dieser Ansatz zuverlässig funktionieren würde, so hätte man zumindest wie beim Verfahren des goldenen Schnitts das Problem, daß es nicht mehr brauchbar parallelisierbar wäre.

Implementiert wurde hier sowohl die parallele Variante der Armijo-Schrittweite als auch die Fletchersche Strahlminimierung mit paralleler Gradientenauswertung.

Parallele Armijo-Schrittweite

Seien nun n Prozessoren bzw. Knoten vorgegeben, auf denen die Schrittweite parallel bestimmt werden soll. Zu Beginn verfügen alle Prozessoren über die gleichen Informationen, d.h. über den aktuellen Iterationspunkt und die Suchrichtung.
Dann berechnet jeder Prozessor entsprechend dem Armijo-Prinzip und seiner Knotennummer einen Funktionswert auf dem Strahl und prüft, ob diese Schrittweite die Armijo-Bedingung erfüllt. Ziel ist es jetzt, die größte Schrittweite zu bestimmen, die die Armijo-Bedingung erfüllt. Zwar wird beim Ansatz von Nash und Sofer [NaSo92a] nicht die größte Schrittweite ausgewählt, sondern die mit dem kleinsten Funktionswert, aber die Wahl der größten Schrittweite hat sich bei den untersuchten Verfahren und Problemen als günstiger erwiesen.
Die Kommunikation zwischen den Knoten erfolgt entlang der Äste eines Binärbaums. In der ersten Phase, die von den Blättern zur Wurzel des Baums läuft, wird die größte Schrittweite bestimmt, die die Armijo-Bedingung erfüllt. Dazu vergleicht jeder Knoten seine eigenen Daten mit denen, die von den Töchtern zugeschickt werden und sendet seinerseits gemäß dem Suchziel die optimale Schrittweite zu seiner Mutter.
In der zweiten Phase wird das Ergebnis des globalen Vergleichs von der Wurzel wieder über den ganzen Baum verteilt. Wenn die so erhaltene Schrittweite die Armijo-Bedingung erfüllt, bricht das Verfahren ab. Ansonsten wird eine neue Serie von p Punkten berechnet und das gleiche Schema beginnt von vorne.
Der Kommunikationsaufwand in jeder Iteration ist theoretisch von der Ordnung $O(\log n)$. Real wird man diesen Wert nur erreichen, wenn man den Binärbaum optimal auf das zweidimensionale Gitter des Intel Paragon XP/S 10 abbildet, so daß keine Nachrichtenkollisionen auftreten. In den vorgenommenen Implementationen wurde auf solche Optimierungen verzichtet, da sie die Programmierung erheblich verkomplizieren.

Parallele Strahlminimierung nach Fletcher

Die Ausgangssituation ist wieder die gleiche wie bei der Armijo-Schrittweite, d.h. jeder der p Knoten verfügt über den aktuellen Iterationspunkt x und die Suchrichtung d. Der Algorithmus läuft nun wie im Sequentiellen auf allen Prozessoren redundant ab bis zu dem Punkt, wo die Ableitung berechnet werden soll. Hier berechnet jeder Knoten gemäß seiner Knotennummer l nur einen Teil der Komponenten des Gradienten und bildet damit das partielle Skalarprodukt

$$s_l = \sum_{i=j_l}^{j_{l+1}-1} d_i \left(\nabla f(x)\right)_i$$

Die Richtungsableitung ergibt sich dann durch globale Summation der partiellen Skalarprodukte, die mit einer Bibliotheksroutine durchgeführt werden kann. Diese Routine arbeitet ebenfalls mit baumartigen Kommunikationsstrukturen, so daß der Aufwand für die globale Summation wieder logarithmisch von der Knotenzahl abhängt.

Sicherlich gibt es noch eine ganze Reihe von alternativen Ansätzen für die parallele

Strahlminimierung (siehe z.B. [AvWi66, AvWi, BeWi70, KaMi68, LoRa88]), von denen man im einzelnen prüfen müßte, ob sie sich gewinnbringend einsetzen lassen. Der zeitliche Rahmen der Diplomarbeit ließ jedoch eine intensivere Auseinandersetzung mit diesem Problem nicht zu.

6.3.2 Update und Richtungsbestimmung

Bevor an dieser Stelle die Parallelisierung des Update und der Richtungsbestimmung diskutiert werden kann, soll zunächst geklärt werden, wie die Berechnung des Update organisiert wurde und wie sich die Richtungsbestimmung in die Berechnung des Update integrieren läßt. Wegen der leichteren Verständlichkeit soll das Vorgehen am Beispiel des Broyden-Update erläutert werden.

In der modifizierten Schreibweise ist die Update-Formel gegeben durch

$$H^{k+1} = H^k + \kappa^k r^k (r^k)^T \tag{6.2}$$

mit

$$r^k = s^k - H^k y^k \quad \text{und} \quad \kappa^k = ((r^k)^T y^k)^{-1}. \tag{6.3}$$

Die neue Suchrichtung ergibt sich als

$$\begin{aligned} d^{k+1} &= -H^{k+1}\nabla f(x^{k+1}) \\ &= -(H^k + \kappa^k r^k (r^k)^T)\nabla f(x^{k+1}) \\ &= -H^k \nabla f(x^{k+1}) - \kappa^k r^k (r^k)^T \nabla f(x^{k+1}) \end{aligned}$$

und

$$\begin{aligned} r^k &= s^k - H^k y^k \\ &= s^k - H^k(\nabla f(x^{k+1}) - \nabla f(x^k)) \\ &= s^k - H^k \nabla f(x^{k+1}) + H^k \nabla f(x^k) \\ &= s^k - H^k \nabla f(x^{k+1}) - d^k. \end{aligned}$$

Offensichtlich reicht es aus, in jeder Iteration nur das Produkt $H^k \nabla f(x^{k+1})$ zu berechnen, wenn man die Suchrichtung d^k aus der letzten Iteration für das Update benutzt. Dann kann die Berechnung von Update und Suchrichtung wie folgt kombiniert werden:

$$\begin{aligned} t &= H^k \nabla f(x^{k+1}) && (6.4)\\ r^k &= s^k - t - d^k && (6.5)\\ \kappa^k &= ((r^k)^T y^k)^{-1} && (6.6)\\ H^{k+1} &= H^k + \kappa^k r^k (r^k)^T && (6.7)\\ d^{k+1} &= -t - \kappa^k r^k (r^k)^T \nabla f(x^{k+1}) && (6.8) \end{aligned}$$

Das Produkt in (6.8) besteht bei einer Auswertung von rechts nach links nur aus einem Skalarprodukt und der Multiplikation eines Vektors mit einem Skalar, im Gegensatz zu einem Matrix-Vektor-Produkt bei Benutzung der ursprünglichen Formel $d^k = -H^{k+1}\nabla f(x^{k+1})$. Außerdem sind die Berechnung (6.8) der neuen Suchrichtung und das Rang-1-Update (6.7) unabhängig voneinander, so daß sie auch in umgekehrter Reihenfolge erfolgen können, wodurch die neue Suchrichtung vor der neuen Quasi-Newton-Matrix zur Verfügung steht. Dieser Aspekt ist besonders im Hinblick auf die Parallelisierung des Verfahrens interessant.
Analoges Vorgehen liefert das Berechnungsschema für die BFGS-Formel.

$$
\begin{aligned}
t &= H^k \nabla f(x^{k+1}) && (6.9)\\
z &= s^k - t - d^k && (6.10)\\
\gamma &= (s^k)^T y^k \qquad \delta = z^T y^k && (6.11)\\
z &= \frac{z}{\gamma} - \frac{\delta}{2\gamma^2} s^k && (6.12)\\
H^{k+1} &= H^k z (s^k)^T + s^k z^T && (6.13)\\
d^{k+1} &= -t - z(s^k)^T \nabla f(x^{k+1}) - s^k z^T \nabla f(x^{k+1}) && (6.14)
\end{aligned}
$$

Auch hier läßt sich die Berechnung der Suchrichtung dem Rang-2-Update vorziehen, allerdings werden hier zwei Skalarprodukte benötigt statt dem einen bei der Broyden-Formel. Das Straeter-Update läßt sich grundsätzlich wie n aufeinanderfolgende Broyden-Updates organisieren. Der einzige Unterschied ist, daß die Richtungen, entlang derer das Update durchgeführt wird, keine sukzessiven Suchrichtungen sind und somit die Berechnung der Suchrichung nicht in der gleichen eleganten Weise mit dem Update kombiniert werden kann. Vielmehr ist für jedes partielle Update ein Matrix-Vektor-Produkt und eine Gradientenauswertung notwendig, und die Suchrichtung ergibt sich ebenfalls als Matrix-Vektor-Produkt. Selbst wenn analog wie bei den anderen Updates eine geschickte Reorganisation der Berechnung möglich wäre, um das Matrix-Vektor-Produkt zur Richtungsbestimmung zu vermeiden, so würden dann doch n Skalarprodukte gebraucht, was dem gleichen numerischen Aufwand entspricht.

Insgesamt werden die folgenden Operationen benötigt:

- Matrix-Vektor-Produkte
- Skalarprodukte
- Rang-1-Updates, d.h. Operationen der Form
  $H^{k+1} = H^k + uu^T$
- Gradientenberechnung

Bevor man beginnt, diese Operationen zu parallelisieren, ist es wieder sinnvoll zu untersuchen, welchen Zeitanteil sie am gesamten Update haben. Während Tabelle

Tabelle 6.3: Zeitanteile der Operationen im Update bei Anwendung auf das iterierte Raffineriemodell

BFGS-Verfahren			
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Matrix-Vektor-Produkt	0.7	0.9	1.0
Rang-2-Update	1.7	1.8	1.9
Skalarprodukte	< 0.1	< 0.1	< 0.1
Gradientenberechnung	96.3	96.8	96.7

Straeter-Verfahren			
	$R^{(1)}$	$R^{(2)}$	$R^{(3)}$
Matrix-Vektor-Produkt	0.7	0.8	0.9
Rang-n-Update	1.3	1.5	1.6
Skalarprodukte	< 0.1	< 0.1	< 0.1
Gradientenberechnung	97.7	97.6	97.4

Angaben in Prozent, $R^{(N)} = N$-fach iteriertes Raffineriemodell

6.1 den Anteil des Update am gesamten Verfahren darstellt, kann man Tabelle 6.3 entnehmen, wie die Kosten innerhalb des Update verteilt sind. Dabei sei mit Update jetzt die gesamte Berechnung der Quasi-Newton-Matrix und der Suchrichtungen gemeint. Da die Wahl der Strahlminimierung keinen Einfluß auf die Kostenverteilung innerhalb des Update hat, sind hier nur die prozentualen Zeitanteile in Abhängigkeit von der Update-Technik und der Problemgröße aufgetragen.

Die Meßergebnisse zeigen deutlich, daß auch im Update die Gradientenberechnung die Kosten dominiert. Deshalb ist das Potential für den Zeitgewinn durch die Parallelisierung der Gradientenberechnung am größten. Die Skalarprodukte haben nur einen geringen Anteil an den Kosten, und eine Parallelisierung auf Rechnern mit verteiltem Speicher würde wegen des Kommunikations-Overheads erst bei extrem großen Vektoren lohnen.
Der Anteil der Matrix-Operationen an der Rechenzeit liegt schon deutlich höher als der der Skalarprodukte. Obwohl sich Matrix-Operationen gut parallelisieren lassen, müßte man ausprobieren, ob dies bei dem recht kleinen Raffineriemodell schon einen Zeitvorteil gegenüber der sequentiellen Variante bringt. Unabhängig davon sollen die Programme aber auch für größere Probleme ausgelegt werden, bei denen eine Parallelisierung auf jeden Fall lohnend ist.

Ziel ist es also jetzt, sowohl die Gradientenberechnung als auch die Matrix-Operationen zu parallelisieren. Für beide Update-Techniken wurde dazu zunächst ein rein datenparalleler Ansatz mit einer SPMD-Struktur gewählt, der durch die NX Message-Passing-Bibliothek bevorzugt unterstützt wird.
Da der Intel Paragon XP/S 10 ein Rechner mit verteiltem Speicher ist, muß zunächst eine geeignete Datenaufteilung festgelegt werden, die eine Parallelisierung mit mög-

lichst geringem Kommunikationsaufwand ermöglicht.
Auftretende Datenstrukturen sind Matrizen, die in gleich großen Spaltenblöcken auf die Knoten verteilt werden, und Vektoren, die jeweils, sofern global benötigt, vollständig auf jedem Knoten gespeichert werden. Es macht grundsätzlich keinen Unterschied, ob die Quasi-Newton-Matrix in Zeilenblöcke oder Spaltenblöcke aufgeteilt wird, da diese Matrizen symmetrisch sind und man die Spalten genausogut als Zeilen interpretieren kann.

Um die Indizierungen zu vereinfachen, sei nun angenommen, daß bei der Problemgröße n ebenfalls n Prozessoren bzw. Knoten vorhanden sind. Für kleinere Prozessorzahlen werden die Daten analog gleichmäßig auf die Prozessoren aufgeteilt.

Zu Beginn eines Update verfügt jeder Prozessor über den aktuellen Iterationspunkt x^{k+1}, die letzte Suchrichtung d^k (beim Straeter-Update nicht notwendig) und entsprechend seiner Prozessornummer eine Spalte bzw. Zeile der Quasi-Newton-Matrix H^k. Berechnet werden sollen die Quasi-Newton-Matrix H^{k+1}, die neue Suchrichtung d^{k+1} und der Gradient $\nabla f(x^{k+1})$. Dabei sollen die Vektoren d^{k+1} und $\nabla f(x^{k+1})$ nach der Berechnung auf allen Knoten vorliegen.
Auf den Gradienten könnte man ggf. global verzichten, wenn man ein anderes Abbruchkriterium und die Strahlminimierung des goldenen Schnitts benutzt, die wirklich vollständig ableitungsfrei sind.
Da sich die Organisation von BFGS- und Straeter-Update unterscheidet, sollen sie getrennt behandelt werden.

Parallelisierung des BFGS-Update

Wie dem Berechnungsschema 6.9 bis 6.14 zu entnehmen ist, müssen zuerst der Gradient $\nabla f(x^{k+1})$ und das Produkt $t = H^k \nabla f(x^{k+1})$ berechnet werden. Dazu berechnet Prozessor p die p-te Komponente des Gradienten. Das Matrix-Vektor-Produkt kann nun mit zwei verschiedenen Methoden berechnet werden. Entweder berechnet Prozessor p die Komponente t_p des Produktvektors oder einen Summanden von jeder Komponente von t.

Im ersten Fall muß zunächst der Gradientenvektor auf jeden Knoten verteilt werden. Dies kann mit der Bibliotheksfunktion *gcolx()* erfolgen, die es erlaubt, einen Vektor global aus verteilten Komponenten zusammenzusetzen. Erst dann ist es möglich, die Komponente t_p durch ein Skalarprodukt zu berechnen. Der Vektor t wird dann analog zum Gradienten global zusammengesetzt.

Im zweiten Fall berechnet Prozessor p einen Summanden von jeder Komponente von t bzw. den Vektor

$$t^{(p)} = (\nabla f(x^{k+1})_p \begin{pmatrix} h_{1p} \\ \vdots \\ h_{np} \end{pmatrix}$$

Der Vektor t entsteht dann aus der Summe der Vektoren $t^{(p)}$, d.h. $t = \sum_{i=1}^{n} t^{(p)}$.

Diese globale Summe kann ebenfalls mit einer Bibliotheksfunktion (*gdsum()*) gebildet werden.

Die restlichen Operationen im Update können, wenn der Gradient und der Vektor t vollständig vorliegen, lokal berechnet werden. Das gilt insbesondere für das Rang-2-Update, das sogar parallel ohne weitere Kommunikation berechnet werden kann.

Es scheint so, als würde man durch diese Technik eine globale Synchronisation vermeiden können. Leider muß der Gradient aber spätestens am Ende des Update auf allen Knoten vorliegen, weshalb noch eine vom Matrix-Vektor-Produkt unabhängige globale Kommunikation nötig wird.
Beide Funktionen *gdsum()* und *gcolx()* benötigen für Vektoren gleicher Länge etwa die gleiche Zeit, die etwa linear von der Vektorlänge und logarithmisch von der Anzahl der Knoten abhängt. Der Vorteil der zweiten Variante liegt darin, daß der Gradient nicht vor dem Matrix-Vektor-Produkt zusammengesetzt werden muß, sondern dies gleichzeitig geschehen kann. Im Prinzip handelt es sich um ein perfektes Beispiel, wo Kommunikation und Berechnung parallel laufen können. Das Zusammensetzen des Gradienten wird vom Message-Passing-Coprozessor übernommen, während der Hauptprozessor den Teil des Matrix-Vektor-Produkts berechnet. Das Problem ist nur, daß das Release 1.2 des Message-Passing-Systems keine asynchronen globalen Operationen zur Verfügung stellt, weshalb man diese selber programmieren müßte. Dann ist aber zu befürchten, daß der Vorteil, den man durch die Überlappung gewinnt, durch die schlechte Implementation der globalen Kommunikation wieder kompensiert wird. Die optimale Implementation solcher globalen Operationen ist jedoch überaus aufwendig (siehe z.B. [Ge91]) und wurde deshalb auch nicht versucht.
Eine andere denkbare Möglichkeit, das Fehlen asynchroner globaler Operationen zu umgehen, wäre das Multitasking. Das Betriebssystem OSF/1, das auf jedem Knoten des Intel Paragon XP/S 10 läuft, bietet dazu das Konzept sogenannter *pthreads* an, die es erlauben, von einem Prozeß aus weitere "leichtgewichtige" Prozesse zu starten. Die ersten Tests mit Multitasking auf der Basis von *pthreads* waren jedoch nicht erfolgversprechend, so daß dieser Ansatz nicht weiter verfolgt und auf eine Überlappung von Kommunikation und Berechnung verzichtet wurde.

Parallelisierung des Straeter-Update

Die Organisation des Straeter-Update ist der des BFGS-Update naturgemäß sehr ähnlich. Der Hauptunterschied ist jedoch, daß die n Gradienten $\nabla f(x^{k+1} + \epsilon e_j)$, die für die partiellen Updates benötigt werden, nicht global vorliegen müssen. Dadurch kann pro partiellem Update eine globale Vektor-Kombination durch eine globale Summe eines Skalars ersetzt werden.

Zu Beginn des Straeter-Update muß, wie beim BFGS-Update, der Gradient $\nabla f(x^{k+1})$ berechnet werden, was vollkommen analog erfolgt. Obwohl der Gradient parallel zu den Berechnungen des restlichen Update zusammengesetzt werden könnte, soll er aus den gleichen Gründen wie beim BFGS-Update direkt mit einer

globalen Operation (*gcolx()*) zusammengesetzt werden.
Jeder Prozessor verfügt also jetzt über den Vektor x^{k+1} des aktuellen Iterationspunktes und über den Gradienten $\nabla f(x^{k+1})$. Da die n partiellen Updates in ihrer Struktur identisch sind, soll nur demonstriert werden, wie das j-te partielle Update durchgeführt wird, welches aus den folgenden Operationen besteht:

$$
\begin{aligned}
y &= \nabla f(x^{k+1} + \epsilon e_j) - \nabla f(x^{k+1}) && (6.15)\\
t &= V^{k_{j-1}} y && (6.16)\\
r^{k_j} &= e_j - t && (6.17)\\
\kappa^{k_j} &= ((r^{k_j})^T y^k)^{-1} && (6.18)\\
V^{k_j} &= V^{k_{j-1}} + \kappa^k r^k (r^k)^T && (6.19)
\end{aligned}
$$

Nachdem der p-te Prozessor die Komponente y_p des Vektors y berechnet hat, wird das Matrix-Vektor-Produkt wie beim BFGS-Update spaltenorientiert gebildet. Prozessor p berechnet dazu analog wie beim BFGS-Update den Vektor $t^{(p)}$. Das Matrix-Vektor-Produkt t ergibt sich durch eine globale Summation der $t^{(p)}$. Die nächste globale Synchronisation wird für das Skalarprodukt (6.18) benötigt, da hier der Vektor y eingeht, von dem jeder Prozessor nur eine Komponente besitzt. Anstatt die Komponenten von y global im System zu verteilen, wird das Skalarprodukt verteilt berechnet, d.h. Prozessor p berechnet das Produkt $r_p^{k_j} y_p$. Anschließend erhält man $r^{k_j} y$ durch eine globale Summation der $r_p^{k_j} y_p$. Die globale Summation der Skalare benötigt nur etwa logarithmische Zeit in Abhängigkeit von der Anzahl der Knoten, während das Zusammensetzen des Vektors etwa lineare Zeit in der der Vektorlänge brauchen würde.
Das verbleibende Rang-1-Update (6.19) kann wieder vollständig lokal berechnet werden.

Nachdem man n solcher partiellen Update-Schritte durchgeführt hat, muß noch die Suchrichtung d^{k+1} bestimmt werden. Diese ergibt sich aus dem Matrix-Vektor-Produkt $-H^{k+1}\nabla f(x^{k+1})$ und erfordert eine weitere globale Vektorsumme.
Insgesamt werden also für das Straeter-Update $(n+1)$ globale Vektorsummen, eine globale Vektorkombination und n globale, skalare Summationen benötigt.

Unberücksichtigt bleibt bei diesem Ansatz die Tatsache, daß sich die n Gradienten $\nabla f(x^{k+1} + \epsilon e_j)$ unabhängig voneinander und folglich auch parallel berechnen lassen. Ebenso kann ihre Berechnung weitestgehend parallel zu den restlichen Operationen des Update durchgeführt werden. Die vollständige Ausnutzung dieser Parallelismen erfordert die Kombination von Funktions-Parallelismus und Daten-Parallelismus, was sich nicht in der SPMD-Philosophie realisieren läßt.
Von der logischen Seite her wird man zwei Prozesse einrichten, einen für die Gradienten (Gradienten-Prozeß) und einen für die Matrix-Operationen (Update-Prozeß), die von verschiedenen Prozessoren bearbeitet werden. Die Parallelisierung auf dieser Ebene ist ein reiner Funktions-Parallelismus. Beide Prozesse für sich betrachtet nutzen intern wieder daten-parallele Konzepte, so daß real zwei Gruppen von Pro-

zessoren entstehen, die jeweils eine Teilaufgabe, Gradientenberechnung oder Matrix-Operationen, bearbeiten.

Hauptproblem eines solchen Ansatzes ist die Koordination der beiden logischen Prozesse. Idealerweise wäre es so, daß der Update-Prozeß nach einer anfänglichen Wartezeit ununterbrochen mit Gradienten versorgt wird. Die anfängliche Wartezeit wird durch die Zeit bestimmt, die der Gradientenprozeß benötigt, um die Gradienten $\nabla f(x^{k+1}))$ und $\nabla f(x^{k+1} + \epsilon e_1)$ zu berechnen und sie dem Update-Prozeß mitzuteilen. Die reine Rechenzeit für die Gradienten läßt sich dabei auf die Zeit reduzieren, die ein Prozessor braucht, um eine Komponente eines Gradienten zu berechnen.
Da man grundsätzlich alle Gradienten gleichzeitig berechnen kann, wäre es denkbar, daß danach die Zeit pro partiellem Update ausschließlich durch die Matrix-Operationen und die Kommunikationszeiten zwischen den beiden Prozessen bestimmt wird. Um die Effizienz dieses Ansatzes zu optimieren, wird man sicherlich nicht alle Gradienten gleichzeitig berechnen, sondern nur so viele, wie notwendig sind, um den Update-Prozeß lückenlos mit Gradienten zu versorgen.
Die Kommunikationszeiten wiederum lassen sich durch asynchrone Empfangstechniken minimieren, da der Gradienten-Prozeß die Daten schon versenden kann, bevor der Update-Prozeß sie benötigt. Um zu verhindern, daß dabei vom empfangenden Knoten mehr als ein Vektor gepuffert werden muß, wird der Update Prozeß den Empfang eines Gradienten mit einer leeren Nachricht bestätigen.

Als recht kompliziert erweist sich der Entwurf der Kommunikationsstrukturen innerhalb und zwischen den beiden Prozessen, da sich diese gegenseitig beeinflussen und auch auf die Datenverteilung rückwirken.
Global hat man die Wahl, ob die Kommunikation zwischen den beiden Prozessen zentralisiert oder verteilt abläuft. Zentralisiert würde bedeuten, daß die Gradienten innerhalb des Gradienten-Prozesses zusammengesetzt und von einem ausgezeichneten Knoten zum Update-Prozeß geschickt und verteilt werden. Diese Technik bietet sich insbesondere dann an, wenn man die Gradienten auf jedem Knoten des Update-Prozesses vollständig haben will, wodurch sich in jedem partiellen Update eine Synchronisation einsparen ließe.
Im dezentralen Fall würde der Knoten, der eine Komponente eines Gradienten berechnet hat, diese direkt zu dem Knoten im Update-Prozeß schicken, der sie benötigt. Dadurch könnte man den Engpaß vermeiden, der durch die zentrale Kommunikation entsteht, andererseits wird die Koordination zwischen den Prozessen aufwendiger, da auf beiden Seiten variable Prozessorzahlen möglich sind.
Zentrale und dezentrale Kommunikation lassen sich als globale und lokale Master-Worker-Strukturen veranschaulichen. Im zentralen bzw. globalen Fall wäre der gesamte Update-Prozeß der Master, der die Gradientenberechnung an Worker verteilt. Die Worker ihrerseits lösen die Teilprobleme parallel, schicken aber das komplette Ergebnis (den zusammengesetzten Vektor) zurück.
Im dezentralen bzw. lokalen Fall wäre jeder Knoten im Update-Prozeß ein "kleiner" Master, der die Berechnung von Teilen des Gradienten als Auftrag an seine lokalen Worker verteilt. Die lokalen Worker lösen diese Aufgaben wieder parallel und

schicken die zusammengesetzten Teilgradienten zu ihrem Master.

Da dieser Ansatz aus Zeitgründen nicht mehr implementiert werden konnte, wird auf die Diskussion weiterer Details verzichtet. Es wird aber klar, daß dieser Ansatz für massiv parallele Systeme wahrscheinlich das größte Potential hat, dessen optimale Nutzung auch die größten Probleme aufwirft.
Neben dem Balancierungsproblem und der Schwierigkeit, den günstigsten Ansatz unter der Vielzahl von Variationsmöglichkeiten auszuwählen, betrifft das aber auch die konkrete programmtechnische Realisierung.
Dies wird insbesondere dadurch verstärkt, daß solche Multi-Level-Parallelismen nicht durch die Message-Passing-Bibliothek des Intel Paragon XP/S 10 unterstützt werden.

Nachdem die Parallelisierung der Strahlminimierung und der Updates bislang einzeln betrachtet wurde, sollen jetzt die Wechselwirkungen zwischen der Strahlminimierung und dem Update bzw. ihre Koordination im Mittelpunkt stehen.

6.3.3 Koordination von Update und Strahlminimierung

Sowohl die Strahlminimierung als auch das Update wurden so organisiert, daß die Vektoren x^k und d^k, die die Schnittstelle zwischen beiden Programmteilen definieren, jeweils vollständig auf allen Knoten vorhanden sind. Deswegen ist es das naheliegendste, Strahlminimierung und Update auf die gleichen Knoten zu legen und nacheinander abarbeiten zu lassen, so daß beim Wechsel vom einen in den anderen Programmteil keine Datenumverteilung notwendig ist. Für den Fall, daß man eine sequentielle Strahlminimierung benutzt, läßt man diese redundant auf allen Knoten laufen, um zu vermeiden, daß der Vektor x^{k+1} nach Beendigung der Strahlminimierung über alle Knoten verteilt werden muß. Abgesehen vom alternativen Ansatz für das Straeter-Update stellt dies die konsequente Fortsetzung der SPMD-Struktur dar.

Die Kopplung von einzelnen Programmteilen speziell in SPMD-Strukturen ist i. allg. jedoch nicht so problemlos wie es hier erscheinen mag.
So reicht es nicht aus, optimale parallele Algorithmen für die einzelnen Teilprobleme zu entwickeln, sondern es ist immer darauf zu achten, daß der Wechsel von einem zum anderen Programmteil keine aufwendigen Datenumverteilungen notwendig macht. Dies ist insbesondere in iterativen Verfahren wichtig, wo erwartungsgemäß viele solcher Übergänge auftreten werden.
Ein weiteres Problem ist die verschieden gute Skalierbarkeit der Programme zur Lösung der Teilprobleme, d.h. für jedes Teilproblem ist eine andere Prozessorzahl notwendig, um einen optimalen Speedup zu erzielen.
Für die Optimierungsverfahren betrifft das nicht nur die Strahlminimierung und das Update, sondern innerhalb des Update gibt es weitere Teilprobleme, wie das Matrix-Vektor-Produkt, das Rang-1-Update und die Gradientenberechnung, die möglicher-

weise alle mit einer anderen Prozessorzahl am schnellsten zu berechnen sind.
In der Regel wird man diesem Problem dadurch begegnen, daß man von jeder Operation entscheidet, ob sie auf allen Knoten sequentiell oder auf allen Knoten parallel ablaufen soll, was natürlich in der Datenverteilung berücksichtigt werden muß. Für einige Probleme wird diese Vorgehensweise jedoch unbefriedigend sein, so daß man versucht ist, einige Teilprogramme nur auf einer Teilmenge der Knoten durchzuführen. Meistens wird durch eine solche Maßnahme jedoch wieder vor und nach der Abarbeitung des Teilprogramms eine Datenumverteilung notwendig, so daß der Nutzen durch den höheren Speedup gegen diese Kommunikationskosten abzuwägen ist.
Es gibt jedoch auch Fälle, wo sich solche Kommunikations-Overheads vollständig vermeiden lassen, indem man eine Teilaufgabe nicht nur auf einer Teilmenge von Knoten, sondern redundant jeweils auf gleich großen Clustern von Knoten durchführt.
Wenn z.B. für das Update 10 und für die Strahlminimierung 2 Knoten optimal wären, so würde man die Strahlminimierung auf 5 Knotenpaaren durchführen und würde so die Verteilung des Ergebnisses auf die restlichen Knoten vermeiden, die notwendig wäre, wenn man die Strahlminimierung nur auf 2 Knoten laufen ließe. Das setzt jedoch voraus, daß die Clustergröße ein Teiler der gesamten Knotenzahl ist.
Eine reale Anwendung dieses Prinzips findet sich innerhalb des Update. Hier werden für die Gradientenberechnung deutlich mehr Knoten benötigt, als für die Matrix-Operationen optimal sind, so daß man die Matrix-Operationen clustern kann. Das setzt voraus, daß die Quasi-Newton-Matrix nicht mehr gleichmäßig über alle Knoten, sondern gleichmäßig innerhalb der einzelnen Cluster verteilt wird.
Je nach Kontext wird man in solchen Fällen natürlich auch die SPMD-Struktur in Frage stellen müssen, wie es im alternativen Ansatz für das Straeter-Update erfolgte.

Bislang unbeachtet ist geblieben, daß beim BFGS-Update die neue Suchrichtung vorliegt, bevor das Update beendet ist. Die Strahlminimierung und das eigentliche Rang-2-Update (also nur die Operation $H^{k+1} = H^k + uu^T + vv^t$) könnten somit parallel ablaufen. Um das zu realisieren, müßte man allerdings die Strahlminimierung auf andere Knoten legen als das Update. Das hat zwar den zusätzlichen Vorteil, daß man die Knotenzahlen für das Update und die Strahlminimierung jeweils optimal einstellen kann, macht es aber notwendig, daß zwischen den beiden Gruppen von Knoten die Vektoren d^k und x^k via Message-Passing ausgetauscht werden müssen. Pro Iteration muß dann zweimal ein Vektor-Broadcast stattfinden. Es wird also vom speziellen Problem abhängen, ob eine solche Parallelisierung einen Zeitvorteil bringt. Lohnend scheint sie nur, wenn das Rang-2-Update einen ähnlich hohen Anteil an der Rechenzeit hat wie die Strahlminimierung.

6.4 Diskussion der Meßergebnisse

In diesem Abschnitt werden die Meßergebnisse für einen Teil der beschriebenen Parallelisierungsansätze dargestellt. Da der Vergleich der verschiedenen Verfahrenskombinationen im Mittelpunkt der Betrachtungen stehen soll, wird darauf verzichtet, den Effekt von Techniken wie z.B. das Clustern oder die Parallelisierung von Update und Strahlminimierung zu zeigen. Stattdessen wird von jedem Verfahren (außer der Armijo-Schrittweite) nur eine Implementationsvariante getestet.

Bei der Durchführung der Messungen hat sich herausgestellt, daß die Iterationszahlen in den Verfahren bei Anwendung auf das Raffineriemodell z.T. in erheblichem Maße von der Anzahl der Knoten abhängen. Der Grund, daß überhaupt Abhängigkeiten von der Prozessorzahl auftreten, liegt in den globalen Summationen. Wegen der endlichen Rechnerarithmetik ist die Addition nicht assoziativ. Bei den globalen Summationen kommt es in Abhängigkeit von der Prozessorzahl zu Vertauschungen in der Summationsreihenfolge, so daß sich die Ergebnisse geringfügig unterscheiden können. In den meisten Fällen kann man solche Effekte bei einer 64 Bit Arithmetik vollständig vernachlässigen, so daß es zunächst nicht sehr plausibel zu sein scheint, daß die Abweichungen in den Verfahren diese Ursache haben. Kontrollmessungen an anderen Problemen, z.B. der Rosenbrock-Funktion, zeigen dieses Verhalten nicht, so daß Programmfehler als Ursache unwahrscheinlich sind.
Vielmehr ist der Grund für diese Probleme darin zu sehen, daß die Zielfunktion durch die Strafterme sehr schlecht konditioniert ist und sowohl die Gradienten als auch die Quasi-Newton-Matrizen sehr sensibel auf kleine Fehler reagieren. Eine Analyse der Iterationspunkte und Suchrichtungen zeigt, daß es nicht zu einem oszillierenden Verhalten um den stationären Punkt kommt. In diesem Fall könnte man das Problem nämlich durch ein anderes Abbruchkriterium in den Griff bekommen. Stattdessen kann man beim Straeter-Verfahren schon nach zwei Iterationen erhebliche Unterschiede in den Suchrichtungen erkennen, die das Konvergenzverhalten teils positiv, teils negativ beeinflussen. Beim BFGS-Verfahren sind die Unterschiede nicht so groß. Während die extremsten Abweichungen beim Straeter-Verfahren bei ca. 30 % liegen, betragen sie beim BFGS-Verfahren höchstens ca. 10 % (bezogen auf den sequentiellen Meßwert). Offensichtlich ist das Straeter-Verfahren weniger robust als das BFGS-Verfahren.

Weil die Summationsreihenfolge bei sequentiellem Ablauf der Verfahren in keiner Weise ausgezeichnet ist, wird man zwangsläufig die Aussagekraft der sequentiellen Meßergebnisse in Zweifel ziehen müssen. Bis auf kleine Abweichungen spiegeln sie jedoch den Trend für das gedächtnislose Straeter-Verfahren und das BFGS-Verfahren richtig wieder. Einzige Ausnahme ist das gedächtnislose Straeter-Verfahren bei Anwendung auf das dreifach iterierte Raffineriemodell, wo in der Kombination mit der Armijo-Schrittweite deutlich weniger Iterationen benötigt werden als im Durchschnitt.

Um dennoch die Effekte der Parallelisierung am Raffineriemodell untersuchen zu

können, werden die Iterationszahlen der parallelen Programmabläufe auf die Iterationen der sequentiellen Programmabläufe normiert. Dabei wurden keine Korrekturen an den Iterationszahlen vorgenommen. Daß die Kombination Straeter-Armijo-Verfahren beim dreifach iterierten Modell im Vergleich zu den Meßwerten mit anderen Prozessorzahlen zu wenig Iterationen benötigt, wird in der Diskussion der Ergebnisse berücksichtigt.

Bevor BFGS- und Straeter-Verfahren miteinander verglichen werden, soll zunächst wieder für beide Verfahren die beste Strahlminimierung ermittelt werden. Beim BFGS-Verfahren ist im Sequentiellen die Armijo-Schrittweite am schnellsten. Die Kosten für die zusätzlich benötigten Iterationen im Quasi-Newton-Verfahren werden durch die Zeiteinsparung in der Strahlminimierung deutlich übertroffen. Beim Straeter-Verfahren hingegen ist es bei sequentiellem Programmablauf günstiger, die Anzahl der zeitaufwendigen Updates durch eine exaktere und kostspieligere Strahlminimierung zu reduzieren. Welche Kombination im Parallelen optimal ist, wird also davon abhängen, ob sich die Kostenanteile von Update und Strahlminimierung verschieben.

Die Abbildung 6.1 (Seite 81) zeigt die Rechenzeiten der BFGS-Verfahren für die drei Raffineriemodelle in Abhängigkeit von der Anzahl der Prozessoren. Bei allen drei Problemen zeigt sich die Armijo-Schrittweite wieder als überlegen, wobei die parallele Version der Armijo-Schrittweite nochmals besser ist als die sequentielle. Das Verfahren des goldenen Schnitts, das im BFGS-Verfahren je nach Problemgröße einen sequentiellen Anteil von 64 - 45% hat, liefert erwartungsgemäß die schlechtesten Speedups und ist sogar schlechter als die Fletchersche Strahlminimierung.

Die maximal erreichten Speedups und die dazu benötigten Prozessorzahlen p sind der Tabelle 6.4 zu entnehmen, wo sowohl der Speedup bzgl. der Laufzeit desselben Verfahrens auf einem Prozessor

$$S_{Verf}(p) = \frac{T_{Verf}(1)}{T_{Verf}(p)}$$

als auch der Speedup bzgl. des besten sequentiellen Verfahrens

$$S_{opt}(p) = \frac{T_{opt}(1)}{T_{Verf}(p)}$$

angegeben ist. Während $S_{Verf}(p)$ primär eine Aussage über die Skalierbarkeit eines Algorithmus macht, erlaubt der Speedup $S_{opt}(p)$ einen direkten Vergleich der Rechenzeiten verschiedener Verfahren.

Man sieht deutlich, daß die Kombination mit der Fletcherschen Strahlminimierung zwar die besten verfahrensspezifischen Speedups liefert, aber im Vergleich zum besten sequentiellen Verfahren nicht mit der Armijo-Schrittweite konkurrieren kann

und dieser Nachteil mit der Problemgröße zunimmt.
Der Gewinn durch die Parallelisierung der Armijo-Schrittweite nimmt ebenfalls mit der Problemgröße zu, da die Kosten der Funktionsauswertung im Verhältnis zu den Kommunikationskosten größer werden.

Beim Straeter-Verfahren setzt setzt sich der Trend aus den sequentiellen Meßergebnissen nicht uneingeschränkt fort. Die Abbildung 6.2 (Seite 83) zeigt wieder die benötigten Rechenzeiten für die einzelnen Probleme in Abbhängigkeit von der Anzahl der Prozessoren. Beim einfachen Problem kann sich die Armijo-Schrittweite gegenüber dem Verfahren des goldenen Schnitts behaupten, obwohl mehr Iterationen benötigt werden. Das liegt daran, daß der Anteil der Strahlminimierung an den Gesamtkosten noch relativ hoch liegt. Im Sequentiellen beträgt er bei der Fletcherschen Stahlminimierung 44.4 %, 10.8 % beim Verfahren des goldenen Schnitts und nur 2.9 % bei der Armijo-Schrittweite. Deshalb wird der Nachteil der höheren Iterationszahlen durch die bessere Skalierbarkeit der Verfahren kompensiert.
Beim zweifach und dreifach iterierten Problem treten die Kosten für die Strahlminimierung wieder in den Hintergrund, und die Anzahl der Quasi-Newton-Iterationen wird zur dominierenden Größe. Dementsprechend verliert dort die Armijo-Schrittweite ihren Vorteil gegenüber den aufwendigeren Verfahren. Leider wird dieser Trend beim dreifach iterierten Modell durch die vorher erwähnte Schwankung der Iterationszahlen etwas überdeckt. Deswegen sind in Abbildung 6.3 (Seite 84) die gleichen Verfahren bei Anwendung auf die iterierte Rosenbrock-Funktion dargestellt, wo der Vorteil der exakteren Strahlminimierungen sehr deutlich wird.

Tabelle 6.5 zeigt die Speedups für das Straeter-Verfahren in Kombination mit den verschiedenen Strahlminimierungen, wobei wieder berücksichtigt werden muß, daß die Werte für die Armijo-Schrittweite beim dreifach iterierten Problem zu relativieren sind.

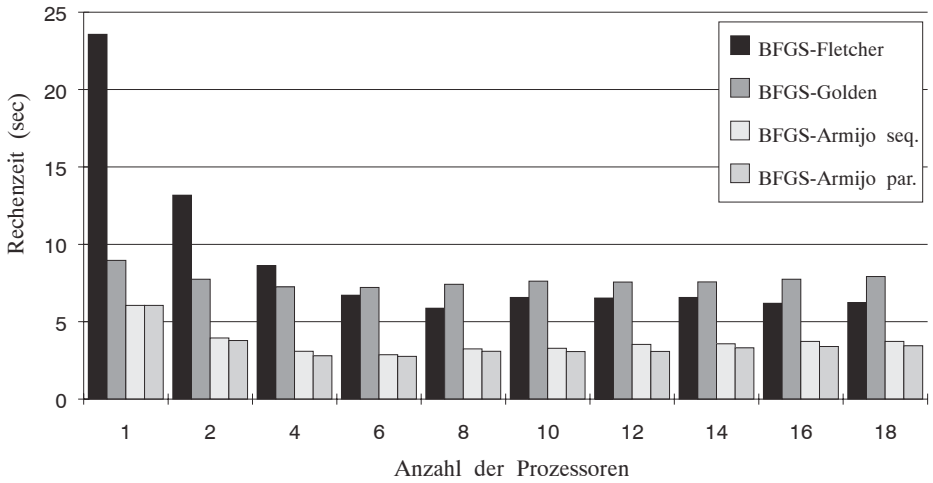
Bei allen Kombinationen sind die verfahrensspezifischen Speedups $S_{Verf}(p)$ deutlich höher als diejenigen, die auf das schnellste sequentielle Verfahren bezogen sind. Welche Verfahrenskombination optimal ist, läßt sich nun nicht eindeutig beantworten. Deswegen wird für die weiteren Vergleiche mit dem BFGS-Verfahren die Variante mit der Strahlminimierung des goldenen Schnitts benutzt. In Tabelle 6.6 sind vergleichend die Speedups $S_{opt}(p)$ und die benötigten Prozessorzahlen vom Straeter-Verfahren und dem BFGS-Verfahren aufgelistet. Man sieht, daß das BFGS-Verfahren mit der parallelen Armijo-Schrittweite nicht nur schneller ist als das Straeter-Verfahren mit der Strahlminimierung des goldenen Schnitts, sondern auch weniger als die Hälfte der Prozessoren dazu benötigt. Das BFGS-Verfahren ist bei diesem speziellen Problem und der gewählten Implementation sowohl schneller als auch effizienter. Wenn man zusätzlich ins Kalkül einbezieht, daß das BFGS-Verfahren numerisch stabiler ist als das Straeter-Verfahren, bleibt kaum mehr ein Grund für die Anwendung des Straeter-Verfahrens.
Es muß jedoch darauf hingewiesen werden, daß diese Aussagen nicht verallgemei-

nerbar sind, sondern sowohl von der Implementation als auch von der Anwendung abhängen.

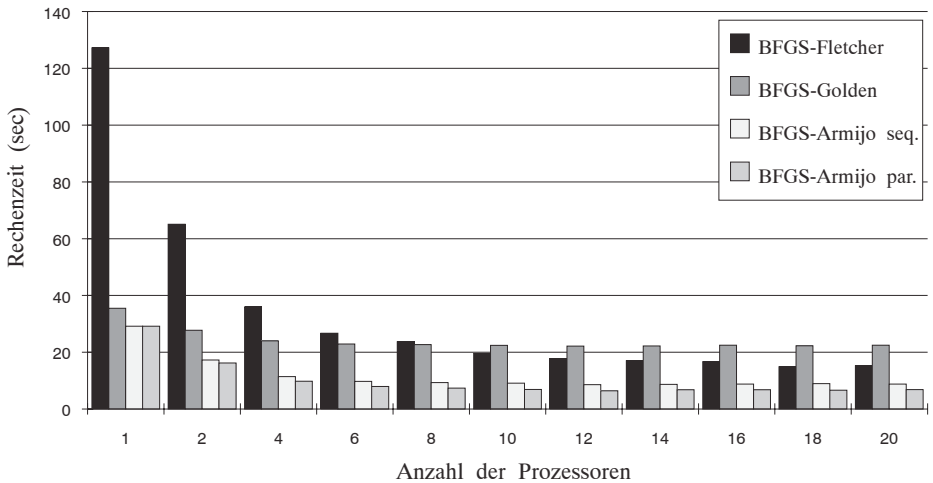
Für eine implementationsunabhängige Einordung der Verfahren ist es sinnvoll, die Anzahl der insgesamt berechneten Gradienten zu vergleichen, da die Kosten der Verfahren primär durch die Gradientenberechnung bestimmt werden.
Tabelle 6.7 zeigt die Anzahl der insgesamt berechneten Gradienten im jeweils günstigsten BFGS- und Straeter-Verfahren.

Man sieht, daß im BFGS-Verfahren wesentlich weniger Gradienten berechnet werden müssen als im Straeter-Verfahren und daß das Verhältnis mit zunehmender Problemgröße für das Straeter-Verfahren schlechter wird.
Es scheint plausibel zu sein, daß das Straeter-Verfahren langamer ist, wenn man nicht in jedem Update mehrere Gradienten gleichzeitig berechnet. Daß das Straeter-Verfahren im Vergleich zum BFGS-Verfahren besser abgeschnitten hat, als die Anzahl der benötigten Gradienten vermuten läßt, liegt an den Einsparungen bei der Strahlminimierung und dem geringeren Synchronisationsaufwand im Straeter-Update.
Wenn man also in jedem Straeter-Update mehrere Gradienten gleichzeitig berechnet, wird es kein Problem sein, die Laufzeiten des Straeter-Verfahrens unter die des BFGS-Verfahrens zu senken.
Das Problem ist nur, daß die Anzahl der zusätzlich zu berechnenden Gradienten mit der Problemgröße zunimmt. Unter den Annahmen, wie sie beim Raffineriemodell zutreffen, ist der Aufwand für die Berechnung eines Gradienten der Dimension n von der Ordung $O(n^3)$, d.h. für große Probleme wird man recht schnell an die Grenzen der üblicherweise zur Verfügung stehenden Hardwareresourcen stoßen, so daß es günstiger ist, den numerischen Aufwand zu senken, als den Parallelisierungsgrad der Verfahren weiter zu erhöhen.

(a) Einfaches Raffineriemodell ($n = 18$)



(b) Zweifaches Raffineriemodell ($n = 36$)



(c) Dreifaches Raffineriemodell ($n = 54$)

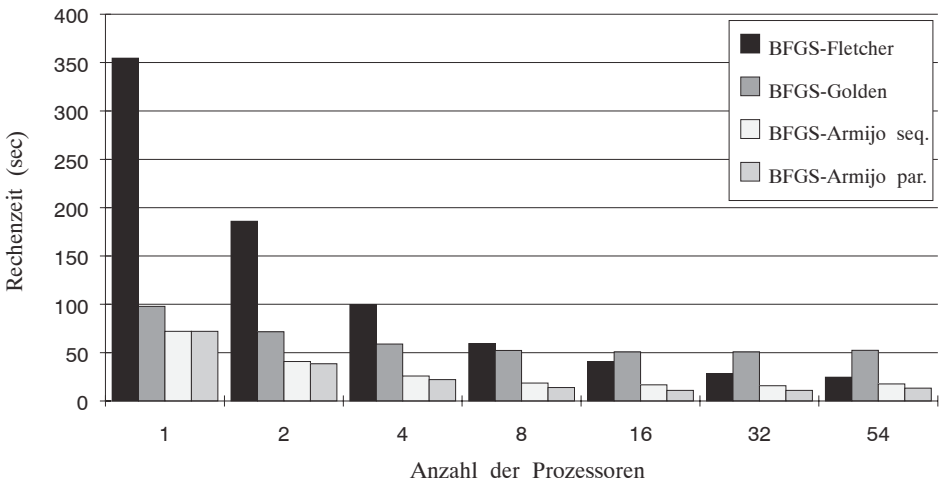


Abbildung 6.1: Rechenzeiten des BFGS-Verfahrens bei Anwendung auf das iterierte Raffineriemodell

Problem	$R^{(1)}$			$R^{(2)}$			$R^{(3)}$		
	$S_{Verf}(p)$	$S_{opt}(p)$	p	$S_{Verf}(p)$	$S_{opt}(p)$	p	$S_{Verf}(p)$	$S_{opt}(p)$	p
Fletcher	4.00	1.03	8	8.55	3.41	18	14.40	2.93	54
Golden Sect.	1.24	0.84	6	1.60	1.31	12	1.93	1.42	16
Armijo seq.	2.11	2.11	6	4.40	4.40	12	4.56	4.56	32
Armijo par.	2.18	2.18	6	4.53	4.53	12	6.65	6.65	16

Tabelle 6.4: **Speedup und benötigte Prozessorzahl im BFGS-Verfahren**

Problem	$R^{(1)}$			$R^{(2)}$			$R^{(3)}$		
	$S_{Verf}(p)$	$S_{opt}(p)$	p	$S_{Verf}(p)$	$S_{opt}(p)$	p	$S_{Verf}(p)$	$S_{opt}(p)$	p
Fletcher	3.99	1.75	14	7.70	3.35	32	14.60	3.37	54
Golden Sect.	2.67	1.74	16	6.22	3.16	32	12.65	4.26	54
Armijo seq.	3.20	1.86	16	7.44	2.41	20	12.38	3.87	54
Armijo par.	3.36	1.95	18	8.02	2.59	26	13.20	4.14	54

Tabelle 6.5: **Speedup und benötigte Prozessorzahl im Straeter-Verfahren**

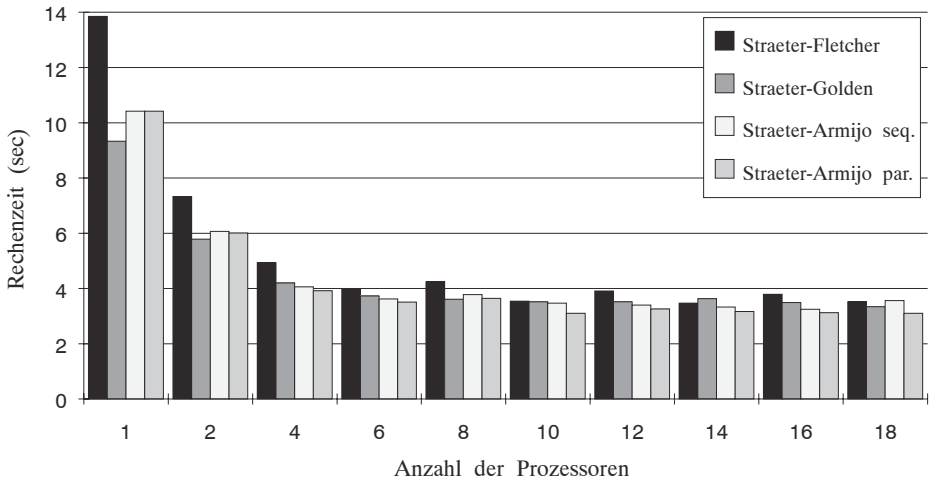
	BFGS Armijo par.	Straeter goldener Schnitt
$R^{(1)}$	2.18 (6)	1.74 (16)
$R^{(2)}$	4.53 (12)	3.16 (32)
$R^{(3)}$	6.65 (16)	4.26 (54)

Tabelle 6.6: **Vergleich des Speedups $S_{opt}(p)$ zwischen dem günstigsten BFGS- und Straeter-Verfahren.**

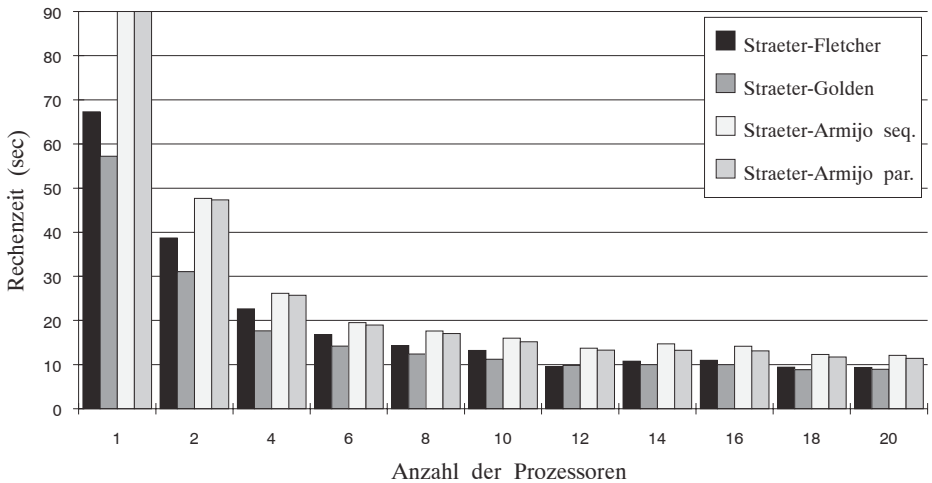
	BFGS Armijo	Straeter goldener Schnitt
$R^{(1)}$	366	627
$R^{(2)}$	500	1295
$R^{(3)}$	580	1925

Tabelle 6.7: **Anzahl der berechneten Gradienten**

(a) Einfaches Raffineriemodell ($n = 18$)



(b) Zweifaches Raffineriemodell ($n = 36$)



(c) Dreifaches Raffineriemodell ($n = 54$)

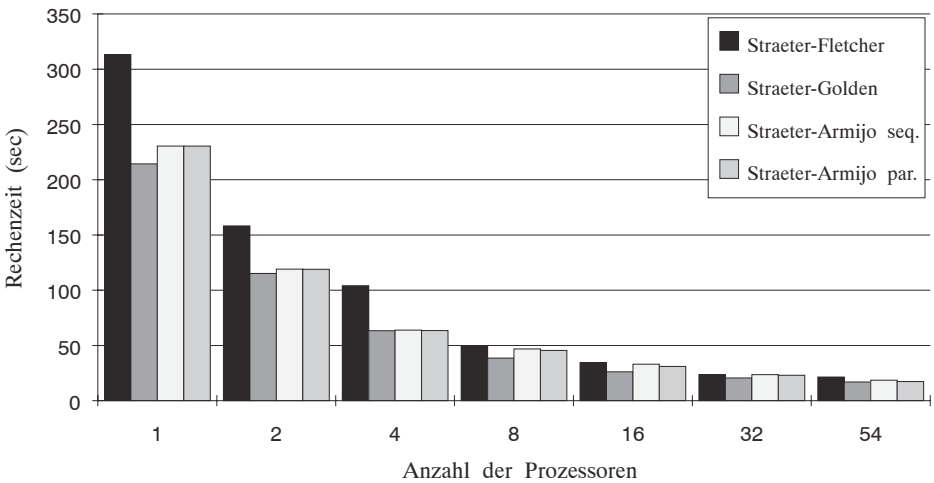


Abbildung 6.2: Rechenzeiten des Straeter-Verfahrens bei Anwendung auf das iterierte Raffineriemodell

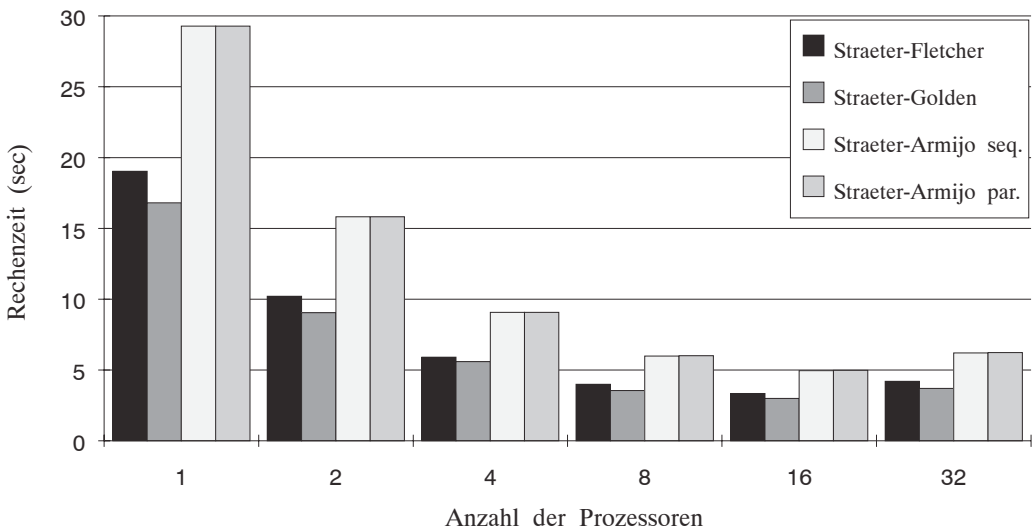


Abbildung 6.3: Rechenzeiten des Straeter-Verfahrens bei Anwendung auf die iterierte Rosenbrockfunktion $n = 128$

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde am Beispiel eines Raffineriemodells der gesamte Prozeß einer nichtlinearen Optimierung von der Modellierung bis zur parallelen Problemlösung dargestellt.
Ziel der Arbeit war insbesondere, die Auswirkungen verschiedener Verfahrenskombinationen auf das Konvergenzverhalten der Verfahren und ihre Performance auf Parallelrechnern zu untersuchen. Dazu wurden das BFGS-Verfahren und einige Varianten des Straeter-Verfahrens mit mehreren Techniken zur eindimensionalen Optimierung kombiniert, mit denen sich gezielt das Konvergenzverhalten und der Parallelisierungsgrad variieren läßt. Der Vergleich der Verfahren erfolgte anhand von Implementationen für den Intel Paragon XP/S 10. Neben den benutzten Parallelisierungsansätzen wurden einige alternative Implementationstechniken und Parallelisierungskonzepte aufgezeigt, deren Nutzen für die Performancesteigerung noch untersucht werden muß.

Das BFGS-Verfahren, das als bestes sequentielles Quasi-Newton-Verfahren gilt [Fl87], konnte sich zusammen mit der Armijo-Schrittweite zur eindimensionalen Optimierung gegenüber allen anderen untersuchten Verfahrensvarianten behaupten. Bei den im Rahmen dieser Arbeit vorgenommenen Implementationen war es schneller, effizienter und numerisch stabiler als die Straeter-Verfahren, obwohl sich mit den letzteren höhere verfahrensspezifische Speedups erzielen ließen. Grundsätzlich ist es mit einer verbesserten Implementation des Straeter-Verfahrens möglich, die Rechenzeiten bei kleinen Problemen unter die des BFGS-Verfahrens zu senken. Der numerische Mehraufwand des Straeter-Verfahrens nimmt jedoch mit der Problemgröße erheblich zu, so daß der Einsatz dieses Verfahrens für große Probleme wegen der schlechten Effizienz nicht mehr vertretbar ist.

Um den erheblichen Mehraufwand des Straeter-Update gegenüber dem BFGS-Update zu vermindern, wurde der Versuch unternommen, in jeder Iteration nur einen Teil der partiellen Updates durchzuführen. Der pro Iteration gewonnene Zeitvorteil wurde jedoch durch das wesentlich schlechtere Konvergenzverhalten vollständig eingebüßt.

Außerdem erwies sich diese Variante bei einigen Problemen als numerisch instabil, was sie für die Anwendung auf das Raffineriemodell unbrauchbar machte.
Obwohl der Ansatz unvollständiger Updates nicht erfolgreich eingesetzt werden konnte, scheint dies doch die einzige Möglichkeit zu sein, das Straeter-Verfahren essentiell zu verbessern. Gegenstand weiterer Untersuchungen können somit Methoden sein, welche die Effizienz bzw. den Nutzen der partiellen Updates steigern.

Trotz der Nachteile gegenüber anderen Quasi-Newton-Verfahren bleibt das Straeter-Verfahren auch deshalb für die Parallelverarbeitung interessant, weil es alternative Ansätze zur Parallelisierung zuläßt. Neben den schon beschriebenen Varianten wäre es denkbar, die partiellen Updates durch asynchron laufende Prozesse durchführen zu lassen. Dadurch könnte wie beim asynchronen Newton-Verfahren (Kapitel 6.1) eine Parallelisierung des Update und der Strahlminimierung erreicht werden. Obwohl derartige Ansätze vereinzelt beschrieben werden (z.B. in [Lo91]), gibt es bislang keine Veröffentlichungen über Implementationen solcher Verfahren.

Ein anderer Weg zur Verbesserung der Optimierungsverfahren wäre eine alternative Berücksichtigung der Nebenbedingungen. Der Ansatz der Straffunktionen führt sowohl zu numerischen Problemen als auch zur extremen Verteuerung der Funktions- und Gradientenberechnung. Insbesondere für große Systeme mit einigen hundert oder tausend Nebenbedingungen wird ein solcher Ansatz nicht mehr sehr effektiv sein. Da im Falle des Raffineriemodells und vieler anderer ökonomischer Modellierungen nur lineare Nebenbedingungen vorliegen, ist es sicherlich vorteilhaft, diese spezielle Struktur auszunutzen. Bei einer Klasse von Verfahren, die das berücksichtigt, werden die Suchrichtungen, die sich mit modifizierten Quasi-Newton-Verfahren erzeugen lassen, auf den linearen Restriktionsbereich projiziert. Die Projektionen lassen sich mit einfachen Matrix-Operationen durchführen, die auf Parallelrechnern effektiv realisiert werden können. Deshalb wäre speziell für große Probleme ein Vergleich mit den Penalty-Verfahren interessant.

Literaturverzeichnis

[Akl89] S.G. Akl, *The Design and Analysis of Parallel Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1989

[AvWi66] M. Avriel, D.J. Wilde, *Optimal search for a maximum with sequences of simultaneous function evaluations*. Management Science 12 (1966) 722-731

[AvWi] M. Avriel, D.J. Wilde, *Golden block search for the maximum of unimodal functions*. Management Science 14 (1968) 307-319

[BaSS93] M.S. Bazaraa, H.D. Sherali, C.M.Shetty, *Nonlinear Programming*, 2nd ed. Wiley New York, 1993

[Be94] R. Berrendorf, H.C. Burg, U. Detert, R. Esser, M. Gerndt, R. Knecht. *Intel Paragon XP/S – Architecture, Software Environment, and Performance*. Interner Bericht KFA-ZAM-IB-9409, Forschungszentrum Jülich – ZAM, Jülich, Mai 1994

[BeTs89] D.P. Bertsekas, J.N. Tsitsiklis, *Parallel and Distributed Computation*. Prentice Hall, Englewood Cliffs, New Jersey, 1989

[BeWi70] J.H. Beamer, D.J. Wilde, *Minimax optimization of unimodal functions by variable block search*. Management Science 16 (1970) 529-541

[Br70] C.G. Broyden, *The convergence of a class of double rank minimization algorithms, parts I and II*, J. Inst. Math. Applns., 6 (1970) 76-90, 222-231

[BSS88] R.H. Byrd, R.B. Schnabel, G.A. Shultz, *Parallel Quasi-Newton Methods for Unconstrained Optimization*. Mathematical Programming 42 (1988) 273-306

[CGM92] D. Conforti, L. Grandinetti, R. Musmanno, *An asynchronous parallel implementation of Newton method for unconstrained optimization*, in Parallel Computing: Problems, Methods and Applications (Eds. P. Messina, A. Murli), Elsevier, Amsterdam 1992

[Da59] W.C. Davidon, *Variable metric method for minimization*. AEC Res. and Dev. Report ANL-5990 (revised)

[DyLT89] M. Dayde, M. Lescrenier, Ph. Toint, *A Comparison between Straeter's Parallel Variable Metric Algorithm and Parallel Discrete Newton Methods*. Report of the ENSEEIHT de Toulouse, France, Département Informatique, 1989

[FiRi88] H. Fisher, K. Ritter, *An asynchronous parallel Newton method*. Mathematical Programming 42 (1988) 363-374

[Fl70] R. Fletcher, *A new approach to variable metric algorithms*. Computer J., 13 (1970) 317-322

[Fl87] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. Wiley New York, 1987

[FlFr77] R. Fletcher, T.L. Freeman, *A modified Newton method for minimization*. Journal of Optimization Theory and Applications, 23 (1977) 357-372

[FlPo63] R. Fletcher, M.J.D. Powell, *A rapidly convergent descent method for minimization*. Computer J., 6 (1963) 163-168

[Ge91] R.A. van de Geijn. *Efficient global combine operations*. In Sixth Distributed Memory Computing Conference Proceedings, pages 291-294, IEEE Computer Society Press, 1991

[GiMu74] P.E. Gill, W. Murray, *Quasi-Newton methods for linearly constrained optimization*, in Numerical Methods for Constrained Optimization (Eds P.E. Gill, W. Murray), Academic Press, London 1974

[Go70] D. Goldfarb, *A family of variable metric methods derived by variational means*. Math. comp., 24 (1970) 23-26

[Gol65] A.A. Goldstein, *On steepest descent*. SIAM J. Control, 3, 147-151

[Gre70] J.L. Greenstadt, *Variations of variable metric methods*. Maths. Comp., 24 (1970) 1-22

[GrTe93] C. Großmann, J. Terno, *Numerik der Optimierung*. Teubner Stuttgart, 1993

[H79] R. Horst, *Nichtlineare Optimierung*. Carl Hanser Verlag München Wien, 1979

[Ho83] F. Hoßfeld, *Parallele Algorithmen*. Informatik-Fachberichte Band 64, Springer, Heidelberg 1983

[HS81] W. Hock, K. Schittkowski, *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 1981

[Hu70] H.Y. Huang, *Unified approach to quadratically convergent algorithms for function minimization*. Journal of Optimization Theory and Applications 5 (1970) 405-423

[JoTr88] H.Th. Jongen, E. Triesch, *Optimierung A*. Augustinus-Buchhandlung Aachen, 1988

[KaMi68] R.M. Karp, W.L. Miranker, *Parallel minimax search for a maximum*. Journal of Combinatorial Theory 4 (1968) 19-35

[La85] P.J.M. van Laarhoven, *Parallel Variable Metric Algorithms for Unconstained Optimization*. Mathematical Programming 33 (1985) 68-81

[Lo91] F.A. Lootsma, *Parallel Newton-Raphson Methods for Unconstrained Minimization with Asynchronous Updates of the Hessian Matrix or its Inverse*, in Parallel Computing and Mathematical Optimization, Proceedings of the Workshop on Parallel Algorithms and Transputers for Optimization, (Eds. M. Grauer, D.B. Pressmar), Springer-Verlag Berlin, 1991

[LoRa88] F.A. Lootsma, K.M. Ragsdell, *State-of-the-art in parallel nonlinear optimization*. Parallel Computing 6 (1988) 133-155

[Lue73] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Reading, Mass., 1973

[Mar63] D.W. Marquardt, *An algorithm for least squares estimation of nonlinear parameters*. SIAM J., 11 (1963) 431-441

[Mu72] W. Murray, *Second derivative methods*, In Numerical Methods for Unconstrained Optimization (Ed. W. Murray). Academic Press, London, 1972

[NaSo89] S.G. Nash, A. Sofer, *Block Truncated-Newton Methods for Parallel Optimization*. Mathematical Programming 45 (1989) 529-546

[NaSo92a] S.G. Nash, A. Sofer, *A parallel line search for Newton-type methods*. In Computer Science and Statistics: Proceedings of the 21st Symposium of the Interface (Alexandria, VA), K. Berk and L. Malone, Eds. ASA, 1989, 134-137

[NaSo92b] S.G. Nash, A. Sofer, *BTN: Software for Parallel Unconstrained Optimization*. ACM Transactions on Mathematical Software, Vol. 18, No. 4, (1992) 414-448

[NM93] L.M. Ni, P.K. McMinley, *A Survey of Wormhole Routing Techniques in Direct Networks*. IEEE Computer, February 1993, 62-76

[Per87] R.H. Perrott, *Parallel Programming*. Addison-Wesley, New York 1987

[RRR83] G.V. Reklaitis, A. Ravindran, K.M. Ragsdell, *Engineering Optimization, Methods and Applications*. Wiley New York, 1983

[SaRa80] E. Sandgren, K.M. Ragsdell, *The Utility of Nonlinear Programming Algorithms: A Comparative Study–Parts 1 and 2*. ASME J. Mech. Des., 102(3), 1980, 540-551

[Sc75] S.B. Schuldt, *A Method of Multipliers for Mathematical Programming Problems with Equality and Inequality Constraints*. JOTA, 17, (1/2), 155-162, 1975

[Sch80] K. Schittkowski, *Nonlinear Programming Codes: Information, Tests, Performance*, in Lecture Notes in Economics and Mathematical Systems, Vol. 183, Springer-Verlag, New York, 1980

[Sh70] D.F. Shanno, *Conditioning of quasi-Newton methods for function minimization*. Maths. Comp., 24 (1970) 647-656

[Stra73] T.A. Straeter, *A parallel variable metric optimization algorithm*. NASA Technical Note D-8020, Langley Research Center, Hampton VA, 1973

Danksagung

Diese Arbeit wurde am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen angefertigt. Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Rechenanlagen und anderen Betriebsmittel des Zentralinstituts für Angewandte Mathematik (ZAM) im Forschungszentrum Jülich nutzen zu können.

Herrn Dr. P. Weidner danke ich für die Betreuung der Arbeit und die Freiheit, die er mir bei der Ausgestaltung des Themas ließ. Insbesondere bin ich ihm zum Dank verpflichtet, weil er bereit war, die Arbeit Korrektur zu lesen, und mit konstruktiver Kritik zur Verbesserung der Ausarbeitung maßgeblich beigetragen hat.

Mein besonderer Dank gilt Irene Merk, die in endlosen Diskussionen an der Entstehung der Arbeit mitgewirkt und das Manuskript Korrektur gelesen hat.

Ich danke Ulrik Brandes für das Lesen des Manuskripts und die wertvollen inhaltlichen Hinweise.

Weiterhin danke ich Barbara Maren Winkler für ihre Anregungen bzgl. der Penalty-Verfahren und allen anderen, die zum Gelingen der Arbeit beigetragen haben.