Christian-Albrechts-Universität zu Kiel

Anordnungsalgorithmen für konvektionsdominante Probleme und deren Implementierung

Diplomarbeit
von
Ronald Kriemann

Institut für Informatik und
und Praktische Mathematik
Prof. Dr. W. Hackbusch

August 1999

Inhaltsverzeichnis

1. Einleitung **5**

1.1. Einführung in die Problematik . . . 5

1.2. Gliederung der Arbeit . . . 6

2. Die Konvektions-Diffusions-Gleichung **7**

2.1. Problemstellung . . . 7

2.2. Variationsformulierung . . . 8

2.3. Finite-Element-Diskretisierung . . . 8

2.4. Diskretisierung des Konvektionsanteils . . . 11

2.5. Repräsentation der Gitter . . . 12

3. Graphentheoretische Grundlagen **15**

3.1. Graphen und Digraphen . . . 15

3.1.1. Planare Graphen . . . 17

3.1.2. Graph einer Matrix . . . 17

3.1.3. Gewichtete Graphen . . . 18

3.1.4. Implementierung . . . 19

3.1.5. Pfade, Zykel und Zusammenhang . . . 20

3.2. Das Feedback Vertex Set Problem . . . 21

3.2.1. Ein heuristischer FVS-Algorithmus . . . 22

3.2.2. Ein FVS-Algorithmus für planare Graphen . . . 28

4. Anordnungsalgorithmen **43**

4.1. Permutationen und Permutationsmatrizen . . . 43

4.2. Bandbreitenreduktion . . . 44

4.3. Anordnungsalgorithmen für konvektionsdominante Probleme . . . 47

4.3.1. Reduktion des Matrixgraphen . . . 48

4.3.2. Anordnungen im $\mathbb{R}^2$ . . . 48

4.3.3. Anordnungen im $\mathbb{R}^3$ . . . 54

5. Iterationsverfahren **63**
5.1. Allgemeines 63
5.2. Die Gauß-Seidel-Iteration 64
5.3. Das BiCG-Stab-Verfahren 65
5.4. Das GMRES-Verfahren 67

6. Numerische Ergebnisse **69**
6.1. Testprobleme 69
6.1.1. Triangulationen 70
6.1.2. Strömungsfelder 71
6.2. Konvergenzraten 73
6.2.1. Experimente im $\mathbb{R}^2$ 73
6.2.2. Experimente im $\mathbb{R}^3$ 76
6.3. Aufwand der Verfahren 79

1. Einleitung

1.1. Einführung in die Problematik

Aufgrund der rasanten Steigerung der Leistungsfähigkeit von Rechnersystemen hat sich die numerische Simulation von technischen oder physikalischen Problemen in den letzten Jahren zu einer wichtigen Alternative zu Experimenten entwickelt. Sie stellt aber nicht nur eine Alternative dar. Bei einigen Problemen gelang die Untersuchung von Theorien erst durch die Modellierung auf dem Computer. Man denke etwa an Prozesse, die in keinem Experiment durchführbar sind, z.B. die Kollision zweier Galaxien.

Den Rahmen dieser Arbeit bildet die numerische Simulation von Strömungen. Vorgänge in denen Strömungen eine Rolle spielen, kommen in vielen Bereichen der Wissenschaft, aber auch des täglichen Lebens vor. Die Beispiele reichen hierbei von mikroskopischen Vorgängen wie dem Fluß von Elektronen in Schaltkreisen, über die Konstruktion von Flugzeugen, speziell deren Aerodynamik, bis hin zum Energie- und Massentransport in Sternen oder Galaxien.

In Strömungen spielen verschiedene physikalische Vorgänge eine Rolle. Durch die *Diffusion* kommt es zu einem ungerichteten Ausgleich von Stoffkonzentrationen, etwa wenn sich zwei Gase vermischen. Ein weiteres Phänomen ist die *Konvektion*, die einen Stofftransport in Strömungsrichtung bewirkt. Ein mathematisches Modell welches diese beiden Vorgänge beinhaltet, ist die *Konvektions-Diffusion-Gleichung*.

Bei der numerische Behandlung des gegebenen Problems ist der unterschiedliche Einfluß der Diffusion und der Konvektion oft entscheidend für den Erfolg des verwendeten Lösungsverfahrens. Bei einer vernachlässigbaren Konvektion stellt die Methode der finiten Elemente in Verbindung mit der Mehrgitteriteration ein stabiles, schnell konvergierendes Verfahren zur Lösung des gegebenen Problemes dar. Wird aber die Konvektion dominanter, so verschlechtert sich die Konvergenz oder das Iterationsverfahren divergiert sogar.

Allerdings sind viele Iterationsverfahren von der Anordnung der Unbekannten im gegebenen Gleichungssystem abhängig. Ein möglicher Ausweg aus dem Dilemma könnte also darin bestehen, eine Anordnung zu finden, durch die das Lösungsverfahren schnell konvergiert. Eine Möglichkeit hierzu ist die Unbekannten so zu numerieren, daß ihre Ordnung der Konvektionsrichtung folgt. Im Idealfall gelingt es, die Matrix des Systems auf Dreiecksgestalt zu bringen, wodurch das System mit einem einzigen Gauß-Seidel-Schritt gelöst werden kann. Algorithmen zur Berechnung einer solchen Anwendung sind Gegenstand dieser Arbeit.

1.2. Gliederung der Arbeit

Zunächst erfolgt in Kapitel 2 die Definition des betrachten Modellproblems, wobei es sich um die bereits erwähnte Konvektions-Diffusions-Gleichung handelt. Anschließend wird auf die Diskretisierung des Problems mit Hilfe der Methode der finiten Elemente eingegangen. In diesem Zusammenhang werden ebenfalls Verfeinerungstechniken für zwei- und dreidimensionale Gitter vorgestellt.

Viele der Anordnungstechniken die in dieser Arbeit vorgestellt werden, basieren auf Graphenalgorithmen. In Kapitel 3 werden die nötigen graphentheoretischen Begriffe eingeführt und das *Feedback-Vertex-Set-Problem* vorgestellt. Bei diesem Problem geht es um die Bestimmung von Knoten eines Graphen, deren Entfernen sämtliche Zykel in diesem Graphen auflöst. Algorithmen zur Berechnung einer solchen Knotenmenge werden ebenfalls behandelt.

Kapitel 4 ist schließlich den eigentlichen Anordnungsalgorithmen gewidmet. Hierbei werden verschiedene Strategien verfolgt, z.B. Bandbreitenminimierung oder die beschriebene Umordnung auf Dreiecksgestalt. Außerdem wird eine Methode vorgestellt, wie sich Probleme aus dem $\mathbb{R}^3$ auf mehrere zweidimensionale Probleme reduzieren lassen.

Zur Lösung der aus der Diskretisierung entstehenden Gleichungssysteme werden in dieser Arbeit Iterationsverfahren auf Basis der *konjugierten Gradienten* verwendet. Die Vorstellung dieser Verfahren erfolgt in Kapitel 5. Auch auf die bereits erwähnte Gauß-Seidel-Iteration wird eingegangen. Diese dient der Beschleunigung der Lösungsverfahren.

In Kapitel 6 erfolgt abschließend ein Test der verschiedenen Anordnungen bzw. Iterationsverfahren. Hierbei werden sowohl Parameter des mathematischen Modells als auch der Diskretisierung verändert, um deren Einfluß auf die Lösungsverfahren zu untersuchen.

Danksagung

Ich danke Prof. Dr. Wolfgang Hackbusch für die Bereitstellung des Themas und der technischen Mittel zur Fertigstellung dieser Arbeit. Ein besonderer Dank gilt meinen Betreuern Dr. Sabine Le Borne und Dr. Thomas Probst für ihre zahlreichen Hinweise, Anregungen und (meist) konstruktive Kritik. Ebenfalls danken möchte ich allen anderen Mitgliedern des Lehrstuhls für deren Hilfe bei kleineren Problemen und für viele erbauliche Diskussionen. Hier seien besonders Jens Burmeister, Steffen Börm, Lars Grasedyck und Kai Helms genannt. Nicht zuletzt gilt ein großer Dank meinen Eltern Doris und Hans-Joachim Kriemann, ohne deren Unterstützung mein Studium und somit diese Arbeit nicht möglich gewesen wäre.

2. Die Konvektions-Diffusions-Gleichung

Das Modellproblem in dieser Arbeit ist die Konvektions-Diffusions-Gleichung bzw. ein Randwertproblem, welches diese Gleichung beinhaltet. Die Vorstellung dieses Randwertproblemes erfolgt in Abschnitt 2.1. Anschließend wird in den Abschnitten 2.2 und 2.3 auf die Diskretisierung des Modellproblemes eingegangen, wobei die Methode der finiten Elemente verwendet wird. Ebenfalls diskutiert werden in diesem Zusammenhang reguläre Verfeinerungstechniken. Auf einen speziellen Aspekt der Diskretisierung des Konvektionsanteils wird in Abschnitt 2.4 eingegangen.

Bei der Implementierung der Diskretisierung ist die Repräsentation des zugrundeliegenden Gitters ein wesentlicher Aspekt. In Abschnitt 2.5 sollen deshalb verschiedene Gitterdarstellungen vorgestellt werden, wobei jeweils der Speicherbedarf und der Verwaltungsaufwand betrachtet werden.

Alternativ zur Konvektions-Diffusions-Gleichung bieten sich auch die *Navier-Stokes-Gleichungen* als Modellproblem an, die die Grundgleichungen der Strömungsmechanik darstellen. Zu deren numerischen Behandlung sei an dieser Stelle auf [Bor99] oder auch [GDN95] verwiesen.

2.1. Problemstellung

Betrachtet wird die zeitunabhängige *Konvektions-Diffusions-Gleichung*

$$-\varepsilon\Delta u + b\cdot\nabla u = f \quad \text{in} \;\; \Omega\subset\mathbb{R}^d \tag{2.1}$$

mit der inhomogenen Dirichlet-Randbedingung

$$u = \varphi \quad \text{auf} \;\; \Gamma = \partial\Omega. \tag{2.2}$$

Dabei bezeichnet Ω ein beschränktes Gebiet des $\mathbb{R}^d$, mit $d = 2$ oder $d = 3$. Das Konvektionsfeld ist durch $b : \mathbb{R}^d \to \mathbb{R}^d$ definiert und durch $0 < \varepsilon \ll 1$ wird die Stärke der Konvektion festgelegt. Mögliche äußere Kräfte werden durch die rechte Seite f beschrieben.

2.2. Variationsformulierung

Für das Randwertproblem (2.1), (2.2) wird nun eine Variationsaufgabe formuliert. Sei V der Raum der einmal schwach differenzierbaren Funktionen mit Nullrandbedingung auf dem Dirichletrand Γ. Multipliziert man Gleichung (2.1) mit einer Testfunktion $v \in V$ und integriert anschließend über Ω, so erhält man

$$-\varepsilon \int_\Omega v \nabla \cdot \nabla u \, dx + \int_\Omega (b \cdot \nabla u) v \, dx = \int_\Omega f v \, dx.$$

Mit Hilfe der Greenschen Formel und unter Beachtung der Tatsache, daß v auf Γ verschwindet, läßt sich diese Gleichung in

$$\varepsilon \int_\Omega \nabla v \cdot \nabla u \, dx + \int_\Omega (b \cdot \nabla u) v \, dx = \int_\Omega f v \, dx$$

umformen. Mit

$$a(u, v) = \varepsilon \int_\Omega \nabla v \cdot \nabla u \, dx + \int_\Omega (b \cdot \nabla u) v \, dx \tag{2.3}$$

und

$$f(v) = \int_\Omega f v \, dx$$

lautet die zu lösende Aufgabe schließlich:

$$\text{Finde } u \in V \text{ so, daß } a(u, w) = f(w) \text{ für alle } w \in V \text{ gilt.} \tag{2.4}$$

Die Lösung $u \in V$ von (2.4) heißt auch *schwache Lösung* des Problems (2.1), (2.2). Zur Existenz und Eindeutigkeit einer Lösung von (2.4) sei auf [Bra97] verwiesen.

2.3. Finite-Element-Diskretisierung

Die Berechnung einer Lösung von Aufgabe (2.4) scheitert i.d.R. daran, daß der Raum V eine unendliche Dimension besitzt. Aus diesem Grund ersetzt man V durch einen endlichdimensionalen Raum $V_h \subset V$ und gelangt somit zur neuen Aufgabe

$$\text{Finde } u_h \in V_h \text{ so, daß } a(u_h, w) = f(w) \text{ für alle } w \in V_h \text{ gilt.} \tag{2.5}$$

Sei $\{\varphi_1, \ldots, \varphi_n\}$ eine Basis von V_h. Mit dem Ansatz

$$u_h = \sum_{i=1}^{n} x_i \varphi_i \tag{2.6}$$

läßt sich (2.5) zu einem linearen Gleichungssystem $Ax = b$ für $x = (x_1, \ldots, x_n)^T$ umformulieren. Dabei sind die *Systemmatrix* A und die rechte Seite b wie folgt definiert:

$$A = (a_{ij})_{i,j=1}^n \quad \text{mit} \quad a_{ij} = a(\varphi_j, \varphi_i), \tag{2.7}$$

$$b = (b_i)_{i=1}^n \quad \text{mit} \quad b_i = f(\varphi_i). \tag{2.8}$$

Teilt man die Bilinearform $a(\cdot,\cdot)$ in einen *Diffusionsteil*

$$a_d(u,v) = \varepsilon \int_\Omega \nabla v \cdot \nabla u \, dx$$

und einen *Konvektionsteil*

$$a_c(u,v) = \int_\Omega (b \cdot \nabla u) v \, dx \tag{2.9}$$

auf, so ergibt sich für A eine analoge Zerlegung

$$A = A^D + A^C, \tag{2.10}$$

wobei A^D und A^C dem Diffusions- bzw. Konvektionsanteil entsprechen.

Die Konstruktion von V_h erfolgt über eine Zerlegung von $\Omega \subset \mathbb{R}^d$ in abgeschlossene Dreiecke ($d = 2$) bzw. Tetraeder ($d = 3$). Dabei wird davon ausgegangen, daß eine solche Zerlegung oder *Triangulation* existiert. Betrachtet werden weiterhin nur *zulässige* Triangulationen.

Definition 2.3.1 Sei $\mathcal{T} = \{t_1, \ldots, t_M\}$ eine Triangulation von Ω. $\mathcal{T}$ heißt *zulässig*, falls

i) $\overline{\Omega} = \bigcup_{i=1}^{M} t_i$,

ii) für $i \neq j$ ist $t_i \cap t_j$ entweder leer, ein gemeinsamer Eckpunkt oder eine gemeinsame Kante (oder ein gemeinsames Dreieck im $\mathbb{R}^3$).

Bezeichnet man mit $h(t)$ die Länge der längsten Kante von $t \in \mathcal{T}$, so läßt sich durch die Größe $h(\mathcal{T}) = \max_{t \in \mathcal{T}} h(t)$ der Triangulation $\mathcal{T}$ ein Diskretisierungsparameter $h = h(\mathcal{T})$ zuweisen, der die "Feinheit" der Zerlegung beschreibt. Anstelle von $\mathcal{T}$ wird auch $\mathcal{T}_h$ geschrieben, falls $h(t) \leq 2h$ für alle $t \in \mathcal{T}$.

Liegt eine Familie von Triangulationen $\{\mathcal{T}_h\}$ vor, so sind häufig weitere Eigenschaften wünschenswert. Sei hierzu $\rho(t)$ der Durchmesser der größten, in t enthaltenen Kugel. Die Familie $\{\mathcal{T}_h\}$ heißt *quasi-uniform*, falls es eine Konstante $\kappa > 0$ gibt, so daß

$$\kappa \leq \frac{\rho(t)}{h(t)} \quad \text{für alle } t \in \mathcal{T}_h.$$

Ersetzt man in dieser Abschätzung $h(t)$ durch h, so gelangt man zu einer weiteren Eigenschaft: $\{\mathcal{T}_h\}$ heißt *uniform*, falls es eine Konstante $\gamma > 0$ gibt, so daß

$$\gamma \leq \frac{\rho(t)}{h} \quad \text{für alle } t \in \mathcal{T}_h.$$

In der Praxis werden Triangulationen häufig konstruiert indem man eine grobe Zerlegung von Ω spezifiziert und diese anschließend *verfeinert*. Im zweidimensionalen Fall kann eine solche Verfeinerung eines Elementes $t \in \mathcal{T}$ z.B. durch das Verbinden der Kantenmittelpunkte erfolgen, wie es in Abbildung 2.3 dargestellt ist. Die so entstehenden Dreiecke, auch *Söhne* genannt, sind kongruent zum ursprünglichen Dreieck, dem *Vaterelement*. Somit ist die Anzahl von Kongruenzklassen der Elemente einer Triangulation durch die Anfangstriangulation bestimmt und insbesondere beschränkt. Solche Verfeinerungen werden auch als *reguläre Verfeinerungen* bezeichnet.

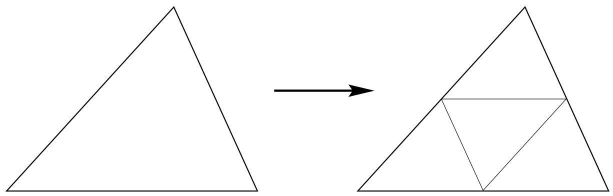

Abbildung 2.1: Reguläre Verfeinerung eines Dreiecks

Die Konstruktion einer regulären Verfeinerung im dreidimensionalen Fall ist schwieriger, da eine Zerlegung eines Tetraeders in 8 kongruente Tetraeder i.allg. nicht möglich ist. In [Bey95] ist ein Verfahren beschrieben, in welchem die Tetraeder in eine endliche Anzahl von Kongruenzklassen zerfallen. Hierbei werden zunächst, wie im zweidimensionalen Fall, die Kantenmittelpunkte jedes Dreiecks miteinander verbunden. An den Eckpunkten des Tetraeders bilden sich so 4 neue Tetraeder, die ähnlich zum Ausgangstetraeder sind. Im Zentrum des Tetraeders bleibt anschließend ein Oktaeder übrig. In Abbildung 2.3 ist diese Situation dargestellt.

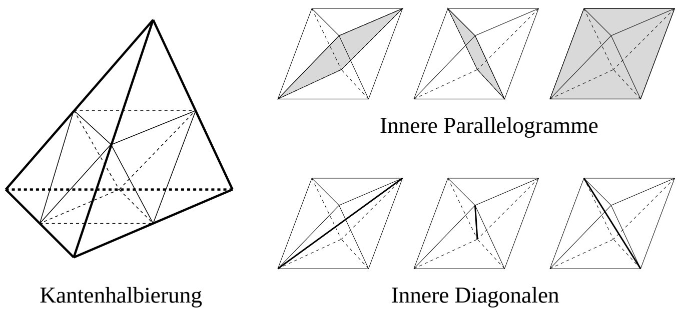

Abbildung 2.2: Reguläre Verfeinerung eines Tetraeders

Im Inneren dieses Oktaeders kreuzen sich drei Parallelogramme. Wird der Oktaeder entlang zwei dieser Parallelogramme aufgeschnitten, so entstehen weitere 4 Tetraeder, welche aber nicht mehr ähnlich zum Ausgangstetraeder sind. Allerdings besitzen alle acht Teiltetraeder das gleiche Volumen.

Das Problem bei der Verfeinerung besteht darin, die richtige Wahl für die beiden Parallelogramme zu treffen, da bei einer ungeschickten Auswahl die Anzahl der Kongruenzklassen der Tetraeder beliebig groß werden kann. Der Algorithmus in [Bey95] löst dieses Problem durch eine feste Numerierung der Eckpunkte der Teiltetraeder. Eine andere Strategie basiert auf den Längen der inneren Diagonalen des Oktaeders. Hierzu beobachtet man zunächst, daß jede Wahl zweier Parallelogramme einer inneren Diagonale des Oktaeders entspricht (siehe auch Abbildung 2.3). Wählt man in jedem Verfeinerungsschritt jeweils die Parallelogramme aus, die der kürzesten Diagonale entsprechen, so gelangt man zu einem Verfahren, welches unter geeigneten Voraussetzungen äquivalent zu der Numerierungsstrategie ist.

Der Raum V_h aus (2.5) wird im folgenden als der Raum der stetigen, stückweise linearen Funktionen über der Triangulierung $\mathcal{T}$ mit der Nebenbedingung

$$\varphi_i(v_j) = \delta_{ij} \tag{2.11}$$

gewählt, wobei v_j ein Eckpunkt eines Elementes von $\mathcal{T}$, ein sog. *Knoten* ist. Durch den stückweise linearen Ansatz sind die Funktionen in V_h durch die Werte an den Knoten bestimmt. Aufgrund dieser Eigenschaft identifiziert man auch häufig die Koeffizienten x_i aus (2.6) mit den entsprechenden Knoten v_i. Die Nebenbedinung (2.11) sorgt für einen kleinen Träger der Basisfunktionen. Dies führt dazu, daß die Systemmatrix A *schwachbesetzt* ist, d.h. nur $\mathcal{O}(n)$ Einträge ungleich Null enthält. Weiterhin erlaubt (2.11) die eindeutige Zuordnung einer Basisfunktion φ_i zu einem Knoten v_i.

2.4. Diskretisierung des Konvektionsanteils

Die im letzten Abschnitt beschriebene Finite-Element-Diskretisierung ist im Falle dominanter Konvektion ($\varepsilon \ll 1$) instabil, was sich durch heftige Oszillationen der diskreten Lösung bemerkbar macht. Die Schwierigkeiten ergeben sich bei der Diskretisierung des Konvektionsterms $b \cdot \nabla u$. Die dabei verwendete Approximation kann interpretiert werden als Annäherung durch zentrale Differenzen.

Eine Möglichkeit zur Stabilisierung der Methode bietet die sog. *Upwind-Strategie*, in welcher die Konvektionsrichtung in die Approximation des Konvektionsterms miteinbezogen wird. Anschaulich geht man dabei zu einseitigen Differenzen über. Hierzu betrachtet man in einem Knoten v_i die negative Konvektionsrichtung $-b(v_i)$. Diese schneidet ein angrenzendes Element, das *Upwind-Dreieck* im $\mathbb{R}^2$ bzw. *Upwind-Tetraeder* im $\mathbb{R}^3$. In Abbildung 2.4 ist die Situation in einem Beispiel dargestellt, dessen Notation im folgenden genutzt wird.

Die Approximation von $b \cdot \nabla u$ in v_i lautet

$$b(v_i) \cdot \nabla u(v_i) \approx \frac{u(v_i) - u(y)}{\|v_i - y\|_2} \|b(v_i)\|_2.$$

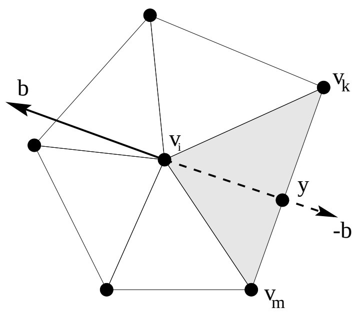

Abbildung 2.3: Upwind-Dreieck

Sei φ_i die zu v_i gehörige, stückweise lineare Basisfunktion aus Abschnitt 2.3 und bezeichne S_i deren Träger. Der Konvektionsanteil $a_c(\cdot,\cdot)$ läßt sich wie folgt annähern:

$$a_c(u, \varphi_v) \approx \frac{S_i}{3} \frac{u(v_i) - u(y)}{\|v_i - y\|_2} \|b(v_i)\|_2$$

Damit ergeben sich nur für die drei Knoten des Upwind-Dreiecks Nicht-Null-Einträge in der Matrix A^C:

$$a_c(\varphi_j, \varphi_i) \approx \begin{cases} \frac{S_i}{3\|v_i - y\|} \|v(v_i)\|_2 & , \quad i = j \\ -\frac{S_i}{3\|v_i - y\|} \varphi_j(y) \|v(v_i)\|_2 & , \quad j = k \text{ oder } j = m \\ 0 & , \quad \text{sonst} \end{cases}$$

Mit zusätzlichen Bedingungen an das zugrundeliegende Gitter folgt eine *M-Matrix-Eigenschaft* der Systemmatrix A.

Für eine vertiefende Behandlung der Upwind-Strategie, sowie zu Alternativen sei auf [Bor99] verwiesen.

2.5. Repräsentation der Gitter

Bei der Implementierung der Methode der finiten Elemente liegt ein wesentlicher Punkt in der Darstellung der Gitter bzw. der Triangulationen. Hier gibt es verschiedene Möglichkeiten, die auf unterschiedliche Aspekte hin optimiert wurden, z.B. Speicherbedarf und Verwaltungsaufwand. In diesem Abschnitt sollen einige dieser Repräsentationen kurz diskutiert werden.

Ausgehend von dem Prinzip der finiten Elemente bietet sich eine *elementorientierte* Speicherung eines Gitters an. Dabei wird jedes Element der Triangulation durch ein einzelnes Objekt repräsentiert. Die Speicherung der Geometrieinformationen eines Elements erfolgt ebenfalls durch das Anlegen eines Objektes für jeden Knoten. Eine Gitterdarstellung, die vergleichsweise wenig Speicher benötigt und mit den beschriebenen Strukturen auskommt, wäre etwa

- **Element** $\rightarrow$ Knoten, Nachbarn, Söhne, Vater
- **Knoten** $\rightarrow$ Koordinaten.

Hierbei wurde schon den Anforderungen an Verfeinerungstechniken Rechnung getragen, indem sowohl eine *horizontale* Kommunikation zwischen Nachbarelementen in einem Gitter, als auch eine *vertikale* Kommunikation zwischen einem Element und seinen Söhnen bzw. seinem Vater realisiert ist.

Die horizonale Kommunikation ist z.B. hilfreich, wenn es um den Abgleich von gemeinsam gespeicherten Objekten geht. Wurde etwa von einem Element während der regulären Verfeinerung aus Abschnitt 2.3 ein Knoten an einem Kantenmittelpunkt angelegt, so kann ein Nachbarelement Zugriff auf diesen Knoten über eine Anfrage an das erste Element erlangen. Ein weiteres Beispiel für horizontale Kommunikation findet sich in adaptiven Verfeinerungen (siehe z.B. [Bey98]). Um eine zulässige Triangulation zu erreichen, erfolgt dort eine Anpassung der Verfeinerungsstrategie eines Elementes in Abhängigkeit von seinen Nachbarelementen. Ebenfalls in adaptiven Verfeinerungen ergibt sich eine Anwendung für die vertikale Kommunikation, falls etwa bereits angelegte Söhne eines Elementes wieder entfernt werden müssen, um das Gitter zu *vergröbern.*

Eine Erweiterung der beschriebenen Gitterdarstellung gelingt durch die Repräsentation von Kanten. Ein Vorteil bei der Verwendung von Kanten ist die Kommunikation von einem Knoten zu seinen Nachbarknoten. Außerdem kann die Verwaltung von Kantenmittelpunkten oder auch der Eckknoten eines Elementes auf die Kanten übertragen werden. Haben dann zwei benachbarte Elemente Zugriff auf das gleiche Kantenobjekt, so steht beiden Elementen ohne entsprechende Kommunikation auch der Kantenmittelpunkt zur Verfügung.

Auf diesen Grundobjekten basierende Gitterdarstellungen sind sehr verbreitet, da sie einen guten Kompromiß zwischen Speicherbedarf und Verwaltungsaufwand darstellen. Als Beispiel seien etwa die Programmpakete *UG* (siehe [Bas96]) oder *AGM3D* (siehe [Bey98]) genannt. Bei UG wird dabei die folgende Gitterrepräsentation verwendet:

- **Element** $\rightarrow$ Knoten, Nachbarn, Söhne, Vater
- **Kante** $\rightarrow$ Knoten (2)
- **Knoten** $\rightarrow$ Koordinaten, Kanten ($< \infty$)

Eine Kante dient hierbei lediglich der Kommunikation zwischen den beiden Endknoten. Weiterhin hat ein Element keinen direkten Zugriff auf die Kantenobjekte. Im AGM3D-Paket wird dagegen die Verwaltung der Kantenmittelpunkte in den Kanten selbst vorgenommen und die Elemente können über die Kanten darauf zugreifen. Die Gitterdarstellung bei AGM3D lautet:

- **Element** → Knoten, Kanten, Söhne, Nachbarn
- **Kante** → Knoten, Mittelpunkt
- **Knoten** → Koordinaten, Kanten ($< \infty$)

In allen bisher beschriebenen Darstellungsarten hatten die Elemente direkt Zugriff auf ihre Geometrieinformationen, da die Knoten in den Elementen gespeichert wurden. Auch wurden Verweise zu Nachbarelementen in den Elementen selbst verwaltet. Ein alternativer, hierarchischer Ansatz verwendet eine eindimensionale Abstufung. Die dabei repräsentierten Strukturen sind Elemente, Flächen, Kanten und Knoten, wobei im zweidimensionalen Fall die Flächen entfallen. Jede Struktur verwaltet dabei solche Objekte, die jeweils eine Dimension kleiner sind. Die einzige Ausnahme bilden Verweise auf Nachbarelemente. Die Kommunikation wird hierbei über die Strukturen ausgeführt, die zwei Elemente trennen, d.h. Kanten im $\mathbb{R}^2$ und Flächen im $\mathbb{R}^3$. Die Gitterdarstellung im $\mathbb{R}^3$ lautet:

- **Element** → Fläche, Söhne, Vater
- **Fläche** → Kanten, Nachbarelemente (2), Söhne
- **Kante** → Knoten (2), Mittelpunkt, Söhne
- **Knoten** → Koordinaten

Im zweidimensionalen Fall ist die Darstellung dagegen wie folgt definiert:

- **Element** → Kanten, Söhne, Vater
- **Kante** → Knoten (2), Mittelpunkt, Nachbarelemente (2), Söhne
- **Knoten** → Koordinaten

Die Sohn-Felder bei Flächen und Kanten dienen der Speicherung von Flächen- bzw. Kanten-Objekten, die bei der Verfeinerung angelegt wurden, etwa den beiden Teilkanten bei der regulären Verfeinerung aus Abschnitt 2.3.

Die eindimensionale Abstufung bietet bei der Verwaltung einer Verfeinerung den Vorteil, daß neu angelegte Objekte allen Strukturen zur Verfügung stehen, ohne das eine Kommunikation stattfinden muß. Ein Beispiel soll dies verdeutlichen: Zwei Nachbarelemente t_1 und t_2 besitzen Zugriff auf die gleiche Fläche. Wird diese von Element t_1 bei der Verfeinerung in entsprechende Teilflächen zerlegt, so stehen auch Element t_2 diese Teilflächen bei der Konstruktion seiner Söhne zur Verfügung, ohne daß t_2 hierzu eine Kommunikation mit t_1 durchführen muß.

Ein wesentlicher Nachteil dieser Art der Gitterdarstellung ist allerdings der vergleichsweise große Speicherbedarf, speziell im dreidimensionalen Fall. Ein Grund hierfür liegt darin, daß die Gesamtzahl der zu speichernden Flächen wesentlich größer ist als die Anzahl der Elemente. Demgegenüber können Informationen die einzelne Flächen betreffen direkt in diesen abgelegt werden. Ein Beispiel hierfür ist der in Abschnitt 4.3.3 vorgestellte Algorithmus zur Konstruktion von Oberflächen, die über Dreiecksflächen in einem Tetraedergitter aufgebaut werden.

3. Graphentheoretische Grundlagen

Die Grundlage der meisten in dieser Arbeit genutzten Algorithmen und Techniken bilden Graphenalgorithmen. In diesem Kapitel werden deshalb die graphentheoretischen Begriffe definiert, die zur Formulierung dieser Verfahren nötig sind. Zunächst erfolgt in Abschnitt 3.1 eine Vorstellung der verwendeten Graphen.

In Abschnitt 3.2 geht es anschließend um ein Problem aus der Graphentheorie, das sog. *Feedback-Vertex-Set-Problem*, auf dessen Lösung viele der Anordnungstechniken in Kapitel 4 aufbauen. Um diese Lösungen auch praktisch nutzen zu können, werden in den Abschnitten 3.2.1 und 3.2.2 verschiedene Algorithmen für deren Berechnung vorgestellt.

3.1. Graphen und Digraphen

Definition 3.1.1 (Graph) Es sei V eine Menge und $E \subseteq V \times V$ eine symmetrische Relation über V. Man bezeichnet das Paar $G = (V, E)$ als *(ungerichteten) Graphen.*

Die Menge V heißt *Knotenmenge* und die Elemente aus V *Knoten* des Graphen G. E heißt *Kantenmenge* und die Elemente aus E *Kanten.*

Aufgrund der Symmetrie von E wird nicht zwischen dem Paar $(u, v) \in E$ bzw. $(v, u) \in E$ unterschieden, d.h. diese Kanten werden als gleich betrachtet.

Sind $u, v \in V(G)$ und $e \in E(G)$ mit $e = (u, v)$, so heißen u und v *Endknoten* der Kante e; man sagt auch, die Kante e *inzidiert* mit den Knoten u und v oder u und v sind durch die Kante e verbunden. Im Falle $u = v$ heißt e *Schlinge* oder *Loop.* Verschiedene Knoten, die durch eine Kante verbunden sind, heißen *benachbart* oder *adjazent.* Inzidieren zwei verschiedene Kanten mit einem gemeinsamen Knoten, so nennt man die Kanten *inzident.*

Man beachte, daß aufgrund der Definition eines Graphen zwei Knoten höchstens durch eine Kante verbunden sein können. Somit sind *Mehrfachkanten*, d.h. verschiedene Kanten mit gleichen Endknoten, ausgeschlossen.

Neben den ungerichteten Graphen spielen in der Graphentheorie die gerichteten Graphen eine große Rolle.

Definition 3.1.2 (Digraph) Es seien V eine Menge und $E \subseteq V \times V$ eine Relation über V. Das Paar $G = (V, E)$ wird als *Digraph* oder als *gerichteter Graph* G bezeichnet.

Die Elemente aus $E = E(G)$ werden auch als *gerichtete* bzw. *orientierte Kanten* bezeichnet. Sind $u, v \in V$ und ist $e = (u, v) \in E$, so heißt u *Anfangsknoten* und v *Endknoten* der Kante e. Die Kante e wird auch als *ausgehende Kante* von u bzw. als *eingehende Kante* von v bezeichnet. Desweiteren heißt u *Vorgänger* von v und v *Nachfolger* von u.

Bemerkung 3.1.3 In dieser Arbeit werden nur Graphen benutzt, deren Knoten- und Kantenmenge endlich ist. Man spricht in diesem Fall auch von einem *endlichen Graphen.*

Definition 3.1.4 (Knotengrad) Ist $G = (V, E)$ ein Graph und v ein Knoten, so wird mit

$$d(v, G) = d(v)$$

die Anzahl der inzidierenden Kanten bezeichnet, wobei Schlingen doppelt gezählt werden. Man nennt $d(v)$ auch *Grad* des Knoten v.

Ist G ein Digraph, so bezeichnen

$$d_a(v, G) = d_a(v) \quad \text{bzw.} \quad d_e(v, G) = d_e(v)$$

die Anzahl der ausgehenden bzw. eingehenden Kanten von v. Dabei heißt $d_a(v)$ *Ausgangsgrad* und $d_e(v)$ *Eingangsgrad* von v.

Definition 3.1.5 (Graphenhomomorphismus) Es seien $G = (V, E)$ ein $G' = (V', E')$ zwei Graphen und $f : V \to V'$ sowie $F : E \to E'$ Abbildungen. Das Paar (f, F) heißt *Graphenhomomorphismus*, wenn für alle $e \in E$ gilt:

$$e = (u, v) \Rightarrow F(e) = (f(u), f(v)).$$

Sind die Abbildungen f und F bijektiv, so heißt (f, F) *Graphenisomorphismus* und die Graphen G und G' zueinander *isomorph.*

Definition 3.1.6 (Teilgraph) Ein Graph $G' = (V', E')$ heißt *Teilgraph* eines Graphen $G = (V, E)$, in Zeichen $G' \subseteq G$, falls $V' \subseteq V$, $E' \subseteq E \cap (V' \times V')$.

Sei $V' \subseteq V$. Der Teilgraph von G, der aus V' und allen Kanten von G besteht, die nur mit Knoten aus V' inzidieren, heißt der von V' *induzierte Teilgraph* $G(V')$ von G. Für $V'' \subseteq V$ sei die Reduktion $G - V''$ von G um V'' definiert durch $G - V'' = G(V - V'')$.

Sei $E' \subseteq E$. Der Teilgraph von G, der aus E' und allen Knoten aus G besteht, die mit Kanten aus E' inzidieren, heißt der von E' *erzeugte Teilgraph* von G, in Zeichen $G(E')$. Analog zur Reduktion um eine Knotenmenge sei die Reduktion um eine Kantenmenge $E'' \subseteq E$ definiert als $G - E'' = G(E - E'')$.

Bemerkung 3.1.7 Die Begriffe und Operationen aus den Definitionen 3.1.5 und 3.1.6 lassen sich analog für Digraphen definieren.

Ein für die Komplexitätsbetrachtung späterer Algorithmen wichtiger Begriff ist Gegenstand der folgenden Definition.

Definition 3.1.8 Sei $G = (V, E)$ ein (Di-) Graph. Dann bezeichnet $|G| = |V| + |E|$ die *Größe* des Graphen G.

Bemerkung 3.1.9 In den folgenden Kapiteln wird auf die Unterscheidung zwischen Graph und Digraph verzichtet, sofern diese aus dem Kontext hervorgeht oder nicht wesentlich für das jeweilige Thema ist.

3.1.1. Planare Graphen

In diesem Abschnitt soll eine Klassen von Graphen vorgestellt werden, deren Eigenschaften wesentlich für die Anwendbarkeit späterer Algorithmen sind.

Definition 3.1.10 Sei $k : [0,1] \to \mathbb{R}^2$ eine stetige Abbildung und $k|_{(0,1)}$ injektiv. Falls $k(0) \neq k(1)$ heißt k *Jordanbogen*. Im Falle von $k(0) = k(1)$ spricht man von einer *Jordankurve*.

Definition 3.1.11 (Ebener Graph) Ein Graph $G = (V, E)$ heißt *ebener Graph*, falls folgende Bedingungen erfüllt sind:

i) $V \subset \mathbb{R}^2$,

ii) Jeder Kante $e_i \in E$ läßt sich entweder eine Jordankurve oder ein Jordanbogen k_i zuordnen, mit: $(k_i(0), k_i(1)) = e$ und $k_i(t) \notin V$ für $0 < t < 1$.

iii) Für je zwei verschiedene Kanten $e_i, e_j \in E$ und für alle $s, t \in (0,1)$ gilt: $k_i(t) \neq k_j(s)$.

Definition 3.1.12 (Planarer Graph) Sei G ein Graph. Existiert ein ebener Graph G' derart, daß G und G' isomorph sind, so bezeichnet man G als *planaren Graphen*.

Ein planarer Graph kann also in der Ebene (dem $\mathbb{R}^2$) auf eine Art und Weise neu gezeichnet werden, so daß sich jedes Paar von Kanten höchstens an den Endknoten trifft.

3.1.2. Graph einer Matrix

Der Graph einer Matrix gibt deren Besetztheitsstruktur wieder. Dabei werden die Indizes als Knoten des Graph betrachtet und die Einträge der Matrix als Kanten. Die genaue Definition lautet:

Definition 3.1.13 (Matrixgraph) Sei I eine endliche Indexmenge und $A = (a_{ij})_{i,j \in I} \in K^{I \times I}$. Der *Matrixgraph* $G(A) := (V, E)$ ist definiert durch

$$V := I \quad \text{und} \quad E := \{(i,j) \in I \times I \,:\, i \neq j \text{ und } |a_{ij}| > 0\}.$$

Besitzt A eine symmetrische Besetztheitsstruktur, so wird $G(A)$ als ungerichteter Graph aufgefaßt.

An Stelle der gesamten Kantenmenge ist man häufig an einer Teilmenge interessiert, die bestimmten Matrixeinträgen entspricht, z.B. nur an “starken” Einträgen, die über ein $\kappa > 0$ bestimmt werden:

$$E' := \{(i,j) \in I \times I \,:\, i \neq j \text{ und } |a_{ij}| > \kappa\}.$$

Betrachtet man z.B. den Matrixgraphen der Systemmatrix aus Abschnitt 2.3, so läßt sich so die Kantenmenge auf die Kanten beschränken, die der Konvektionsrichtung folgen.

3.1.3. Gewichtete Graphen

Eine Erweiterung der in Definition 3.1.1 und 3.1.2 vorgestellten Graphen stellen die gewichteten Graphen dar:

Definition 3.1.14 (Gewichteter Graph) Ist $G = (V, E)$ ein Graph und sind $w_V : V \to \mathbb{R}_+$ und $w_E : E \to \mathbb{R}_+$ Abbildungen, so bezeichnet man $G' = (V, E, w_V, w_E)$ als *gewichteten Graph* und w_V und w_E als *Gewichtsfunktionen.*

Liegt lediglich die Gewichtsfunktion über den Knoten vor, so spricht man von einem *knotengewichteten Graphen* $G_V = (V, E, w_V)$. Analog wird der Graph $G_E = (V, E, w_E)$ als *kantengewichteter Graph* bezeichnet.

Ein wichtiges Beispiel für gewichtete Graphen ergibt sich als Erweiterung der in Abschnitt 3.1.2 definierten Matrixgraphen:

Definition 3.1.15 Sei I eine endliche Indexmenge und $A = (a_{ij})_{i,j \in I} \in K^{I \times I}$. Der Graph $G_w(A) = (V, E, w_V, w_E)$ mit $V = V(G(A))$ und $E = E(G(A))$ und den Gewichtsfunktionen $w_V : V \to \mathbb{R}_+$ und $w_E : E \to \mathbb{R}_+$ definiert durch

$$w_V(v_i) = |a_{ii}| \quad \text{bzw.} \quad w_E((v_i, v_j)) = |a_{ij}|$$

wird als *gewichteter Matrixgraph* von A bezeichnet.

3.1.4. Implementierung

Für die Komplexitätsbetrachtung der späteren Algorithmen (speziell derer in Abschnitt 3.2) ist die verwendete Implementierung eines Graphen wichtig. Besondere Beachtung gilt dabei den elementaren Operationen wie z.B. dem Einfügen und Löschen eines Knotens bzw. einer Kante im Graphen. An dieser Stelle soll deshalb näher auf diesen Sachverhalt eingegangen werden.

Eine mögliche Grundlage für die Implementierung bildet die *Adjazenzmatrix* eines Graphen.

Definition 3.1.16 (Adjazenzmatrix) Sei $G = (V, E)$ ein Graph mit $V = \{v_1, \ldots, v_n\}$. Die *Adjazenzmatrix* $A = A(G) \in \mathbb{B}^{n \times n}$ sei definiert durch:

$$a_{ij} = \begin{cases} 1 & , \quad (v_i, v_j) \in E \\ 0 & , \quad \text{sonst} \end{cases}$$

Die Adjazenzmatrix stellt somit das Gegenstück zu den in Abschnitt 3.1.2 definierten Matrixgraphen dar und gibt an, welche Knoten des Graphen über eine Kante miteinander verbunden sind. Ein Eintrag auf der Hauptdiagonalen entspricht hierbei einer Schlinge.

Die in den späteren Anwendungen vorkommenden Graphen verfügen über eine Eigenschaft, die man sich bei der Speicherung zunutze machen kann. Diese Eigenschaft betrifft die Anzahl der adjazenten Kanten eines Knotens, welche sich durch eine von der Größe des Graphen unabhängige Konstante abschätzen läßt. Die Ursache hierfür liegt in der Schwachbesetztheit der Matrizen, aus denen die Graphen gebildet werden (siehe Abschnitt 2.3). Diese Schwachbesetztheit überträgt sich auf die Adjazenzmatrix.

In der Praxis gibt es verschiedene Techniken für das Abspeichern schwachbesetzter Matrizen, wie etwa *Compressed-Row/Column-Storage*, *Diagonal-Storage* oder *Skyline-Storage*, die auf unterschiedliche Anwendungen hin optimiert wurden. Ein Problem bei diesen Darstellungen ist die Modifikation einer bestehenden Matrix und somit des Graphen, etwa durch das Hinzufügen oder das Entfernen eines Knoten, da hierzu i.d.R. die gesamte Matrix umkopiert werden muß. Auch ist in den, in dieser Arbeit auftretenden Graphen nicht immer eine Zuweisung eines Knotennamens $n \in \mathbb{N}$ möglich bzw. sinnvoll.

Aus diesen Gründen wurde auf eine Matrix-orientierte Darstellung verzichtet und statt dessen die Kantenrelation des Graphen über Verkettungen der Knoten realisiert. Hierbei besteht der Graph aus einer doppelt verketten Liste von Knoten. Jeder einzelne Knoten enthält weiterhin jeweils eine Liste seiner ausgehenden und eingehenden Kanten. Jede dieser Kanten besitzt eine Referenz[1] auf ihren Anfangs- und Endknoten. In Abbildung 3.1.4 ist die beschriebene Struktur noch einmal graphisch dargestellt.

[1] Es wird an dieser Stelle von einer Programmiersprache ausgegangen, die ein zu Zeigern oder Referenzen auf Daten äquivalentes Konzept implementiert, etwa Pascal oder C.

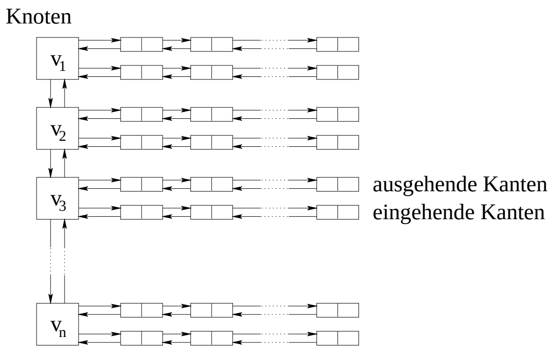

Abbildung 3.1: Darstellung eines Graphen über verkettete Listen

Durch die Verwendung von Listen liegt der Aufwand für das Hinzufügen eines Knotens bzw. einer Kante in $\mathcal{O}(1)$. Das Löschen eines Knotens hat ebenfalls einen $\mathcal{O}(1)$-Aufwand, falls das entsprechende Listenelement vorliegt. Anderenfalls ist zunächst eine Suche mit Aufwand $\mathcal{O}(n)$ (bei binärer Suche $\mathcal{O}(\log n)$) nötig. Hierbei bezeichnet n die Anzahl der Elemente der Liste.

Ein weiterer Vorteil der beschriebenen Graphenrepräsentation ist der Zugriff auf den sogenannten *transponierten Graphen*, in dem die Kantenrelation invertiert ist. Dies ist möglich, da auch die eingehenden Kanten für jeden Knoten abgespeichert werden.

3.1.5. Pfade, Zykel und Zusammenhang

Definition 3.1.17 Sei $G = (V, E)$ ein Graph, $k \in \mathbb{N}$ und $e_1, \ldots, e_k \in E$. Falls $e_i = (v_{i-1}, v_i)$ für $i = 1, \ldots, k$, so heißt $p = (e_1 \ldots, e_k)$ *Kantenfolge* oder *Pfad* von v_0 nach v_k der Länge k. Für Pfade wird folgende Schreibweise verwendet:

$$p = (e_1, \ldots, e_k) = (v_0, \ldots, v_k) = (v_0, e_1, v_1, \ldots, e_k, a_k) = v_0 v_1 \ldots v_k.$$

Der Knoten v_0 heißt *Anfangsknoten*, der Knoten v_k *Endknoten* von p. Die Länge k von p wird auch mit $|p|$ bezeichnet. Der Pfad p heißt *geschlossen*, falls $v_0 = v_p$ und *offen*, falls $v_0 \neq v_p$.

Sind in einem Pfad alle Kanten paarweise verschieden, so spricht man von einem *Kantenzug*. Sind zusätzlich alle Knoten paarweise verschieden, so bezeichnet man den Kantenzug als *Weg*. Ein geschlossener Kantenzug $\zeta = v_0 \ldots v_k$ mit $k > 0$, heißt *Kreis* oder *Zykel*, falls $v_0 \ldots, v_{k-1}$ einen Weg bilden. Enthält ein Graph keine Zykel, so nennt man ihn auch *azyklisch*.

Definition 3.1.18 Sei $G = (V, E)$ ein Graph. Zwei Knoten $v, u \in V$ heißen *zusammenhängend*, falls ein Weg von v nach u existiert. Dies definiert auf der Knotenmenge eine Äquivalenzrelation. Jeder von einer Äquivalenzklasse induzierte Teilgraph heißt *Zusammenhangskomponente* oder *Komponente* von G. Besteht G nur aus einer einzigen Zusammenhangskomponente, so bezeichnet man den Graphen als *zusammenhängend*.

Definition 3.1.19 Sei $G = (V, E)$ ein Digraph. Analog zu Definition 3.1.17 seien *orientierte Kantenfolgen*

$$p = (v_0, e_1, v_1, \ldots, e_k, v_k) = (v_0, \ldots, v_k) = v_0 \ldots v_k$$

von v_0 nach v_k der Länge $k = |p|$ mit $v_i \in V$, $e_i \in E$ und $e_i = (v_{i-1}, v_i)$, *orientierte Kantenzüge*, *orientierte Wege*, *offene* und *geschlossene orientierte Kantenfolgen* und *orientierte Kreise* definiert.

Seien $u, v \in V$. Existiert ein Weg von u nach v, so heißt v von u aus *erreichbar*. Ist sowohl v von u aus erreichbar, als auch u von v aus, so bezeichnet man u und v als *stark zusammenhängend*. Wie im Fall der ungerichteten Graphen definiert der starke Zusammenhang auf der Knotenmenge eine Äquivalenzrelation. Die durch die Äquivalenzklassen induzierten Teilgraphen heißen *starke Zusammenhangskomponenten* von G.

Definition 3.1.20 Seien $G = (V, E)$ ein Graph und $\bigcup_{i=1}^{n} G_i$ eine disjunkte Zerlegung von G. Der Graph $G_Q = (V_Q, E_Q)$ mit

$$\begin{aligned} V_Q &= \{G_i \,:\, i = 1, \ldots, n\} \quad \text{und} \\ E_Q &= \{(G_i, G_j) \,:\, \exists\, v_i \in V(G_i), v_j \in V(G_j) \text{ mit } (v_i, v_j) \in E\} \end{aligned}$$

heißt *Quotientengraph* von G.

Ein Beispiel für einen Quotientengraph ist der Graph der starken Zusammenhangskomponenten. Da starke Zusammenhangskomponenten maximal sind, d.h. nicht Teilgraph einer anderen Komponente, ist dieser Quotientengraph azyklisch.

3.2. Das Feedback Vertex Set Problem

In diesem Abschnitt soll das Problem definiert werden, welches in vielen der späteren Anordnungsalgorithmen zu lösen sein wird.

Definition 3.2.1 (Feedback Vertex Set) Sei $G = (V, E)$ ein Graph. Sei $F \subseteq V$. Ist $G - F$ azyklisch, so bezeichnet man F als *Feedback Vertex Set* oder kurz *FVS* von G. Die Knoten in F heißen *Feedbacknoten*. F wird *minimales Feedback Vertex Set* genannt, falls F ein FVS von G ist und für alle anderen FVSs F' von G gilt: $|F| \leq |F'|$.

Bemerkung 3.2.2 Äquivalent zu "$G - F$ ist azyklisch" in Definition 3.2.1 ist: F enthält aus jedem Zykel in G mindestens einen Knoten.

Das klassische FVS-Problem besteht darin, für einen gegebenen Graphen G und ein $k \in \mathbb{N}$ zu bestimmen, ob G ein FVS der Kardinalität k besitzt. Alternativ hierzu und für die Anwendung in dieser Arbeit eher geeignet ist das Problem, für einen gegebenen Graphen ein minimales FVS aufzufinden. Dieses ist i.d.R. nicht eindeutig bestimmt, z.B. bei einem Graphen, der aus einem Zykel mit Länge ≥ 2 besteht.

Die Hauptschwierigkeit bei der Lösung des FVS-Problems ist dessen **NP**-Vollständigkeit (siehe [GJ79]), somit existiert unter der Annahme $\mathbf{P} \neq \mathbf{NP}$ kein effizienter, also in Polynomzeit arbeitender Lösungsalgorithmus. Allerdings lassen sich für spezielle Klassen von Graphen Algorithmen angeben, die das FVS-Problem effizient lösen.

Falls ein solcher Lösungsalgorithmus nicht vorliegt, genügt in der Regel auch ein sogenannter *approximativer Algorithmus.* Darunter versteht man ein Verfahren, welches ein FVS berechnet, das höchstens durch einen konstanten Faktor von der Kardinalität eines minimalen FVS abweicht. So ermittelt der in Abschnitt 3.2.2 vorgestellte Algorithmus ein FVS, welches höchstens doppelt so groß wie ein minimales FVS ist.

Für eine andere Klasse von Algorithmen, die sogenannten *heuristischen* Verfahren, sind solche Abschätzungen nicht möglich. Ihre Anwendung ist nur durch die guten Resultate begründet, die mit ihnen erzielt werden. Ein Beispiel für einen solchen Algorithmus wird in Abschnitt 3.2.1 vorgestellt.

3.2.1. Ein heuristischer FVS-Algorithmus

Der in diesem Abschnitt beschriebene Algorithmus geht zurück auf ein in [LSW88] beschriebenes Verfahren, welches das FVS-Problem für eine spezielle Klasse von Graphen in Polynomzeit löst. Dabei wird der Graph verschiedenen Transformationen unterworfen, um die Feedbackknoten zu bestimmen. Diese Transformationen und die sich daraus ableitende Klasse von Graphen sollen in den beiden nächsten Definitionen vorgestellt werden.

Definition 3.2.3 Sei $G = (V, E)$ ein Graph und sei $v \in V$. Seien die Graphentransformationen $t_1, \ldots, t_5$ wie folgt definiert:

- t_1 – Ist v ein Knoten mit einer Schlinge, dann setze $G = G - \{v\}$.
- t_2 – Ist v ein Knoten mit $d_a(v) = 0$, dann setze $G = G - \{v\}$.
- t_3 – Ist v ein Knoten mit $d_e(v) = 0$, dann setze $G = G - \{v\}$.
- t_4 – Ist v ein Knoten mit genau einem Nachfolger $u \neq v$, dann füge Kanten zwischen allen Vorgängern von v und u zu E hinzu und setze $G = G - \{v\}$.
- t_5 – Ist v ein Knoten mit genau einem Vorgänger $u \neq v$, dann füge Kanten zwischen allen Nachfolgern von v und u zu E hinzu und setze $G = G - \{v\}$.

Bezeichne $t(G, v)$ den Graphen, der sich nach Anwendung von Transformation t auf G und v ergibt. Eine Transformation t heißt auf einen Knoten $v \in V$ *anwendbar*, falls $t(G, v) \neq G$.

Die Transformationen t_4 und t_5 sind in Abbildung 3.2 an einem Beispiel illustriert.

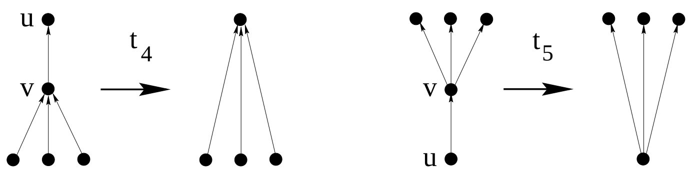

Abbildung 3.2: Transformationen t_4 und t_5

Definition 3.2.4 Sei G ein Graph. Eine *T-Sequenz* von G ist eine Folge $(T_0, v_0), \ldots, (T_k, v_k)$ von Transformationen und Knoten mit $v_i \in V(G_i)$ und $T_i \in \{t_1, \ldots, t_5\}$, so daß T_i auf v_i anwendbar ist. Dabei sei $G_0 = G$ und $G_{i+1} = T_i(G_i, v_i)$. Eine T-Sequenz heiße *vollständig*, falls $|V(G_k)| = 0$. Besitzt G eine vollständige T-Sequenz, so bezeichnet man G als *zwei-Wege-reduzierbar* bzw. als *zwei-Wege-reduzierbaren Graphen*.

Aufbauend auf den Transformationen ist in [LSW88] ein Algorithmus vorgestellt worden, der für einen gegebenen zwei-Wege-reduzierbaren Graphen ein minimales FVS berechnet. Dazu wird eine vollständige T-Sequenz aufgebaut, indem sukzessive für alle Knoten des Graphen die jeweils anwendbaren Transformationen ausgeführt werden. Das FVS wird von allen Knoten v_i der T-Sequenz mit $T_i = t_1$ gebildet. Das genaue Verfahren ist in Algorithmus 3.1 dargestellt. Wie in [LSW88] bemerkt wird, arbeitet dieses Verfahren unabhängig von der Reihenfolge der Knoten.

```
Eingabe: Graph G ohne Mehrfachkanten
i := 0; G_0 := G; F := ∅;
while V(G_i) ≠ ∅ do
    if "keine Transformation auf Knoten anwendbar" then
        Abbruch; { G ist nicht zwei-Wege-reduzierbar }
    else
        wähle v ∈ V und t ∈ {t_1, ..., t_5}, so daß t auf v anwendbar;
    endif;
    if t = t_1 then F := F ∪ {v}
    G_{i+1} = t(G_i, v); i := i + 1;
endwhile;
FVS = F;
```

Algorithmus 3.1: FVS-Algorithmus für zwei-Wege-reduzierbare Graphen

Die Korrektheit und Vollständigkeit des Verfahrens beruhen auf den folgenden beiden Sätzen, die in [LSW88, Abschnitt 5] bewiesen werden.

Satz 3.2.5 Sei G ein Graph. Falls G zwei-Wege-reduzierbar und T eine vollständige T-Sequenz von G ist, so bildet $\{w : (w, t_1) \in T\}$ ein minimales FVS von G.

Satz 3.2.6 Sei $G = (V, E)$ ein Graph und $v \in V$. Falls G zwei-Wege-reduzierbar und die Transformation $t \in \{t_1, \ldots, t_5\}$ auf v anwendbar ist, existiert eine vollständige T-Sequenz von G mit (t, v) als ihrem ersten Element.

Um Algorithmus 3.1 auch auf allgemeine Graphen anwenden zu können, muß der Abbruch im Falle eines nicht-zwei-Wege-reduzierbaren Graphen durch eine geeignete, zusätzliche Transformation ersetzt werden.

Definition 3.2.7 Sei $G = (V, E)$ ein Graph und sei $v \in V$. Sei die Transformation t_6 definiert durch

- t_6 – Sei v ein Knoten mit $t_1 \ldots t_5$ nicht auf v anwendbar und $d(v) = \max_{u \in V} d(u)$. Setze $G = G - \{v\}$.

Eine *$\hat{T}$-Sequenz* von G ist eine Folge $(T_0, v_0), \ldots, (T_k, v_k)$ mit $v_i \in V(G_i)$, $T_i \in \{t_1, \ldots, t_6\}$, T_i auf v_i anwendbar, $G_0 = G$ und $G_{i+1} = T_i(G_i, v_i)$ für alle $i \in \{0, \ldots, k\}$. Eine $\hat{\text{T}}$-Sequenz heiße *vollständig*, falls $|V(G_k)| = 0$.

Transformation t_6 setzt eine einfache Heuristik für das FVS-Problem um: Entferne sukzessive die Knoten mit maximalen Grad bis der Graph azyklisch ist. Die Begründung für dieses Vorgehen bildet Bemerkung 3.2.2 und die Annahme, daß der Grad eines Knotens mit der Anzahl der Zykel, die diesen Knoten enthalten, korreliert.

Mit dem folgenden Lemma folgt die Korrektheit und Vollständigkeit von Algorithmus 3.2.

Lemma 3.2.8 Sei $G = (V, E)$ ein Graph.

i) Algorithmus 3.2 terminiert und berechnet eine vollständige $\hat{\text{T}}$-Sequenz $T = (T_1, v_1), \ldots, (T_k, v_k)$ von G.

ii) Sei F das durch Algorithmus 3.2 berechnete FVS von G. Dann gilt: $G - F$ ist azyklisch.

Beweis:

zu i) Algorithmus 3.2 terminiert nach $|V(G_0)|$ Schritten, da in jedem Schritt ein Knoten aus dem Graphen entfernt wird. Somit wird auch eine vollständige $\hat{\text{T}}$-Sequenz berechnet.

zu ii) Annahme: $G - F$ ist nicht azyklisch. Dann existiert mindestens ein Zykel ζ in $G - F$. Sei $v \in V(\zeta)$ und somit $v \notin F$. Es gilt: $\exists\, (t, v) \in T : t \neq t_1 \wedge t \neq t_6$. Da $v \in V(\zeta)$

und somit mindestens einen Vorgänger und einen Nachfolger besitzt, folgt: $\exists\,(t,v) \in T\,:\, t = t_4 \vee t = t_5$. Betrachte $(T_i, u_1) \in T$ mit $T_i \in \{t_4, t_5\}$, $u_1 \in V(\zeta)$ und $V(G_i) \cap V(\zeta) = \{u_1, u_2\}$. Nach Definition von t_4 und t_5 gilt: u_1u_2 ist Zykel in G_i und $(u_2, u_2) \in E(G_{i+1})$. Dann ist aber nur noch t_1 auf u_2 anwendbar. Es ergibt sich ein Widerspruch und somit ist $G - F$ azyklisch.

□

Eingabe: Digraph $G = (V, E)$
$i := 0$; $G_0 := G$; $F := \emptyset$; $F' = \emptyset$;
while $V(G_i) \neq \emptyset$ **do**
  if "keine Transformation auf Knoten anwendbar" **then**
    $v := u$ mit $u \in \{v \in V\,:\, d(v) \geq d(w)$ für alle $w \in V\}$;
    $t := t_6$; $F' := F' \cup \{v\}$;
  else
    wähle $v \in V$ und $t \in \{t_1, \ldots, t_5\}$, so daß t auf v anwendbar;
  endif;
  if $t = t_1$ **then** $F := F \cup \{v\}$;
  $G_{i+1} = t(G_i, v)$; $i := i + 1$;
endwhile;
FVS $= F \cup F'$;

Algorithmus 3.2: FVS-Algorithmus für Digraphen

Wie aus Satz 3.2.5 folgt, wird in Algorithmus 3.2 ein minimales FVS für einen gegebenen Graphen berechnet, falls die Transformation t_6 nicht angewendet wird.

3.2.1.1. Implementierung

In der bisherigen Form ist Algorithmus 3.2 nur bedingt für eine Implementierung geeignet. Ein Grund hierfür liegt in dem Nichdeterminismus bei der Auswahl eines zu transformierenden Knotens. Hier kann es zu ungünstigen Laufzeiten kommen, falls zunächst ein passender Knoten gesucht werden muß. Dieses Verhalten läßt sich vermeiden, da es Abhängigkeiten zwischen der Transformation eines Knotens und der Anwendbarkeit von Transformationen auf seine Vorgänger bzw. Nachfolger gibt. Außerdem soll auf Alternativen bei der Wahl des Knotens in Transformation t_6 eingegangen werden, die Auswirkungen auf die Laufzeit des Algorithmus haben.

Anwendung der Transformationen

Zunächst werden in einem ersten Schritt alle Knoten auf die Anwendbarkeit einer der Transformationen $t_1, \ldots, t_5$ hin getestet und die entsprechende Transformation angewandt. Dies führt dazu, daß nur noch Knoten im Graphen vorhanden sind, die mögliche Kandidaten für t_6 sind. Insbesondere besitzt kein Knoten in G eine Schlinge.

Die nächste auszuführende Transformation muß t_6 sein. Dies impliziert das Entfernen eines Knotens v aus dem Graphen. Somit ändert sich der Eingangs- bzw. Ausgangsgrad aller Vorgänger und Nachfolger von v. Folglich ist ein Test auf die Anwendbarkeit der Transformationen $t_2, \ldots, t_5$ auf alle zu v adjazenten Knoten angebracht.

Gleiches gilt für die Transformationen t_1, t_2 und t_3, in denen ebenfalls Knoten aus dem Graphen entfernt werden und somit der Grad benachbarter Knoten geändert wird. Bei t_2 und t_3 ist allerdings zu beachten, daß nur der Ausgangs- bzw. der Eingangsgrad der adjazenten Knoten geändert wird und es somit genügt, auf die Anwendbarkeit der Transformationen t_2 und t_4 bzw. t_3 und t_5 hin zu testen.

Nachdem im ersten Schritt des Algorithmus alle Schlingen im Graphen entfernt wurden, besteht nur durch die Anwendung der Transformationen t_4 oder t_5 die Möglichkeit zur Konstruktion einer Schlinge. Somit genügt ein entsprechender Test bzgl. t_1 nach dem Einfügen der neuen Kanten in t_4 bzw. t_5.

In Algorithmus 3.3 sind die Prozeduren für die Transformationen t_2 und t_3 angegeben, die sich aus den dargestellten Abhängigkeiten ergeben. Zunächst wird die Anwendbarkeit der Transformation auf den Knoten v überprüft. Da der Knoten aus dem Graphen entfernt wird, erfolgt eine Sicherung der Nachfolger bzw. Vorgänger von v in einer Liste. Schließlich werden alle Knoten in dieser Liste, sofern sie nicht schon transformiert wurden, auf die Anwendbarkeit der Transformationen t_2 und t_4 bzw. t_3 und t_5 hin überprüft.

```
procedure t_2( v,G )
    if d_a(v) = 0 then
        P := {u ∈ V(G) : u ist Vorgänger von v};
        G := G − {v};
        for all u ∈ P do
            if u ∈ V(G) then t_2( u,G );
            if u ∈ V(G) then t_4( u,G );
endfor; endif; end;

procedure t_3( v,G )
    if d_e(v) = 0 then
        S := {u ∈ V(G) : u ist Nachfolger von v};
        G := G − {v};
        for all u ∈ S do
            if u ∈ V(G) then t_3( u,G );
            if u ∈ V(G) then t_5( u,G );
endfor; endif; end;
```

Algorithmus 3.3: Prozeduren für die Transformationen t_2 und t_3

Analog ergeben sich die Prozeduren für die Transformationen t_4 und t_5 in Algorithmus 3.4. Man beachte, daß aufgrund der Definition einer Kantenmenge (siehe Abschnitt 3.1) durch das Hinzufügen von Kanten zwischen den Vorgängern und Nachfolgern keine Mehrfachkanten entstehen. Je nach Implementierung eines Graphen ist dies entsprechend zu überprüfen. Nachdem der Knoten aus dem Graphen entfernt wurde, wird getestet, ob eine Schlinge erzeugt wurde. Der anschließende, rekursive Aufruf der Transformationen mit den Nachfolgern bzw. Vorgängern führte bei den in Abschnitt 6 vorkommenden Graphen zu signifikant besseren Ergebnissen bei der Größe des entstehenden FVS, ist aber im allgemeinen Fall nicht zu begründen.

```
procedure t_4( v,G )
    if d_a(v) = 1 then
        u := Nachfolger von v;
        for all Vorgänger w von v do E(G) := E(G) ∪ {(w,u)};
        G := G − {v};
        t_1(u,G);
        { if u ∈ V(G) then t_4( u,G ) }
endfor; endif; end;

procedure t_5( v,G )
    if d_e(v) = 1 then
        u := Vorgänger von v;
        for all Nachfolger w von v do E(G) := E(G) ∪ {(u,w)};
        G := G − {v};
        t_1(u,G);
        { if u ∈ V(G) then t_5( u,G ) }
endfor; endif; end;
```

Algorithmus 3.4: Prozedur für die Transformation t_4

In Algorithmus 3.5 sind die Prozeduren für die Transformation t_1 und das Hinzufügen eines neuen Feedbackknotens v dargestellt. Bei letzterer werden nach dem Entfernen von v aus G alle Vorgänger und Nachfolger von v auf die Anwendbarkeit einer Transformation $t_2, \ldots, t_5$ hin überprüft.

```
procedure t_1( v,G )
    if v hat Schlinge then add_to_FVS( v,G );
end;

procedure add_to_FVS( v,G )
    S := {u ∈ V(G) : u ist Nachfolger von v};
    P := {u ∈ V(G) : u ist Vorgänger von v};
    G := G − {v};
    FVS := FVS ∪{v};
    for all u ∈ S ∪ P do
        if u ∈ V(G) then t_2(u,G);
        if u ∈ V(G) then t_3(u,G);
        if u ∈ V(G) then t_4(u,G);
        if u ∈ V(G) then t_5(u,G);
endfor; end;
```

Algorithmus 3.5: Prozeduren für t_1 und das Hinzufügen eines Feedbackknotens

Die obigen Prozeduren sind schließlich in Algorithmus 3.6 zu einer Prozedur zur Bestimmung eines FVS für einen gegebenen Graphen G zusammengefaßt. Zunächst wird der Graph mit den Transformationen $t_1, \ldots, t_5$ soweit wie möglich reduziert, sodaß nach diesem Schritt nur noch Knoten vorhanden sind, auf die lediglich die Transformation t_6 anwendbar ist.

```
procedure buildFVS( G )
    for all v ∈ V(G) do
        t_1( v, G );
        if v ∈ V(G) then t_2( v, G );
        if v ∈ V(G) then t_3( v, G );
        if v ∈ V(G) then t_4( v, G );
        if v ∈ V(G) then t_5( v, G );
    endfor;
    while V(G) ≠ ∅ do
        v := Knoten mit maximalem Grad;
        add_to_FVS( v, G );
endwhile; end;
```

Algorithmus 3.6: Prozedur für Algorithmus 3.2

Alternative Knotenwahl

Nach Definition 3.2.7 wird ein Knoten mit maximalem Grad gewählt. Dies erfordert allerdings eine Suche in $V(G)$, womit die Komplexität des Algorithmus in $\mathcal{O}(n^2)$ mit $n = |G|$, liegt. Ein besseres Laufzeitverhalten ergibt sich, wenn man die globale Suche nach einem Knoten mit maximalem Grad auf eine lokale Suche beschränkt. Hierbei werden nur die Knoten in die Suche miteinbezogen, die auf die Anwendbarkeit von t_4 und t_5 hin getestet, also bei einem Aufruf der Prozeduren t_4 und t_5 übergeben werden. Ein entsprechend modifizierter Algorithmus erreicht ein fast lineares Laufzeitverhalten. Kleine Abweichungen vom optimalen Aufwand ergeben sich dadurch, daß in den verschiedenen Prozeduren auch Nachbarknoten auf die Anwendbarkeit einer Transformation hin getestet werden. Dies führt teilweise zu Mehrfachtests einzelner Knoten.

Weiterhin erkauft man sich den besseren Aufwand bei der lokalen Suche mit einem im allgemeinen größeren FVS. In durchgeführten Tests waren die Abweichungen aber vergleichsweise gering.

3.2.2. Ein FVS-Algorithmus für planare Graphen

In diesem Abschnitt wird ein Algorithmus vorgestellt, der für planare Graphen ein FVS berechnet, dessen Kardinalität höchstens zweimal so groß wie die Kardinalität eines minimalen FVS ist. Zusätzlich zur Planarität wird dabei an die Graphen eine Bedingung über die Verteilung der eingehenden und ausgehenden Kanten gestellt.

Der Algorithmus bildet die Grundlage für viele der Anordnungstechniken in Kapitel 4 und ist in [Hac97] eingehend beschrieben.

3.2.2.1. One-Flow-Direction Bedingung

In einem planaren Graphen lassen sich die Kanten eines Knoten nach dem Winkel ordnen, den sie zu einer festen Achse einnehmen. Hieraus leitet sich die folgende Bedingung für gerichtete, planare Graphen ab.

Definition 3.2.9 Sei $G = (V, E)$ ein planarer Graph. G erfüllt die *One-Flow-Direction-Bedingung* oder kurz OFD-Bedinung, falls es für alle Knoten $v \in V$ eine Liste $(e_1, \ldots, e_k)$ aller zu v inzidenten Kanten gibt, so daß:

i) für eine feste Achse $x \in \mathbb{R}^2$ gilt: $\measuredangle(x, e_i) \leq \measuredangle(x, e_{i+1})$ für alle $1 \leq i < k$, wobei der Winkel durch die Einbettung in den $\mathbb{R}^2$ definiert ist,

ii) ein $m \in \{0, \ldots, k\}$ existiert, so daß alle e_i mit $1 \leq i \leq m$ ausgehende Kanten sind und alle e_i mit $m < i \leq k$ e_i eingehende Kanten.

Anschaulich bedeutet Definition 3.2.9, daß man den eingehenden und den ausgehenden Kanten jeweils zwei zusammenhängende, disjunkte Bereiche zuordnen kann (siehe auch Abbildung 3.3 (i)). Eine nicht erlaubte Situation ist in Abbildung 3.3 (ii) dargestellt.

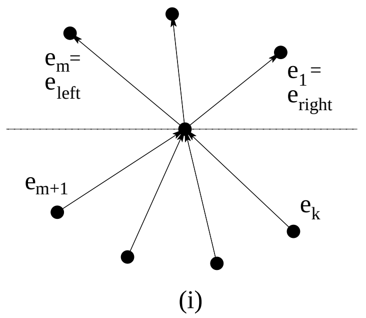

Abbildung 3.3: OFD-Bedingung (i) und nicht erlaubte Situation (ii)

Bemerkung 3.2.10 Sei $G = (V, E)$ ein Graph. Für die Zuordnung der geordneten Kantenliste zu einem Knoten entsprechend Definition 3.2.9 genügt es, wenn G *lokal planar* ist. Dabei heißt G lokal planar, wenn für alle Knoten $v \in V(G)$ der Graph $G_v = (V_v, E_v)$ mit $V_v := \{v\} \cup \{u \in V(G) : u \text{ ist adjazent zu } v\}$ und $E_v := \{e \in E(G) : e \text{ ist inzident zu } v\}$ [2] planar ist.

[2]Anschaulich ist G_v der Untergraph der Nachbarknoten von v.

In den folgenden Abschnitten bezeichne G stets einen planaren Graphen mit OFD-Bedingung.

3.2.2.2. Reduktion des Graphen

Nach Bemerkung 3.2.2 sind für ein FVS nur Knoten von Interesse, die auch auf einem Zykel liegen. Der hier vorgestellte Algorithmus bestimmt für jeden Knoten in G die Zugehörigkeit zu einem Zykel. Dabei bekommt jeder Knoten eine Markierung F, L oder C. Die Markierung C zeigt an, daß ein Knoten auf einem Zykel oder auf einem Pfad zwischen zwei Zykeln liegt. Die Markierungen F und L bedeuten demgegenüber, daß der Knoten nicht mit einem Zykel assoziiert ist. Insbesondere ist kein Knoten mit Markierung F bzw. L in einem Zykel enthalten[3].

```
procedure SetMarks( G )
    for all v ∈ V(G) do v.mark := C;      { Initialisierung }
    for all v ∈ V(G) do
        if v.mark = C then SetF( v );
        if v.mark = C then SetL( v );
endfor; end;

procedure SetF( v )
    if alle Vorgänger von v haben die Markierung F then
        v.mark = F;
        for all Nachfolger u von v do
            if u.mark = C then SetF( u );
endif; end;

procedure SetL( v )
    if alle Nachfolger von v haben die Markierung L then
        v.mark = L;
        for all Vorgänger u von v do
            if u.mark = C then SetL( u );
endif; end;
```

Algorithmus 3.7: Algorithmus zur Bestimmung der Knotenmarkierungen

Sei $G_C = G_C(G)$ die Einschränkung von G auf Knoten mit der Markierung C. Entsprechend Algorithmus 3.7 hat jeder Knoten in G_C mindestens einen Vorgänger und einen Nachfolger. Außerdem ist jeder Zykel in G auch ein Zykel in G_C. Im folgenden wird deshalb nur der Graph G_C betrachtet, der Einfachheit halber aber wieder mit G bezeichnet.

[3] Auf die genaue Bedeutung der Markierungen F und L wird in Zusammenhang mit den entsprechenden Anordnungsalgorithmen in Abschnitt 4.3.2.1 näher eingegangen.

3.2.2.3. Minimale und maximale Zykel

Aufgrund der OFD-Bedinung läßt sich G weiter reduzieren, indem für jeden Knoten nur bestimmte Kanten betrachtet werden, die bestimmend für die Struktur der Zykel in G sind.

Nach Definition 3.2.9 kann jedem Knoten $v \in V(G)$ eine geordnete Kantenliste $\{e_1, \ldots, e_m\}$ zugeordnet werden. Bezeichne $e_{right}(v)$ die Kante e_1 und $e_{left}(v)$ die Kante e_m (siehe auch Abb. 3.3 (i)). Die reduzierten Graphen G^+ und G^- besitzen die gleiche Knotenmenge wie G, ihre Kantenmenge wird aber reduziert auf $e_{left}(v)$ bzw. $e_{right}(v)$:

$$\begin{aligned} V(G^+) := V(G) \quad &, \qquad E(G^+) := \{e_{left}(v) \,:\, v \in V(G)\} \\ V(G^-) := V(G) \quad &, \qquad E(G^-) := \{e_{right}(v) \,:\, v \in V(G)\} \end{aligned}$$

Die Menge der Zykel in G^+ (G^-) sei mit C^+ (C^-) bezeichnet.

Aufgrund der Definition von G_C und G^+ besitzt jeder Knoten $v \in V(G^+)$ genau einen Nachfolger, definiert durch $e_{left}(v)$. Somit definiert die Rekursion

$$v_0 = v, \quad v_{i+1} = \text{Nachfolger von } v_i \tag{3.1}$$

eine eindeutige, unendliche Folge von Knoten in G^+. Da G und somit auch G^+ endlich sind, gibt es ein $i \geq 0$ und ein $j > i$ mit $v_i = v_j$ und $v_k \neq v_j$ für $i < k < j$. Nach Definition 3.1.19 bilden $(v_i, v_{i+1}, \ldots, v_j)$ einen Zykel. Da die Rekursion für alle Knoten in G^+ möglich ist, läßt sich eine surjektive Abbildung $\zeta_+ : V(G^+) \to C^+$ definieren, die jedem Knoten einen Zykel zuweist.

Aufgrund der Planarität von G und der damit verbundenen Einbettung von G in den $\mathbb{R}^2$ können das *Innere* bzw. das *Äußere* eines Zykels definiert werden. Sei hierzu $\zeta \in C^+$ und $v \in V(\zeta)$ mit Vorgänger $v_p \in V(\zeta)$ und Nachfolger $v_s \in V(\zeta)$. Alle Kanten in G, die in dem Winkel zwischen den Kanten (v_p, v) und (v, v_s) (entgegen dem Uhrzeigersinn) liegen, heißen *linke Kanten*. Sie bilden die Menge $E_{left}(v, \zeta)$. Die Kanten auf der anderen Seite, die *rechten Kanten*, bilden die Menge $E_{right}(v, \zeta)$ (siehe auch Abb. 3.4).

Sei G stark zusammenhängend und $\zeta \in C^+$. Seien G_i die starken Zusammenhangskomponenten von $G - \zeta$. Dann besitzt jede Komponente G_i einen Knoten $v_i \in V(G_i)$, der nicht auf ζ liegt, aber in G mit einem Knoten auf ζ verbunden ist:

$$\exists\, v \in V(\zeta)\, \exists\, v_i \in V(G_i) : (v_i, v) \in E(G) \text{ oder } (v, v_i) \in E(G). \tag{3.2}$$

Falls $(v_i, v) \in E_{left}(v, \zeta)$ bzw. $(v, v_i) \in E_{left}(v, \zeta)$, so gilt dies auch für jede andere Wahl eines Knotens $v_i \in V(G_i)$, der (3.2) erfüllt. Somit liegt der Teilgraph G_i auf der *linken Seite*, dem *Inneren* von ζ. Analog befindet sich G_i auf der *rechten Seite*, dem *Äußeren* von ζ, falls $(v_i, v) \in E_{right}(v, \zeta)$ bzw. $(v, v_i) \in E_{right}(v, \zeta)$. Die Vereinigung aller G_i im Inneren von ζ wird mit $\text{Int}(\zeta)$ bezeichnet. $\text{Ext}(\zeta)$ bezeichnet die Vereinigung aller G_i im Äußeren von ζ. Der Abschluß von $\text{Int}(\zeta)$ und $\text{Ext}(\zeta)$ sei definiert als: $\overline{\text{Int}}(\zeta) = \text{Int}(\zeta) \cup \zeta$ und $\overline{\text{Ext}}(\zeta) = \text{Ext}(\zeta) \cup \zeta$.

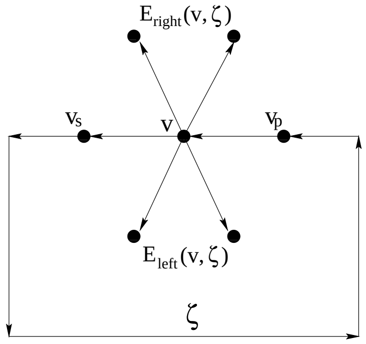

Abbildung 3.4: Linke und rechte Kanten

Aufgrund des folgenden Lemmas werden die Zykel in C^+ auch als *minimale Zykel* bezeichnet, da in ihrem Inneren keine weiteren (kleineren) Zykel enthalten sind.

Lemma 3.2.11 Sei G stark zusammenhängend. Dann gilt für alle $\zeta \in C^+$: $\mathrm{Int}(\zeta) = \emptyset$.

Beweis: Siehe Beweis zu Theorem 3.5 in [Hac97]. □

Ein analoges Resultat erhält man für die Zykel in C^-, die auch als *maximale Zykel* bezeichnet werden:

Lemma 3.2.12 Sei G stark zusammenhängend. Dann gilt für alle $\zeta \in C^-$: $\mathrm{Ext}(\zeta) = \emptyset$.

3.2.2.4. Konzentrische Zykel

Für G läßt sich iterativ eine Hierarchie von minimalen Zykeln konstruieren. Hierzu wird ein $\zeta \in C^+$ aus dem Graphen entfernt und die Konstruktion 3.1 der minimalen Zykel auf $G_C(G - \zeta)$ angewandt. Algorithmus 3.8 konstruiert somit eine Menge $\{\zeta_0, \ldots, \zeta_k\}$ von minimalen Zykeln.

Eingabe: planarer Graph G mit OFD-Bedingung
$G_0 := G_C(G)$; $i := 0$;
while $V(G_i) \neq \emptyset$ **do**
  wähle $v \in V(G_i)$;
  $\zeta_i := \zeta_+(v)$;
  $G'_i := G_i - \zeta_i$;
  $G_{i+1} := G_C(G'_i)$;
  $i := i + 1$;
endwhile;

Algorithmus 3.8: Algorithmus zur Bestimmung der minimalen Zykel

Unter geeigneten Voraussetzungen gelten für die Zykel $\zeta_0, \ldots, \zeta_k$: $\zeta_i \subseteq \zeta_{i+1}$. Dies motiviert die folgende partielle Ordnung für $\zeta, \zeta' \in \{\zeta_0, \ldots, \zeta_k\}$:

$$\zeta' > \zeta \quad :\Longleftrightarrow \quad \overline{\mathrm{Int}}(\zeta) \subseteq \mathrm{Int}(\zeta') \text{ und } \overline{\mathrm{Ext}}(\zeta') \subseteq \mathrm{Ext}(\zeta). \tag{3.3}$$

Für die weiteren Betrachtungen wird mittels dieser Ordnungsrelation ein Graph definiert, der die Hierarchie der in Algorithmus 3.8 konstruierten Zykel wiederspiegelt. Sei $Z^+ = Z^+(G) = (V(Z^+), E(Z^+))$ dieser Graph mit

$$\begin{aligned} V(Z^+) &= \{\zeta_0, \ldots, \zeta_k\} \\ E(Z^+) &= \{(\zeta, \zeta') \in V(Z^+) \times V(Z^+) : \zeta' > \zeta \\ &\qquad \text{und es existiert kein } \zeta'' \in V(Z^+) \text{ mit } \zeta' > \zeta'' > \zeta\}. \end{aligned}$$

Z^+ ist aufgrund der Ordnungsdefinition azyklisch, und jeweils zwei Zykel $\zeta, \zeta' \in V(Z^+)$ mit $\zeta > \zeta'$ sind durch einen eindeutigen Pfad verbunden. In Abbildung 3.5 sind einige Beispiele für Zykel und dazugehörige Z^+-Graphen aufgeführt.

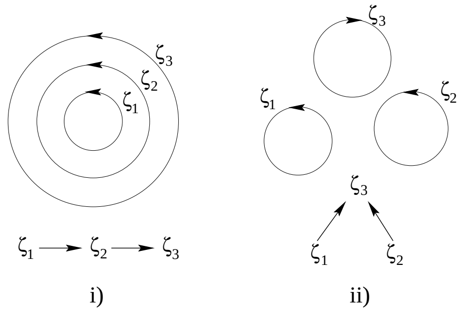

Abbildung 3.5: Beispiele für minimale Zykel und zugehörige Z^+-Graphen

Wird die Rekursion (3.1) auf Knoten in G^- angewandt, so erhält man eine surjektive Abbildung der Knoten in $V(G^-)$ in die Menge der maximalen Zykel C^-. Diese Abbildung sei mit ζ_- benannt. Ersetzt man in Algorithmus 3.8 ζ_+ durch ζ_-, werden maximale Zykel konstruiert. Die so erhaltene Menge von Zykeln sei $\{\zeta_0^-, \ldots, \zeta_l^-\}$. In Entsprechung zu Z^+ bildet man den Graphen $Z^- = Z^-(G) = (V(Z^-), E(Z^-))$ mit

$$\begin{aligned} V(Z^-) &= \{\zeta_0^-, \ldots, \zeta_l^-\} \\ E(Z^-) &= \{(\zeta', \zeta) \in V(Z^-) \times V(Z^-) : \zeta' > \zeta \\ &\qquad \text{und es exitiert kein } \zeta'' \in V(Z^-) \text{ mit } \zeta' > \zeta'' > \zeta\}. \end{aligned}$$

Durch die Definition des Inneren und Äußeren von Zykeln kann eine wichtige Relation zwischen den Knoten in Z^+ und Z^- eingeführt werden.

Definition 3.2.13 Seien $\zeta^+ \in V(Z^+(G))$ und $\zeta^- \in V(Z^-(G))$. Falls

$$\zeta^+ \cap \zeta^- \neq \emptyset \quad \text{und} \quad \mathrm{Int}(\zeta^+) \cap \zeta^- = \zeta^+ \cap \mathrm{Ext}(\zeta^-) = \emptyset,$$

so *berührt* ζ^+ den Zykel ζ^-. Abgekürzt wird die Relation durch: $\zeta^+ \circ \zeta^-$.

Das folgende Lemma zeigt einige Eigenschaften der Relation $\circ$ auf.

Lemma 3.2.14 i) Für jeden Zykel ζ in G existiert mindestens ein Zykel $\zeta_+ \in V(Z^+)$ mit $\zeta_+ \circ \zeta$ und $\zeta_+ \subset \overline{\mathrm{Int}}(\zeta)$.

ii) Für jeden Zykel ζ in G existiert mindestens ein Zykel $\zeta_- \in V(Z^-)$ mit $\zeta_- \circ \zeta$ und $\zeta_- \subset \overline{\mathrm{Ext}}(\zeta)$.

iii) Für jeden Zykel $\zeta_+ \in V(Z^+)$ existiert mindestens ein Zykel $\zeta_- \in V(Z^-)$ mit $\zeta_+ \circ \zeta_-$.

Beweis: Siehe Beweis zu Lemma 5.2 in [Hac97]. □

3.2.2.5. Eine Bewertungsfunktion und Konstruktion des FVS

In diesem Abschnitt wird eine Bewertungsfunktion für Knoten in G eingeführt. Sie definiert eine Knotenmenge, aus der innerhalb des späteren Algorithmus der jeweils nächste Knoten für das FVS entnommen werden kann und erlaubt es, die Kardinalität eines FVS für G nach oben abzuschätzen.

Sei $v \in V(G)$. Die Bewertungsfunktion setzt sich zusammen aus:

$$\varphi_+(v, Z^+) = \frac{1}{2}\left(|\{\zeta \in V(Z^+) \,:\, v \in \mathrm{Ext}(\zeta)\}| + |\{\zeta \in V(Z^+) \,:\, v \in \overline{\mathrm{Ext}}(\zeta)\}|\right)$$

und

$$\varphi_-(v, Z^-) = \frac{1}{2}\left(|\{\zeta \in V(Z^-) \,:\, v \in \mathrm{Int}(\zeta)\}| + |\{\zeta \in V(Z^-) \,:\, v \in \overline{\mathrm{Int}}(\zeta)\}|\right).$$

Dabei entspricht $\varphi_+(v)$ der gemittelten Anzahl minimaler Zykel für die v im Inneren bzw. im Abschluß des Inneren liegt. Analog entspricht $\varphi_-(v)$ der gemittelten Anzahl maximaler Zykel für die v im Äußeren bzw. im Abschluß des Äußeren liegt.

Die Bewertungsfunktion φ ist schließlich definiert als:

$$\varphi(v) = \varphi(v, Z^+, Z^-) = \varphi_+(v, Z^+) + \varphi_-(v, Z^-). \tag{3.4}$$

Die Funktion φ kann aufgrund der Einbettung von G in den $\mathbb{R}^2$ auch für ein beliebiges $x \in \mathbb{R}^2$ definiert werden. Insbesondere kann φ für bereits aus dem Graphen entfernte Knoten ausgewertet werden.

In der folgenden Bemerkung sind einige Eigenschaften von φ zusammengefaßt, die in den späteren Algorithmus einfließen werden.

Bemerkung 3.2.15 i) G ist azyklisch $\Longleftrightarrow \exists\, v \in V(G) : \varphi(v) = 0$.

ii) Für alle $v \in V(G)$ gilt: $\varphi(v) \leq |V(Z^+)| + |V(Z^-)|$.

iii) Für alle $v \in V(G)$ gilt: $\varphi(v) \geq$ maximale Anzahl disjunkter Zykel in G.

Beweis: Siehe Beweis zu Bemerkung 5.4 in [Hac97]. □

Wie bereits angesprochen kann mit Hilfe von φ eine Knotenmenge von G definiert werden, aus der in jedem Iterationsschritt des Algorithmus der nächste Knoten für das FVS entnommen werden kann. Das folgende Lemma gibt diese Menge an.

Lemma 3.2.16 Das Minimum von φ wird auf der Menge

$$T := \{v \in V(G) \; : \; v \in \zeta_+ \cap \zeta_- \text{ für } \zeta_+ \in V(Z^+) \text{ und } \zeta_- \in V(Z^-) \text{ mit } \zeta_+ \circ \zeta_-\}$$

angenommen.

Beweis: Siehe Abschnitt 5.4 in [Hac97]. □

Algorithmus 3.9 wählt nun sukzessive Knoten aus T für das FVS. Sei aber zunächst

$$\varphi(v, G) := \varphi(v, Z^+, Z^-), \qquad \text{für } v \in V(G).$$

Dabei sind Z^+ und Z^- bezüglich G gemeint, da sich der Graph während des Algorithmus ändert. Weiter wird das Minimum von φ über $V(G)$ bezeichnet mit

$$\varphi_{min}(G) := \min\{\varphi(v, G) \; : \; v \in V(G)\}.$$

Für einen gegebenen Graphen G mit OFD-Bedingung iteriert Algorithmus 3.9, bis $\varphi_{min}(G_i) = 0$. Nach Bemerkung 3.2.15 ist dies äquivalent mit dem Fehlen von Zykeln in G. In jedem Iterationsschritt wird ein Knoten v_i aus $T(G_i)$ ausgewählt, der die Bewertungsfunktion minimiert. Dieser Knoten wird dem FVS hinzugefügt und aus dem Graphen entfernt.

Eingabe: planarer Graph G mit OFD-Bedingung
$i := 0$; $G_0 = G$; $F_0 = \emptyset$;
while $\varphi_{min}(G) > 0$ **do**
    $v_i :=$ ein $v \in T(G_i)$ mit $\varphi(v, G_i) = \varphi_{min}(G_i)$;
    $F_{i+1} := F_i \cup \{v_i\}$;
    $G_{i+1} := G_i - \{v_i\}$;
    $i := i + 1$;
endwhile;
FVS $:= F_i$;

Algorithmus 3.9: FVS-Algorithmus für planare Graphen mit OFD-Bedingung

Das nachfolgende Lemma gibt eine obere Schranke für die Anzahl der benötigten Iterationsschritte und somit für die Größe des FVS an.

Lemma 3.2.17 Sei G ein planarer Graph mit OFD-Bedingung, $v \in T(G)$ und $G' = G - \{v\}$. Weiter seien G_i die Graphen aus Algorithmus 3.9 bezüglich Eingabe G. Dann gilt:

i) $\varphi(v, G') = \varphi(v, G) - 1$.

ii) $\varphi_{min}(G_{i+1}) \leq \varphi_{min}(G_i) - 1$.

iii) Algorithmus 3.9 terminiert nach höchstens $i = \varphi_{min}(G)$ Schritten und somit gilt: $|\text{FVS}| \leq \varphi_{min}(G)$

Beweis: Siehe Beweis zu Lemma 6.3 in [Hac97]. □

Faßt man die obigen Ergebnisse zusammen, so ergibt sich die nachfolgende Aussage über die Kardinalität des berechneten FVS.

Lemma 3.2.18 Sei $F \subseteq V$ ein durch Algorithmus 3.9 berechnetes FVS von G und F_{min} ein minimales FVS von G. Dann gilt:

$$|F| \;\leq\; 2 \cdot |F_{min}|.$$

Beweis: Sei $\mu = \max\{\,|Z| \,:\, Z \text{ ist eine Menge disjunkter Zykel von } G\}$. Da jedes FVS von G mindestens einen Knoten aus jedem Zykel einer Menge von disjunkten Zykeln enthalten muß, gilt: $|F_{min}| \geq \mu$. Da $V(Z^+)$ und $V(Z^-)$ Mengen disjunkter Zykel sind, folgt: $|V(Z^+)| + |V(Z^-)| \leq 2 \cdot \mu$. Mit Bemerkung 3.2.15 gilt somit:

$$\mu \;\leq\; \varphi \;\leq\; |V(Z^+)| + |V(Z^-)| \;\leq\; 2 \cdot \mu \;\leq\; 2 \cdot |F_{min}|.$$

Mit Lemma 3.2.17 folgt: $|F| \;\leq\; \varphi_{min}(G) \;\leq\; 2 \cdot \mu \;\leq\; 2 \cdot |F_{min}|$. □

3.2.2.6. Implementierung

In diesem Abschnitt soll näher auf die Implementierung des FVS-Algorithmus 3.9 eingegangen werden. Oberstes Ziel ist dabei ein linearer Aufwand für das Verfahren.

Der Algorithmus läßt sich in zwei Abschnitte einteilen. Der erste Teil entspricht dabei dem Aufbau der Graphen $Z^+(G)$ und $Z^-(G)$. Im zweiten Abschnitt werden die Feedbackknoten bestimmt. Vorausgesetzt wird dabei, daß alle Kanten eines Knotens im gegebenen Graphen G entsprechend der OFD-Bedingung sortiert sind, d.h., daß für jeden Knoten die Liste $(e_1, \ldots, e_k)$ von adjazenten, dem Winkel nach geordneten Kanten zur Verfügung steht. Desweiteren wird davon ausgegangen, daß der Graph G eine starke Zusammenhangskomponente darstellt. Hierzu ist es gegebenenfalls nötig, zunächst einen Algorithmus zur Konstruktion der Zusammenhangskomponenten aufzurufen. Ein Verfahren hierfür, welches lineare Komplexität besitzt, wird z.B. in [Tar72] beschrieben.

Reduktion des Graphen

Bei der Konstruktion der Graphen $Z^+(G)$ und $Z^-(G)$ wird das Anlegen der Knoten, d.h. der minimalen bzw. maximalen Zykel von G, getrennt von dem Bestimmen der Kantenrelation[4]. Zur Konstruktion der Zykel wird Algorithmus 3.8 genutzt. Kritisch für die lineare Komplexität ist dabei die Reduzierung des aktuellen Graphen G_i auf die Menge der Knoten mit Markierung C mittels Algorithmus 3.7. In der bisherigen Variante wird hierfür die gesamte Knotenmenge von G durchlaufen.

Allerdings kann eine Veränderung der Markierungen nur durch das Entfernen eines Knotens hervorgerufen werden. Aus diesem Grund genügt es, nur die Knoten zu betrachten, die von ζ_i aus erreichbar sind. Die Prozedur *SetMarks* in Algorithmus 3.7 erhält hierzu ζ_i anstatt G als Eingabe. Weiterhin kann die Initialisierung der Knotenmarkierungen entfallen, da Knoten mit den Attributen F oder L diese auch in G_{i+1} besitzen, alle anderen Knoten aber über die Markierung C verfügen.

In den Prozeduren *SetF* und *SetL* sind die Attribute der Vorgänger bzw. Nachfolger nicht in G_i, sondern in G'_i zu überprüfen, da ζ_i bereits aus dem Graphen entfernt wurde. Die modifizierten Varianten lauten

procedure LokalSetF(v)
    if $v \in V(\zeta_i)$ oder alle Vorgänger von v in $V(G'_i)$ haben die Markierung F **then**
        v.mark = F;
        for all Nachfolger u von v in $V(G_i)$ **do**
            if u.mark = C **then** LokalSetF(u);
endif; **end**;

procedure LokalSetL(v)
    if $v \in V(\zeta_i)$ oder alle Nachfolger von v in $V(G'_i)$ haben die Markierung L **then**
        v.mark = L;
        for all Vorgänger u von v in $V(G_i)$ **do**
            if u.mark = C **then** LokalSetL(u);
endif; **end**;

Der so veränderte Graphendurchlauf erreicht nur Knoten in $G_{i-1} - V(G_i)$, da die Knoten auf dem neuen Zykel ζ_{i+1} die Markierung C beibehalten. Somit liegt auch der Aufwand nur in $\mathcal{O}(|G_{i-1} - V(G_i)|)$. Für den Gesamtaufwand der Zykelbestimmung in Algorithmus 3.8 ergeben sich somit lineare Kosten in der Größe des Graphen.

Aufbau der Kantenrelation in Z^+ und Z^-

Der Aufbau der Kanten in $Z^\pm$ erfolgt durch einen Erreichbarkeitstest zwischen den Zykeln über Pfade in G. Falls z.B. für zwei Zykel ζ_1^+, ζ_2^+ in Z^+ ein Pfad π zwischen den Knoten $v_1 \in V(\zeta_1^+)$ und $v_2 \in V(\zeta_2^+)$ existiert und π sich in $\overline{\mathrm{Ext}}(\zeta_1^+) \cap \overline{\mathrm{Int}}(\zeta_2^+)$ befindet, so gilt für die Zykel: $\overline{\mathrm{Int}}(\zeta_1^+) \subseteq \mathrm{Int}(\zeta_2)$ und $\overline{\mathrm{Ext}}(\zeta_2^+) \subseteq \mathrm{Ext}(\zeta_1^+)$. Dies entspricht der Definition

[4] Um Verwechslungen zwischen den Knoten in G bzw. in $Z^\pm$ zu vermeiden, wird im folgenden stets von (minimalen/maximalen) Zykeln gesprochen, wobei aber die Knoten in $Z^\pm$ gemeint sind.

der partiellen Ordnung $>$ zwischen Zykeln (siehe Gleichung (3.3)). Liegt π weiterhin im Inneren aller minimalen Zykel ζ^+ mit $\zeta_1^+ < \zeta^+$, so gibt es auch kein $\zeta_3^+ \in V(Z^+)$ mit $\zeta_1^+ < \zeta_3^+ < \zeta_2^+$, und somit gilt: $(\zeta_1^+, \zeta_2^+) \in E(Z^+)$. Die letzte Bedingung kann beim Aufbau des Pfades π bzw. beim entsprechenden Durchlauf durch den Graphen G erreicht werden, indem man sich für alle minimalen Zykel entweder auf die Knoten im Inneren oder auf die Knoten im Äußeren beschränkt. Dies bedeutet, daß beim Erreichen eines Zykels über eine linke Kante nur weitere linke Kanten (und Kanten des Zykels) beim Graphendurchlauf betrachtet werden. In Abbildung 3.6 i) sind die für die Pfadkonstruktion erlaubten Bereiche in einem Beispiel dargestellt.

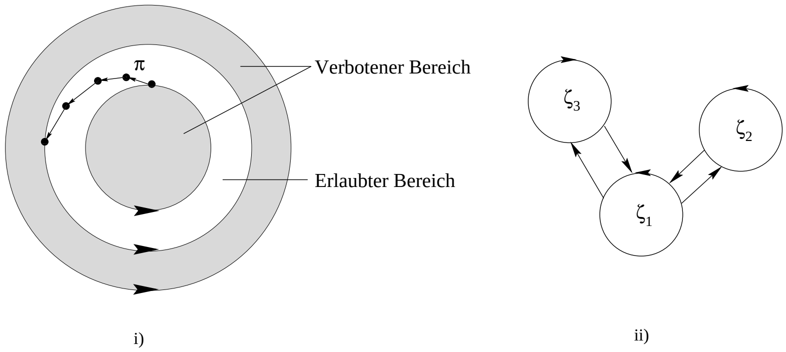

Abbildung 3.6: Beispiele für die Pfadkonstruktion

Die Erreichbarkeit aller Zykel wird durch den starken Zusammenhang des Graphen sichergestellt. Man beachte aber, daß diese Erreichbarkeit auch über Pfade erfolgen kann, die auf nichtvergleichbaren Zykeln liegen, wie Abbildung 3.6 ii) verdeutlicht. In dem dort angegebenen Beispiel gilt: $\zeta_1 < \zeta_3$ und $\zeta_2 < \zeta_3$. In einem Graphendurchlauf ist ζ_3 von ζ_2 aber nur über ζ_1 erreichbar.

Für das Berechnen von $E(Z^-)$ gelten analoge Bedingungen wie hier für $E(Z^+)$ beschrieben. Aufgrund der Einschränkungen beim Graphendurchlauf für das Bestimmen der Kantenrelation ergibt sich ein ähnlich lokaler Aufwand wie für die Konstruktion der Zykel selbst. Die Komplexität des Verfahrens ist somit ebenfalls linear in der Größe des Graphen. Der eigentliche Algorithmus entspricht wie bereits angesprochen einem Graphendurchlauf, z.B. Breiten- oder Tiefensuche, und soll hier nicht weiter behandelt werden.

Bestimmung der Feedbackknoten

Nachdem die Graphen Z^+ und Z^- zur Verfügung stehen, läßt sich entsprechend Algorithmus 3.9 ein Knoten für das FVS bestimmen. Dabei kann die Menge $T(G)$ aus Bemerkung 3.2.16 noch weiter eingeschränkt werden:

Lemma 3.2.19 Sei $B^+ = B^+(G)$ die Menge aller Knoten $\zeta \in V(Z^+)$ mit $d_e(\zeta) = 0$. Das Minimum von φ wird angenommen auf der Menge

$$T_+(G) = \{v \in V(G) \,:\, v \in \zeta_+ \cap \zeta \text{ für } \zeta_+ \in B^+ \text{ und } \zeta \in V(Z^-)\}. \tag{3.5}$$

Beweis: Siehe Beweis zu Lemma 6.6 in [Hac97]. □

Die Menge B^+ entspricht den Zykeln in $C^+(G)$ (siehe Abschnitt 3.2.2.3).

Eine Möglichkeit, weitere Knoten für das FVS zu finden, ohne die Graphen Z^+ und Z^- nach dem Entfernen des Feedbackknotens neu zu konstruieren, bietet das folgende Lemma:

Lemma 3.2.20 Seien $\zeta^+ \in V(Z^+)$ und $\zeta^- \in V(Z^-)$ zwei Zykel mit $\zeta^+ \circ \zeta^-$. Besitzt ζ^+ einen Nachfolger in Z^+ und ζ^- einen Vorgänger in Z^-, so berühren sich diese Zykel ebenfalls.

Beweis: Siehe Beweis zu Lemma 5.6 und Abschnitt 6.4 in [Hac97]. □

Es genügt also, die Nachfolger/Vorgänger der minimalen/maximalen Zykeln zu bestimmen, die den aktuellen Feedbackknoten enthalten, und aus deren Schnitt einen weiteren Knoten für das FVS zu entnehmen. Falls auch die neuen Zykel Nachfolger bzw. Vorgänger besitzen, kann der Prozeß fortgesetzt werden. Bilden die Knoten in Z^+ und Z^- eine Kette wie in Abbildung 3.5 i), so erlaubt es Lemma 3.2.20, ein FVS des Graphen zu konstruieren, ohne die Graphen Z^+ und Z^- erneut zu berechnen.

Auch im allgemeinen Fall kann der Aufwand für die Neubestimmung von Z^+ und Z^- nach dem Entfernen eines Feedbackknotens aus dem Graphen G beschränkt werden.

Bemerkung 3.2.21 Sei $v_i \in V(G_i)$ mit $\varphi(v_i, G) = \varphi_{\min}(G_i)$ ein Feedbackknoten entsprechend Algorithmus 3.9. Sei Z_i^+ die Menge aller Zykel $\zeta^+ \in V(Z^+(G_i))$ mit $v_i \in \mathrm{Ext}(\zeta^+)$ und analog Z_i^- die Menge aller Zykel $\zeta^- \in V(Z^-(G_i))$ mit $v_i \in \mathrm{Int}(\zeta^-)$. Nach Definition der minimalen (maximalen) Zykel sind für deren Konstruktion nur die Teilgraphen von G_i in deren Inneren (Äußeren) notwendig. Somit folgt: $Z_i^+ \subseteq V(Z^+(G_{i+1}))$ und $Z_i^- \subseteq V(Z^-(G_{i+1}))$.

Sei v_i ein Feedbackknoten entsprechend Algorithmus 3.9, und sei $\zeta^- \in V(Z^-(G_i))$ ein maximaler Zykel mit $v_i \in V(\zeta^-)$. Nach Bemerkung 3.2.21 sind Vorgänger von ζ^- in $Z^-(G_i)$ ebenfalls in $Z^-(G_{i+1})$ enthalten und brauchen somit nicht mehr bei der Neukonstruktion von $Z^-(G_{i+1})$ betrachtet werden. Für $\zeta^+ \in V(Z^+(G_i))$ mit $v_i \in \zeta^+$ sind diese Betrachtungen nicht relevant, da nach Lemma 3.2.19 ζ^+ keine Vorgänger in Z^+ besitzt.

Eine Neukonstruktion der minimalen (maximalen) Zykeln beschränkt sich somit auf das Äußere (Innere) von ζ^+ (ζ^-). Diese Mengen lassen sich weiter beschränken, da man nur Zykel berechnen muß, die für die Bestimmung eines neuen Feedbackknotens nötig sind. Hierzu macht man sich klar, daß ein neukonstruierter Zykel in

$$[\bigcup_{\zeta \in S^+} \overline{\mathrm{Int}}(\zeta)] \setminus \mathrm{Int}(\zeta^+), \tag{3.6}$$

der Vereinigung der abgeschlossenen Inneren aller Nachfolger von ζ^+ liegt. Dies folgt, da die Nachfolger selbst Kandidaten für Zykel in Z_{i+1}^+ sind. In dem Beispiel in Abbildung 3.2.2.6 ist der Bereich aus Gleichung 3.6 das Innere von ζ_4.

Diese Menge kann weiter reduziert werden. Sei hierzu S^+ die Menge der Nachfolger von ζ_i und $\mathrm{Pred}(S^+)$ die Menge der Vorgänger von Zykeln in S^+. Nach Bemerkung 3.2.21 sind alle Zykel, die Vorgänger eines Zykels in $Pred(S^+)$ sind, auch Zykel in $Z^+(G_{i+1})$. Somit braucht man das Innere solcher Zykel für die Zykelkonstruktion nicht zu betrachten. In Abbildung 3.2.2.6 ist dies $\overline{\mathrm{Int}}(\zeta_2)$, der für das Update von Z^+ relevante Bereich ist in dem entsprechenden Z^+-Graphen angegeben.

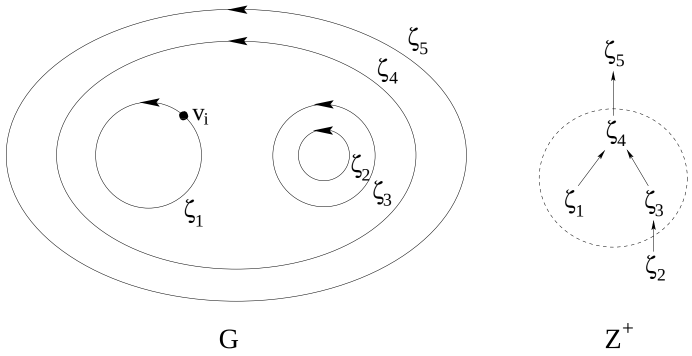

Abbildung 3.7: Beispiel für die Neukonstruktion von Z^+

Für den Graphen Z^- gelten die obigen Aussagen analog. Die hierbei zu betrachtende Menge ist

$$[\bigcup_{\zeta \in S^-} \overline{\mathrm{Ext}}(\zeta)] \setminus \mathrm{Ext}(\zeta^-), \tag{3.7}$$

wobei S^- die Menge aller Nachfolger von ζ^- in $Z^-(G_i)$ ist.

Für die Konstruktion der Zykel selbst kann die Rekursion 3.1 genutzt werden, wobei lediglich die Menge der Knoten auf die angegebenen Bereiche reduziert wird. Gegebenenfalls sind vorher die Markierungen der involvierten Knoten zu aktualisieren, da der Feedbackknoten aus dem Graphen entfernt wurde.

Nachdem die Zykel in den entsprechenden Gebieten (3.6) und (3.7) konstruiert wurden, kann die Kanteninformation auf die gleiche Weise bestimmt werden, wie bei der erstmaligen Konstruktion der Graphen.

Wie aus (3.6) und (3.7) folgt, ist auch für die Neukonstruktion von Z^+ und Z^- lediglich ein lokaler Aufwand nötig. Insgesamt gilt:

Bemerkung 3.2.22 Die Komplexität von Algorithmus 3.9 zur Bestimmung eines FVS eines gegebenen, planaren Graphen G liegt in $\mathcal{O}(|G|)$.

3.2.2.7. Graphen ohne OFD-Bedingung

Wesentlich für die Abschätzung der Kardinalität des durch Algorithmus 3.9 berechneten FVS war die Eigenschaft, die Kanten eines Knotens entsprechend der OFD-Bedingung anordnen zu können. Sollte diese Eigenschaft nicht vorliegen, so läßt sich eine alternative Anordnung gewinnen, falls der Graph gewichtet ist (siehe Abschnitt 3.1.3).

Sei $G = (V, E, w_V, w_E)$ ein gewichteter Graph. Jedem Knoten $v \in V$ läßt sich eine Liste $(e_1, \ldots, e_m)$ von adjazenten, ausgehenden Kanten zuordnen, mit

$$w_E(e_1) \leq w_E(e_2) \leq \cdots \leq w_E(e_m). \tag{3.8}$$

Somit entspricht die Kante $e_{right}(v) = e_1$ der ausgehenden Kante mit dem kleinsten Gewicht. Analog handelt es sich bei der Kante $e_{left}(v) = e_m$ um die ausgehende Kante mit dem größten Gewicht.

In (3.8) wurde lediglich von den Kantengewichten Gebrauch gemacht. Alternativ lassen sich auch die Gewichte der Knoten in die Definition der Anordnung einbeziehen. Die Liste der adjazenten, ausgehenden Kanten $(e_1, \ldots, e_m)$ sei dann definiert durch:

$$w_E(e_1) + w_V(v_1) \leq w_E(e_2) + w_V(v_2) \leq \cdots \leq w_E(e_m) + w_V(v_m), \tag{3.9}$$

mit $e_i = (v, v_i)$ mit $1 \leq i \leq m$.

Führt man einen entsprechend modifizierten Algorithmus 3.9 mittels einer der Anordnungen (3.8) und (3.9) durch, so lassen sich allerdings die Ergebnisse bezüglich der Größe des berechneten FVS nicht mehr übernehmen. Es handelt sich somit um einen heuristischen Algorithmus.

4. Anordnungsalgorithmen

Viele Iterationsverfahren, wie z.B. die Gauß-Seidel-Iteration, sind abhängig von der gewählten Anordnung der Unbekannten innerhalb des zu lösenden Gleichungssystems. Insbesondere bei den konvektionsdominanten Problemen, die Gegenstand dieser Arbeit sind, kann eine geeignete Anordnung entscheidend zur Beschleunigung der Konvergenz der Lösungsverfahren beitragen, wie die Ergebnisse in Kapitel 6 verdeutlichen.

In diesem Kapitel sollen die verwendeten Anordnungstechniken vorgestellt werden. Hierzu folgen zunächst in Abschnitt 4.1 Definitionen im Zusammenhang mit Anordnungen bzw. *Permutationen*. In Abschnitt 4.2 wird ein Algorithmus vorgestellt, der die *Bandbreite* oder das *Profil* einer Matrix reduziert. Anschließend werden in Abschnitt 4.3 Algorithmen und Anordnungen für konvektionsdominante Probleme im $\mathbb{R}^2$ und $\mathbb{R}^3$ eingeführt.

4.1. Permutationen und Permutationsmatrizen

Den Ausgangspunkt für die zu berechnenden Anordnungen der Unbekannten stellt die, aus der Diskretisierung des Modellproblems stammende Anordnung dar. Diese ist abhängig von der Triangulation des Gebietes und der Art der Verfeinerung der Gitter. Als Ergebnis der Diskretisierung entsteht das Gleichungssystem

$$Ax = b, \tag{4.1}$$

mit $A = (a_{ij})_{i,j=1}^n \in \mathbb{R}^{n \times n}$ und $b \in \mathbb{R}^n$.

Für die Umordnung der Unbekannten in diesem Gleichungssystem werden *Permutationen* und *Permutationsmatrizen* verwendet.

Definition 4.1.1 Sei $n \in \mathbb{N}$. Eine bijektive Abbildung $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ heißt *Permutation*. Die durch π bestimmte *Permutationsmatrix* $P = P_\pi$ ist komponentenweise definiert durch:

$$P_{ij} = \delta_{\pi(i),j}.$$

Hierbei ist δ_{ij} das Kroneckerdelta.

Das durch die Permutation π definierte, umgeordnete System ergibt sich durch Multiplikation mit der zugehörigen Permutationsmatrix:

$$P\,A\,P^T\,x' = P\,b,$$

wobei sich die ursprüngliche Lösung von (4.1) durch $x = P^T x'$ ergibt. Die Permutation selbst wird in den folgenden Abschnitten stets durch die Reihenfolge der Unbekannten angegeben, in der diese im umgeordneten System auftreten sollen.

4.2. Bandbreitenreduktion

In diesem Abschnitt soll ein Algorithmus vorgestellt werden, der der Beschleunigung von direkten Lösungsverfahren für Gleichungssysteme auf Basis der Gaußelimination dient, indem die Anzahl der Rechenoperationen gesenkt wird. Abgesehen davon stellt er ein robustes Instrument zur Beschleunigung von iterativen Verfahren dar.

Definition 4.2.1 Sei $n \geq 1$ und $A = (a_{ij})_{i,j=1}^n \in \mathbb{R}^{n\times n}$. Bezeichne für $i \leq n$

$$\begin{aligned} m_l^i &= \max\{j < i \,:\, a_{i,i-j} \neq 0\} \quad \text{und} \\ m_u^i &= \max\{j < i \,:\, a_{i-j,i} \neq 0\} \end{aligned}$$

die Abstände des am weitesten links bzw. oberhalb der Diagonale stehenden Nichtnullelementes in jeder Zeile bzw. Spalte.

Die *untere* (*obere*) *Semibandbreite* von A ist definiert als $m_l := \max_{i\leq n} m_l^i$ ($m_u := \max_{i\leq n} m_u^i$). Die Bandbreite von A ist $m_l + m_u + 1$.

Besitzt A eine symmetrische Besetztheitsstruktur, so stimmen m_l und m_u überein. Die Bandbreite von A ist entsprechend $2m + 1$. Desweiteren wird mit $\sum_{i=1}^n (m_l^i + 1)$ das *Profil* von A bezeichnet.

Da während der Gaußelimination nur Einträge nach dem ersten Nichtnullelement einer Zeile bzw. Spalte eliminiert werden müssen, führt eine kleine Bandbreite zu einer Verringerung des Aufwandes der Gaußelimination. Aus den gleichen Gründen schließt man, daß ein kleines Profil zu einer Aufwandsverringerung führt.

Im folgenden besitze die Matrix A eine symmetrische Besetztheitsstruktur und sei $G = G(A) = (V, E)$ der (ungerichtete) Matrixgraph von A.

Ein Weg zur Reduktion der Bandbreite von A besteht darin, die Knoten von G in sogenannte *Levelsets* S_i aufzuteilen. Dabei besteht S_1 nur aus einem Knoten, dem Startknoten. Die Menge S_2 enthält alle Knoten, die zum Startknoten adjazent sind. Allgemein besteht für $i > 2$ die Menge S_i aus allen Knoten, die zu Knoten in S_{i-1} adjazent sind und nicht in $S_{i-2} \cup S_{i-1}$ enthalten sind. Das resultierende Verfahren ist in Algorithmus 4.1 dargestellt. Wird A entsprechend der Reihenfolge der Mengen S_i permutiert, also zuerst alle Knoten in S_1, dann S_2 usw., erhält A eine Blocktridiagonalgestalt, wobei die Diagonalblöcke den Mengen S_i entsprechen.

Ein auf diesem Grundalgorithmus beruhendes Verfahren wurde von Cuthill und McKee (siehe Kapitel 8.4 in [DER86]) vorgestellt, bei dem zusätzlich die Knoten in den Mengen S_i angeordnet werden. Hierzu wählt man zuerst die Knoten, die adjazent zum ersten Knoten der Menge S_{i-1} sind, dann alle Knoten, die adjazent zum zweiten Knoten in S_{i-1} sind usw. Die Reihenfolge der Knoten ergibt sich, indem man die Menge $S_i = \{v_1, v_2, \ldots, v_m\}$ mit der Liste $S_i = (v_1, v_2, \ldots, v_m)$ gleichsetzt.

Eingabe: Graph $G = (V, E)$
$v :=$ geeigneter Startknoten;
$i := 1$; $S_1 = \{v\}$;
while $S_i \neq \emptyset$ **do**
  $S_{i+1} := \emptyset$;
  for all $u \in S_i$ **do**
    for all Nachbarknoten w von u **do**
      if w noch nicht besucht **then** $S_{i+1} := S_{i+1} \cup \{w\}$;
  endfor; **endfor**;
  $i := i + 1$;
endwhile;

Algorithmus 4.1: Konstruktion der Levelsets

Wird die Reihenfolge der berechneten Permutation aus dem Verfahren von Cuthill und McKee umgekehrt, gewinnt man oft eine Anordnung der Matrix mit einem kleineren Profil. Der so entstehende Algorithmus wird *Reverse Cuthill-McKee-* oder kurz *RCM-Algorithmus* genannt. In Abbildung 4.1 ist ein Beispiel für die Anordnung einer Matrix mittels RCM dargestellt. Der Startknoten ist hierbei der Knoten "1".

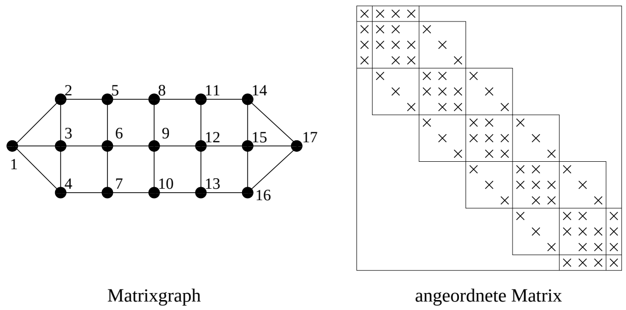

Abbildung 4.1: Beispiel für Reverse Cuthill-McKee

Bemerkung 4.2.2 Der Lauf durch den Matrixgraphen im RCM-Algorithmus entspricht einer Breitensuche. Entsprechend beträgt die minimale Anzahl von Kanten zwischen Knoten in der Menge S_i und dem Startknoten $i-1$.

Innerhalb des RCM-Algorithmus ist die Wahl des Startknotens entscheidend für die Bandbreite der angeordneten Matrix. Wählt man in dem Beispiel von Abbildung 4.1 Knoten "10" als Startknoten, so ergeben sich die Levelsets $S_1 = \{10\}, S_2 = \{7, 9, 13\}, S_3 = \{4, 6, 8, 12, 16\}, S_4 = \{1, 3, 5, 11, 15, 17\}$ und $S_5 = \{2, 14\}$. Die entsprechend angeordnete Matrix ist in Abbildung 4.2 zu sehen. Deutlich erkennbar ist die größere Bandbreite im Vergleich zur Matrix in Abbildung 4.1.

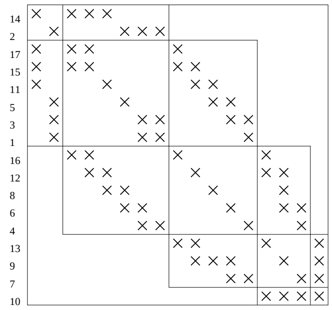

Abbildung 4.2: Beispiel für eine ungünstige Wahl des Startknotens

Gesucht ist also ein Startknoten, der die Anzahl der Mengen S_i maximiert und somit die Kardinalität jedes einzelnen Levelsets minimiert. Ein iteratives Verfahren zum Auffinden eines geeigneten Knotens verwendet zunächst einen beliebigen Startknoten und konstruiert zu diesem die Mengen S_i. Dann wird für alle Knoten v in der zuletzt konstruierten Menge S_k untersucht, ob eine Anwendung des RCM-Algorithmus mit v als Startknoten mehr als k Mengen S_i' produziert. Ist dies der Fall, so wird das Verfahren mit diesem Knoten und den Mengen S_i' wiederholt, bis keine Verbesserung mehr eintritt.

Dieser Algorithmus läßt sich kombinieren mit einem Verfahren, bei dem Knoten mit minimalem Grad betrachtet werden. Hierzu wählt man in der Menge S_k nur den Knoten als möglichen Kandidaten für einen neuen Startknoten, der den kleinsten Grad aller Knoten in S_k besitzt. Auf diese Weise verringert sich der Aufwand für die Suche nach einem passenden Startknoten. Man beachte aber, daß der von beiden Verfahren ermittelte Startknoten nicht notwendigerweise die optimale Wahl darstellt.

Algorithmus 4.2 faßt die obigen Erläuterungen zusammen. Dabei wird bei der Bestimmung eines passenden Startknotens von Bemerkung 4.2.2 Gebrauch gemacht, indem das Maximum der minimalen Kantenzahl vom Startknoten zu allen Knoten im Graphen und die dazugehörige Knotenmenge betrachtet wird, um die Anzahl der entstehenden Levelsets zu berechnen.

procedure StartNode($G = (V, E)$)
    sei $v \in V(G)$; $l_{old} = \infty$;
    while true **do**
        bfs(G, v); $l := \max\{$minimale Kantenzahl von v zu u $: u \in V(G)\}$;
        if $l > l_{old}$ **then**
            $l_{old} := l$;
            $v :=$ Knoten u mit $u.dist = l$ und $d(u) = \min\{d(w) : w \in V(G)\}$;
        else
            v ist passender Startknoten; **return** v;
        endif;
endwhile; **end**;

Algorithmus 4.2: Bestimmung eines geeigneten Startknotens

Abschließend ist in Abbildung 4.3 die Anwendung des RCM-Algorithmus an einer Systemmatrix dargestellt, die sich aus der in Abschnitt 2.3 beschriebenen Diskretisierung der Konvektions-Diffusions-Gleichung (2.1) ergibt. Zum Vergleich wurde die aus der Verfeinerung stammende Anordnung gegenübergestellt.

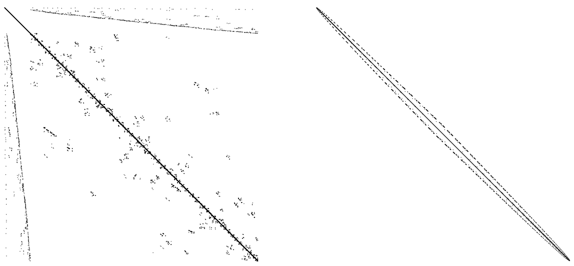

Abbildung 4.3: Matrix ohne und mit RCM-Anordnung

4.3. Anordnungsalgorithmen für konvektionsdominante Probleme

In diesem Abschnitt sollen Algorithmen vorgestellt werden, die speziell auf Probleme mit starker Konvektion zugeschnitten sind. Den Ausgangspunkt bildet hierbei die Systemmatrix $A = A^C + A^D$ (siehe Abschnitt 2.3) bzw. deren Konvektionsanteil A^C. Die wesentliche Idee der folgenden Algorithmen ist, die Unbekannten derart umzuordnen, daß ihre Anordnung der Konvektionsrichtung folgt. Im Idealfall ergibt sich hierbei eine Dreiecksmatrix, die in einem Schritt gelöst werden kann.

Zunächst werden in Abschnitt 4.3.1 die Graphen definiert, die als Eingabe für die Anordnungsalgorithmen genutzt werden. Dann erfolgt in Abschnitt 4.3.2 eine Vorstellung von

Algorithmen für Probleme im $\mathbb{R}^2$. In Abschnitt 4.3.3 wird eine Methode eingeführt, die es ermöglicht, diese 2D-Algorithmen auf Probleme im $\mathbb{R}^3$ anzuwenden.

Einen anderen Ansatz als die in diesem Abschnitt vorgestellten Algorithmen bzw. Anordnungen, die jeweils Gebrauch von der Konvektionsrichtung machen, findet man in [Tur92]. Dort werden verschiedene Strategien untersucht, die auf Gitteranordnungen beruhen.

4.3.1. Reduktion des Matrixgraphen

Ein erster Schritt in den folgenden Algorithmen besteht in der Konstruktion eines Graphen aus der Matrix A. Hierzu werden die (gewichteten) Matrixgraphen $G = G_W(A)$ (siehe auch Abschnitt 3.1.2 und 3.1.3) aufgestellt. Weiterhin macht man von der Beobachtung Gebrauch, daß große Außerdiagonaleinträge der Matrix A^C Kanten im Gitter entsprechen, die einen kleinen Winkel zu der Konvektionsrichtung einnehmen. Aufgrund der gewählten Diskretisierung fallen diese Gitterkanten mit Kanten in G zusammen und bilden somit die Hauptinformationsträger für die Konvektion. Aus diesem Grund wird der Graph auf sie beschränkt.

Zur Definition der reduzierten Graphen werden verschiedene Kantenmengen genutzt:

Definition 4.3.1 Seien $E_i \subseteq E$, $i \in \{0,1\}$, definiert durch

$$\begin{aligned} E_0 &= \{(v_i,v_j) \in E : a_{ij} > \kappa_0 a_{ii}\}, \quad 0 < \kappa_0 < 1, \quad \text{und} \\ E_1 &= \{(v_i,v_j) \in E : a_{ij} > \kappa_1 a_{ji}\}, \quad 1 \leq \kappa_1. \end{aligned}$$

Die Graphen $G_i = G_i(G) = (V, E_i)$, $i \in \{0,1\}$, seien die Untergraphen von G bzgl. der Mengen E_i.

Eine Kante $e \in E$ heißt *stark* (bzgl. E_i), falls $e \in E_i$ und *schwach*, falls $e \in E \setminus E_i$.

In Definition 4.3.1 werden nicht die Einträge von A^C sondern von A betrachtet, da sich die Beobachtungen über die großen Matrixeinträge von A^C unter der Voraussetzung einer dominanten Konvektion auf A übertragen.

In den folgenden Abschnitten dient, sofern nicht anders definiert, stets einer der Graphen G_i als Eingabe für die Graphenalgorithmen.

4.3.2. Anordnungen im $\mathbb{R}^2$

Die Grundlage der in diesem Abschnitt vorgestellten Anordnungen bilden die Verfahren aus Abschnitt 3.2.1 und 3.2.2 zur Bestimmung von Feedbackknoten. Dabei werden verschiedene Strategien verfolgt. In Abschnitt 4.3.2.2 wird das berechnete FVS direkt genutzt, um die Matrix geeignet umzuordnen. Dagegen werden in Abschnitt 4.3.2.3 die in Algorithmus 3.8 berechneten Zykel verwendet.

In Abschnitt 4.3.2.4 ist ein Verfahren angegeben, welches einen gegebenen planaren Graphen mit OFD-Bedingung vor der Anwendung eines der oben genannten Algorithmen reduziert, indem benachbarte Knoten zusammengefaßt werden.

4.3.2.1. Ein Numerierungsalgorithmus

Ein wesentlicher Bestandteil der folgenden Verfahren bildet Algorithmus 3.7, im folgenden auch *Numerierungsalgorithmus* genannt. In Abschnitt 3.2.2.2 diente er der Bestimmung von Knoten, die auf Zykeln liegen bzw. auf Pfaden zwischen Zykeln. Er läßt sich aber auch zur Berechnung einer Anordnung verwenden, bei der die Matrix die folgende Gestalt annimmt:

$$\begin{pmatrix} F & * & * \\ 0 & C & * \\ 0 & 0 & L \end{pmatrix}. \tag{4.2}$$

Dabei sind F und L obere Dreiecksmatrizen, die die Knoten mit Markierung F (von *first*) und L (von *last*) beinhalten. Innerhalb des Blockes F werden die Knoten entsprechend der Reihenfolge angeordnet in der sie beim Durchlauf des Numerierungsalgorithmus auftreten. Im Block L wird die Reihenfolge der Knoten mit Markierung L dagegen genau umgekehrt. Der Block C repräsentiert schließlich alle Knoten mit Markierung C (von *cyclic*). Die Reihenfolge der Knoten in diesem Block entspricht wieder der Reihenfolge ihres Auftretens innerhalb des Numerierungsalgorithmus. Ist der Graph azyklisch, verschwindet der Block C und die Matrix ist eine obere Dreiecksmatrix.

In [BW97] ist ein sog. *Downwind-Numbering*-Verfahren beschrieben, welches ein ähnliches Resultat wie der Numerierungsalgorithmus liefert: Ist der Matrixgraph der Systemmatrix A azyklisch, so kann A in Dreieckgestalt gebracht werden. Hierbei wird versucht, eine maximale Anzahl von Knoten mit der Markierung L zu versehen. Dazu dient die Menge aller Knoten ohne eingehende Kanten als Eingabe für Prozedur `SetMarks` in Algorithmus 3.7. In einem azyklischen Graphen gelingt auf diese Weise die Markierung aller Knoten. Besitzt der Matrixgraph dagegen Zykel, so wird statt dessen der (azyklische) Graph der starken Zusammenhangskomponenten betrachtet (siehe Definition 3.1.20). Diese Anordnung führt allerdings zu einer feineren Blockung von A als in (4.2), da die Knoten einer Zusammenhangskomponente jeweils als Block betrachtet werden.

4.3.2.2. Anordnungen mittels FVS

Bei einer Anordnung mittels FVS werden die Feedbackknoten in einem Block zusammengefaßt. Aufgrund der Definition eines FVS F eines Graphen G ist der Restgraph $G - F$ azyklisch. Faßt man die Knoten in diesem Restgraphen ebenfalls zu einem Block zusammen, so läßt sich dieser mit Hilfe des Numerierungsalgorithmus in eine obere Dreiecksmatrix umordnen. Insgesamt ergibt sich somit folgende Gestalt für die Matrix:

$$\begin{pmatrix} F & C \\ D & \hat{A} \end{pmatrix}, \tag{4.3}$$

wobei F den Block mit den Feedbackknoten und $\hat{A}$ den Block mit den Knoten des Restgraphen darstellt. Ist der Block F leer, so entspricht die entstehende Anordnung der durch den Numerierungsalgorithmus berechneten Permutation. Ein Beispiel für die Anwendung der Anordnung mittels FVS findet sich in Abbildung 4.4.

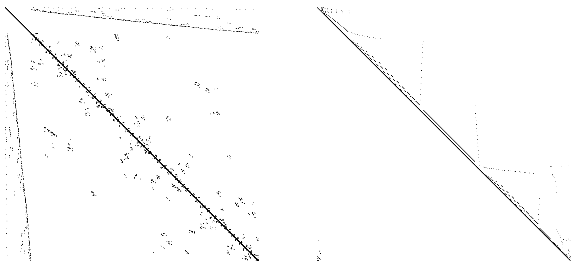

Abbildung 4.4: Matrix ohne und mit FVS-Anordnung

Für die Bestimmung eines FVS kann einer der Algorithmen aus den Abschnitten 3.2.1 oder 3.2.2 genutzt werden. Da für das heuristische FVS-Verfahren lediglich der Matrixgraph vorliegen muß, kann eine auf diesem Algorithmus basierende Anordnungstechnik auch für Probleme aus dem $\mathbb{R}^3$ genutzt werden, ohne die in Abschnitt 4.3.3 beschriebene Konstruktion von 2D-Oberflächen zu benutzen. In diesem Fall wird dann von einem *globalen FVS* gesprochen.

In [RR96] ist eine Weiterentwicklung des Downwind-Numbering-Algorithmus aus Abschnitt 4.3.2.1 beschrieben, welche ebenfalls ein FVS zur Bestimmung einer geeigneten Anordnung benutzt. Im zyklischen Fall wird dabei allerdings nicht der Graph der starken Zusammenhangskomponenten betrachtet, sondern aus der Menge der Knoten mit Markierung C geeignete Feedbackknoten bestimmt. Diese werden aus dem Graphen entfernt und das Verfahren nach einer erneuten Bestimmung der Knotenmarkierungen wiederholt. Im Gegensatz zu (4.3) erfolgt aber eine Trennung der Knoten, die in jedem Iterationsschritt markiert werden.

4.3.2.3. Anordnungen mittels konzentrischer Zykel

Die in diesem Abschnitt beschriebene Anordnung nutzt die durch Algorithmus 3.8 berechneten minimalen Zykel $\zeta_0, \ldots, \zeta_k$. Vor der Konstruktion eines Zykels wird in Algorithmus

3.8 der Graph auf die Knoten reduziert, die auf Zykeln bzw. auf Pfaden zwischen Zykeln liegen. Hierzu kann der Numerierungsalgorithmus 3.7 genutzt werden.

Seien F_i und L_i, $0 \leq i \leq k$ die Mengen der Knoten mit Markierung F bzw. L, die *vor* der Konstruktion des Zykels ζ_i durch den Numerierungsalgorithmus markiert werden. Die nach dem Entfernen des letzten Zykels verbleibenden Knoten seien dann, entsprechend ihrer Markierung durch den Numerierungsalgorithmus, in den Mengen F_{k+1} und L_{k+1} enthalten. Seien weiterhin $C_i = V(\zeta_i)$ mit $0 \leq i \leq k$ die Mengen der Knoten auf den minimalen Zykeln. Die Anordnung der Knoten innerhalb der Mengen F_i, L_i und C_i entspreche der in Abschnitt 4.3.2.1 beschriebenen Sortierung. In der Matrix A^C führt dies zu zyklisch bidiagonalen Blöcken C_i. In $A^C + A^D$ nehmen die Blöcke C_i eine zyklisch tridiagonale Gestalt an.

Die entstehenden Mengen F_i, L_i und C_i können außerdem untereinander verschieden angeordnet werden. Mögliche Beispiele sind

$$F_0, \ldots, F_{k+1}, C_0, \ldots, C_k, L_{k+1}, \ldots, L_0 \tag{4.4}$$

oder

$$F_0, C_0, F_1, C_1 \ldots, F_k, C_k, F_{k+1}, L_{k+1}, \ldots, L_0.$$

In Abbildung 4.5 ist die Anordnung (4.4) in einem Beispiel dargestellt.

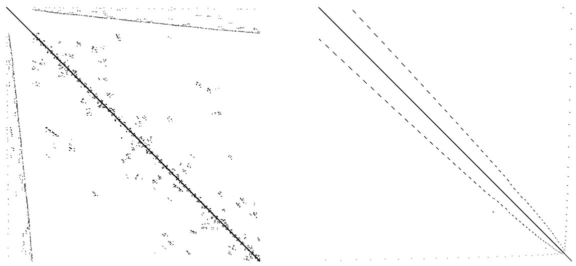

Abbildung 4.5: Matrix ohne und mit Anordnung mittels konzentrischer Zykel

Wie in Abschnitt 4.3.2.2, so gilt auch bei der Anordnung mittels konzentrischer Zykel: Bei einem azyklischen Graphen entspricht die entstehende Anordnung der durch den Numerierungsalgorithmus berechneten Permutation.

4.3.2.4. Planare Reduktion

Ist der Graph $G = G_w(A) = (V, E, w_V, w_E)$ planar und erfüllt zusätzlich die OFD-Bedingung (siehe Definition 3.2.9), so läßt sich der Graph reduzieren. Die Idee hierbei ist, die "Struktur" des Graphen, die durch die Konvektion definiert wird, mit wenigen Knoten zu charakterisieren. So überträgt sich etwa eine kreisförmige Konvektion auf den Graphen in Abbildung 4.6 (links), dort als konzentrische Zykel erkennbar. Die Information über die Konvektion läßt sich in einem einzigen Zykel ausdrücken. Hierzu faßt man Knoten, die durch eine schwache Kante, normal zur Konvektion, miteinander verbunden sind, zu einem Knoten zusammen (Abbildung 4.6, rechts). Bei diesem Verschmelzen der Knoten darf allerdings die OFD-Bedingung nicht verletzt werden.

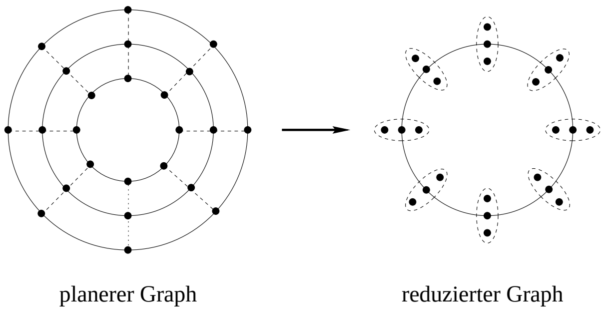

Abbildung 4.6: Planare Reduktion

Für eine allgemeine Definition der planaren Reduktion sei $G_r = (V, E_r)$ einer der reduzierten Graphen aus Definition 4.3.1 und $v, u \in V$ Knoten, die in G durch schwache Kanten $(v,u),(u,v) \in E \setminus E_r$ miteinander verbunden sind. Der Graph G'_r, der durch das Verschmelzen der Knoten v und u aus G_r entsteht, sei definiert durch:

$$\begin{aligned} V(G'_r) &= V - \{u\} \quad \text{und} \\ E(G'_r) &= \Big[E_r \cap \Big(V(G'_r) \times V(G'_r)\Big)\Big] \cup E', \end{aligned}$$

wobei

$$E' = \{(v,w) \,:\, (u,w) \in E_r\} \cup \{(w,v) \,:\, (w,u) \in E_r\}.$$

Man beachte, daß nur Knoten zusammengefaßt werden, die durch eine schwache Hin- und Rückkante miteinander verbunden sind.

Aufgrund der Konstruktion von G'_r folgert man, daß mit G_r auch G'_r ein planarer Graph ist, womit eine Eigenschaft für die Anwendung der Algorithmen in Abschnitt 3.2.2 bestehen bleibt. Für den Erhalt der OFD-Bedingung in G'_r sind in der folgenden Bemerkung hinreichende Kriterien aufgelistet.

Bemerkung 4.3.2 G'_r erfüllt die OFD-Bedingung falls die Knoten v und u eine der folgenden Bedingungen erfüllen:

i) In dem Graphen $G_r \cup \{(v,u),(u,v)\}$ gilt:

$$\begin{array}{ccc} (e_{left}(v) = (v,u) & \text{und} & e_{right}(u) = (u,v)) \\ & \text{oder} & \\ (e_{right}(v) = (v,u) & \text{und} & e_{left}(u) = (u,v)). \end{array}$$

ii) Falls $d_e(u, G_r) = 0$ dann erfüllt v in $G_r \cup \{(v,u)\}$ die OFD-Bedingung.

iii) Falls $d_a(u, G_r) = 0$ dann erfüllt v in $G_r \cup \{(u,v)\}$ die OFD-Bedingung.

In Abbbildung 4.7 sind die Bedingungen i) und iii) an einem Beispiel dargestellt.

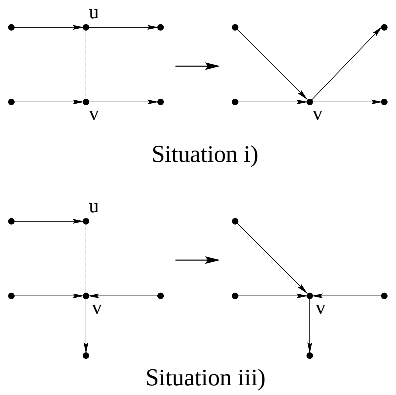

Abbildung 4.7: Beispiele für das Verschmelzen von Knoten

Nach Definition der OFD-Bedingung ließen sich die zu v in G_r adjazenten Kanten $(e_1, \ldots, e_m, \ldots, e_k)$ nach ihrem Winkel zu einer festen Achse ordnen, wobei $e_{right}(v) = e_1$ und $e_{left}(v) = e_m$. Diese Anordnung fließt in eine weitere Bedingung für das Verschmelzen der Knoten, da die inzidierende, schwache Kante normal zur Konvektion liegen soll. Somit folgt, daß die Kante (v,u) (bzw. (u,v)) zwischen den Kanten e_m und e_{m+1} (linker Sektor) oder zwischen e_k und e_1 (rechter Sektor) liegen muß (siehe auch Abbildung 4.8). Weiterhin wird die Anzahl der Knoten, die mit v zusammengefaßt werden, auf maximal einen zu v adjazenten Knoten pro Sektor beschränkt. In dem Beispiel in Abbildung 4.8 folgt somit, daß höchstens ein Knoten l_i und ein Knoten r_i mit v zusammengefaßt werden können. Da durch die Verschmelzung mit einem Nachbarn neue, schwache, mit v inzidente Kanten entstehen können, wird dieser gesamte Prozeß erneut auf v angewandt.

Sei $V(v) = \{r_t, \ldots, r_1, v, l_1, \ldots, l_{z-1}, l_z\}$ die Menge aller Knoten (plus v), die mit v verschmolzen wurden. Hierbei seien r_i die Knoten, die im rechten Sektor reduziert wurden entsprechend ihrer Reihenfolge (r_1 der erste Knoten usw.). Analog seien l_i die Knoten, die im linken Sektor mit v verschmolzen wurden. Diese Knotenmenge repräsentiert einen Block in der Matrix A, indem die Knoten entsprechend ihres Zusammenhangs über schwache Kanten angeordnet werden. Es ergibt sich somit die Reihenfolge: $r_t, \ldots, r_1, v, l_1, \ldots, l_z$.

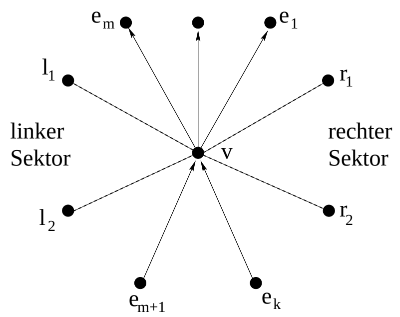

Abbildung 4.8: Zulässige Knoten für das Verschmelzen

Nachdem in G_r alle Knoten durch obige Konstruktion mit ihren Nachbarn zusammengefaßt wurden, ergibt sich ein planarer Graph, der die OFD-Bedingung erfüllt. Auf diesen Graphen läßt sich einer der beschriebenen Anordnungsalgorithmen (FVS oder mit konzentrischen Zykeln) anwenden. Allerdings ist die resultierende Anordnung nicht eindeutig, sondern abhängig von der Reihenfolge der Knoten innerhalb des Verschmelzungsalgorithmus. Ein Beispiel für die Anwendung der planaren Reduktion findet sich in Abbildung 4.9. Die "Grundstruktur" des Graphen entspricht dabei der in Abbildung 4.6 gezeigten Struktur. Nach der Reduktion wurde ein FVS-Algorithmus genutzt, um eine Anordnung für die Matrix zu berechnen. Das Ergebnis ist ebenfalls dargestellt.

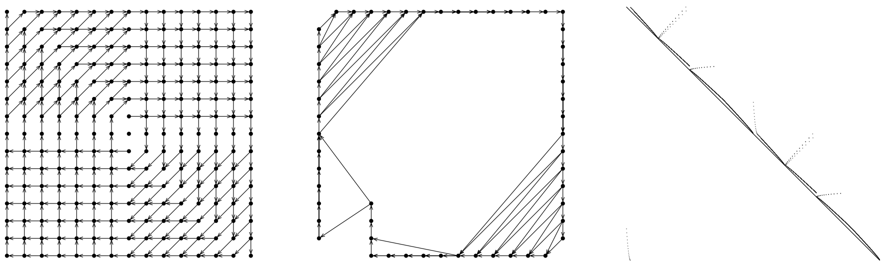

Abbildung 4.9: Graph vor und nach planarer Reduktion und angeordnete Matrix

4.3.3. Anordnungen im $\mathbb{R}^3$

Mit der in diesem Abschnitt diskutierten Technik soll versucht werden, innerhalb eines dreidimensionalen Gebietes zweidimensionale Oberflächen zu konstruieren, die die Struktur der Konvektion wiedergeben. Auf diese Oberflächen können dann die Anordnungstechniken aus Abschnitt 4.3.2 angewandt werden.

4.3.3.1. Oberflächen in Tetraedergittern

Im folgenden wird stets ein Tetraedergitter $\mathcal{T} = \mathcal{T}(V, E, F)$ mit einer Knotenmenge $V(\mathcal{T})$, einer Kantenmenge $E(\mathcal{T})$ und einer Menge von Dreiecksflächen $F(\mathcal{T})$ betrachtet.

Zunächst eine Definition bezüglich der verwendeten Notation.

Definition 4.3.3 i) Zwei Flächen $f, g \in F(\mathcal{T})$ heißen *benachbart*, falls sie eine gemeinsame Kante $e \in E(\mathcal{T})$ besitzen: $f \cap g = e$. Man sagt auch, die Flächen f und g sind mit der Kante e *verbunden*.

ii) Sei $e \in E(\mathcal{T})$. Der *Grad* $d(e)$ von e sei definiert als die Anzahl der Flächen in $\mathcal{T}$, die mit e verbunden sind: $d(e) := |\{f \in F(\mathcal{T}) \, : \, e \cap f \neq \emptyset\}|$.

In der folgenden Definition sind die grundlegenden Eigenschaften von Oberflächen als Teilmenge des Tetraedergitters zusammengefaßt.

Definition 4.3.4 Sei $\mathcal{S} \subseteq \mathcal{T}$. $\mathcal{S}$ heißt *Oberfläche*, falls die folgenden Bedingungen erfüllt sind:

i) Für alle $e \in E(\mathcal{S})$ gilt: $d(e) \leq 2$.

ii) Für alle $f, g \in F(\mathcal{S})$ existieren Flächen $f_0, \ldots, f_k \in F(\mathcal{S})$ mit $f = f_0$, $g = f_k$ und für alle $0 \leq i \leq k-1$ gilt: f_i und f_{i+1} sind benachbart.

iii) Für alle $f, g \in F(\mathcal{S})$ mit einem gemeinsamen Knoten v existieren Flächen $f_0, \ldots, f_k \in F(\mathcal{S})$ mit $f = f_0$, $g = f_k$, f_i und f_{i+1} sind benachbart für alle $0 \leq i \leq k-1$ und $v \in \bigcap_{i=0}^{k} f_i$.

Der *Graph einer Oberfläche* $G(\mathcal{S})$ wir durch die Knoten und Kanten von $\mathcal{S}$ gebildet: $G(\mathcal{S}) = (V(\mathcal{S}), E(\mathcal{S}))$.

Bedingung (i) verhindert, daß mit jeder Kante mehr als 2 Flächen verbunden sind und sich die Oberfläche damit in mehr als einer Ebenen ausbreitet. Bedingung (ii) sorgt für den Zusammenhang der Oberfläche über Kanten und Bedingung (iii) schließt "Löcher" in der Oberfläche aus. In Abbildung 4.10 sind ein Beispiel für eine Oberfläche und drei Gegenbeispiele angegeben, wobei in den Gegenbeispielen (i) - (iii) die jeweilige Bedingung aus Definition 4.3.4 verletzt ist.

Zusätzlich zu den in Definition 4.3.4 angegebenen Eigenschaften einer Oberfläche wird im folgenden auch stets von deren *Orientierbarkeit* ausgegangen. Gemeint ist hierbei die Möglichkeit, einer Oberfläche eine eindeutige obere und untere Seite zuweisen zu können.

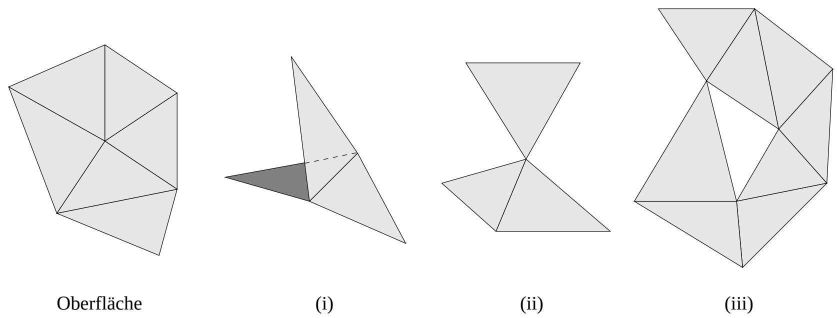

Abbildung 4.10: Ein Beispiel für eine Oberfläche und drei Gegenbeispiele

Hierdurch sind Strukturen ähnlich einem Möbiusband ausgeschlossen. Der Grund für diese Forderung liegt in der Anwendbarkeit der Algorithmen aus Abschnitt 3.2.2, die die Planarität der betrachteten Graphen zugrundelegen. Da durch die Orientierbarkeit eine Einbettung in den $\mathbb{R}^2$ gelingt, läßt sich hierdurch die Planarität des Graphen der Oberfläche sicherstellen. Eine algorithmische Betrachtung zum Test der Orientierbarkeit einer Oberfläche findet man in Abschnitt 4.3.3.4.

4.3.3.2. Flußoberflächen

Im folgenden sollen orientierbare Oberflächen definiert werden, die die Hauptinformation über den Fluß tragen, die sogenannten *Flußoberflächen*. Der Grundgedanke dabei ist, der Richtung der Konvektion im Mittelpunkt einer Kante zu folgen und ihr so ein "geeignetes", adjazentes Dreieck zuzuordnen. Die Konvektionsrichtung $b : \mathbb{R}^3 \to \mathbb{R}^3$ muß hierfür explizit vorliegen.

Definition 4.3.5 Sei $e \in E(\mathcal{T})$ und seien $v, u \in V(\mathcal{T})$ die Endpunkte von e. Bezeichne $m = m(e) = \frac{1}{2}(v + u)$ den Mittelpunkt der Kante e.

Seien $\sigma \in \mathbb{R}_{>0}$ und $\kappa \in [0, \pi]$. Man nennt e *stark*, falls für die Konvektionsrichtung b im Kantenmittelpunkt gilt: $\|b(m)\|_2 > \sigma$. Im Fall $\|b(m)\|_2 \leq \sigma$ heißt e auch *schwach*. Gilt für den Winkel zwischen e und $b(m)$: $\measuredangle(e, b(m)) < \kappa$ für ein $\kappa \in [0, \frac{\pi}{2}]$, so bezeichnet man e als *flußparallel*.

Sei $\tilde{E}(\mathcal{T}) \subseteq E(\mathcal{T})$ die Menge aller starken Kanten mit einem Grad größer als 1. Weiter sei $\tilde{E}_1(\mathcal{T}) \subseteq \tilde{E}(\mathcal{T})$ die Menge aller nicht flußparallelen Kanten in $\tilde{E}(\mathcal{T})$ und $\tilde{E}_2(\mathcal{T}) = \tilde{E}(\mathcal{T}) \setminus \tilde{E}_1(\mathcal{T})$.

Im folgenden wird jeder Kante in $\tilde{E}$ durch die Funktionen $\varphi_f, \varphi_b : \tilde{E}_1(\mathcal{T}) \to F(\mathcal{T})$ und $\varphi_p : \tilde{E}_2(\mathcal{T}) \times F(\mathcal{T}) \to F(\mathcal{T})$ mittels der Konvektionsrichtung ein mit e verbundenes Dreieck zugewiesen. Die Indizes f, b und p stehen dabei für *forward*, *backward* und *parallel* und entsprechen der Richtung, in der der Konvektion gefolgt wird (*vorwärts* und *rückwärts*) bzw. der Art der Kante (flußparallel).

Zur Definition von φ_f, φ_b und φ_p sei $\gamma \in [0, \frac{\pi}{2})$. Für eine Kante $e \in \tilde{E}_1(\mathcal{T})$ sei $\varphi_f(e)$ definiert als die mit e verbundene Fläche, die den kleinsten Winkel zur Ebene einnimmt, die von e und der Konvektionsrichtung $b(m(e))$ aufgespannt wird. Vorausgesetzt wird hierbei, daß dieser Winkel kleiner ist als γ. Entsprechend ist $\varphi_b(e)$ definiert als die Fläche, die den kleinsten Winkel zu der von e und der negativen Konvektionsrichtung $-b(m(e))$ aufgespannten Ebene einnimmt, sofern dieser kleiner als γ ist.

Man beachte, daß die Richtung der Konvektion (positiv oder negativ) entscheidend in die Berechnung des Winkels α zwischen der Dreiecksfläche und der jeweiligen Ebene eingeht, da hierfür sowohl α als auch $\pi - \alpha$ in Frage kommen. In Abbildung 4.11 ist ein Beispiel angegeben. Der Winkel zwischen $b(m(e))$ und f beträgt dort 0 und somit ist $\varphi_f(e) = f$. Dagegen nimmt der Winkel zwischen $b(m(e))$ und g den Wert π an bzw. ist 0 zwischen $-b(m(e))$ und g, womit $\varphi_b(e) = g$ folgt.

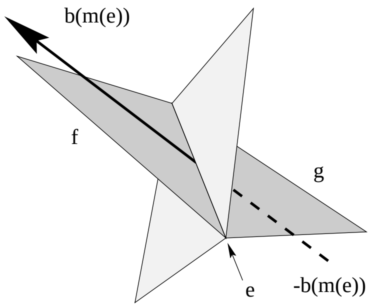

Abbildung 4.11: Beispiel für φ_f und φ_b

Für Kanten $e \in \tilde{E}_2(\mathcal{T})$ ist die Definition einer Ebene zwischen e und $b(m(e))$ aus numerischer Sicht nicht sinnvoll oder nicht eindeutig. Aus diesem Grunde wird der Wert von $\varphi_p(e,g)$ bezüglich einer mit e verbundenen Fläche $g \in F(\mathcal{T})$ definiert und sei die mit e verbundene Dreiecksfläche, für deren Winkel α zu g gilt: $|\alpha - \pi| \leq \gamma$ und $|\alpha - \pi| = \min\{|\pi - \measuredangle(g,f)|, f \text{ ist mit } e \text{ verbunden}\}$. Dabei ist der maximale Winkel zwischen den durch die Dreiecksflächen definierten Ebenen gemeint. Das Beispiel in Abbildung 4.11 verdeutlicht diese Definition. Darin gilt: $\varphi_p(e,g) = f$.

Durch die Funktionen φ_f, φ_b und φ_p läßt sich die Menge der Dreiecke im Tetraedergitter einschränken. Sei hierzu

$$\begin{aligned}
\tilde{F}_f(\mathcal{T}) &= \varphi_f(\tilde{E}_1(\mathcal{T})) \\
\tilde{F}_b(\mathcal{T}) &= \varphi_b(\tilde{E}_1(\mathcal{T})) \\
\tilde{F}_p(\mathcal{T}) &= \varphi_p(\tilde{F}_f(\mathcal{T}) \times \tilde{E}_2(\mathcal{T})) \cup \varphi_p(\tilde{F}_b(\mathcal{T}) \times \tilde{E}_2(\mathcal{T})).
\end{aligned}$$

Die Menge

$$\tilde{F}(\mathcal{T}) = \tilde{F}_f(\mathcal{T}) \cup \tilde{F}_b(\mathcal{T}) \cup \tilde{F}_p(\mathcal{T}) \tag{4.5}$$

läßt sich nach obigen Definitionen als die Menge der “im Fluß liegenden” Dreiecke betrachten und bildet die Grundlage für die Flußoberflächen.

Definition 4.3.6 Sei $\alpha \in [0, \pi]$ und sei $\mathcal{S} \subseteq \mathcal{T}$ eine orientierbare Oberfläche mit $F(\mathcal{S}) \subseteq \tilde{F}(\mathcal{T})$. $\mathcal{S}$ heißt *Flußoberfläche mit minimalen Winkel* α, falls für alle Kanten $e \in E(\mathcal{S})$ und mit e verbundenen Dreiecksflächen $f, g \in F(\mathcal{S})$ gilt: $\measuredangle(f, g) \geq \alpha$.

4.3.3.3. Konstruktion der Flußoberflächen

Die einzelnen Funktionen φ_f, φ_b und φ_p werden im folgenden zu einer *Flußfunktion* $\varphi : \tilde{E}(\mathcal{T}) \times F(\mathcal{T}) \to F(\mathcal{T})$ zusammengefaßt:

$$\varphi(e, f) = \begin{cases} \varphi_p(e, f) & , \quad e \in \tilde{E}_2(\mathcal{T}) \\ \varphi_f(e) & , \quad e \in \tilde{E}_1(\mathcal{T}) \text{ und } \measuredangle(\varphi_f(e), f) > \measuredangle(\varphi_b(e), f) \\ \varphi_b(e) & , \quad \text{sonst} \end{cases} \tag{4.6}$$

Die Konstruktion einer Flußoberfläche ist in Algorithmus 4.3 dargestellt. Hierbei werden, ausgehend von einer gegebenen Kante $e \in \tilde{E}_1(\mathcal{T})$, mittels φ sukzessive Dreiecksflächen an die Oberfläche angefügt, wobei die Eigenschaft einer Flußoberfläche erhalten bleiben soll. Eine mögliche Implementierung für diesen Test ist in Abschnitt 4.3.3.4 angegeben.

procedure Surface(e, f, $\mathcal{S}$)
    $g := \varphi(e, f)$;
    if $g \notin \mathcal{S}$ und $\mathcal{S} \cup \{g\}$ ist eine Flußoberfläche **then**
        $\mathcal{S} := \mathcal{S} \cup \{g\}$;
        for all Kanten $e' \neq e$ in g **do** Surface(e',g,$\mathcal{S}$);
endif; **end**;

Algorithmus 4.3: Algorithmus zur Konstruktion einer Flußoberfläche

Algorithmus 4.3 kann benutzt werden, um das gesamte Gitter $\mathcal{T}$ in disjunkte Flußoberflächen zu zerlegen. Das entstehende Verfahren ist in Algorithmus 4.4 angegeben.

procedure Decompose($\mathcal{T}$)
    $i := 0$;
    while $\tilde{E}_1(\mathcal{T}) \neq \emptyset$ **do**
        sei $e \in \tilde{E}_1(\mathcal{T})$ mit $f = \varphi_f(e) \in \tilde{F}(\mathcal{T})$;
        $\mathcal{S}_i = \{f\}$;
        for all Kanten e' in f **do** Surface(e', f, $\mathcal{S}_i$);
        lösche alle Dreiecksflächen in $\mathcal{T}$ die mindestens einen Knoten in $\mathcal{S}_i$ besitzen;
        $i := i + 1$;
endwhile; **end**;

Algorithmus 4.4: Algorithmus zur Zerlegung von $\mathcal{T}$ in Flußoberflächen

Ein Beispiel für die Anwendung von Algorithmus 4.4 findet sich in Abbildung 4.12. Auf den konstruierten Flußoberflächen wurden anschließend Anordnungen mittels konzentrischer Zykel (siehe Abschnitt 4.3.2.3) und FVS (siehe Abschnitt 4.3.2.2) berechnet. Die permutierten Matrizen sind ebenfalls dargestellt. Dabei wurde die Blockung der Matrix, die von der Aufteilung der Unbekannten durch die Flußoberflächen herrührt, ebenfalls dargestellt.

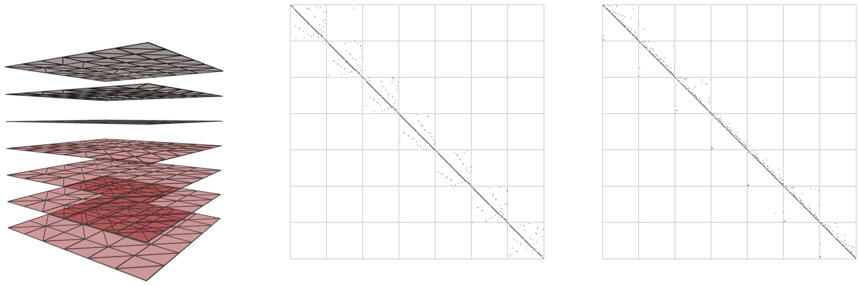

Abbildung 4.12: Konstruierte Flußoberflächen und Anordung mittels Zykel und FVS

4.3.3.4. Implementierung

Orientierung einer Oberfläche

In Abschnitt 4.3.3.1 wurde auf die Orientierbarkeit einer Oberfläche als wichtige Forderung für die Anwendbarkeit der Algorithmen in Abschnitt 3.2.2 hingewiesen. Die folgende Definition gibt eine Möglichkeit an, diese Bedingung algorithmisch sicherzustellen.

Definition 4.3.7 Sei $f \in F(\mathcal{T})$ und seien $v_0, v_1, v_2 \in V(\mathcal{T})$ mit $v_0, v_1, v_2 \in f$ die drei Eckpunkte von f. Eine *Orientierung* von f sei definiert durch eine Permutation (i_0, i_1, i_2) von $(0, 1, 2)$. Bezeichne $o(f)$ diese Permutation.

Sei $\mathcal{S} \subseteq \mathcal{T}$ eine Oberfläche. $\mathcal{S}$ ist orientierbar, falls für alle $f, g \in F(\mathcal{S})$ mit f und g sind benachbart und $o(f) = (i_0, i_1, i_2)$ und $o(g) = (j_0, j_1, j_2)$, gilt:

$$\exists\, l, k \in \{0, 1, 2\} \,:\, v_{i_l} = v_{j_k} \text{ und } (v_{i_{l+1}} = v_{j_{k-1}} \text{ oder } v_{i_{l-1}} = v_{j_{k+1}}). \tag{4.7}$$

Dabei ist die Addition bzw. Subtraktion in den Indizes modulo 3 zu verstehen.

Bedingung (4.7) gibt an, daß die Anordnung der Eckpunkte der Dreiecksflächen f und g bezüglich der jeweiligen Orientierung entgegengesetzt ist (siehe auch Abbildung 4.13). Durch die Orientierung eines Dreiecks ist eine äußere bzw. innere Normalenrichtung gegeben, etwa durch das Vektorprodukt $(v_{i_0} - v_{i_1}) \times (v_{i_2} - v_{i_1})$. Somit lassen sich die bereits angesprochene Unter- bzw. Oberseite des Dreiecks und der gesamten Oberfläche angeben.

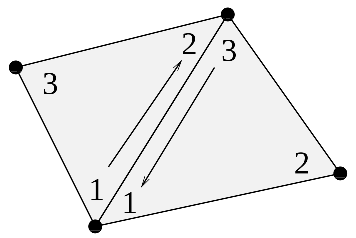

Abbildung 4.13: Beispiel für eine Orientierung benachbarter Dreiecke

Innerhalb von Algorithmus 4.3 wird dem Anfangsdreieck eine beliebige Orientierung zugewiesen. Jede neue Dreiecksfläche erhält dann eine Orientierung in Abhängigkeit von dem zuletzt besuchten Dreieck, entsprechend der Definition einer orientierbaren Oberfläche. Nach Konstruktion der gesamten Flußoberfläche läßt sich schließlich mittels Definition 4.3.7 überprüfen, ob die einzelnen Orientierungen der Dreiecke passend sind.

Test auf Flußoberfläche

Im folgenden seien für alle Kanten $e \in E(\mathcal{T})$ Datenfelder *noOfFaces* angenommen, die die Anzahl der mit e verbundenen Dreiecke innerhalb einer Flußoberfläche speichern. Analog gebe es für alle Knoten $v \in V(\mathcal{T})$ bool'sche Datenfelder *inSurface*, die `true` sind, falls v in einer Flußoberfläche enthalten ist.

Seien e die Kante und g das Dreiecke aus Algorithmus 4.3. Hinreichend für "$\mathcal{S} \cup \{g\}$ *ist eine Flußoberfläche*" in Algorithmus 4.3 sind die folgenden Bedingungen:

i) für alle Kanten $e' \neq e$ von g gilt: e'.noOfFaces < 2,

ii) es existiert eine Kante $e' \neq e$ von g mit e'.noOfFaces $= 1$ oder für alle Knoten v in g mit $v \notin e$ gilt: v.inSurface $=$ `false` und

iii) für alle Kanten $e' \neq e$ von g mit e'.noOfFaces $= 1$ gilt: für das mit e' verbundene Dreieck $f \neq g$ gilt: $\measuredangle(f, g) \geq \alpha$.

Bedingung i) entspricht der Bedingung i) aus Definition 4.3.4. Die Bedingung ii) sorgt für die Einhaltung der Bedingungen ii) und iii) aus Definition 4.3.4. Bedingung iii) ist eine direkte Umsetzung der Winkelbedingung für Flußoberflächen.

Man beachte, daß Bedingung ii) auch für die Orientierbarkeit der konstruierten Flußoberfläche sorgt. Dies folgt aus der Eigenschaft, daß alle Mengen $\mathcal{S}$ aus Algorithmus 4.3 während der Konstruktion Oberflächen sind und somit jedem neuen Dreieck eine geeignete Orientierung zugewiesen werden kann.

Dies bedeutet aber zugleich auch eine Einschränkung der konstruierten Flußoberflächen. Strukturen wie Bänder sind ausgeschlossen, da hierzu zum Schließen der Oberfläche während der Konstruktion ein Dreieck angefügt werden muß, wobei ein Knoten des neuen Dreiecks bereits in der Oberfläche enthalten ist. Hierdurch ist aber Bedingung ii) verletzt. Diese Situation ist in Abbildung 4.14 dargestellt.

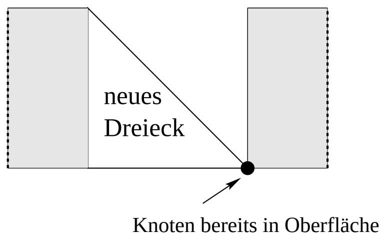

Abbildung 4.14: Situation beim Schließen einer Oberfläche

5. Iterationsverfahren

In diesem Kapitel sollen die in dieser Arbeit verwendeten Iterationsverfahren vorgestellt werden. Zum Einsatz kommen hierbei nichtlineare Verfahren auf Basis der konjugierten Gradienten. Diese gehören, neben den Mehrgitterverfahren, zu den am schnellsten konvergierenden Verfahren zur Lösung von großen, schwachbesetzten Gleichungssystemen.

Nachdem in Abschnitt 5.1 einige Begriffe eingeführt werden, erfolgt in Abschnitt 5.2 eine Vorstellung der Gauß-Seidel-Iteration. Diese wird in den Lösungsiterationen als *Präkonditionierer* verwendet. Anschließend werden in Abschnitt 5.3 das BiCG-Stab-Verfahren und in Abschnitt 5.4 das GMRES-Verfahren vorgestellt.

5.1. Allgemeines

Das durch die Iterationsverfahren zu lösende System sei im folgenden stets

$$Ax = b,$$

mit regulärem $A \in \mathbb{R}^{n \times n}$ und $b \in \mathbb{R}^n$.

Um die Spektraleigenschaften der Systemmatrix und somit auch das Konvergenzverhalten einiger Iterationsverfahren zu verbessern, wird häufig ein *Präkonditionierer* verwendet. Dies ist eine reguläre Matrix $W = W_1 W_2 \in \mathbb{R}^{n \times n}$. Dabei wird W so gewählt, daß sie eine Approximation von A darstellt, sich aber einfacher invertieren läßt, z.B. entspricht in der *Jacobi-Präkonditionierung* W der Diagonalen von A. Das mittels W transformierte Gleichungssystem lautet schließlich:

$$W_1^{-1} A W_2^{-1} (W_2 x) = W_1^{-1} b.$$

Durch die Zerlegung von W in W_1 und W_2 lassen sich zusätzliche Forderungen an das neue System erfüllen. Sind etwa A und W symmetrisch, so läßt sich durch die Wahl $W_1 = W_2^T$ diese Eigenschaft auch auf die Matrix $W_1^{-1} A W_2^{-1}$ übertragen.

5.2. Die Gauß-Seidel-Iteration

Die unterschiedlichen Varianten der Gauß-Seidel-Iteration basieren auf einer Zerlegung der Matrix A in

$$A = D - E - F, \tag{5.1}$$

wobei D den Diagonalanteil von A bildet und E und F strikte untere bzw. obere Dreiecksmatrizen sind. Ein Schritt der Iteration ist durch die Vorschrift

$$x_{i+1} = x_i - \vartheta\, W^{-1}(Ax_i - b) \tag{5.2}$$

definiert, wobei der Parameter $\vartheta \in \mathbb{R}$ einer geeigneten Dämpfung der Korrektur dient. Die Matrix W ist durch die einzelnen Varianten der Gauß-Seidel-Iteration bestimmt. Das *Vorwärts-Gauß-Seidel*-Verfahren ergibt sich durch die Wahl

$$W^{FGS} = D - E,$$

während die Matrix

$$W^{BGS} = D - F$$

der *Rückwärts-Gauß-Seidel*-Iteration entspricht. Führt man jeweils einen Vorwärts- und einen Rückwärts-Gauß-Seidel-Schritt aus, so ergibt sich das *symmetrische Gauß-Seidel*-Verfahren. Die Matrix W aus (5.2) entspricht hierbei der Matrix

$$W^{SGS} = (D - E)D^{-1}(D - F).$$

Bei den obigen Iterationen spricht man auch von der *punktweisen* Gauß-Seidel-Iteration. Besitzt die Matrix A eine Blockstruktur und sind die Matrizen D, E und F in der Zerlegung (5.1) als Blockdiagonale bzw. untere und obere Blockdreiecksmatrizen gewählt, so ergibt sich die *Block-Gauß-Seidel*-Iteration.

Betrachtet man die in Kapitel 4 vorgestellten Anordnungen für die Matrix A, so wird eine Eignung einzelner Gauß-Seidel-Varianten für die verschiedenen Anordnungen deutlich. Dies betrifft sowohl den Einsatz als Löser, Präkonditionierer oder als Glätter in einer Mehrgitteriteration.

Ist der Graph der Matrix A azyklisch, so ergibt sich durch den Numerierungsalgorithmus eine Anordnung, die A zu einer oberen Dreiecksmatrix umordnet (siehe Abschnitt 4.3.2.1). Hierfür bietet sich die Rückwärts-Gauß-Seidel-Iteration an. Auch bei der Anordnung mittels FVS (siehe Abschnitt 4.3.2.2) entsteht eine Anordung von A, für die sich die Rückwärts-Gauß-Seidel-Iteration eignet, sofern das FVS des Graphen klein ist. Hier wird auch die Notwendigkeit für Verfahren zur Bestimmung von minimalen bzw. sehr kleinen FVSs deutlich.

Das symmetrische Block-Gauß-Seidel-Verfahren ist hingegen für die Anordnung mittels konzentrischer Zykel (siehe Abschnitt 4.3.2.3) zu empfehlen, da hierbei tridiagonale Diagonalblöcke entstehen. Diese können jeweils effizient gelöst werden.

5.3. Das BiCG-Stab-Verfahren

Für positiv definite Systeme ist das *Verfahren der konjugierten Gradienten* oder *CG-Verfahren* (siehe [Hac93]) als Lösungsverfahren eine geeignete Wahl. Ein Grund hierfür liegt in der Minimierung der Fehlernorm

$$\|x - x_i\|_A$$

in jedem Iterationsschritt. Bei nichtsymmetrischen Systemen verliert man allerdings diese Minimierungseigenschaft. Ist die Systemmatrix A nicht positiv definit, so definiert $\|\cdot\|_A$ keine Norm und eine Minimierung von $\|x - x_i\|_A$ ist wenig sinnvoll.

Innerhalb der CG-Iteration lassen sich das aktuelle Residuum r_i und die Suchrichtung p_i jeweils durch Polynome vom Grad kleiner oder gleich i ausdrücken. Diese Polynome erfüllen ähnliche Orthogonalitätsbedingungen wie die Vektoren r_i und p_i bezüglich einer geeigneten Bilinearform. Durch eine spezielle Wahl dieser Bilinearform entsteht das *BiCG-Verfahren*. Allerdings besitzt das BiCG-Verfahren nicht die Minimierungseigenschaft der CG-Iteration. Außerdem erfolgen innerhalb des Algorithmus Multiplikationen mit der Transponierten A^T und im Falle einer Präkonditionierung auch mit dem transponierten Präkonditionierer.

Neben den Residuen r_i wird im BiCG-Verfahren eine weitere Folge $\hat{r}_i$ von Vektoren berechnet. Dabei beobachtet man im Falle der Konvergenz, daß die Vektoren r_i und $\hat{r}_i$ nach Null konvergieren, allerdings nur die Konvergenz von r_i ausgenutzt wird. Das *CGS-Verfahren* konzentriert durch eine Modifikation alle Anstrengungen auf die Minimierung von r_i. Hierbei wir die Darstellung von r_i und $\hat{r}_i$ im BiCG-Verfahren durch Polynome φ_i vom Grad kleiner oder gleich i ausgenutzt:

$$r_i = \varphi_i(A) r_0 \quad \text{und} \quad \hat{r}_i = \varphi_i(A^T) r_0'.$$

Dabei ist r_0' ein Vektor mit $\langle r_0, r_0' \rangle \neq 0$. Im CGS-Verfahren werden nun anstelle der $\hat{r}_i$ neue Vektoren $\tilde{r}_i$ berechnet:

$$\tilde{r}_i = \varphi_i^2(A) r_0.$$

Betrachtet man $\varphi_i(A)$ als *Reduktionsoperator* bzgl. der Norm von r_0, so wird dieser im CGS-Verfahren zweifach verwendet, wodurch sich auch der Name CG-*Squared* erklärt. Die Methode hat zudem den Vorteil, daß die Multiplikationen mit A^T nicht mehr erforderlich sind.

Sowohl das BiCG- als auch das CGS-Verfahren liefern bei exakter Arithmetik nach spätestens n Schritten die Lösung des Gleichungssystems. Allerdings haben beide Verfahren, u.a. bedingt durch das Fehlen einer Minimierungseigenschaft, ein sehr unregelmäßiges Konvergenzverhalten. Dieses ist beim CGS-Verfahren noch ausgeprägter als bei der BiCG-Iteration, wobei allerdings das CGS-Verfahren häufig schneller konvergiert. Auch besteht bei beiden Verfahren die Möglichkeit, daß zwei verschiedene *Breakdowns* auftreten.

Um das unregelmäßige Konvergenzverhalten zu glätten, kann man untersuchen wie $\varphi_i(A)$ als Reduktionsoperator wirkt. Hierbei stellt man fest, daß $\varphi_i(A)$ zwar bzgl. des Startresiduums r_0 ein Reduktionsoperator ist, daß dies aber nicht unbedingt auch für $\varphi_i(A)r_0$ gilt. Es lassen sich sogar Beispiele konstruieren, in denen $\varphi_i(A)r_0$ eine kleine Norm besitzt, $\varphi_i^2(A)r_0$ aber eine größere Norm als r_0 hat. Das *BiCG-Stab-Verfahren* benutzt deshalb nicht $\varphi_i(A)$ als Reduktor bzgl. $\varphi_i(A)r_0$, sondern ein Polynom λ_i der Gestalt

$$\lambda_i(x) = (1 - \omega_1 x)(1 - \omega_2 x) \cdots (1 - \omega_i x).$$

Die Koeffizienten $\omega_j \in \mathbb{R}$ werden dabei innerhalb des Algorithmus so berechnet, daß $r_i = \lambda_i(A)\varphi_i(A)r_0$ in der euklidischen Norm minimiert wird. Das resultierende Verfahren ist in Algorithmus 5.1 dargestellt. Dabei ist W ein geeigneter Präkonditionierer. Man beachte dabei, daß jede Korrektur der Iterierten als einzelner Schritt innerhalb des BiCG-Stab-Verfahrens angesehen wird. Entsprechend kann auch ein Abbruch der Iteration jeweils nach einer der beiden Korrekturen erfolgen.

procedure BiCGStab($b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{n\times n}$ regulär)
  Sei x_0 bel.; $r_0 := b - Ax_0$; wähle $\tilde{r}$ mit $\langle r_0, \tilde{r}\rangle \neq 0$;
  $\rho_{-1} := \alpha_{-1} := \omega_{-1} := 1$; $v_{-1} := p_{-1} := 0$;
  for $i = 0, 1, 2, \ldots$ Stopkriterium erreicht **do**
    $\rho_i := \langle \tilde{r}, r_i\rangle$; $\beta := \frac{\rho_i}{\rho_{i-1}} \frac{\alpha_{i-1}}{\omega_{i-1}}$;
    $p_i := r_i + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$;
    $y := W^{-1} p_i$; $v_i := Ay$;
    $\alpha_i := \frac{\rho_i}{\langle \tilde{r}, v_i\rangle}$;
    $x_{i+\frac{1}{2}} := x_i + \alpha_i y$; { *1. BiCG-Stab Schritt* }
    $r_{i+\frac{1}{2}} := r_i - \alpha_i v_i$;
    $y := W^{-1} r_{i+\frac{1}{2}}$; $t := Ay$;
    $\omega_i := \frac{\langle t, r_{i+\frac{1}{2}}\rangle}{\langle t, t\rangle}$;
    $x_{i+1} := x_{i+\frac{1}{2}} + \omega_i y$; { *2. BiCG-Stab Schritt* }
    $r_{i+1} := r_{i+\frac{1}{2}} - \omega_i t$;
endfor; **end**;

Algorithmus 5.1: Präkonditioniertes BiCG-Stab-Verfahren

In Anwendungen beobachtet man neben dem geglätten Konvergenzverhalten, daß das BiCG-Stab-Verfahren häufig schneller als das CGS-Verfahren konvergiert. Leider können aber auch bei dem BiCG-Stab-Verfahren die gleichen Breakdowns wie beim CGS-Verfahren auftreten, jeweils bei der Berechnung von β und α_i. Analog zu dem BiCG- und dem CGS-Verfahren berechnet Algorithmus 5.1 nach spätestens n Schritten die exakte Lösung.

Mehr zur Herleitung der BiCG-, CGS- und BiCG-Stab-Verfahren findet man z.B. in der Übersichtsarbeit [Gut94] oder in [Son89] und [vdV92].

5.4. Das GMRES-Verfahren

Innerhalb der *GMRES-Iteration* ("*Generalized-Minimal-Residual*"), die auch auf Systeme mit nichtsymmetrischer Matrix angewandt werden kann, wird mit Hilfe des Gram-Schmidt-Verfahrens sukzessive ein Orthonormalsystem $\{v_1, \ldots, v_k\}$ des Krylov-Unterraumes $K_k = \mathrm{span}\{v_1, Av_1, \ldots, A^{k-1}v_1\}$ aufgebaut. Dabei ist $v_1 = \frac{r_0}{\|r_0\|}$, mit dem Startresiduum $r_0 = b - Ax_0$. Die Iteration für den Aufbau der Basis lautet:

for $j = 1, 2, \ldots, k$ **do**
    for $i = 1, 2, \ldots, j$ **do** $h_{ij} := \langle Av_j, v_i \rangle$;
    $v'_{j+1} := Av_j - \sum_{i=1}^{j} h_{ij} v_i$;
    $h_{j+1,j} := \|v'_{j+1}\|$;
    $v_{j+1} := \frac{v'_{j+1}}{h_{j+1,j}}$;
endfor

Die Koeffizienten h_{ij} bilden dabei die obere $(k+1) \times k$ Hessenbergmatrix $\hat{H}_k$. Die Einschränkung von $\hat{H}_k$ auf die ersten k Zeilen sei H_k.

Die Korrektur z_k für die Iterierte x_0 wird anschließend innerhalb von K_k bestimmt, wozu das Minimierungsproblem

$$\min_{z \in K_k} \|b - A(x_0 + z)\| = \min_{z \in K_k} \|r_0 - Az\| \tag{5.3}$$

gelöst wird. Sei V_k die $n \times k$ Matrix, deren Spalten die Vektoren v_i darstellen. Setzt man $z = V_k y$ mit $y \in \mathbb{R}^k$, so läßt sich das Problem (5.3) mit

$$AV_k = V_{k+1} \hat{H}_k$$

auf das Minimierungsproblem

$$\begin{aligned} \min_{y \in \mathbb{R}^k} \|\beta v_1 - AV_k y\| &= \min_{y \in \mathbb{R}^k} \|V_{k+1}(\beta e_1 - \hat{H}_k y)\| \\ &= \min_{y \in \mathbb{R}^k} \|\beta e_1 - \hat{H}_k y\| \end{aligned} \tag{5.4}$$

reduzieren. Dabei sind $e_1 = (1\ 0 \ldots 0)^T$ und $\beta = \|r_0\|$. Die Lösung von (5.3) und somit die Approximation für die Lösung des Gleichungssystems ist dann

$$x_k = x_0 + V_k y_k,$$

wobei y_k die Lösung von (5.4) darstellt.

Das so definierte Verfahren terminiert nach höchstens n Schritten, wobei eine exakte Arithmetik vorausgesetzt wird. Außerdem ist, anders als beim BiCG-Stab-Verfahren, kein Breakdown möglich. Auch besitzt das Konvergenzverhalten der GMRES-Iteration nicht die Unregelmäßigkeiten des BiCG-Stab-Verfahrens.

Das GMRES-Verfahren hat allerdings den Nachteil, daß für wachsendes k der Aufwand für die Berechnung der j Skalarprodukte und der Speicherbedarf für die Vektoren v_i ansteigt. Um dies zu umgehen, läßt sich der Algorithmus iterativ verwenden, wozu nach jeweils m Schritten ein Neustart mit der aktuellen Iterierten durchgeführt wird. Das resultierende Verfahren ist in Algorithmus 5.2 angegeben. Die Matrix W ist dabei ein geeigneter Präkonditionierer.

procedure GMRES($m \in \mathbb{N}$, $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{n\times n}$ regulär)
    Sei x_0 beliebig;
    while Stopkriterium nicht erreicht **do**
        $r_0 := b - Ax_0$; $v_1 := \frac{r_0}{\|r_0\|}$
        for $j = 1, 2, \ldots, m$ **do**
            $p := AW^{-1}v_j$; $v'_{j+1} := p$
            for $i = 1, \ldots, j$ **do**
                $h_{ij} := \langle p, v_i \rangle$;
                $v'_{j+1} := v'_{j+1} - h_{ij}v_i$;
            endfor;
            $h_{j+1,j} := \|v'_{j+1}\|$;
            $v_{j+1} := \frac{v'_{j+1}}{h_{j+1,j}}$;
        endfor;
        $x_m := x_0 + W^{-1}V_m y_m$, wobei y_m (5.4) minimiert;
        $x_0 := x_m$;   { *Neustart* }
endwhile; **end**;

Algorithmus 5.2: Präkonditioniertes GMRES mit Neustart

Für die Implementierung von GMRES (mit und ohne Neustart) ist die effiziente Lösung des Minimierungsproblems (5.4) wichtig. Die hierzu notwendige Zerlegung von $\hat{H}_k$ in $\hat{H}_k = Q_k R_k$ mittels Givensrotationen läßt sich aufgrund der Struktur von $\hat{H}_k$ während der Iteration berechnen. Hierzu wird in jedem Schritt das Element $h_{j+1,j}$ elemeniert. Durch die gleichzeitige Transformation von βe_1 gewinnt man zusätzlich die Residuumsnorm der aktuellen Iterierten x_j, denn mit $g^j = Q_j \beta e_1$ gilt:

$$|g^j_{j+1}| = \|b - Ax_j\|.$$

Bei dem GMRES-Verfahren ohne Neustart läßt sich somit der Zeitpunkt des Abbruchs der Iteration steuern, ohne die Iterierte x_k zu berechnen. Aber auch bei dem GMRES-Verfahren mit Neustart erlaubt diese Methode die Einsparung einer Berechnung der Residuumsnorm, um etwa Stopkriterien für die Iteration zu testen.

Weitere Informationen zur Herleitung und zu den Eigenschaften des GMRES-Verfahrens sind in [SS86] zu finden.

6. Numerische Ergebnisse

In diesem Kapitel sollen die in den vorausgegangenen Kapiteln beschriebenen Algorithmen und Anordnungen in numerischen Experimenten angewandt werden. Dabei werden sowohl verschiedene Parameter des zugrundeliegenden mathematischen Problems, als auch der Diskretisierung verändert, um deren Auswirkungen auf die Verfahren zu untersuchen.

In Abschnitt 6.1 erfolgt zunächst eine Vorstellung der einzelnen Testprobleme mit den verwendeten Konvektionsfeldern und den verschiedenen Triangulationen. In Abschnitt 6.2 werden die ermittelten Konvergenzraten für die verschiedenen Iterationsverfahren vorgestellt und analysiert. In Abschnitt 6.3 erfolgt schließlich eine Gegenüberstellung der Kosten der einzelnen Verfahren.

6.1. Testprobleme

Ausgangspunkt für die Experimente bildet die Konvektions-Diffusions-Gleichung aus Kapitel 2

$$\begin{aligned} -\varepsilon\Delta u + b\cdot\nabla u &= f \quad \text{in } \Omega \subset \mathbb{R}^d \\ u &= u_0 \quad \text{auf } \Gamma = \partial\Omega. \end{aligned}$$

Dabei werden die Dirichlet-Randbedingungen zu $u_0(\mathbf{x}) = \sum_{i=1}^d x_i^2$ gesetzt. Die rechte Seite f wird anschließend so gewählt, daß sich als Lösung des Problems $u(\mathbf{x}) = \sum_{i=1}^d x_i^2$ ergibt.

Das Gebiet Ω ist im Fall des $\mathbb{R}^2$ das Einheitsquadrat. Im $\mathbb{R}^3$ entspricht Ω dem Einheitswürfel.

Für die numerischen Tests können verschiedene Parameter der Gleichung und der Diskretisierung variiert werden. Im Einzelnen sind dies etwa

- die Starttriangulierung,
- der Parameter ε bzw. die Dominanz der Konvektion,
- das Strömungsfeld und
- die Schrittweite der Diskretisierung und somit die Anzahl der Unbekannten.

Eine Veränderung der Starttriangulation, des Parameters ε und des Konvektionsfeldes dient der Untersuchung der Abhängigkeit der Anordnungstechniken von diesen Variablen.

Innerhalb der Tests wird die aus der Diskretisierung stammende Anordnung mit **NO** abgekürzt. **RCM** steht für die Anordnung mittels Reverse-Cuthill-McKee (siehe Abschnitt 4.2). Eine Anordnung auf Basis des heuristischen/approximativen FVS-Algorithmus (siehe Abschnitt 4.3.2.2) wird mit **hFVS/aFVS** bezeichnet und mit **CC** die Anordnung mittels konzentrischen Zykel (siehe Abschnitt 4.3.2.3).

Im $\mathbb{R}^3$ wird die Verwendung von Flußoberflächen (siehe Abschnitt 4.3.3.1) durch ein **(FS)** kenntlich gemacht, während im $\mathbb{R}^2$ eine vorherige Anwendung der planaren Reduktion (siehe Abschnitt 4.3.2.4) durch ein **(p)** gekennzeichnet wird. Man beachte, daß die Anordnung **hFVS** im $\mathbb{R}^3$ dem in Abschnitt 4.3.2.2 beschriebenen globalen FVS entspricht.

6.1.1. Triangulationen

Als Starttriangulationen im Falle des $\mathbb{R}^2$ werden die in Abbildung 6.1 dargestellten Gitter verwendet. Hierauf aufbauend werden die feineren Gitter mittels der in Abschnitt 2.3 beschriebenen regulären Verfeinerung von Dreiecken konstruiert. Es ergeben sich folgenden Anzahlen von Unbekannten für die einzelnen Verfeinerungsstufen und Gitter[1]:

Stufe	3	4	5	6	7	8
`QUAD1/QUAD2`	225	961	3969	16129	65025	261121
`QUAD3`	157	665	2737	11105	44737	179585

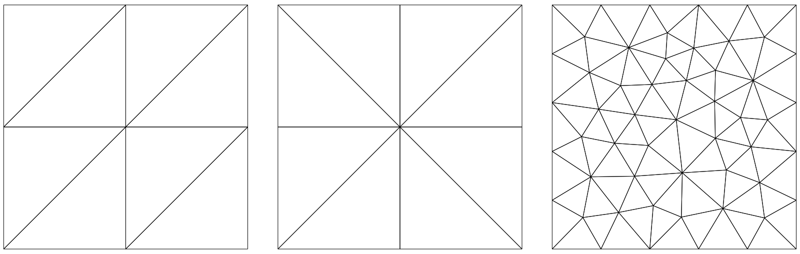

Abbildung 6.1: Starttriangulationen `QUAD1`, `QUAD2` und `QUAD3`

Im $\mathbb{R}^3$ dient die *Kuhn-Triangulierung* des Einheitswürfels (siehe auch [Bey98]) als Starttriangulation. Bei der Kuhn-Triangulierung geht man von der Beobachtung aus, daß der n-dimensionale Einheitswürfel insgesamt $n!$ verschiedene Möglichkeiten bietet, über n Kanten vom Nullpunkt zum Punkt $(1, 1, \ldots, 1)^T$ zu gelangen, und dabei, einschließlich Start- und

[1] Für das Gitter `QUAD3` wurden die Stufen an die Gitter `QUAD1/2` angepaßt, um eine bessere Übersichtlichkeit zu erreichen.

Endpunkt, $n+1$ Eckpunkte zu passieren. Ein Element wird dann durch die Eckpunkte eines solchen Weges definiert. In Abbildung 6.2 ist die Kuhn-Triangulierung des Einheitswürfels im $\mathbb{R}^3$ dargestellt und entspricht Gitter `CUBE1`.

Bei einem weiteren Gitter, der Triangulation `CUBE2`, wurde zunächst eine Zerlegung des Einheitswürfels in Oktaeder spezifiziert und anschließend jeder einzelne Oktaeder mittels Kuhn-Triangulierung in Tetraeder unterteilt. In Abbildung 6.2 ist die Zerlegung in Oktaeder dargestellt.

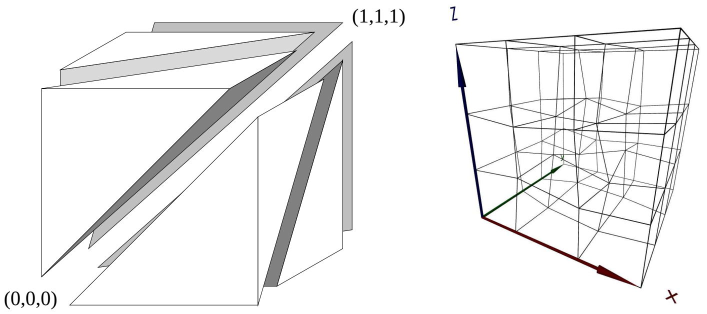

Abbildung 6.2: Gitter `CUBE1` und Oktaedergitter `CUBE2`

Als Verfeinerungsalgorithmus wird die in Abschnitt 2.3 vorgestellte reguläre Verfeinerung für Tetraeder verwendet. Für die Gitter `CUBE1` und `CUBE2` ergeben sich somit folgende Zahlen für die Unbekannten auf den einzelnen Stufen:

Stufe	3	4	5	6
`CUBE1`	343	3375	29791	250047
`CUBE2`	125	1331	12167	103823

6.1.2. Strömungsfelder

Um den Einfluß des Strömungsfeldes auf den Nutzen einer Anordnungstechnik zu ermitteln, werden hierfür verschiedene Felder mit jeweils unterschiedlichen Eigenschaften verwendet. Es handelt sich dabei sowohl um azyklische, als auch um zyklische Strömungsfelder mit konstanter, wie nichtkonstanter Konvektionsrichtung.

Die im $\mathbb{R}2$ verwendeten Konvektionsrichtungen sind in Abbildung 6.3 graphisch dargestellt. Ihr genaue Definition lautet:

- **XLINE** (konstanter, azyklischen Fluß):

$$b = (1,0)^T$$

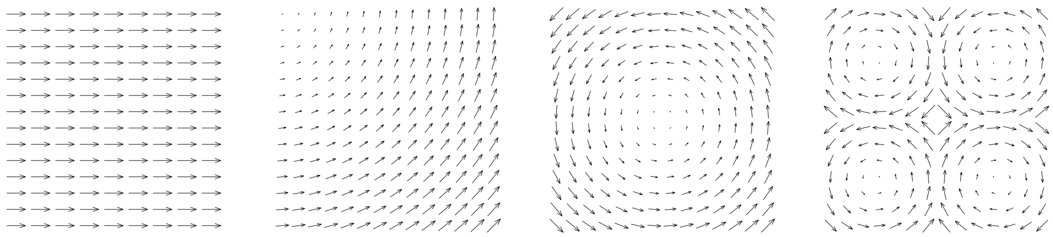

Abbildung 6.3: Strömungsfelder `XLINE`, `CURVE`, `CIRCLE` und `4CIRCLES`

- **CURVE** (nichtkonstanter, azyklischer Fluß):

$$b(x,y) = \begin{pmatrix} 1-y \\ x \end{pmatrix}$$

- **CIRCLE** (zyklischer Fluß mit einem Kreis):

$$b(x,y) = \begin{pmatrix} 0.5-y \\ x-0.5 \end{pmatrix}$$

- **4CIRCLES** (zyklischer Fluß mit vier Kreisen):

$$b(x,y) = \begin{cases} (y-0.25, 0.25-x)^T, & (x,y) \in [0,5]^2 \\ (0.75-y, x-0.25)^T, & (x,y) \in [0,5] \times (0.5,1) \\ (y-0.75, 0.75-x)^T, & (x,y) \in [0.5,1]^2 \\ (0.25-y, x-0.75)^T, & \text{sonst} \end{cases}$$

Im $\mathbb{R}^3$ werden die Konvektionsfelder `XLINE`, `CURVE`, `CIRCLE` und `4CIRCLES` übernommen, wobei die z-Komponente zu 0 gesetzt wird. Zusätzlich wird die folgende Flußrichtung verwendet:

- **DIAGCIRCLE** (zyklischer, dreidimensionaler Fluß):

$$b(x,y,z) = \begin{pmatrix} z-y \\ x-z \\ y-x \end{pmatrix}.$$

`DIAGCIRCLE` entspricht dem Strömungsfeld `CIRCLE`, wobei die Rotationsachse aber nicht $(0.5, 0.5, 0)^T$, sondern die Diagonale $(1,1,1)^T$ ist.

6.2. Konvergenzraten

In den anschließenden Tests werden die in Kapitel 5 vorgestellten BiCG-Stab-Iteration und das GMRES-Verfahren (mit Restartparameter $m = 10$, siehe Algorithmus 5.2) als Löser eingesetzt. Als Präkonditionierer wird sowohl die symmetrische Gauß-Seidel-Iteration (SGS), als auch das Rückwärts-Gauß-Seidel-Verfahren (BGS) verwendet. Die Tests selbst verlaufen in zwei Abschnitten. Zuerst werden die Anordnungstechniken in Abhängigkeit von der Starttriangulation untersucht und anschließend die Abhängigkeit von der Dominanz der Konvektion.

Für den Test mit den verschiedenen Triangulationen werden im $\mathbb{R}^2$ die Gitter bis zur Stufe 8 verfeinert. Im $\mathbb{R}^3$ erfolgt der Test auf Stufe 6. Für den Parameter ε gilt: $\varepsilon = 1_{10}{-}5$.

Die Experimente bezüglich der Dominanz der Konvektion erfolgen im $\mathbb{R}^2$ auf dem Gitter `QUAD1` und im $\mathbb{R}^3$ auf dem Gitter `CUBE1`. Diese Tests werden auf mehreren Stufen durchgeführt, um zusätzlich die Abhängigkeit des Konvergenzverhaltens von der Schrittweite zu untersuchen.

In allen Tests wird sowohl die Anzahl der Iterationsschritte k die nötig sind, um das Startresiduum um $1_{10}{-}4$ zu reduzieren, als auch (in Klammern) die Konvergenzrate $\gamma = (\|r_i\|/\|r_0\|)^{1/k}$ ausgegeben, wobei r_i das i-te Residuum bezeichnet. Die Anzahl der Iterationsschritte ist allerdings auf maximal 400 begrenzt. Als Startwert für die Iterationsverfahren wurde stets $x_0 = (1000, 1000, \dots, 1000)^T$ gewählt.

Ist das Strömungsfeld azyklisch (`XLINE` und `CURVE`), erfolgen, neben der Anordnung **NO** und **RCM**, lediglich Untersuchungen mittels approximativem FVS. Diese Anordnung ist, wie in Abschnitt 4.3.2.2 und 4.3.2.3 beschrieben, identisch mit den Anordnungen **CC** bzw. **aFVS**.

6.2.1. Experimente im $\mathbb{R}^2$

Zunächst werden die Tests mit den azyklischen Strömungsfeldern `XLINE` und `CURVE` durchgeführt. In der folgenden Tabelle sind die Konvergenzraten für die verschiedenen Anordnungstechniken aufgelistet. Als Präkonditionierer kommt dabei das symmetrische Gauß-Seidel-Verfahren zum Einsatz.

Strömungsfeld `XLINE` / SGS

Anordnung		**NO**	**RCM**	**aFVS**
QUAD1	BiCG-S	div	12 (0.45)	11 (0.42)
	GMRES	400 (0.98)	9 (0.33)	9 (0.32)
QUAD2	BiCG-S	div	13 (0.44)	11 (0.43)
	GMRES	400 (0.97)	9 (0.33)	9 (0.31)
QUAD3	BiCG-S	400 (0.99)	104 (0.92)	10 (0.39)
	GMRES	352 (0.97)	98 (0.91)	8 (0.25)

Strömungsfeld `CURVE` / SGS

Anordnung		**NO**	**RCM**	**aFVS**
QUAD1	BiCG-S	div	55 (0.84)	17 (0.47)
	GMRES	361 (0.97)	56 (0.85)	11 (0.43)
QUAD2	BiCG-S	400 (0.99)	133 (0.92)	17 (0.47)
	GMRES	400 (0.98)	103 (0.91)	11 (0.43)
QUAD3	BiCG-S	400 (0.99)	71 (0.88)	12 (0.46)
	GMRES	338 (0.97)	84 (0.89)	10 (0.34)

Man erkennt eine erhebliche Beschleunigung der Iterationsverfahren durch die Anordnung mittels FVS. Bei der RCM-Anordnung fällt die Beschleunigung dagegen moderater aus. Auch ist diese vom Gitter abhängig wie die Tests mit `XLINE` auf den Gittern `QUAD1` und `QUAD2` zeigen. Die FVS-Anordnung zeigt sich dagegen unempfindlicher bzgl. einer Änderung der Triangulation.

Bei dem folgenden Experiment mit unterschiedlicher Dominanz der Konvektion wird die FVS-Anordnung verwendet, da diese die besten Beschleunigungsresultate liefert.

Strömungsfeld `XLINE` / BGS

ε	$1_{10}{-1}$		$1_{10}{-5}$		$1_{10}{-9}$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 6	193 (0.95)	266 (0.97)	4 (0.06)	3 (0.05)	1 ($3_{10}{-6}$)	1 ($3_{10}{-6}$)
Stufe 7	393 (0.98)	400 (0.98)	6 (0.20)	5 (0.12)	1 ($9_{10}{-6}$)	1 ($9_{10}{-6}$)
Stufe 8	400 (0.98)	400 (0.98)	13 (0.39)	9 (0.32)	1 ($3_{10}{-5}$)	1 ($3_{10}{-5}$)

Strömungsfeld `CURVE` / BGS

ε	$1_{10}{-1}$		$1_{10}{-5}$		$1_{10}{-9}$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 6	169 (0.95)	277 (0.97)	4 (0.08)	4 (0.06)	1 ($4_{10}{-6}$)	1 ($4_{10}{-6}$)
Stufe 7	341 (0.97)	400 (0.98)	8 (0.26)	6 (0.18)	1 ($1_{10}{-5}$)	1 ($1_{10}{-5}$)
Stufe 8	400 (0.98)	400 (0.98)	17 (0.47)	12 (0.43)	1 ($3_{10}{-5}$)	1 ($3_{10}{-5}$)

Man erkennt deutlich, daß die Iterationsverfahren im Grenzfall $\varepsilon \to 0$ zu einem exakten Löser werden (siehe auch Abbildung 6.4). Dieses Verhalten ist unabhängig von der Anzahl der Unbekannten, obwohl die Konvergenzraten selbst abhängig von der Schrittweite sind.

In den folgenden Experimenten werden die zyklischen Strömungsfelder `CIRCLE` und `4CIRCLES` untersucht. Zunächst erfolgt wieder der Test bzgl. der verschiedenen Anordnungstechniken. Als FVS-Algorithmus wird dabei nur das approximative Verfahren genutzt, da der heuristische Algorithmus einen vergleichbaren Aufwand hat (siehe Abschnitt 6.3) und ähnliche Resultate liefert.

Strömungsfeld `CIRCLE` / SGS

Anordnung		**NO**	**RCM**	**CYC**	**CYC (p)**	**aFVS**	**aFVS (p)**
QUAD1	BiCG-S	400 (0.99)	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)
	GMRES	400 (0.99)	400 (0.98)	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)
QUAD2	BiCG-S	400 (0.99)	400 (0.99)	400 (0.99)	400 (0.98)	400 (0.98)	breakdown
	GMRES	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)
QUAD3	BiCG-S	400 (0.99)	400 (0.98)	breakdown	400 (0.98)	400 (0.98)	389 (0.98)
	GMRES	400 (0.99)	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)

Strömungsfeld `4CIRCLES` / SGS

Anordnung		**NO**	**RCM**	**CYC**	**CYC (p)**	**aFVS**	**aFVS (p)**
QUAD1	BiCG-S	400 (0.99)	370 (0.98)	400 (0.98)	289 (0.97)	313 (0.97)	335 (0.97)
	GMRES	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.99)	361 (0.97)	400 (0.98)
QUAD2	BiCG-S	breakdown	400 (0.99)	400 (0.98)	389 (0.98)	251 (0.96)	255 (0.96)
	GMRES	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)	279 (0.97)
QUAD3	BiCG-S	breakdown	400 (0.98)	400 (0.99)	400 (0.98)	138 (0.94)	219 (0.96)
	GMRES	400 (0.99)	400 (0.98)	400 (0.98)	400 (0.98)	133 (0.93)	218 (0.96)

Bei dem Strömungsfeld `CIRCLE` ist durch keine Anordnung eine spürbare Beschleunigung der Konvergenz festzustellen. Anders verhält es sich bei den Tests mit dem Konvektionsfeld `4CIRCLES`. Die hier erkennbare Verbesserung des Konvergenzverhaltens fällt allerdings moderater aus als bei den azyklischen Feldern. Die stabilste Technik scheint auch hier die Anordnung mittels FVS zu sein. Das RCM-Verfahren ist dagegen im zyklischen Fall nicht brauchbar. Ähnliches gilt für die Anordnung mittels konzentrischer Zykel, die nur mit zusätzlicher planarer Reduktion nennenswerte Ergebnisse produziert. Leider ist die Reduktion abhängig von der Triangulation.

Analog wie im azyklischen Falls wird auch bei den folgenden Tests die Anordnung mittels FVS genutzt, um das Verhalten bei unterschiedlicher Dominanz der Konvektion und die Abhängigkeit der Lösungsverfahren von der Schrittweite zu testen. Der verwendete Präkonditionierer ist das Rückwärts-Gauß-Seidel-Verfahren.

Strömungsfeld `CIRCLE` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 6	143 (0.94)	338 (0.97)	30 (0.72)	29 (0.72)	13 (0.44)	10 (0.33)
Stufe 7	281 (0.97)	400 (0.98)	181 (0.95)	76 (0.88)	17 (0.55)	17 (0.58)
Stufe 8	400 (0.98)	400 (0.98)	400 (0.98)	400 (0.98)	25 (0.65)	20 (0.63)

Strömungsfeld `4CIRCLES` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 6	157 (0.94)	318 (0.97)	35 (0.77)	29 (0.73)	35 (0.76)	28 (0.72)
Stufe 7	317 (0.97)	400 (0.98)	73 (0.88)	65 (0.87)	51 (0.83)	44 (0.81)
Stufe 8	400 (0.98)	400 (0.98)	217 (0.96)	400 (0.98)	69 (0.87)	61 (0.86)

Anders als bei den azyklischen Konvektionsfeldern werden die Iterationsverfahren nicht zu exakten Lösern bei dominanter werdender Konvektion. In Abbildung 6.4 ist dieses unterschiedliche Verhalten graphisch dargestellt. Dort wurde der Präkonditionierer als Lösungsiteration genutzt und die Entwicklung des Konvergenzverhaltens für $\varepsilon \to 0$ aufgezeichnet.

Auch wenn kein exakter Löser vorliegt, so ist die Beschleunigung der Konvergenz doch auf allen Stufen vorhanden. Allerdings ist die Konvergenzrate selbst abhängig von der Schrittweite.

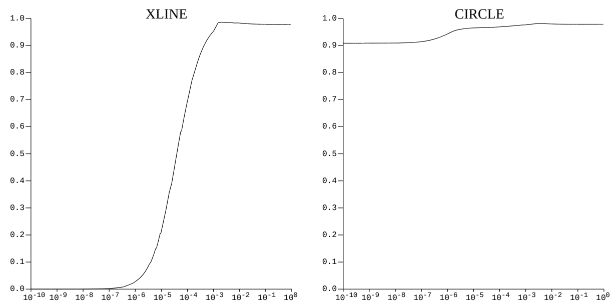

Abbildung 6.4: Konvergenzentwicklung bei Rückwärts-Gauß-Seidel für $\varepsilon \to 0$

6.2.2. Experimente im $\mathbb{R}^3$

In den nachfolgenden Tabellen sind die Konvergenzraten für die Strömungsfelder `XLINE` und `CURVE` bei unterschiedlicher Anordnung aufgeführt. Da beide Felder azyklisch sind, entspricht die FVS-Anordnung der Anordnung mittels Numerierungsalgorithmus.

Strömungsfeld `XLINE` / SGS

Anordnung		**NO**	**RCM**	**hFVS**
CUBE1	BiCG-S	65 (0.86)	31 (0.58)	3 (0.03)
	GMRES	78 (0.89)	43 (0.81)	3 (0.02)
CUBE2	BiCG-S	41 (0.78)	22 (0.63)	3 (0.03)
	GMRES	61 (0.86)	25 (0.69)	3 (0.03)

Strömungsfeld `CURVE` / SGS

Anordnung		**NO**	**RCM**	**CYC**
CUBE1	BiCG-S	59 (0.84)	24 (0.67)	3 (0.04)
	GMRES	68 (0.87)	30 (0.73)	3 (0.03)
CUBE2	BiCG-S	37 (0.77)	15 (0.52)	3 (0.05)
	GMRES	49 (0.83)	14 (0.49)	3 (0.05)

Analog zu den Resultaten im zweidimensionalen Fall ergibt sich auch hier eine wesentliche Beschleunigung der Konvergenz durch die FVS-Anordnung. Aber auch das RCM-Verfahren führt zu einer spürbaren Verbesserung des Konvergenzverhaltens.

Die FVS-Anordnung wird auch in den folgenden Test mit unterschiedlicher Schrittweite und Konvektionsstärke verwendet.

Strömungsfeld `XLINE` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 4	29 (0.72)	31 (0.74)	2 (0.01)	2 (0.01)	1 ($2_{10}-7$)	1 ($2_{10}-7$)
Stufe 5	63 (0.86)	63 (0.86)	2 (0.01)	2 (0.01)	1 ($6_{10}-7$)	1 ($6_{10}-7$)
Stufe 6	117 (0.92)	124 (0.93)	3 (0.03)	3 (0.02)	1 ($2_{10}-6$)	1 ($2_{10}-6$)

Strömungsfeld `CURVE` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 4	33 (0.74)	33 (0.75)	3 (0.04)	3 (0.04)	1 ($5_{10}-5$)	1 ($5_{10}-5$)
Stufe 5	70 (0.88)	71 (0.88)	5 (0.12)	5 (0.12)	1 ($6_{10}-5$)	1 ($6_{10}-5$)
Stufe 6	114 (0.92)	157 (0.94)	3 (0.04)	3 (0.03)	1 ($2_{10}-6$)	1 ($2_{10}-6$)

Die aufgelisteten Resultate geben ein, zum zweidimensionalen Fall vergleichbares Bild ab: Für $\varepsilon \to 0$ werden die Iterationsverfahren zu exakten Lösern. Abbildung 6.5 stellt dieses Verhalten anhand der Rückwärts-Gauß-Seidel-Iteration graphisch dar.

Bei den zyklischen Strömungsfeldern werden zusätzlich zu den bisherigen Anordnungen auch Anordnungen mit vorheriger Zerlegung in Flußoberflächen verwendet. Auf den einzelnen Flußoberflächen kommen anschließend die Verfahren mittels konzentrischer Zykel und approximativem FVS zum Einsatz.

Strömungsfeld `CIRCLE` / SGS

Anordnung		**NO**	**RCM**	**hFVS**	**CYC (FS)**	**aFVS (FS)**
CUBE1	BiCG-S	309 (0.97)	180 (0.95)	45 (0.81)	113 (0.92)	21 (0.63)
	GMRES	400 (0.98)	286 (0.97)	40 (0.79)	160 (0.94)	17 (0.57)
CUBE2	BiCG-S	171 (0.95)	71 (0.87)	57 (0.84)	49 (0.83)	42 (0.80)
	GMRES	290 (0.97)	90 (0.90)	51 (0.83)	48 (0.82)	40 (0.79)

Strömungsfeld `4CIRCLES` / SGS

Anordnung		**NO**	**RCM**	**hFVS**	**CYC (FS)**	**aFVS (FS)**
CUBE1	BiCG-S	165 (0.94)	97 (0.91)	30 (0.73)	61 (0.85)	15 (0.52)
	GMRES	216 (0.96)	111 (0.92)	26 (0.69)	51 (0.83)	13 (0.49)
CUBE2	BiCG-S	93 (0.90)	51 (0.83)	34 (0.76)	32 (0.75)	27 (0.70)
	GMRES	122 (0.93)	49 (0.83)	30 (0.73)	29 (0.73)	22 (0.65)

Bei diesen Strömungsfeldern, die im wesentlichen zweidimensionalen Charakter haben (die Z-Komponente ist jeweils 0), ergibt sich eine wesentliche Verbesserung der Konvergenzraten durch die Verwendung von Flußoberflächen. Hierbei ist die Beschleunigung durch die FVS-Anordung besonders hervorzuheben. Die Anordnung mittels konzentrischer Zykel ist dagegen abhängig von der Triangulation. Als robuste Technik präsentiert sich abermals die Anordnung mittels globalem FVS.

Strömungsfeld `CIRCLE` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 4	33 (0.75)	32 (0.75)	11 (0.42)	10 (0.36)	15 (0.54)	13 (0.47)
Stufe 5	61 (0.86)	79 (0.89)	26 (0.70)	21 (0.64)	37 (0.77)	49 (0.83)
Stufe 6	83 (0.89)	182 (0.95)	46 (0.82)	41 (0.80)	51 (0.83)	46 (0.82)

Strömungsfeld `4CIRCLES` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 4	33 (0.75)	33 (0.75)	11 (0.39)	10 (0.36)	14 (0.51)	13 (0.45)
Stufe 5	62 (0.86)	76 (0.88)	17 (0.51)	14 (0.49)	21 (0.64)	19 (0.61)
Stufe 6	103 (0.91)	177 (0.95)	35 (0.74)	26 (0.70)	34 (0.76)	24 (0.72)

Bei den Tests mit variierender Konvektionsstärke ergibt sich ein interessantes Resultat: Nach anfänglicher Beschleunigung der Konvergenz bei zunehmender Konvektion, verschlechtert sich diese bei sehr starker Konvektionsdominanz zunehmend. In Abbildung 6.5 ist dieses Verhalten auch bei dem Rückwärts-Gauß-Seidel-Verfahren sichtbar. Ebenfalls erkennbar ist die Abhängigkeit der Konvergenzraten von der Schrittweite. Verwendet wurde bei diesen Versuchen die Anordnung mittels globalem FVS.

Abschließend erfolgen die Versuche mit dem Konvektionsfeld `DIAGCIRCLE`. Zunächst der Vergleich der verschiedenen Anordnungstechniken.

Strömungsfeld `DIAGCIRCLE` / SGS

Anordnung		**NO**	**RCM**	**hFVS**	**CYC (FS)**	**aFVS (FS)**
CUBE1	BiCG-S	223 (0.96)	115 (0.92)	105 (0.92)	127 (0.92)	125 (0.92)
	GMRES	400 (0.98)	204 (0.96)	171 (0.95)	223 (0.96)	222 (0.96)
CUBE2	BiCG-S	154 (0.94)	61 (0.86)	61 (0.86)	85 (0.89)	85 (0.89)
	GMRES	253 (0.96)	73 (0.88)	73 (0.88)	106 (0.92)	106 (0.92)

Obwohl auch `DIAGCIRCLE` eine im wesentlichen zweidimensionalen Grundstruktur besitzt, können die Anordungen mittels Flußoberflächen das Lösungsverfahren nicht so stark beschleunigen, wie in den vorherigen Beispielen. Der Grund hierfür liegt in der Abhängigkeit der Konstruktion der Flußoberflächen vom zugrundeliegenden Gitter, die bei `CIRCLE` und `4CIRCLES` zu einer günstigeren Zerlegung führten.

Bei dem folgenden Test bzgl. der Abhängigkeit der Lösungsiterationen von der Dominanz der Konvektion wurde wieder die Anordnung mittels globalem FVS genutzt.

Strömungsfeld `DIAGCIRCLE` / BGS

ε	$1_{10}-1$		$1_{10}-5$		$1_{10}-9$	
	BiCG-S	GMRES	BiCG-S	GMRES	BiCG-S	GMRES
Stufe 4	28 (0.72)	31 (0.74)	31 (0.72)	27 (0.71)	31 (0.72)	27 (0.71)
Stufe 5	51 (0.83)	76 (0.88)	61 (0.86)	93 (0.91)	61 (0.86)	92 (0.90)
Stufe 6	85 (0.89)	181 (0.95)	99 (0.91)	174 (0.95)	99 (0.91)	170 (0.95)

Im Gegensatz zu den bisherigen Resultaten erkennt man bei diesem Beispiel kaum eine Verbesserung des Konvergenzverhaltens mit abnehmendem ε. In einigen Fällen verschlechtert sich dieses sogar. Auch beim Rückwärts-Gauß-Seidel-Verfahren in Abbildung 6.5 ergibt sich nur eine minimale Beschleunigung.

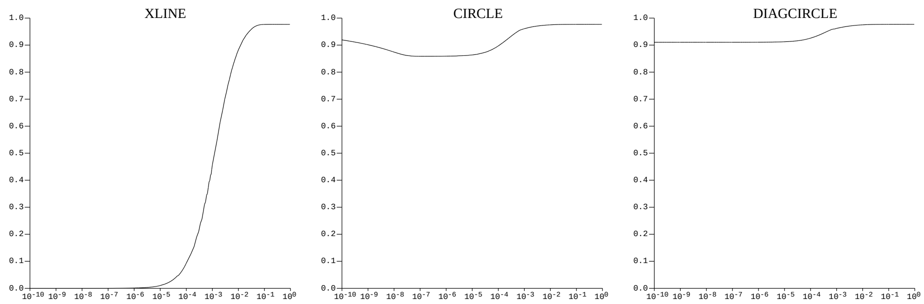

Abbildung 6.5: Konvergenzentwicklung bei Rückwärts-Gauß-Seidel für $\varepsilon \to 0$

6.3. Aufwand der Verfahren

Für einen abschließenden Vergleich der verschiedenen Anordnungstechniken sind die jeweiligen Kosten zu beachten. Hierbei ergeben sich teils gravierende Unterschiede zwischen den Verfahren.

Zunächst werden die unterschiedlichen Iterationsverfahren verglichen. Die Zahlen sind dabei jeweils in Relation zu einem Rückwärts-Gauß-Seidel-Schritt angegeben. Diese Einheit wird auch bei den folgenden Tabellen beibehalten. Als zusätzlichen Vergleichswert seien die Kosten für die Mehrgitteriteration benannt, wie sie z.B. in [Bor99] zu finden sind. Dabei betragen die Kosten eines V-Zyklus-Schrittes mit einer Vor- und einer Nachglättung (jeweils Symmetrisch-Gauß-Seidel) etwa 6 Rückwärts-Gauß-Seidel-Schritte.

Aufwand der Iterationsverfahren			
BGS	**SGS**	**BiCG-Stab**	**GMRES**
1	1.5	1.5	2

Innerhalb der BiCG-Stab-Iteration wurden die beiden Einzelschritte extra gezählt (siehe Algorithmus 5.1). Beim GMRES-Verfahren wird jeder Schritt beim Aufbau des Orthogonalsystems im Krylov-Raum als Einzelschritt gewertet (siehe Algorithmus 5.2). Dabei ist allerdings zu beachten, daß der Aufwand vom Restartparameter abhängt. Der angegebene Zahlenwert bezieht sich auf 10 Orthogonalisierungsschritte.

Sowohl beim BiCG-Stab-, als auch beim GMRES-Verfahren ist weiterhin zu berücksichtigen, daß der Aufwand für einen Präkonditionierungsschritt in den entsprechenden Kosten bereits enthalten ist.

Vergleicht man die Anordnungstechniken im zweidimensionalen Problem, so stellt man den relativ hohen Aufwand für die planare Reduktion des Matrixgraphen (siehe Abschnitt 4.3.2.4) im Vergleich zu den anderen Verfahren fest.

Aufwand im $\mathbb{R}^2$				
RCM	**CYC**	**hFVS**	**aFVS**	**planare Red.**
11	13	28	30	63

Im $\mathbb{R}^3$ sind erkennt man eine Verbesserung des Aufwandes für den heuristischen FVS-Algorithmus, wobei hierbei das globale FVS aus Abschnitt 4.3.2.2 gemeint ist. Eine Erklärung hierfür findet sich im heuristischen Charakter des Verfahrens. Dies äußert sich darin, daß der Algorithmus bei den Testbeispielen im dreidimensionalen Fall relativ zur Anzahl der Knoten im Graph ein deutlich größeres FVS berechnet als bei den Beispielen im $\mathbb{R}^2$.

Weiterhin erkennt man die enormen Kosten für den Aufbau der Flußoberflächen. Hierzu ist zu sagen, daß der implementierte Algorithmus einen Großteil der Zeit für Winkelberechnungen zwischen Kanten und Dreiecksflächen im Tetraedergitter verwendet.

Aufwand im $\mathbb{R}^3$				
RCM	**hFVS**	**CYC (FS)**	**aFVS (FS)**	**Aufbau der FS**
12	14	111	122	96

Es sei an dieser Stelle allerdings darauf hingewiesen, daß eine spezielle, auf einen bestimmten Anordnungsalgorithmus hin optimierte Implementierung zu (deutlichen) Verbesserungen in den Laufzeiten führen kann. Bei der Programmierung in Zusammenhang mit dieser Arbeit wurde mehr Wert auf eine homogene Datenstruktur als auf eine spezielle Optimierung der Algorithmen gelegt.

Literaturverzeichnis

[Bas96] Peter Bastian. *Parallele adaptive Mehrgitterverfahren.* Teubner Skripten zur Numerik. B.G. Teubner, Stuttgart, 1996.

[Bey95] Jürgen Bey. Tetrahedral grid refinement. *Computing*, 55:355–378, 1995.

[Bey98] Jürgen Bey. *Finite-Volumen- und Mehrgitter-Verfahren für elliptische Randwertprobleme.* B.G. Teubner, Stuttgart, 1998.

[Bor99] Sabine Le Borne. *Multigrid methods for convection-dominated problems.* PhD thesis, Christian-Albrechts-Universität zu Kiel, 1999.

[Bra97] Dietrich Braess. *Finite elements.* Cambridge University Press, 1997.

[BW97] J. Bey and G. Wittum. Downwind Numbering: Robust Multigrid for Convection-Diffusion Problems. *Applied Numerical Mathematics*, 23:177–192, 1997.

[DER86] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Oxford Science Publications, Oxford, 1986.

[GDN95] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerische Simulation in der Strömungsmechanik: eine praxisorientierte Einführung.* Vieweg, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[Gut94] Sabine Gutsch. *Ein Vergleich CG-ähnlicher Verfahren zur Lösung indefiniter Probleme.* Diplomarbeit, Christian-Albrechts-Universität zu Kiel, 1994.

[Hac93] Wolfgang Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssystem.* B.G. Teubner, 1993.

[Hac97] Wolfgang Hackbusch. On the feedback vertex set problem for a planar graph. *Computing*, 58(2):129–155, 1997.

[LSW88] Errol L. Lloyd, Mary Lou Soffa, and Ching-Chy Wang. On locating minimum feedback vertex sets. *Journal of Computer and System Sciences*, 37:292–311, 1988.

[RR96] H. Rentz-Reichert. *Robuste Mehrgitterverfahren zur Lösung der inkompressiblen Navier-Stokes Geichung: Ein Vergleich.* PhD thesis, Institut für Computeranwendungen der Universität Stuttgart, 1996.

[Son89] P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat Comput.*, 10:36–52, 1989.

[SS86] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Comput.*, 7(3):856–869, 1986.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2), 1972. 146-160.

[Tur92] Stefan Turek. On ordering strategies in a multigrid algorithm. In *Notes on Numerical Fluid Mechanics*, volume 41. Vieweg, 1992.

[vdV92] H. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of bicg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat Comput.*, 13:631–644, 1992.