Loadbalancing auf Parallelrechnern mit Hilfe endlicher Dimension-Exchange-Verfahren

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

am Fachbereich Mathematik der
Bergischen Universität Wuppertal
genehmigte

Dissertation

von

Dipl.-Math. Holger Arndt

Tag der mündlichen Prüfung:	24. Juli 2003
Referent:	Prof. Dr. A. Frommer
Korreferent:	Prof. Dr. B. Monien

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Vorwort

Loadbalancing-Verfahren werden verwendet, um Berechnungen auf ungleichmäßig ausgelasteten Parallelrechnern besser auf die einzelnen Prozessoren zu verteilen. Obwohl im Deutschen auch die Begriffe *Lastausgleich* und *Lastverteilung* existieren, wird hier dennoch das verbreitete englische *Loadbalancing* verwendet werden.

Der Begriff *Loadbalancing* wird für eine Reihe verschiedener Aufgabenstellungen verwendet. Am häufigsten taucht er auf im Zusammenhang mit der Verteilung vieler Anfragen aus dem Internet auf mehrere Webserver. Ein ähnliches Beispiel ist die Verteilung neu gestarteter Prozesse auf die Prozessoren eines (Shared-Memory)-Parallelrechners. Beiden gemeinsam ist, dass neue Lasten (Anfragen bzw. Prozesse) einmal einem Prozessor zugewiesen werden und dort verbleiben; man spricht hier daher auch vom dynamischen *Mapping*-Problem. In dieser Arbeit dagegen wird das sog. dynamische Loadbalancing-Problem betrachtet. Es wird davon ausgegangen, dass bereits eine Anzahl von Lasten (Einzelproblemen) auf die Prozessoren eines Parallelrechners verteilt ist. Fallen durch Lösung einzelner Probleme Lasten weg oder entstehen woanders neue Lasten, so führt dies zu einem Ungleichgewicht, das durch Verschiebung einzelner Lasten ausgeglichen werden soll. In den Modellen hierzu wird angenommen, dass alle Lasten gleich groß sind, oder anders ausgedrückt, dass alle Probleme gleich schnell zu lösen sind. Einen guten Überblick über die verschiedenen Aspekte des Loadbalancings geben Diekmann, Monien und Preis in [DMP99].

Im Wesentlichen lassen sich Loadbalancing-Verfahren in zwei Klassen aufteilen, nämlich die sog. Diffusions- und die Dimension-Exchange-Verfahren; beide Arten wurden zuerst von Cybenko in [Cyb89] beschrieben. Beide Verfahren arbeiten iterativ, der Unterschied zwischen ihnen besteht darin, welche Werte in jedem Iterationsschritt zur Aufdatierung verwendet werden und lässt sich vergleichen mit dem Unterschied zwischen Jacobi- und Gauß-Seidel-Verfahren zur Lösung linearer Gleichungssysteme. Für die Klasse der Diffusionsverfahren sind zudem endliche Verfahren entwickelt worden [DFM99, EFMP99], in Analogie zum cg-Verfahren bei linearen Gleichungssystemen. Ziel dieser Arbeit ist die Entwicklung und Bewertung endlicher Dimension-Exchange-Verfahren. Untersucht werden die Schnelligkeit und Genauigkeit dieser Verfahren sowie die Frage, wie viele Lasten (u. U. unnötig) verschoben werden. Sämtliche Messergebnisse, die hier gezeigt werden, sind mit Hilfe paralleler Programme erzielt worden.

Die Arbeit ist folgendermaßen gegliedert: Kapitel 1 gibt eine mathematische Definition der Problemstellung und einen Überblick über Parallelrechnerarchitekturen sowie über Anwendungsfälle für Loadbalancing-Verfahren. Außerdem werden einige Standardgraphen sowie Matrizen zur Beschreibung von Graphen definiert. Die Graphen dienen dazu, die Topologie von Parallelrechnern zu modellieren.

Das Kapitel 2 beschäftigt sich mit den verschiedenen Diffusionsverfahren, die seit 1989 veröffentlicht worden sind. Begonnen wird mit einfacheren Verfahren erster und zweiter Ordnung. Danach folgen endliche Verfahren, die auf der Kenntnis von Eigenwerten bestimmter Matrizen zu Graphen basieren. Bei einem der endlichen Verfahren wird untersucht, wie sich die Reihenfolge dieser Eigenwerte auf die numerische Stabilität des Verfahrens auswirkt.

Das zentrale Kapitel der vorliegenden Arbeit ist das Kapitel 3 über Dimension-Exchange-Verfahren. Der wesentliche Unterschied zu Diffusionsverfahren liegt darin, dass die einzelnen Iterationsschritte weiter unterteilt werden in Teilschritte, wobei stets die aktuellsten Lastinformationen herangezogen werden. Die Abschnitte 3.1 und 3.2 widmen sich der hierfür notwendigen Kantenfärbung der Graphen sowie Matrizen zur Beschreibung gefärbter Graphen. In den Abschnitten 3.3 bis 3.5 wird gezeigt, wie man aus den verschiedenen Diffusionsverfahren entsprechende Dimension-Exchange-Varianten gewinnt. Die Idee des Dimension-Exchange-Prinzips ist genauso alt wie die Diffusionsverfahren (siehe [Cyb89]), neu in dieser Arbeit sind endliche Dimension-Exchange-Verfahren. Von jeder Verfahrensvariante gibt es zwei Arten, eine mit unsymmetrischer und eine mit symmetrischer Iterationsmatrix. Bei den endlichen unsymmetrischen Algorithmen wird jeweils die Korrektheit nachgewiesen, bei den symmetrischen ist dies wegen der engeren Verwandtschaft zu Diffusionsverfahren nicht nötig. Wie im Diffusionsfall müssen auch für die endlichen Dimension-Exchange-Verfahren vorneweg Eigenwerte bestimmter Graphmatrizen ausgerechnet werden; in Abschnitt 3.6 werden für einige wichtige Graphen konkrete Formeln für diese Eigenwerte hergeleitet. Bei vielen Graphen ist zudem die Anzahl verschiedener Eigenwerte geringer als im Diffusionsfall, was sich unmittelbar auf die Laufzeiten der Verfahren auswirkt. Der konkrete Ablauf des Lastaustausches wird durch Flüsse auf den Graphen beschrieben. Während Diffusionsverfahren Flüsse erzeugen, die eine minimale l_2-Norm besitzen, entstehen beim Dimension-Exchange höhere Flüsse. Abschnitt 3.7 demonstriert, wie man die Flüsse durch zusätzliche Rechnungen, aber mit wenig zusätzlicher Kommunikation, verringern kann. In Abschnitt 3.8 werden wichtige Resultate zur Berechnung und Abschätzung von Flüssen hergeleitet. Zu einem gegebenem Graphen und einem gegebenen Verfahren lassen sich Matrizen konstruieren, die angewandt entweder auf eine Ausgangslastverteilung oder einen mit irgendeinem anderen Verfahren berechneten Fluss den gesuchten Fluss liefern. Die Spektralnormen dieser mit Hilfe von Moore-Penrose-Inversen bestimmten Matrizen sind ein Maß dafür, um wie viel die berechneten Flüsse im schlimmsten Fall über dem Minimum liegen. Für einige spezielle Graphen lassen sich diese Matrixnormen theoretisch berechnen, für alle anderen wichtigen Fälle werden experimentell berechnete Werte angegeben. Das Kapitel endet mit Betrachtungen zu Aufwand und Stabilität sowie den diesbezüglichen Vorteilen von Dimension-Exchange- gegenüber Diffusionsverfahren.

Kapitel 4 ist den Verfahren der alternierenden Richtungen gewidmet, einer speziellen Verfahrensklasse für so genannte Produktgraphen wie Gitter und Torus. Basierend auf mehreren bereits existierenden Diffusionsverfahren werden sowohl neue Diffusionsvarianten als auch Dimension-Exchange-Verfahren für Produktgraphen konstruiert. Die Ergebnisse zur Berechnung von Flüssen aus Kapitel 3 lassen sich zumindest auf einen Teil der Verfahren für Produktgraphen übertragen, ebenso wie Methoden zur Flussreduktion.

Kapitel 5 enthält Hinweise zur Implementierung von Loadbalancing-Verfahren sowie ausführliche Messergebnisse zum Vergleich der verschiedenen Algorithmen.

Während es bis hierher nur darum ging, eine Gleichverteilung zu bestimmen, beschäftigt sich Kapitel 6 mit dem Scheduling, d. h. mit dem tatsächlichen Verschieben von Lasten. Neben bewährten Strategien wird ein neues Verfahren vorgestellt, das die für Dimension-Exchange-Verfahren sowieso notwendige Kantenfärbung der Graphen verwendet. Die neue Methode ist in vielen Fällen gleichwertig und in einigen Situationen sogar schneller als die bisherigen Verfahren.

Abschließend gibt Kapitel 7 Ausblicke darauf, wie die Verfahren modifiziert werden müssen, um verschiedene Knoten- und Kantengewichte zu berücksichtigen. Hiermit lassen sich verschieden schnelle Prozessoren bzw. Netzwerkverbindungen modellieren.

Die vorliegende Dissertation entstand im Zeitraum März 1999 bis April 2003 am Fachbereich Mathematik der Bergischen Universität Wuppertal.

Mein besonderer Dank an dieser Stelle gilt meinem Doktorvater Prof. Dr. Andreas Frommer für die Aufnahme in seine Arbeitsgruppe, für die interessante Themenstellung sowie seine Diskussionsbereitschaft. Seine wertvollen Anmerkungen und kritischen Kommentare trugen zum Gelingen dieser Arbeit bei.

Bei Prof. Dr. Burkhard Monien bedanke ich mich für die Übernahme des Korreferates sowie die Möglichkeit, den Parallelrechner der Universität Paderborn zu benutzen.

Ich danke allen Mitgliedern der Arbeitsgruppe für die gute Zusammenarbeit und angenehme Arbeitsatmosphäre.

Diese Arbeit wurde unterstützt von der Deutschen Forschungsgemeinschaft (DFG), Projekt FR 755/14-1.

Wuppertal, im Juli 2003 Holger Arndt

1 Einleitung

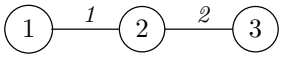
Dieses Kapitel beginnt mit einer mathematischen Definition des Loadbalancing-Problems. Es folgen Einführungen zum allgemeinen Ablauf des Loadbalancings, Anwendungsfällen und Parallelrechnerarchitekturen bzw. -topologien sowie hierzu passenden Verfahrensklassen. Abgeschlossen wird das Kapitel mit einigen weiteren Definitionen spezieller Graphen und Matrizen.

1.1 Das Loadbalancing-Problem und Matrizen für Graphen

Für die gesamte Arbeit gelte die folgende Generalvoraussetzung.

Definition 1.1. Es sei $G = (V, E)$ ein zusammenhängender ungerichteter Graph mit $n = |V|$ Knoten und $N = |E|$ Kanten. Die Knoten werden mit $1, \ldots, n$ und die Kanten mit $e_1, \ldots, e_N$ bezeichnet.

Graph-Beispiel 1.2. Alle in dieser Arbeit folgenden „Graph-Beispiele“ werden sich auf diesen Pfad der Länge 3 (P_3) beziehen.



Hier ist also $V = \{1, 2, 3\}$ und $E = \{\{1, 2\}, \{2, 3\}\}$.

Zur Beschreibung solcher Graphen wird nun die *Inzidenzmatrix* definiert.

Definition 1.3. Die zum Graphen G gehörende *Knoten-Kanten-Inzidenzmatrix* $A \in \{0; 1; -1\}^{n \times N}$ mit $A = (a_{ij})$ ist definiert durch

$$a_{ij} = \begin{cases} 1 & \text{falls } e_j = \{i, k\} \text{ existiert mit } i < k \\ -1 & \text{falls } e_j = \{i, k\} \text{ existiert mit } i > k \\ 0 & \text{sonst.} \end{cases}$$

Bemerkung 1.4. Entgegen der Definition 1.1 wird durch Definition 1.3 jeder Kante eine Richtung zugeordnet. Die hier gewählten Richtungen sind rein willkürlich und könnten auch anders festgesetzt werden, ohne dass sich hierdurch im Folgenden Entscheidendes ändert.

Graph-Beispiel 1.5.

$$A = \begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{pmatrix}$$

Vor Ausführung eines Loadbalancing-Verfahrens besitze jeder Knoten i, $i = 1, \ldots, n$ die Last $w_i^0 \in \mathbb{N}_0$. Als *Ausgangslastverteilung* bezeichnet wird der Vektor

$$w^0 = \left(w_1^0, \ldots, w_n^0\right)^T .$$

Zu berechnen ist der Vektor der Gleichverteilung

$$\overline{w} = \frac{1}{n} \sum_{i=1}^{n} w_i^0 \cdot (1, \ldots, 1)^T .$$

Es sei $x \in \mathbb{R}^N$ ein Fluss auf dem Graphen G. Die Richtung des Flusses wird bestimmt durch die von der Inzidenzmatrix A festgelegten Richtungen der Kanten und die Vorzeichen der Einträge in x; $x_e > 0$ steht für einen Fluss in Richtung der Kante e und $x_e < 0$ für einen Fluss in entgegengesetzter Richtung.

Definition 1.6. Ein Fluss $x \in \mathbb{R}^N$ auf einem Graphen G heißt ausgleichender Fluss, falls gilt:

$$Ax = w^0 - \overline{w}$$

Bemerkung 1.7. Der ausgleichende Fluss ist genau dann eindeutig bestimmt, wenn der Graph G keine Zyklen hat, also ein Baum ist. Andernfalls könnten nämlich die Flusswerte entlang eines Zyklus um einen konstanten Wert verändert werden, ohne dass sich die Eigenschaft des ausgleichenden Flusses ändert. Ist umgekehrt der Fluss nicht eindeutig, dann gibt es zwei verschiedene Flüsse x_1, x_2 mit $Ax_1 = Ax_2$. Damit gilt aber $A(x_1 - x_2) = 0$ und $x_1 - x_2$ ist ein zirkulärer Fluss.

1.2 Allgemeines Vorgehen

Die Knoten des oben definierten Graphen entsprechen den Prozessoren eines Parallelrechners. Der Datenaustausch zwischen diesen erfolgt ausschließlich über die Kanten des Graphen, also nur zwischen adjazenten Knoten.

Prinzipiell ließe sich ein iteratives Loadbalancing-Verfahren so organisieren, dass in jedem Iterationsschritt Lasteinheiten zwischen den Prozessoren ausgetauscht werden, bis eine Gleichverteilung hergestellt ist [Cyb89, MGS96]. Aus Effizienz- und algorithmischen Gründen findet üblicherweise jedoch eine Unterteilung in zwei Phasen statt [DFM99, HB95]. Zunächst wird keine Last bewegt. Stattdessen werden nur einzelne Zahlen (Lastwerte) versandt und ein ausgleichender Fluss bestimmt. Erst in der zweiten Phase, dem *Scheduling*, werden tatsächlich Lasten anhand dieses Flusses verschoben. Dies hat zum einen den Vorteil, dass Loadbalancingalgorithmen nicht nur mit ganzzahligen, sondern auch reellen, negativen und sogar komplexen Lastwerten rechnen können. Zum anderen wird so verhindert, dass einzelne Lasten in aufeinander folgenden Schritten unnötig zwischen zwei benachbarten Prozessoren hin- und hergeschoben werden, vergleiche Abschnitt 5.1 in [DFM99].

Unter Umständen wird das Scheduling nochmals unterteilt in die Bestimmung des Ablaufs (in welchem Schedule-Schritt werden wie viele Lasten von wo nach wo verschoben) und dessen Durchführung (Bestimmung, welche konkreten Lasten verschoben werden und eigentliches Verschieben). In diesem Fall spricht man vom *Drei-Phasen-Modell* [Els02].

Es wird grundsätzlich davon ausgegangen, dass die übergeordnete Berechnung während des Lastausgleichs angehalten wird, so dass in dieser Zeit keine neuen Lasteinheiten entstehen oder alte wegfallen.

1.3 Anwendungen für Loadbalancing

Loadbalancing kann immer dann eingesetzt werden, wenn bei der Verteilung der Rechenlast auf mehrere Prozessoren von vornherein nicht bekannt ist, welcher Teil wie viel Rechenzeit benötigt. Im Folgenden werden einige Beispiele aufgeführt.

Loadbalancing-Verfahren werden benötigt in parallelen adaptiven Finite-Elemente-Verfahren. Hierbei wird zunächst ein Gebiet diskretisiert und in gleich große Teilgebiete zerlegt, die dann auf die Prozessoren verteilt werden. Im Laufe der Rechnung ergibt sich oft die Notwendigkeit, das Gitter in einigen Regionen zu verfeinern, während in anderen Regionen auf einen Teil der Gitterpunkte verzichtet werden kann. Das so entstehende Ungleichgewicht muss nun durch ein Loadbalancing-Verfahren ausgeglichen werden. In diesem Anwendungsfall ergibt sich allerdings noch ein zusätzliches Problem dadurch, dass die einzelnen Teilgebiete zusammenhängend bleiben sollten und einen „glatten“ Rand behalten sollten. Aus diesem Grund wird hier zur Modellierung ein Graph verwendet, bei dem jeder Knoten für ein Teilgebiet (also einen Prozessor) steht und die Kanten die Nachbarschaft zweier Teilgebiete repräsentieren. Werden Lasten nun im Modell ausschließlich über diese Kanten verschoben, so bedeutet dies einen Wechsel von einem Gebiet in ein angrenzendes Gebiet. Realisiert ist dies zum Beispiel in der Software PadFEM [Pad, DDNR96].

Weitere Anwendungsbeispiele sind Lösungen von Optimierungsproblemen mit parallelen Branch-and-Bound-Verfahren [LM92], Spielbaumsuchalgorithmen *(game tree search)* zum Beispiel in Schachprogrammen [Fel97, FM98] oder auch das Rendern hochaufgelöster Bilder [SR99, Cro97].

1.4 Parallelrechner

Parallelrechner sind zunächst einmal dadurch charakterisiert, dass sie mehr als einen Prozessor besitzen. Diese Prozessoren müssen in der Lage sein, gemeinsam an einem Problem zu arbeiten. Dazu müssen sie entweder auf dieselben Daten zugreifen können oder sich gegenseitig Daten zugänglich machen können. Hieraus ergibt sich eines der wichtigsten Unterscheidungsmerkmale bei der Klassifizierung von Parallelrechnern: gemeinsamer oder verteilter Speicher. Bei gemeinsamem Speicher *(shared memory)* teilen sich alle Prozessoren einen gemeinsamen Speicher. Ein expliziter Datenaustausch ist somit nicht nötig und insbesondere besteht keine Notwendigkeit für die in dieser Arbeit

behandelten Loadbalancing-Verfahren. Im Gegensatz dazu hat bei Rechnern mit verteiltem Speicher *(distributed memory)* jeder Rechner seinen eigenen, von den anderen Prozessoren unabhängigen Speicher. Der Datenaustausch erfolgt durch das Versenden von Nachrichten *(message passing)* über ein Netzwerk.

Bei den verwendeten Netzwerken sind viele verschiedene Topologien anzutreffen. Besonders auf etwas älteren Rechnern sind die Prozessoren oftmals in Form eines Gitters, Torus oder Hypercubes angeordnet, manchmal auch als Zyklus oder als Baum. Während der Datenaustausch zwischen zwei Prozessoren, die in dieser Struktur benachbart sind, in der Regel sehr schnell ist, kann die Kommunikation zwischen zwei beliebigen Knoten um ein Vielfaches länger dauern. In solchen Fällen ist der für das Loadbalancing verwendete Graph dieser Topologie anzupassen. Beispiele für Rechner mit solch fester Topologie sind die Cray T3E (3-D Torus), Fujitsu AP3000 (2D-Torus), SGI Origin 2000 (Hypercube mit 4 Prozessoren pro Knoten) oder der schon etwas ältere (1993) nCUBE 2S (Hypercube der maximalen Dimension 12).

Vielfach werden in neueren Parallelrechnern aber Switches bzw. Crossbars eingesetzt, so dass jeder Prozessor mit jedem beliebigen anderen Prozessor in etwa gleich schnell kommunizieren kann. Bei mehr als etwa 64 Prozessen werden in der Regel mehrstufige Crossbars verwendet. Beispiele hierfür sind Fujitsu VPP5000 (einstufiger Crossbar), Hitachi SR8000 (bis zu dreistufiger Crossbar) und NEC SX-6 (mehrstufiger Crossbar). Diese Architektur lässt sich zwar am ehesten durch einen vollständigen Graphen beschreiben; allerdings ist es nicht möglich, dass jeder Prozessor mit allen anderen Prozessoren gleichzeitig kommuniziert. Der Vorteil ist vielmehr, dass durch Auswahl einzelner Verbindungen beliebige Teilgraphen verwendet werden können. In diesem Fall kann man entweder einen für die Anwendung passenden Graphen wählen (z. B. bei Finite-Elemente-Anwendungen) oder einen an die zur Verfügung stehende Anzahl an Prozessoren angepassten Graphen, also beispielsweise bei einer Zweierpotenz einen Hypercube, oder sonst einen mehrdimensionalen Torus.

In diese zweite Kategorie von Rechnern ohne bestimmte Topologie fallen auch die immer häufiger anzutreffenden Cluster. Hierbei werden aus Standardkomponenten bestehende Rechner (z. B. normale PCs) miteinander vernetzt, im einfachsten Fall mit normalen Ethernet-Karten, falls bessere Datentransferraten erzielt werden sollen mit spezialisierter Hardware wie zum Beispiel einem Myrinet-Netzwerk. Mit dem Linux NetworX (2304 Intel-Xeon-Prozessoren, 11 TFlop/s Peak-Performance) und dem HPTi (1536 Prozessoren) befinden sich sogar zwei Cluster unter den ersten zehn Plätzen der TOP500-Liste vom November 2002.

Eine ganz andere Art von „Cluster“ stellen lose vernetzte Rechner dar, die nur gelegentlich gemeinsam für große Rechenaufgaben benutzt werden und daher keine besonders schnelle Netzwerkanbindung besitzen. Ein weiteres mögliches Anwendungsgebiet für Loadbalancing in der Zukunft stellt das Grid-Computing dar, bei dem mehrere, räumlich unter Umständen weit entfernte, Rechner gemeinsam benutzt werden. Bei diesen letzten beiden Fällen ist ein schnelles Loadbalancing-Verfahren noch wichtiger als bei echten Parallelrechnern mit schneller spezialisierter Kommunikationshardware.

Einen guten Überblick über Parallelrechner-Topologien liefert das Einführungskapitel in [ALO02], ausführlichere Informationen zur Hardware finden sich in [vdSD02]. Dieser

Bericht wird jährlich aktualisiert.

1.5 Kommunikationsmodelle und Verfahrensklassen

Neben der im letzten Abschnitt erläuterten Kategorisierung von Parallelrechnern gibt es noch ein weiteres Unterscheidungsmerkmal, das für Loadbalancing-Algorithmen von großer Wichtigkeit ist, nämlich ob der Nachrichtenaustausch immer auf zwei Prozessoren beschränkt ist oder nicht. Beim so genannten *All-Port-Modell* wird davon ausgegangen, dass ein Prozessor mit all seinen Nachbarn gleichzeitig kommunizieren kann. Hierauf basieren die in dieser Arbeit zuerst vorgestellten *Diffusionsverfahren*. Die meisten tatsächlichen Parallelrechner entsprechen dagegen eher dem *One-Port-Modell*, bei dem der Datenaustausch immer paarweise zwischen zwei Prozessoren erfolgt. Die *Dimension-Exchange-Verfahren*, denen der wesentliche Teil dieser Arbeit gewidmet ist, sind speziell für dieses Modell konstruiert.

1.6 Anforderungen an Loadbalancing-Verfahren

Eine erste Anforderung ergibt sich dadurch, dass die übergeordnete Rechnung für die Dauer des Loadbalancings unterbrochen werden muss, die Bestimmung des ausgleichenden Flusses (erste Phase) sollte also möglichst schnell sein. Bei der Geschwindigkeit heutiger Rechner und Netzwerkhardware wird die Schnelligkeit des Verfahrens allerdings erst bei sehr vielen Prozessoren zu einem wichtigen Kriterium. Zum zweiten sollte ein möglichst kleiner ausgleichender Fluss berechnet werden. Sind die Flüsse unnötig groß, bedeutet dies, dass Lasten auf Umwegen oder gar im Kreis verschoben werden und dann die zweite Phase (das Scheduling) zu lang dauert. Des weiteren sollte der Loadbalancing-Algorithmus numerisch stabil arbeiten, was bei unüberlegter Anwendung einiger Verfahren nicht garantiert ist.

Eine gleichzeitige Minimierung der Laufzeit und des Flusses hat sich bisher als unmöglich herausgestellt. Das Ziel in den nachfolgenden Kapiteln ist es daher, diesen Anforderungen möglichst nahe zu kommen.

1.7 Häufig verwendete Standardgraphen

Es gibt eine Reihe von Standardgraphen, die im Laufe der Arbeit immer wieder verwendet werden. Dies sind: der Pfad der Länge n (P_n), der Zyklus der Länge n (C_n), $k \times l$-Gitter ($G_{k,l}$), $k \times l$-Torus ($T_{k,l}$), Hypercube der Dimension d (H_d), Stern aus n Knoten (S_n), kompletter Graph aus n Knoten (K_n) und zufälliger Graph mit n Knoten und N Kanten ($R_{n,N}$). Siehe hierzu auch Tabelle 1.1. Quadratische Gitter $G_{k,k}$ und Tori $T_{k,k}$ werden im Folgenden mit G_k bzw. T_k bezeichnet.

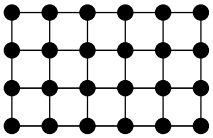
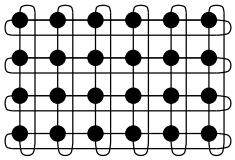
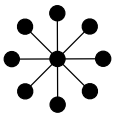
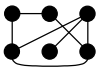
Graph	Abk.	Abbildung eines Beispiels
Pfad	P_6	
Zyklus	C_6	
Gitter	$G_{6,4}$	
Torus	$T_{6,4}$	
Hypercube	H_3	
Stern	S_9	
kompletter Graph	K_4	
zufälliger Graph	$R_{6,7}$	

Tabelle 1.1: Häufig verwendete Standardgraphen

1.8 Produktgraphen

Unter den für Loadbalancing verwendeten Graphen spielt die Klasse der so genannten Produktgraphen eine wichtige Rolle. Das Kapitel 4 ist einer speziell hierauf angepassten Klasse von Verfahren gewidmet, aber auch bei den einfacheren Verfahren lässt sich diese Struktur ausnutzen.

Definition 1.8. Es seien $G^{(1)} = (V^{(1)}, E^{(1)})$ und $G^{(2)} = (V^{(2)}, E^{(2)})$ zusammenhängende ungerichtete Graphen. Dann ist der Produktgraph $G = G^{(1)} \times G^{(2)}$ mit $G = (V, E)$ definiert durch

$$V = V^{(1)} \times V^{(2)}$$

und

$$E = \left\{ \left((u^{(1)}, u^{(2)}), (v^{(1)}, v^{(2)}) \right) \mid \left(u^{(1)} = v^{(1)} \wedge (u^{(2)}, v^{(2)}) \in E^{(2)} \right) \vee \left((u^{(1)}, v^{(1)}) \in E^{(1)} \wedge u^{(2)} = v^{(2)} \right) \right\}.$$

Bemerkung 1.9. Zum Teil werden zur Vereinfachung der Notation die Knoten mit einfachen Nummern statt mit Paaren bezeichnet.

Beispiel 1.10. Gitter und Torus lassen sich auffassen als Produkt zweier Pfade bzw. Zyklen und der Hypercube als mehrfaches Produkt von Pfaden der Länge zwei:

$$G_{k,l} = P_k \times P_l\,, \qquad T_{k,l} = C_k \times C_l\,, \qquad H_d = P_2 \times \cdots \times P_2\,.$$

Im Zusammenhang mit Produktgraphen werden an verschiedenen Stellen Kroneckerprodukte verwendet.

Definition 1.11. Für $\Bbbk \in \{\mathbb{R}, \mathbb{C}\}$ seien $A \in \Bbbk^{p\times q}$ und $B \in \Bbbk^{r\times s}$ zwei Matrizen. Das *Kroneckerprodukt* von A und B ist durch folgende Blockmatrix definiert:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1q}B \\ \vdots & \ddots & \vdots \\ a_{p1}B & \cdots & a_{pq}B \end{pmatrix} \in \Bbbk^{pr\times qs}$$

1.9 Bezeichnungen für spezielle Matrizen

Einheitsmatrizen der Dimension n werden mit I_n bezeichnet, Matrizen aus $\mathbb{R}^{n\times n}$, deren Einträge sämtlich eins sind, mit J_n. Wo die Dimension n offensichtlich ist, wird deren Angabe teilweise weggelassen.

2 Diffusionsverfahren

Bei den Diffusionsverfahren handelt es sich um die Verfahrensklasse, die speziell auf das All-Port-Rechnermodell ausgerichtet ist. Nach einigen Definitionen werden verschiedene Diffusionsverfahren vorgestellt, die alle innerhalb der letzten etwa zehn Jahre veröffentlicht wurden. Es folgen einige Bemerkungen zu Eigenwerten von Matrizen zur Beschreibung von Graphen sowie zu den Auswirkungen verschiedener Anordnungen der Eigenwerte auf die numerische Stabilität der Diffusionsverfahren.

2.1 Matrizen für Diffusionsverfahren

Zunächst werden mehrere Matrizen zur Beschreibung von Graphen definiert.

Definition 2.1. Es sei G ein zusammenhängender ungerichteter Graph und $\alpha \in (0;1)$ ein konstantes Kantengewicht. Definiere die Matrix $M^{\mathsf{Diff}} \in \mathbb{R}^{n\times n}$, $M^{\mathsf{Diff}} = \left(m_{ij}^{\mathsf{Diff}}\right)$ durch

$$m_{ij}^{\mathsf{Diff}} = \begin{cases} 1-\alpha\deg(i) & \text{falls } i=j \\ \alpha & \text{falls } \{i,j\}\in E \\ 0 & \text{sonst,} \end{cases}$$

wobei $\deg(i)$ den Grad des Knotens i bezeichnet. Sind alle $m_{ij}^{\mathsf{Diff}} \geq 0$, so nennt man M^{Diff} eine *Diffusionsmatrix* des Graphen G.

Bemerkung 2.2. Die Matrix M^{Diff} ist symmetrisch und doppelt stochastisch, das heißt alle Zeilensummen und Spaltensummen ergeben 1.

Graph-Beispiel 2.3.

$$M^{\mathsf{Diff}} = \begin{pmatrix} 1-\alpha & \alpha & 0 \\ \alpha & 1-2\alpha & \alpha \\ 0 & \alpha & 1-\alpha \end{pmatrix}$$

Im folgenden sei stets die nachstehende Zusatzvoraussetzung erfüllt. Sie garantiert die Konvergenz der Diffusionsverfahren, vergleiche Bemerkung 2.12.

Voraussetzung 2.4. Es gelte eine der beiden Bedingungen:

1. Ein Diagonalelement m_{ii}^{Diff} der Diffusionsmatrix M^{Diff} ist positiv.
2. Der Graph G ist nicht bipartit.

Definition 2.5. Die *Laplace-Matrix* $L^{\mathsf{Diff}} \in \mathbb{Z}^{n\times n}$, $L^{\mathsf{Diff}} = \left(l_{ij}^{\mathsf{Diff}}\right)$, von G wird definiert durch

$$l_{ij}^{\mathsf{Diff}} = \begin{cases} \deg(i) & \text{falls } i = j \\ -1 & \text{falls } \{i,j\} \in E \\ 0 & \text{sonst}\,. \end{cases}$$

Graph-Beispiel 2.6.

$$L^{\mathsf{Diff}} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

Bemerkung 2.7. Es gilt

$$M^{\mathsf{Diff}} = I - \alpha L^{\mathsf{Diff}}\,.$$

Der Sinn der nachfolgenden Definition wird später deutlich werden, wenn andere Verfahren betrachtet werden, bei denen die entsprechenden Matrizen (hier A und A^{Diff}) nicht mehr identisch sind.

Definition 2.8. Für einen Graphen G, auf den ein Diffusionsverfahren angewandt werden soll, definiert man

$$A^{\mathsf{Diff}} = A\,.$$

Bemerkung 2.9. Es gilt

$$L^{\mathsf{Diff}} = AA^{\mathsf{Diff}\,T}\,.$$

Mit diesen Bezeichnungen entspricht die Berechnung der Gleichverteilung der Lösung des singulären linearen Gleichungssystems $L^{\mathsf{Diff}}\overline{w} = 0$. Die Matrix L^{Diff} hat Rang $n-1$, nach [HJ85, Th. 8.4.4] ist nämlich 1 ein einfacher Eigenwert von M^{Diff} und 0 damit einfacher Eigenwert von L^{Diff}. Eindeutig wird die Lösung des Systems durch die Forderung, dass die Gesamtlast unverändert bleibt. Äquivalent hierzu ist die Suche nach dem Fixpunkt $\overline{w}$ in der Gleichung $M\overline{w} = \overline{w}$.

Die Laplace- und Diffusionsmatrizen der in Definition 1.8 eingeführten Produktgraphen lassen sich leicht aus den Matrizen der kleineren „Faktorgraphen“ bestimmen.

Lemma 2.10. *Es sei $G = G^{(1)} \times G^{(2)}$ ein Produktgraph mit $\left|V^{(1)}\right| = p$ und $\left|V^{(2)}\right| = q$. Dann ergeben sich die Diffusions- und Laplace-Matrix von G auf folgende Weise aus den Matrizen der Faktorgraphen $G^{(1)}$ und $G^{(2)}$, wobei für beide dasselbe Kantengewicht α verwendet wird:*

$$L^{\mathsf{Diff}} = I_q \otimes L^{\mathsf{Diff}\,(1)} + L^{\mathsf{Diff}\,(2)} \otimes I_p$$
$$M^{\mathsf{Diff}} = I_q \otimes M^{\mathsf{Diff}\,(1)} + M^{\mathsf{Diff}\,(2)} \otimes I_p - I_{pq}$$

Beweis. Die Gleichung für L stammt aus [EFMP99], die für M lässt sich hiermit leicht nachrechnen. □

Für einige der später zu betrachtenden Verfahren wird die Kenntnis der Eigenwerte der Diffusions- bzw. Laplace-Matrix vorausgesetzt.

Definition 2.11. Es seien M^{Diff} und L^{Diff} wie oben. Die Anzahl paarweise verschiedener Eigenwerte dieser Matrizen werde mit m bezeichnet. Die zu M^{Diff} gehörenden Eigenwerte seien so geordnet, dass

$$\mu_1^{\mathsf{Diff}} > \mu_2^{\mathsf{Diff}} > \cdots > \mu_m^{\mathsf{Diff}}$$

und die von L^{Diff} entsprechend

$$\lambda_1^{\mathsf{Diff}} < \lambda_2^{\mathsf{Diff}} < \cdots < \lambda_m^{\mathsf{Diff}}$$

mit $\mu_i^{\mathsf{Diff}} = 1 - \alpha\lambda_i^{\mathsf{Diff}}$.

Bemerkung 2.12. Es ist der größte Eigenwert $\mu_1^{\mathsf{Diff}} = 1$ mit zugehörigem Eigenvektor $(1, \ldots, 1)^T$ [HJ85, Cor. 8.1.30]. Unter Voraussetzung 2.4 gilt außerdem $\mu_m^{\mathsf{Diff}} > -1$ [Cyb89, Theorem 1].

Definition 2.13. Mit $\gamma^{\mathsf{Diff}} = \max\left\{\left|\mu_2^{\mathsf{Diff}}\right|, \left|\mu_m^{\mathsf{Diff}}\right|\right\}$ werde der betragsmäßig zweitgrößte Eigenwert von M^{Diff} bezeichnet.

Lemma 2.14 ([DFM99, Lemma 1]). *Es seien w^0 und $\overline{w}$ die Vektoren der anfänglichen Lastverteilung bzw. der Gleichverteilung. Ferner sei*

$$w^0 = \sum_{i=1}^{m} z_i$$

eine Darstellung von w^0 durch (nicht-normalisierte) Eigenvektoren z_i von M^{Diff}, d. h. $M^{\mathsf{Diff}} z_i = \mu_i^{\mathsf{Diff}} z_i$ für $i = 1, \ldots, m$. Dann gilt

$$\overline{w} = z_1.$$

Definition 2.15. Ein *polynomiales Loadbalancing-Verfahren* ist ein Verfahren, bei dem sich die Last im Schritt k ausdrücken lässt als

$$w^k = p_k(M^{\mathsf{Diff}})w^0$$

mit $p_k \in \overline{\Pi}_k$. Hierbei bezeichnet $\overline{\Pi}_k$ die Menge aller Polynome p vom Grad $\deg(p) \leq k$, für die $p(1) = 1$ ist.

Alle Verfahren, die hier betrachtet werden, sind polynomiale Verfahren im Sinne obiger Definition. Die Bedingung $p_k(1) = 1$ bewirkt, dass alle Matrizen $p_k(M^{\mathsf{Diff}})$ doppelt stochastisch sind. Dies stellt sicher, dass die Gesamtlast unverändert bleibt, d. h. dass $\sum_{i=1}^n w_i^k = \sum_{i=1}^n w_i^0$ ist.

Definition 2.16. Mit $e^k = w^k - \overline{w}$ werde der *Fehler* nach dem k-ten Iterationsschritt bezeichnet.

Lemma 2.17 ([DFM99, Lemma 2]). *Es sei $w^0 = \sum_{i=1}^{m} z_i$ wie in Lemma 2.14. Dann gilt:*

$$\begin{aligned} e^0 &= \sum_{i=2}^{m} z_i \\ e^k &= p_k(M^{\mathsf{Diff}})e^0 \\ &= \sum_{i=2}^{m} p_k(\mu_i^{\mathsf{Diff}})z_i, \quad k = 0, 1, 2, \ldots \end{aligned}$$

2.2 Einfache Diffusionsverfahren: FOS, SOS und Čebyšev

Das erste Diffusionsverfahren wurde 1989 von Cybenko entwickelt [Cyb89]. In jedem Schritt vergleichen die Prozessoren ihre aktuelle Last mit der aller ihrer Nachbarn. Über jede Kante wird das α-fache der Lastdifferenz in Richtung des Knotens mit der geringeren Last verschoben. Dieses Verfahren wird auch als *First Order Scheme (FOS)* bezeichnet. Für Prozessor i ergibt sich im Schritt k Algorithmus 2.1.

---

for all $e = \{i, j\} \in E$ **do**
  $y_e^{k-1} = \alpha\left(w_i^{k-1} - w_j^{k-1}\right)$
  $x_e^k = x_e^{k-1} + y_e^{k-1}$
end for
$w_i^k = w_i^{k-1} - \sum_{e=\{i,j\}\in E} y_e^{k-1}$

Algorithmus 2.1: Schritt k des FOS-Verfahrens für Prozessor i

Betrachtet man das Verfahren nun nicht mehr lokal aus Sicht eines Prozessors sondern global, so erhält das FOS-Verfahren bei Verwendung der Diffusionsmatrix M^{Diff} die einfache Gestalt aus Algorithmus 2.2.

$x^0 = 0$
for $k = 1, 2, \ldots$ **do**
  $w^k = M^{\mathsf{Diff}} w^{k-1}$
  $x^k = x^{k-1} + \alpha A^{\mathsf{Diff}^T} w^{k-1}$
end for

Algorithmus 2.2: FOS-Verfahren, Matrix-Version

Bei allen folgenden Verfahren wird jeweils nur noch die Matrix-Version des Algorithmus angegeben, die Beziehung zwischen lokaler und globaler Sicht ist in allen Fällen mit dem FOS vergleichbar.

Für die zugehörigen Polynome p_k^{FOS} mit $w^k = p_k^{\mathsf{FOS}}(M^{\mathsf{Diff}})w^0$ gilt

$$p_k^{\mathsf{FOS}}(t) = t^k;$$

diese erfüllen also die kurze Rekursion

$$p_0^{\mathsf{FOS}}(t) = 1$$
$$p_k^{\mathsf{FOS}}(t) = t p_{k-1}^{\mathsf{FOS}}(t) \qquad k = 1, 2, \ldots .$$

In [Cyb89] wird gezeigt, dass das FOS-Verfahren genau dann konvergiert, wenn die Zusatzvoraussetzung 2.4 erfüllt ist. Dann gilt nämlich für $\gamma = \gamma^{\mathsf{Diff}}$ aus Definition 2.13 die Beziehung $\gamma < 1$ und für die Fehler gilt

$$\left\| e^k \right\|_2 \leq \gamma^k \left\| e^0 \right\|_2 .$$

Der Wert γ wird dann möglichst klein und damit die Konvergenz im Sinne der obigen Abschätzung möglichst gut, wenn $\alpha = \frac{2}{\lambda_2 + \lambda_m}$ ist, wobei λ_2 und λ_m der zweitkleinste bzw. der größte Eigenwert der Laplace-Matrix L^{Diff} sind [Cyb89, EFMP99].

Falls γ nahe bei 1 liegt, konvergiert das FOS-Verfahren sehr langsam. Um die Konvergenz zu verbessern, haben Ghosh, Muthukrishnan und Schultz das *Second Order Scheme (SOS)* [MGS96] entwickelt, vgl. Algorithmus 2.3. Die Polynome erfüllen die Rekursion

$$p_0^{\mathsf{SOS}}(t) = 1$$
$$p_1^{\mathsf{SOS}}(t) = t$$
$$p_k^{\mathsf{SOS}}(t) = \beta t p_{k-1}^{\mathsf{SOS}}(t) + (1-\beta)\, p_{k-2}^{\mathsf{SOS}}(t).$$

Dabei ist $\beta \in (0; 2)$ ein fest gewählter Parameter.

$w^1 = M^{\mathsf{Diff}} w^0$
$x^1 = \alpha {A^{\mathsf{Diff}}}^T w^0$
for $k = 2, 3, \ldots$ **do**
  $w^k = \beta M^{\mathsf{Diff}} w^{k-1} + (1-\beta)\, w^{k-2}$
  $x^k = \beta x^{k-1} + \beta\alpha {A^{\mathsf{Diff}}}^T w^{k-1} + (1-\beta)\, x^{k-2}$
end for

Algorithmus 2.3: SOS-Verfahren

Das SOS-Verfahren konvergiert für jedes $\beta \in (0; 2)$, die beste Konvergenz ergibt sich für

$$\beta = \beta_{opt} = \frac{2}{1 + \sqrt{1 - \gamma^2}} ;$$

in diesem Fall gilt für die Fehler [MGS96]

$$\left\| e^k \right\|_2 \leq (\beta_{opt} - 1)^{\frac{k}{2}} \left(1 + k\sqrt{1-\gamma^2}\right) \left\| e^0 \right\|_2 .$$

Eng verwandt mit dem SOS-Verfahren ist das Čebyšev-Verfahren [DFM99]. Statt eines festen Parameters β werden in den einzelnen Schritten verschiedene Werte verwendet, nämlich

$$\beta_1 = 1, \quad \beta_2 = \frac{2}{2 - \gamma^2}, \quad \beta_k = \frac{4}{4 - \gamma^2 \beta_{k-1}}, \; k = 3, 4, \ldots .$$

Die zugehörige Fehlerabschätzung lautet

$$\left\|e^k\right\|_2 \leq 2\frac{(\beta_{opt}-1)^{\frac{k}{2}}}{1+(\beta_{opt}-1)^k}\left\|e^0\right\|_2 .$$

Das Čebyšev-Verfahren ist also geringfügig besser als SOS.

2.3 Endliche Diffusionsverfahren: OPS und OPT

Bei den bisher vorgestellten Verfahren handelt es sich um nicht-endliche Iterationsverfahren, die mehr oder weniger schnell gegen die Gleichverteilung $\overline{w}$ konvergieren. In [DFM98, DFM99] wurde ein neues Verfahren vorgestellt, das, ähnlich dem cg-Verfahren zur Lösung linearer Gleichungssysteme, in endlich vielen Schritten die exakte Lösung bestimmt. Dieses Verfahren wird als *Optimal Polynomial Scheme (OPS)* bezeichnet.

Das Verfahren basiert darauf, dass Polynome p_k^{OPS} aus der Menge $\overline{\Pi}_k$ berechnet werden, die bezüglich des folgenden Innenproduktes orthogonal sind:

$$\langle p,q\rangle = \sum_{j=2}^{m} \omega_j p(\mu_j^{\mathsf{Diff}}) q(\mu_j^{\mathsf{Diff}}) \tag{2.1}$$

mit $\omega_j = 1-\mu_j^{\mathsf{Diff}}$. Solche Orthogonalpolynome lassen sich mit Hilfe einer Dreitermrekursion berechnen. Das $(m-1)$-te Polynom ist zu sich selbst orthogonal, d. h. es ist $\langle p_{m-1}^{\mathsf{OPS}}, p_{m-1}^{\mathsf{OPS}}\rangle = 0$ und somit gilt $p_{m-1}^{\mathsf{OPS}}(\mu_i^{\mathsf{Diff}}) = 0$ für $i = 2,\ldots,m$.

Um das Verfahren durchführen zu können, müssen zunächst alle Eigenwerte μ_i^{Diff} der Diffusionsmatrix M^{Diff} bekannt sein. Außerdem müssen vorab Parameter α_i, β_i und γ_i berechnet werden. Für $k = 0,\ldots,m-1$ sind die Polynome p_k^{OPS} gegeben durch

$$\begin{aligned}
p_0^{\mathsf{OPS}}(t) &= 1 \\
p_1^{\mathsf{OPS}}(t) &= \frac{1}{\gamma_1}\left[(\alpha_1 - t)\, p_0^{\mathsf{OPS}}(t)\right] \\
p_k^{\mathsf{OPS}}(t) &= \frac{1}{\gamma_k}\left[(\alpha_k - t)\, p_{k-1}^{\mathsf{OPS}}(t) - \beta_k p_{k-2}^{\mathsf{OPS}}(t)\right], \qquad k = 2,\ldots,m-1
\end{aligned}$$

mit

$$\alpha_k = \frac{\langle t p_{k-1}^{\mathsf{OPS}}, p_{k-1}^{\mathsf{OPS}}\rangle}{\langle p_{k-1}^{\mathsf{OPS}}, p_{k-1}^{\mathsf{OPS}}\rangle}, \qquad k = 1,\ldots,m-1 \tag{2.2}$$

$$\beta_k = \gamma_{k-1}\frac{\langle p_{k-1}^{\mathsf{OPS}}, p_{k-1}^{\mathsf{OPS}}\rangle}{\langle p_{k-2}^{\mathsf{OPS}}, p_{k-2}^{\mathsf{OPS}}\rangle}, \qquad k = 2,\ldots,m-1 \tag{2.3}$$

$$\gamma_1 = \alpha_1 - 1, \quad \gamma_k = \alpha_k - 1 - \beta_k, \qquad k = 2,\ldots,m-1\,. \tag{2.4}$$

Sind diese Größen einmal berechnet, dann kann das eigentliche in Algorithmus 2.4 dargestellte OPS-Verfahren durchgeführt werden.

$w^1 = \frac{1}{\gamma_1}\left[\alpha_1 w^0 - M^{\mathsf{Diff}} w_0\right]$
$x^1 = \frac{1}{\gamma_1}\alpha {A^{\mathsf{Diff}}}^T w^0$
for $k = 2, \ldots, m-1$ **do**
  $w^k = \frac{1}{\gamma_k}\left[\alpha_k w^{k-1} - M^{\mathsf{Diff}} w^{k-1} - \beta_k w^{k-2}\right]$
  $x^k = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x^{k-1} - \alpha {A^{\mathsf{Diff}}}^T w^{k-1} - \beta_k x^{k-2}\right]$
end for

Algorithmus 2.4: OPS-Verfahren

Allgemein gilt bei polynomialen Loadbalancing-Verfahren nach Lemma 2.17 für die Fehler

$$e^k = \sum_{i=2}^{m} p_k(\mu_i^{\mathsf{Diff}}) z_i \,,$$

wobei die z_i wieder Eigenvektoren von M^{Diff} sind. Beachtet man, dass die z_i paarweise orthogonal sind, erhält man nach Übergang zur euklidischen Norm

$$\left\|e^k\right\|_2^2 = \sum_{i=2}^{m} p_k(\mu_i^{\mathsf{Diff}})^2 \left\|z_i\right\|_2^2 \leq \left(\sum_{i=2}^{m} p_k(\mu_i^{\mathsf{Diff}})^2\right) \max_{i=2}^{m} \left\|z_i\right\|_2^2 \,.$$

In [DFM99] wird gezeigt, dass der Faktor $\sum_{i=2}^{m} p_k(\mu_i^{\mathsf{Diff}})^2$ aus der letzten Abschätzung für obige Wahl der ω_i in jedem Schritt minimal wird. Insbesondere ist $e^{m-1} = \sum_{i=2}^{m} p_{m-1}(\mu_i^{\mathsf{Diff}}) z_i = 0$ und damit $w^{m-1} = \overline{w}$.

Die Konvergenz der verschiedenen Diffusionsverfahren (FOS, SOS, Čebyšev und OPS) ist in Abbildung 2.1 am Beispiel eines Zyklus der Länge 12 dargestellt. Für die Parameter α bzw. β wurden in den entsprechenden Verfahren jeweils die optimalen Werte gewählt. In dem Diagramm wird die Überlegenheit des OPS-Verfahrens deutlich, SOS und Čebyšev unterscheiden sich nur geringfügig, FOS ist nicht konkurrenzfähig. In der Abbildung wie auch in allen anderen Beispielen, sofern nicht anders angegeben, beträgt die Gesamtlast $100 \cdot n$, wobei n die Anzahl der Knoten ist. Die Ausgangsverteilung wurde zufällig erzeugt. Bei den nicht-endlichen Verfahren wird jeweils iteriert, bis der absolute Fehler in der l_2-Norm garantiert kleiner ist als 0,5. Denn schließlich muss am Ende der Rechnung jeder Lastwert auf die nächste natürliche Zahl gerundet werden, da nur ganze Lasteinheiten verschoben werden können. In den Beispielen bleibt also ein relativer Fehler von 0,5 %. Beim Vergleich der Verfahren sollte man also nicht vergessen, dass die Anzahl der Schritte bei den nicht-endlichen Verfahren mit wachsender Gesamtlast zunimmt, bei den endlichen Verfahren dagegen konstant ist.

Das OPS-Verfahren ist im Gegensatz zu den anderen Verfahren etwas aufwändiger zu implementieren, da zunächst die Skalare α_k, β_k und γ_k aus den Eigenwerten berechnet werden müssen. In [EFMP99] wird mit dem *OPT*-Verfahren ein anderer Algorithmus vorgestellt, der wesentlich einfacher und unmittelbar verständlich ist. Er kommt ohne die Dreitermrekursion aus, vgl. Algorithmus 2.5. Auch wenn die Eigenwerte im Algorithmus aufsteigend durchlaufen werden, hat die Reihenfolge bei exakter Rechnung keinen

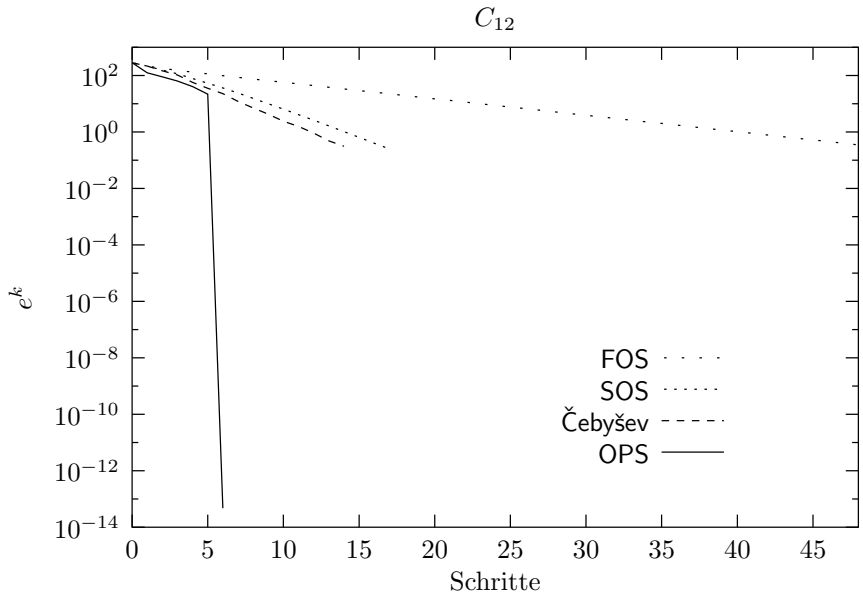


Abbildung 2.1: Konvergenz verschiedener Diffusionsverfahren: Fehler nach jedem Schritt beim Zyklus C_{12}

Einfluss auf das Ergebnis. Aus Gründen der numerischen Stabilität sollten allerdings spezielle Anordnungen gewählt werde, siehe hierzu Abschnitt 2.5.

$x^0 = 0$
for $k = 1, \ldots, m-1$ **do**
$w^k = \left(I - \frac{1}{\lambda_{k+1}^{\mathsf{Diff}}} L^{\mathsf{Diff}}\right) w^{k-1}$
$x^k = x^{k-1} + \frac{1}{\lambda_{k+1}^{\mathsf{Diff}}} {A^{\mathsf{Diff}}}^T w^{k-1}$
end for

Algorithmus 2.5: OPT-Verfahren

Satz 2.18 ([EFMP99]). *Das OPT-Verfahren endet nach $m-1$ Iterationsschritten mit $w^{m-1} = \overline{w}$.*

Beweis. Es sei $w^0 = \sum_{i=1}^{m} z_i$ wie in Lemma 2.14:

$$\begin{aligned} w^{m-1} &= \prod_{k=m-1}^{1} \left(I - \frac{1}{\lambda_{k+1}^{\mathsf{Diff}}} L^{\mathsf{Diff}}\right) \sum_{i=1}^{m} z_i \\ &= \sum_{i=1}^{m} \prod_{k=m-1}^{1} \left(1 - \frac{\lambda_i^{\mathsf{Diff}}}{\lambda_{k+1}^{\mathsf{Diff}}}\right) z_i \\ &= z_1 \end{aligned}$$

Nach Lemma 2.14 aber ist $z_1 = \overline{w}$. □

Um zu sehen, dass auch OPT ein polynomiales Verfahren im Sinne von Definition 2.15 ist, lässt sich die Iterationsvorschrift auch auf folgende Weise unter Verwendung der Matrix M^{Diff} schreiben:

$$w^k = \left(\frac{1}{1-\mu_{k+1}^{\mathsf{Diff}}} M^{\mathsf{Diff}} - \frac{\mu_{k+1}^{\mathsf{Diff}}}{1-\mu_{k+1}^{\mathsf{Diff}}} I \right) w^{k-1} \tag{2.5}$$

Bemerkung 2.19. Die nach $m-1$ Schritten von OPS und OPT berechneten Polynome sind identisch, denn es gibt nur ein eindeutig bestimmtes Polynom vom Grad $m-1$ mit den Eigenschaften $p(\mu_i^{\mathsf{Diff}}) = 0$ für $i = 2, \ldots, m$ und $p(1) = 1$, siehe auch [DFM99]:

$$p_{m-1}^{\mathsf{OPS}}(t) = p_{m-1}^{\mathsf{OPT}}(t) = \prod_{i=2}^{m} \frac{t - \mu_i^{\mathsf{Diff}}}{1 - \mu_i^{\mathsf{Diff}}}$$

Bei exakter Rechnung sind die beiden Verfahren OPS und OPT also gleichwertig bezüglich der Anzahl der benötigten Schritte und des Flusses.

Bemerkung 2.20. Wegen der engen Verwandtschaft von OPS und OPT gelten viele der im folgenden untersuchten Eigenschaften für beide gleichermaßen. In solchen Fällen wird abkürzend OPX geschrieben.

2.4 Eigenwerte von Graphen

Es mag als ein großer Nachteil des OPS- bzw. OPT-Verfahrens erscheinen, dass vor dem eigentlichen Loadbalancing erst alle Eigenwerte des Graphen berechnet werden müssen. Für die wichtigsten Standardgraphen, die als Parallelrechnertopologien in der Praxis auftreten, sind die Eigenwerte jedoch bekannt. Tabelle 2.1 enthält die Eigenwerte der meisten in Abschnitt 1.7 eingeführten Standardgraphen. Die angegeben Formeln stammen aus [CDS95, Els02]. Bei den Formeln, die für Zyklus, Gitter und Torus angegeben

Graph	Abk.	$\lambda_i^{\mathsf{Diff}}$
Pfad	P_n	$2 - 2\cos\left(\frac{\pi}{n} j\right),\ j = 0, \ldots, n-1$
Zyklus	C_n	$2 - 2\cos\left(\frac{2\pi}{n} j\right),\ j = 0, \ldots, n-1$
Gitter	G_{n_1,n_2}	$4 - 2\left(\cos\left(\frac{\pi}{n_1} j_1\right) + \cos\left(\frac{\pi}{n_2} j_2\right)\right),\ j_i = 0, \ldots, n_i - 1, i = 1, 2$
Torus	T_{n_1,n_2}	$4 - 2\left(\cos\left(\frac{2\pi}{n_1} j_1\right) + \cos\left(\frac{2\pi}{n_2} j_2\right)\right),\ j_i = 0, \ldots, n_i - 1, i = 1, 2$
Hypercube	H_d	$0, 2, \ldots, 2d$
Stern	S_n	$0, 1, n$
Kompl. Gr.	K_n	$0, n$

Tabelle 2.1: Eigenwerte der Laplace-Matrizen einiger Standardgraphen

sind, ist zu beachten, dass einige Eigenwerte mehrfach auftreten können. Falls der Graph

G keinem der obigen Standardgraphen entspricht, müssen die Eigenwerte zu Beginn der Rechnung *einmal* bestimmt werden, anschließend kann das OPS- oder OPT-Verfahren mit diesen Eigenwerten beliebig oft durchgeführt werden.

Das folgende Resultat erklärt in Verbindung mit Lemma 2.10 nochmals die Eigenwertformeln für Gitter, Tori und (bei mehrfacher Anwendung) auch Hypercubes. Ferner erlaubt es, die Eigenwerte beliebiger anderer Produktgraphen effizient zu berechnen.

Lemma 2.21 ([HJ91, Theorem 4.4.5]). *Es sei $A \in \mathbb{C}^{p\times p}$ und $B \in \mathbb{C}^{q\times q}$. Sind λ und μ Eigenwerte von A bzw. B mit zugehörigen Eigenvektoren x und y, dann ist $\lambda+\mu$ Eigenwert des* Kronecker-Produktes *$I_q \otimes A + B \otimes I_p$ mit Eigenvektor $y \otimes x$. Jeder Eigenwert von $I_q \otimes A + B \otimes I_p$ lässt sich als ein solches Produkt schreiben.*

Wie lange das Loadbalancing mit OPS oder OPT dauert, hängt ab von der Anzahl m der verschiedenen Eigenwerte von L^{Diff} bzw. M^{Diff}. Für einige Graphen, bei denen Formeln für die Eigenwerte existieren, sind die entsprechenden Werte von m in Tabelle 2.2 abgedruckt. Die Werte für Pfad, Zyklus, Hypercube und Stern ergeben sich unmittelbar aus den Formeln in Tabelle 2.1. Für quadratische Gitter gilt offenbar $m(G) \leq \frac{1}{2}m(P)\,(m(P)+1)$ und für Tori entsprechendes. Hiermit erhält man die Faktoren $\frac{1}{2}$ bzw. $\frac{1}{8}$ vor k^2. Für die tatsächlichen Zahlen ist zu beachten, dass weitere Eigenwerte doppelt auftreten. Die Obergrenzen für Gitter und Tori in der Tabelle sind experimentell ermittelt.

Graph		Anzahl
P_n		n
C_n	$2 \mid n$	$\frac{1}{2}n+1$
	$2 \nmid n$	$\frac{1}{2}n+\frac{1}{2}$
G_k	$2 \mid k$	$\leq \frac{1}{2}k^2+1$
	$2 \nmid k$	$\leq \frac{1}{2}k^2+\frac{3}{2}$
T_k	$4 \mid k$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+1$
	$2 \mid k, 4 \nmid k$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{2}$
	$2 \nmid k$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{8}$
H_d		$d+1$
S_n		3
K_n		2

Tabelle 2.2: Anzahl der verschiedenen Eigenwerte bei Standardgraphen

2.5 Stabilität und Sortierung der Eigenwerte

Im Gegensatz zum OPS-Verfahren hat die Reihenfolge der Eigenwerte $\lambda_i^{\mathsf{Diff}}$ beim OPT-Verfahren entscheidenden Einfluss auf die numerische Stabilität. Sortiert man die Eigenwerte aufsteigend, so ergeben sich nach wenigen Schritten sehr große Fehler e^k. Bei

etwas größeren Graphen können die Werte 10^{16} übersteigen, selbst bei moderaten Anfangslasten unter 100. Bei Rechnung mit doppelter Genauigkeit liegt dann auch der abschließende Fehler e^{m-1} über 10^0, d. h. das Resultat ist völlig unbrauchbar. Sortiert man die Eigenwerte in absteigender Reihenfolge, so nehmen die Fehlerwerte zuerst monoton ab. Der abschließende Fehler ist jedoch in vielen Fällen nur knapp unterhalb 10^0, bei größeren Graphen lässt sich sogar ein starker Anstieg in den letzten Iterationsschritten beobachten.

Zur Erzielung besserer Ergebnisse wird in [EFMP99] eine Sortierung „von der Mitte nach außen“ vorgeschlagen, die auf Young (1953) zurückgeht.

Beispiel 2.22.

- $\text{SortYoung}\left((1,2,3,4,5,6,7,8,9,10,11)\right) = (6,5,7,4,8,3,9,2,10,1,11)$,
- $\text{SortYoung}\left((1,2,3,4,5,6,7,8,9,10,11,12)\right) = (6,7,5,8,4,9,3,10,2,11,1,12)$

Bei einigen (größeren) Graphen liefert aber auch diese Sortierung unbrauchbare Resultate.

Eine Sortierung, die sich für fast alle praktisch relevanten Fälle als sehr gut erwiesen hat, beruht auf gewichteten *Leja-Punkten* [Rei91]. Diese Sortierung wurde zuerst in [EFMP99] mit OPT in Verbindung gebracht und in [Els02] bereits näher untersucht.

Definition 2.23 (Leja-Punkte). Es sei $\{a_1, \ldots, a_n\}$ eine Menge reeller oder komplexer Zahlen und $\omega : \mathbb{C} \to \mathbb{R}^+$ eine Gewichtsfunktion. Die Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ sei folgendermaßen definiert: Wähle $\pi(1)$ so, dass

$$\left|a_{\pi(1)}\right| = \max_{i=1}^{n} |a_i|\, \omega\left(a_i\right)$$

und für $j = 2, \ldots, n$ wähle $\pi(j)$ so, dass

$$\prod_{l=1}^{j-1} \left|1 - \frac{a_{\pi(j)}}{a_{\pi(l)}}\right| \omega\left(a_{\pi(j)}\right) = \max_{i \in \{1,\ldots,n\} \setminus \{\pi(1),\ldots,\pi(j-1)\}} \prod_{l=1}^{j-1} \left|1 - \frac{a_i}{a_{\pi(l)}}\right| \omega\left(a_i\right).$$

Bemerkung 2.24. Speichert man in jedem Schritt die betrachteten Produkte zwischen, so beträgt der Aufwand zur Berechnung der Leja-Sortierung $\mathcal{O}\left(n^2\right)$ verglichen mit $\mathcal{O}\left(n \log n\right)$ bei einfachen Sortierungen.

Reichel [Rei91] schlägt als Gewichtsfunktionen $\omega(z) = 1$ und $\omega(z) = |z|$ vor. Später werden als Verallgemeinerung Funktionen $\omega(z) = |z|^g$ betrachtet werden. Diese Sortierungen werden im folgenden mit $\text{Leja}^{(g)}$ bezeichnet; die beiden Spezialfälle heißen also $\text{Leja}^{(0)}$ bzw. $\text{Leja}^{(1)}$.

Beispiel 2.25. Die folgenden Beispiele zeigen Leja-Sortierungen der Zahlen von 1 bis 12 mit verschiedenen Gewichtsfunktionen.

- $\text{Leja}^{(0)}\left((1,2,3,4,5,6,7,8,9,10,11,12)\right) = (12,1,6,9,3,11,2,8,4,10,5,7)$
- $\text{Leja}^{(\frac{1}{2})}\left((1,2,3,4,5,6,7,8,9,10,11,12)\right) = (12,4,8,1,11,2,6,10,3,9,5,7)$

- $\mathsf{Leja}^{(1)}\left((1,2,3,4,5,6,7,8,9,10,11,12)\right) = (12,6,3,10,1,8,11,2,5,9,4,7)$
- $\mathsf{Leja}^{(10)}\left((1,2,3,4,5,6,7,8,9,10,11,12)\right) = (12,11,9,10,7,8,5,6,4,3,2,1)$

Bei größeren Exponenten dominiert die Gewichtsfunktion und die Sortierung wird irgendwann zur absteigenden Sortierung.

Wie sich die verschiedenen Sortierungen auf die numerische Stabilität auswirken, ist anhand zweier beispielhafter Graphen in Abbildung 2.3 dargestellt. Dort sind aufsteigende, absteigende Sortierung sowie Young- und Leja-Sortierung gegenübergestellt. Mit der Leja-Sortierung werden eindeutig die besten Ergebnisse erzielt; dabei steigt der Fehler bei $\mathsf{Leja}^{(0)}$ im Gegensatz zu $\mathsf{Leja}^{(1)}$ anfangs zwar an, der abschließende Fehler ist aber in den Beispielen wie in der Regel mit $\mathsf{Leja}^{(0)}$ etwas geringer.

Nachdem gezeigt wurde, dass das OPT-Verfahren mit der Leja-Sortierung die besten Resultate liefert, soll dieses nun mit dem (bei exakter Rechnung äquivalenten) OPS-Verfahren verglichen werden. Wie aus Abbildung 2.4 ersichtlich ist, liefern die beiden Verfahren für praktisch relevante Graphen in etwa gleich gute Ergebnisse. Lediglich bei extrem großen Graphen wie dem 24×24-Gitter zeigt sich die Überlegenheit von OPS. Neben solchen besonders strukturierten Graphen werden in Abbildung 2.4 auch zwei Zufallsgraphen betrachtet. Die Diagramme zeigen, dass bei „dünnen“ Zufallsgraphen, insbesondere bei Zufallsbäumen, alle bisher betrachteten Verfahren numerische Probleme haben.

2.6 Flüsse

In den bisher besprochenen Algorithmen ist neben der Gleichverteilung der Last jeweils auch schon der dazugehörende ausgleichende Fluss mit bestimmt worden. In [DFM99] und [HB99] ist nachgewiesen worden, dass diese Flüsse in der l_2-Norm minimal sind. Man könnte sich nun fragen, ob ein l_1- oder l_∞-minimaler Fluss nicht „besser“ wäre. Ein l_1-minimaler Fluss bewirkt, dass die Last entlang kürzester Wege verschoben wird, l_∞-Minimalität bedeutet, dass der maximale Fluss über jede einzelne Kante möglichst klein ist. Die l_2-Minimalität stellt einen guten Kompromiss zwischen diesen Alternativen dar und kostet keinerlei zusätzliche Rechenzeit. Vergleiche hierzu auch Abbildung 2.2.

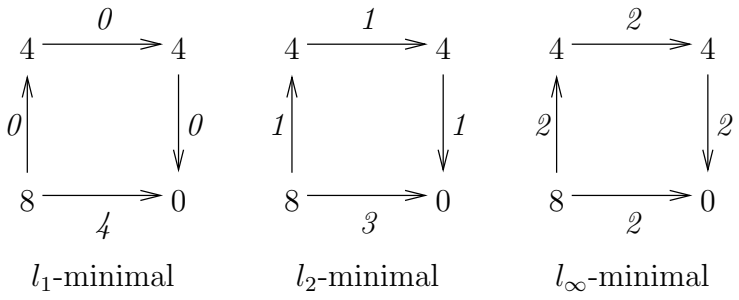


Abbildung 2.2: l_1-, l_2- und l_∞-minimaler Fluss am Beispiel des Zyklus C_4

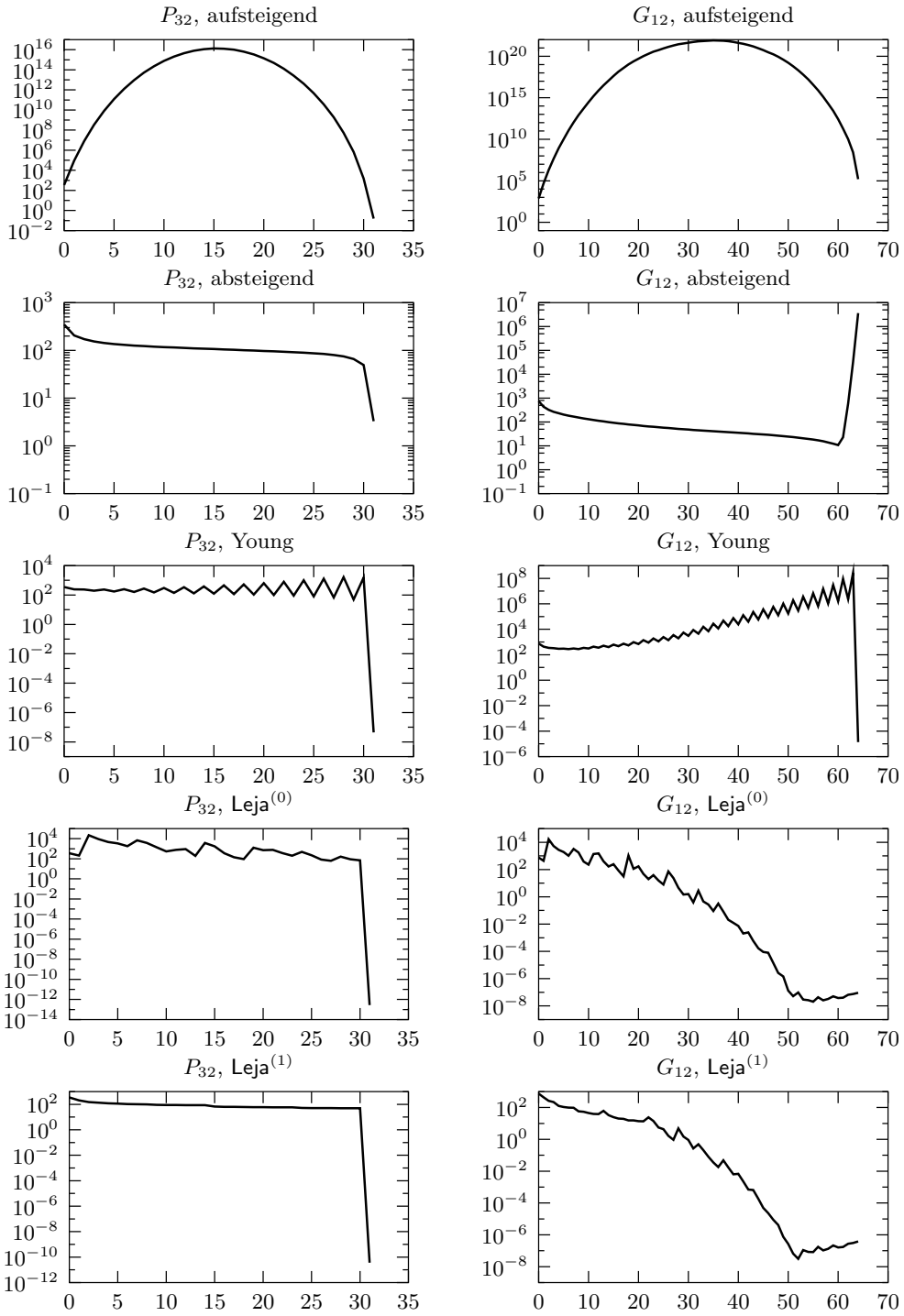


Abbildung 2.3: Numerische Stabilität bei verschiedenen Sortierungen der Eigenwerte am Beispiel des Pfades P_{32} und des Gitters G_{12}; dargestellt ist der Fehler nach jedem Schritt des Verfahrens

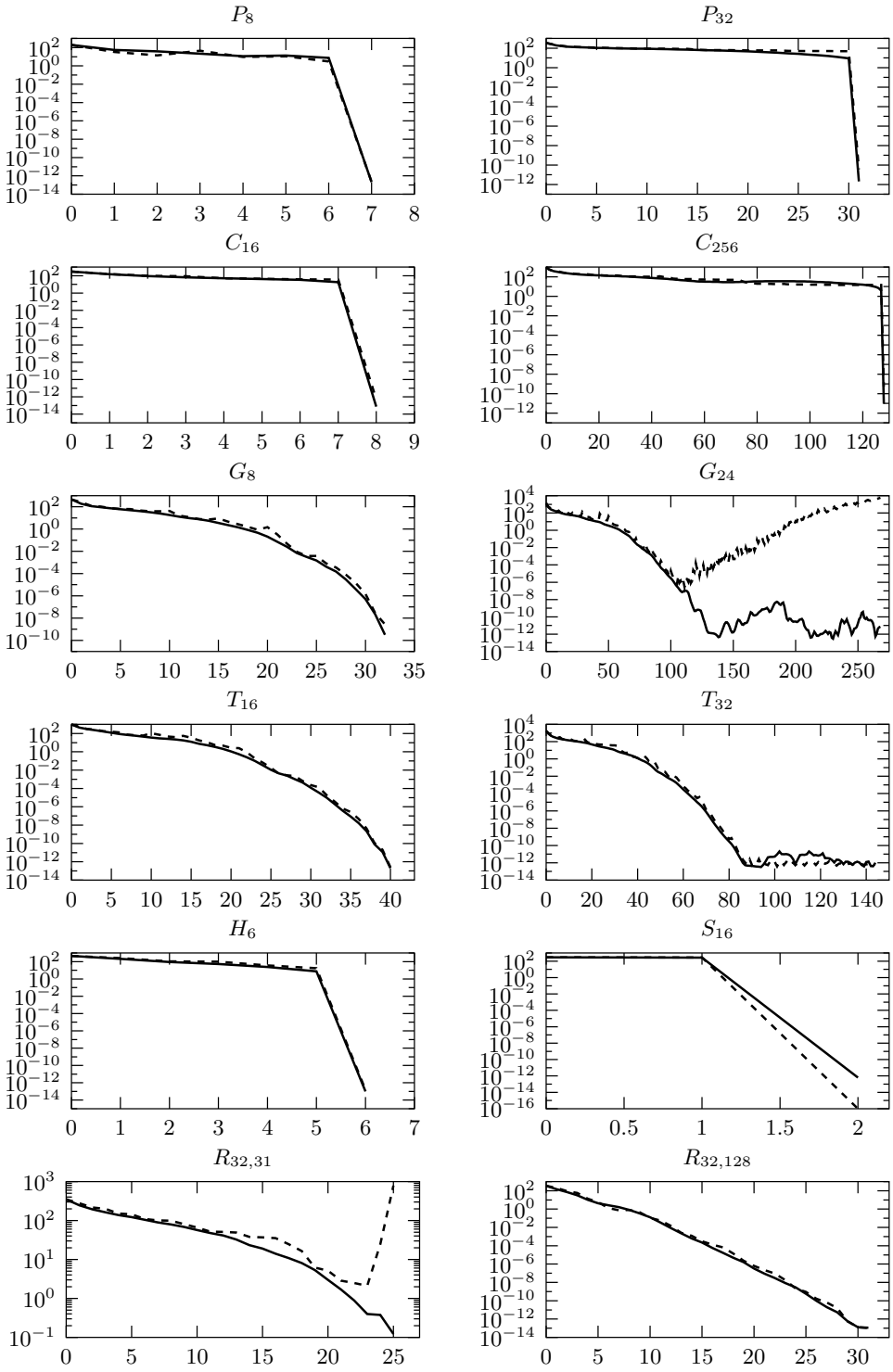


Abbildung 2.4: Vergleich von OPS (durchgezogene Linie) und OPT (gestrichelt)

3 Dimension-Exchange-Verfahren

In diesem Kapitel wird gezeigt, wie man aus den Diffusionsverfahren des letzten Kapitels sowie einigen vorhandenen Ansätzen zu Dimension-Exchange-Verfahren erster Ordnung neue Algorithmen konstruieren kann. Diese neuen Verfahren werden danach mit ihren Diffusionsgegenstücken verglichen. Besondere Aufmerksamkeit wird dabei der Frage geschenkt, inwieweit man auch hier Eigenwerte mit einfachen Formeln berechnen kann und wie die Flüsse bei den neuen Verfahren aussehen.

3.1 Motivation und Einfärbung von Graphen

Bei den Diffusionsverfahren wird davon ausgegangen, dass jeder Prozessor in einem Schritt mit allen benachbarten Prozessoren gleichzeitig kommuniziert, d. h. Diffusionsverfahren sind auf das All-Port-Modell spezialisiert. Bei den meisten Parallelrechnern ist dies jedoch nicht möglich. Stattdessen muss ein Prozessor nacheinander mit seinen Nachbarn kommunizieren (One-Port-Modell). Aus diesem Grund ist es sinnvoll, die einzelnen Schritte der Verfahren weiter aufzuteilen in Teilschritte, ein Teilschritt für jeden Nachbarn. In jedem Teilschritt können dann die aktuellen Informationen aus dem jeweils letzten Teilschritt verwendet werden.

Dieses Vorgehen wurde von Cybenko in [Cyb89] zuerst für den Hypercube vorgeschlagen. Ein Teilschritt entspricht dann genau einer Richtung oder Dimension, er umfasst also alle parallelen Kanten. Daher wird das Verfahren als *Dimension Exchange* bezeichnet.

Es sei i ein Knoten eines Hypercubes H_d und $\text{nachbar}(i,l)$ der Nachbar von i in der Dimension l, (bei geeigneter Nummerierung der Knoten unterscheiden sich i und $\text{nachbar}(i,l)$ genau im l-ten Bit). Damit ergibt sich das in Algorithmus 3.1 aus Sicht des Knotens i dargestellte Verfahren:

---

for $k = 1, 2, \ldots$ **do**
  for $l = 1, \ldots, d$ **do**
    $w_i^{k-1+\frac{l}{d}} = (1-\alpha)\, w_i^{k-1+\frac{l-1}{d}} + \alpha w_{\text{nachbar}(i,l)}^{k-1+\frac{l-1}{d}}$
  end for
end for

Algorithmus 3.1: Dimension Exchange für den Hypercube H_d für Prozessor i

In [Cyb89] wird gezeigt, dass für $\alpha = \frac{1}{2}$ ein Schritt ausreicht, d. h. es ist $w^1 = \overline{w}$.

Beim Hypercube werden die einzelnen Teilschritte durch die verschiedenen Dimensionen (Richtungen der Kanten) festgelegt. Will man den Dimension-Exchange-Ansatz auf beliebige Graphen übertragen, so muss sichergestellt werden, dass jeder Prozessor pro Teilschritt nur mit höchstens einem Nachbarn kommuniziert. Dies wird durch eine Einfärbung der Kanten erreicht. Diese Idee findet sich bereits in [HLM+90] und [XL92].

Definition 3.1. Eine *Einfärbung der Kanten* (kurz Einfärbung) des Graphen $G = (V, E)$ mit c Farben ist eine Aufteilung der Kantenmenge E in c paarweise disjunkte, nicht-leere Teilmengen E_i,

$$E = E_1 \cup \cdots \cup E_c \,,$$

wobei für jedes E_i gilt, dass keine zwei Kanten aus E_i mit demselben Knoten inzident sind.

Die Anzahl der zur Einfärbung verwendeten Farben bestimmt die Anzahl der Teilschritte pro komplettem Schritt und beeinflusst so die Gesamtlaufzeit der Verfahren. Es stellt sich also die Frage, wie viele Farben notwendig sind, um einen gegebenen Graphen einzufärben.

Definition 3.2. Es sei $G = (V, E)$ ein Graph. Der *chromatische Index* von G ist definiert als die minimale Anzahl von Farben, die notwendig ist, um die Kanten von G gemäß Definition 3.1 einzufärben. Der chromatische Index wird mit $\chi'(G)$ bezeichnet.

Der folgende Satz, dessen Beweis z. B. in [FW78] zu finden ist, gibt nun Antwort auf obige Frage.

Satz 3.3. *Es sei G ein Graph mit maximalem Grad ϱ. Dann gilt für den chromatischen Index von G*

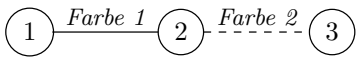
$$\varrho \leq \chi'(G) \leq \varrho + 1 \,.$$

Wie in [Arj82] festgestellt wird, ist das Problem der Bestimmung einer Einfärbung aus ϱ Farben, falls diese existiert, jedoch NP-vollständig. Eine Einfärbung mit $\varrho + 1$ Farben ist dagegen in polynomialer Zeit berechenbar.

3.2 Matrizen für gefärbte Graphen

Definition 3.4. Es sei $G = (V, E)$ ein Graph mit Einfärbung $E_1, \ldots, E_c$. Diese Färbung definiert *Teilgraphen* $G_i = (V, E_i)$, $i = 1, \ldots, c$. Die Diffusions- und Laplace-Matrizen zu G_i werden mit M_i bzw. L_i bezeichnet. Die Inzidenzmatrix $A_i \in \{0; 1; -1\}^{n \times N}$ erhält man, indem man alle Spalten von A, die nicht zur Farbe i gehören, durch Nullspalten ersetzt; im Gegensatz zur üblichen Definition einer Inzidenzmatrix werden die Nullspalten hier also nicht entfernt.

Graph-Beispiel 3.5.



$$E_1 = \{\{1,2\}\} \qquad E_2 = \{\{2,3\}\}$$

$$A_1 = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 0 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1-\alpha & \alpha & 0 \\ \alpha & 1-\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1-\alpha & \alpha \\ 0 & \alpha & 1-\alpha \end{pmatrix}$$

$$L_1 = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad L_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

Das folgende sofort herleitbare Lemma fasst einige Eigenschaften der neu eingeführten Matrizen zusammen.

Lemma 3.6. *Für die Laplace- und Inzidenzmatrizen der Teilgraphen gilt:*

$$\begin{aligned} L_i &= A_i A_i^T \\ A = A^{\mathsf{Diff}} &= \sum_{i=1}^{c} A_i \\ L^{\mathsf{Diff}} &= \sum_{i=1}^{c} L_i \\ A_i A_j^T &= 0 \qquad \text{für } i \neq j \end{aligned}$$

Die neu einzuführenden Verfahren beruhen alle darauf, die Diffusionsmatrizen der einzelnen Farben hintereinander auf den Lastvektor anzuwenden. Um eine möglichst kompakte Notation zu ermöglichen, werden die folgenden Matrizen eingeführt.

Definition 3.7. Es seien $M_1, \ldots, M_c$ die Diffusionsmatrizen der Teilgraphen zu jeweils einer Farbe. Dann werden die folgenden Matrizen definiert:

$$\begin{aligned} M^{\mathsf{DE}} &= M_c \cdots M_1 \\ M^{\mathsf{SDE}} &= M_1 \cdots M_c \cdot M_c \cdots M_1 = {M^{\mathsf{DE}}}^T M^{\mathsf{DE}} \\ L^{\mathsf{DE}} &= \frac{1}{\alpha}\left(I - M^{\mathsf{DE}}\right) \\ L^{\mathsf{SDE}} &= \frac{1}{\alpha}\left(I - M^{\mathsf{SDE}}\right) \end{aligned}$$

Hierbei steht (S)DE für (symmetrisches) Dimension Exchange.

Graph-Beispiel 3.8.

$$M^{\mathsf{DE}} = M_2 M_1 = \begin{pmatrix} 1-\alpha & \alpha & 0 \\ \alpha(1-\alpha) & (1-\alpha)^2 & \alpha \\ \alpha^2 & \alpha(1-\alpha) & 1-\alpha \end{pmatrix}$$

$$M^{\mathsf{SDE}} = M^{\mathsf{DE}^T} M^{\mathsf{DE}} = \begin{pmatrix} (1+\alpha^2)(1-\alpha)^2+\alpha^4 & (\alpha+\alpha^3)(1-\alpha)+\alpha(1-\alpha)^3 & 2\alpha^2(1-\alpha) \\ (\alpha+\alpha^3)(1-\alpha)+\alpha(1-\alpha)^3 & \alpha^2+(1-\alpha)^4+\alpha^2(1-\alpha)^2 & 2\alpha(1-\alpha)^2 \\ 2\alpha^2(1-\alpha) & 2\alpha(1-\alpha)^2 & \alpha^2+(1-\alpha)^2 \end{pmatrix}$$

$$L^{\mathsf{DE}} = \frac{1}{\alpha}\left(I - M^{\mathsf{DE}}\right) = \begin{pmatrix} 1 & -1 & 0 \\ \alpha-1 & 2-\alpha & -1 \\ -\alpha & \alpha-1 & 1 \end{pmatrix}$$

$$L^{\mathsf{SDE}} = \frac{1}{\alpha}\left(I - M^{\mathsf{SDE}}\right) = \begin{pmatrix} 2\left(1-\alpha+\alpha^2-\alpha^3\right) & -2+4\alpha-4\alpha^2+2\alpha^3 & 2\alpha(\alpha-1) \\ -2+4\alpha-4\alpha^2+2\alpha^3 & 4-8\alpha+6\alpha^2-2\alpha^3 & -2(1-\alpha)^2 \\ 2\alpha(\alpha-1) & -2(1-\alpha)^2 & 2(1-\alpha) \end{pmatrix}$$

Die Matrizen M^{DE} und M^{SDE} sind als Produkt doppelt stochastischer Matrizen selbst wieder doppelt stochastisch. Die Matrix M^{DE} ist in aller Regel unsymmetrisch und die Eigenwerte können daher komplex werden. Um dies zu umgehen werden in dieser Arbeit Varianten des Dimension Exchange betrachtet, die die symmetrisierte Matrix M^{SDE} verwenden.

Wie im Diffusionsfall werden auch hier die Eigenwerte der Iterationsmatrix benötigt.

Definition 3.9. Analog zu Definition 2.11 werden die Eigenwerte der Matrizen M^{DE}, M^{SDE}, L^{DE} und L^{SDE} mit μ_i^{DE}, μ_i^{SDE}, λ_i^{DE} bzw. λ_i^{SDE} bezeichnet.

3.3 Allgemeines Vorgehen zur Konstruktion von Dimension-Exchange-Verfahren aus Diffusionsverfahren

In den folgenden beiden Abschnitten 3.4 und 3.5 werden einige nicht-endliche und endliche Dimension-Exchange-Verfahren konstruiert. An dieser Stelle wird zunächst ein allgemeines „Kochrezept“ gegeben, um aus einem bestehenden Diffusionsverfahren (in Matrix-Schreibweise) ein Dimension-Exchange-Verfahren zu erhalten: Ersetze die Diffusionsmatrix oder die Laplace-Matrix durch eine der neu definierten Matrizen aus Definition 3.7. Falls es sich um ein eigenwertbasiertes Verfahren handelt, ersetze die Eigenwerte entsprechend durch die der neuen Matrix. Kurz: Mache aus jedem $\square^{\mathsf{Diff}}$ ein $\square^{\mathsf{DE}}$ oder $\square^{\mathsf{SDE}}$.

Definition 3.10. Ist ALG ein Diffusionsverfahren, dann werden die aus diesem wie oben beschrieben konstruierten Dimension-Exchange-Verfahren mit DE-ALG bzw. SDE-ALG bezeichnet, je nachdem ob sie die Matrix M^{DE} oder M^{SDE} verwenden.

3.4 Ein erstes Dimension-Exchange-Verfahren: DE-FOS

Der erste Vertreter dieser Verfahren, der hier vorgestellt werden soll, ist die Dimension-Exchange-Version des FOS-Verfahrens. Algorithmus 3.2 zeigt die Version, die die Matrix

M^{DE} verwendet, Algorithmus 3.3 enthält die einzelnen Teilschritte inklusive der Berechnung des Flusses. Dieses Verfahren ist nicht neu, es wurde erstmalig in [HLM+90] für

---

for $k = 1, 2, \ldots$ **do**
  $w^k = M^{\mathsf{DE}} w^{k-1}$
end for

Algorithmus 3.2: DE-FOS-Verfahren, Matrix-Version

$x = 0$
for $k = 1, 2, \ldots$ **do**
  $\hat{w}^0 = w^{k-1}$
  for $j = 1, \ldots, c$ **do** *{Farbschleife}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$
    $x = x + \alpha A_j^T \hat{w}^{j-1}$
  end for
  $w^k = \hat{w}^c$
end for

Algorithmus 3.3: DE-FOS-Verfahren, Version mit der Schleife über die Farben

den Spezialfall $\alpha = \frac{1}{2}$ beschrieben. Xu und Lau haben das Verfahren für beliebiges α unter dem Namen *Generalized Dimension Exchange Method (GDE)* in [XL92, XL95] beschrieben und näher untersucht. Aus Gründen der Systematik wird hier die Bezeichnung DE-FOS verwendet. Das wichtigste Resultat aus [XL95] ist in folgendem Satz zusammengefasst. (Hier wie auch im Rest dieses Abschnittes wird eine konkrete Einfärbung der Graphen vorausgesetzt, die später erklärt wird.)

Satz 3.11. *Der optimale Wert für α beim DE-FOS-Verfahren, der den zweitgrößten Eigenwert $\gamma(M^{\mathsf{DE}})$ minimiert, ist für den Pfad P_k, den Zyklus C_{2k} sowie das Gitter $G_{k,l}$ und den Torus $T_{2k,2l}$ mit $k > l$*

$$\alpha_{opt} = \frac{1}{1 + \sin\left(\frac{\pi}{k}\right)}.$$

Die symmetrische Variante, SDE-FOS erhält man, indem man die Matrix M^{SDE} statt M^{DE} verwendet. Wie in Algorithmus 3.4 ersichtlich ist, müssen die Farben pro Schritt zweimal durchlaufen werden, einmal vorwärts und einmal rückwärts. Zum SDE-FOS-Verfahren gibt es noch keine mit Satz 3.11 vergleichbaren theoretischen Ergebnisse. Experimente haben jedoch folgendes gezeigt:

Beobachtung 3.12. Das SDE-FOS-Verfahren konvergiert am schnellsten für $\alpha = \frac{1}{2}$. In diesem Fall unterscheidet sich das Konvergenzverhalten kaum von DE-FOS. Allerdings besteht ein Schritt des SDE-FOS aus doppelt so vielen Teilschritten wie bei DE-FOS.

Außerdem sind die Resultate bei DE-FOS für $\alpha = \frac{1}{2}$ mitunter noch recht weit vom Optimum entfernt, vergleiche hierzu auch Tabelle 3.1. Beim einfachen First-Order-Scheme ist die symmetrische Variante also nicht konkurrenzfähig.

$x = 0$
for $k = 1, 2, \ldots$ **do**
  $\hat{w}^0 = w^{k-1}$
  for $j = 1, \ldots, c$ **do** *{1. Farbschleife (vorwärts)}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$
    $x = x + \alpha A_j^T \hat{w}^{j-1}$
  end for
  for $j = c, \ldots, 1$ **do** *{2. Farbschleife (rückwärts)}*
    $\hat{w}^{2c-j+1} = M_j \hat{w}^{2c-j}$
    $x = x + \alpha A_j^T \hat{w}^{2c-j}$
  end for
  $w^k = \hat{w}^{2c}$
end for

Algorithmus 3.4: SDE-FOS-Verfahren, Version mit den Schleifen über die Farben

In der Regel ist DE-FOS sogar schneller als SOS und Čebyšev, siehe Tabelle 3.1. Allerdings sollte man hierbei nicht übersehen, dass die Bestimmung des optimalen Wertes von α für Graphen, über die Satz 3.11 keine Aussage trifft (z. B. der T_7), aufwändig sein kann.

Graph	FOS	SOS	Čebyšev	DE-FOS $\alpha = \alpha_{opt}$	DE-FOS $\alpha = \frac{1}{2}$	SDE-FOS $\alpha = \frac{1}{2}$
C_{12}	48	17	14	6	23	23
G_8	173	33	27	9	44	44
T_8	47	17	14	4	10	10
T_7	34	14	12	6	8	9

Tabelle 3.1: Anzahl der Schritte verschiedener Verfahren an vier Beispielgraphen bei jeweils identischer, zufälliger Ausgangslastverteilung

In Abbildung 3.1 ist an Hand von vier Graphen dargestellt, wie die Anzahl der benötigten Schritte bei verschiedenen Verfahren von α abhängt. Da globale Kommunikation während des Ablaufs der Verfahren grundsätzlich vermieden werden soll, wird die Anzahl der Schritte im Voraus (nach einer globalen Summe) mit Hilfe einer Fehlerabschätzung berechnet. Die Ausgangslastverteilung war zufällig, aber für alle Testfälle identisch — am Ende sollte jeder Knoten 100 Lasteinheiten haben. Bei den Diffusionsverfahren darf α nicht größer werden als der Kehrwert des maximalen Knotengrades, da die Diffusionsmatrizen sonst negative Diagonalelemente hätten.

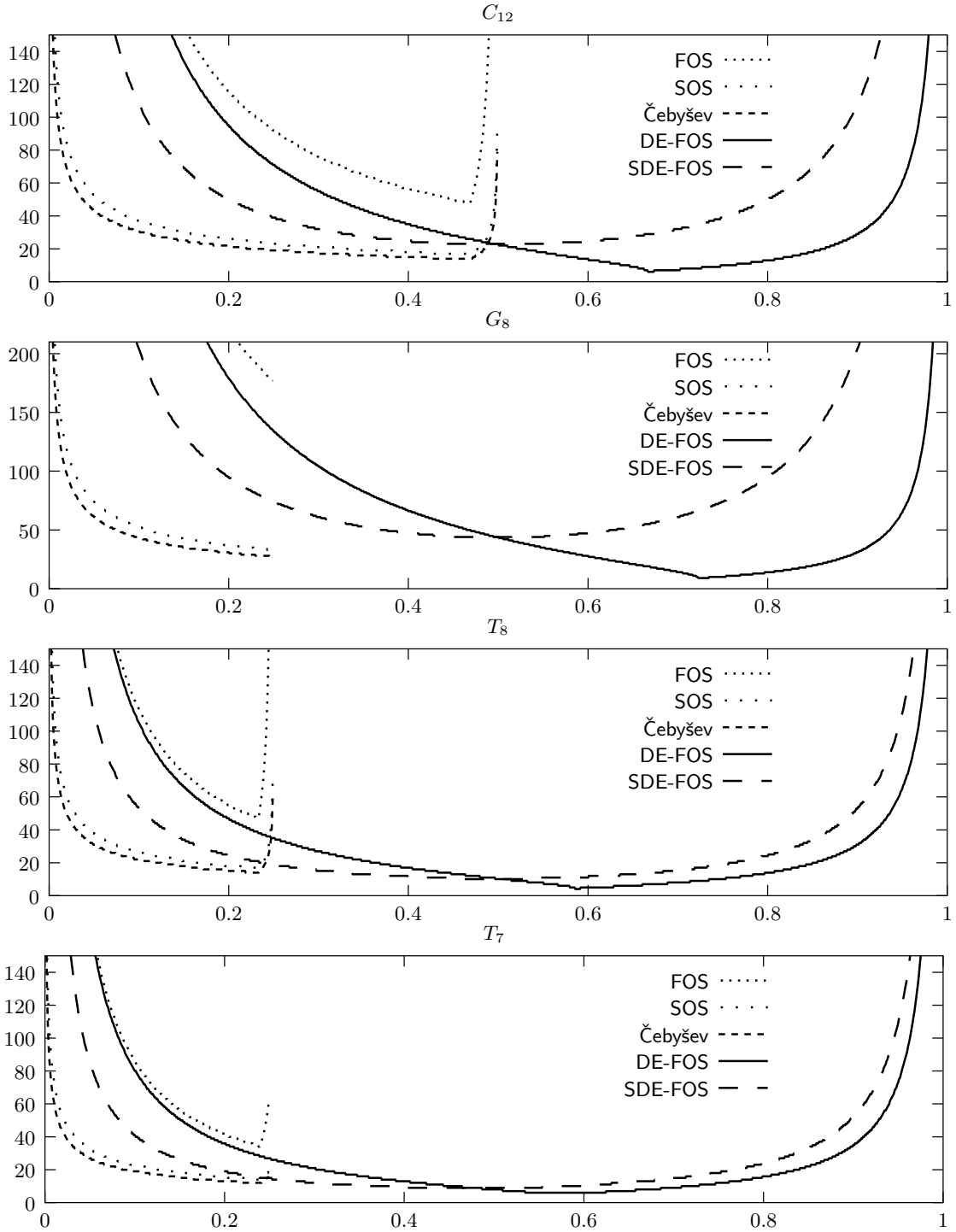


Abbildung 3.1: Anzahl der benötigten Schritte verschiedener Verfahren in Abhängigkeit vom Parameter α anhand von vier Beispielen

3.5 Endliche Dimension-Exchange-Verfahren

Der letzte Abschnitt hat gezeigt, dass die Dimension-Exchange-Technik dem Diffusionsansatz bei den einfachen FOS-Verfahren klar überlegen ist. In diesem Abschnitt werden endliche Dimension-Exchange-Verfahren konstruiert und auf ihre Korrektheit untersucht.

Sofern nichts Anderes erwähnt wird, wird immer im Folgenden immer vorausgesetzt, dass die Matrix M^{DE} diagonalisierbar ist. Der nicht-diagonalisierbare Fall wird in Abschnitt 3.5.1 behandelt.

Zuerst wird das Verfahren DE-OPT betrachtet, wie es global in Algorithmus 3.5 und aus der Sicht von Prozessor i in Algorithmus 3.6 dargestellt ist. Für den Nachweis der Korrektheit werden zunächst einige Aussagen über die Eigenwerte und -vektoren der Matrix M^{DE} benötigt.

---

for $k = 1, \ldots, m-1$ **do**
  $\hat{w}^0 = w^{k-1}$
  for $j = 1, \ldots, c$ **do** *{Schleife über die Farben}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$
    $x = x + \frac{1}{\lambda_{k+1}^{\mathsf{DE}}} A_j^T \hat{w}^{j-1}$
  end for
  $w^k = \left(1 - \frac{1}{\alpha \lambda_{k+1}^{\mathsf{DE}}}\right) w^{k-1} + \frac{1}{\alpha \lambda_{k+1}^{\mathsf{DE}}} \hat{w}^c$
end for

Algorithmus 3.5: DE-OPT-Verfahren

---

for $k = 1, \ldots, m-1$ **do**
  $\hat{w}_i^0 = w_i^{k-1}$
  for $j = 1, \ldots, c$ **do** *{Schleife über die Farben}*
    if existiert $e = \{i, t\} \in E_j$ **then**
      $y_e = \alpha \left(\hat{w}_i^{j-1} - \hat{w}_t^{j-1}\right)$
      $\hat{w}_i^j = \hat{w}_i^{j-1} - y$
      $x_e = x_e + \frac{1}{\alpha \lambda_{k+1}^{\mathsf{DE}}} y_e$
    end if
  end for
  $w_i^k = \left(1 - \frac{1}{\alpha \lambda_{k+1}^{\mathsf{DE}}}\right) w_i^{k-1} + \frac{1}{\alpha \lambda_{k+1}^{\mathsf{DE}}} \hat{w}_i^c$
end for

Algorithmus 3.6: DE-OPT-Verfahren für Prozessor i

Wie früher bereits erwähnt sind die Matrizen $M^{(\mathsf{S})\mathsf{DE}}$ als Produkt der doppelt stochastischen Matrizen M_j selbst ebenfalls doppelt stochastisch. Insbesondere ist also $M^{(\mathsf{S})\mathsf{DE}} e = e$, wobei e der Vektor aus lauter Einsen ist. Nach dem Satz von Perron-Frobenius [HJ85, Cor. 8.1.30] folgt damit, dass der Spektralradius $\varrho\left(M^{(\mathsf{S})\mathsf{DE}}\right) = 1$ ist.

Der Eigenwert $\varrho\left(M^{(\mathsf{S})\mathsf{DE}}\right) = 1$ ist laut [HJ85, Th. 8.4.4] einfach, falls die Matrix $M^{(\mathsf{S})\mathsf{DE}}$ irreduzibel ist.

Für eine beliebige Matrix $A = (a_{ij})$ definiere $\operatorname{struct}(A) = \{(i,j) \mid a_{ij} \neq 0\}$ die Struktur von A. Es ist $M^{\mathsf{DE}} = M_c \cdots M_1$, wobei alle M_i wegen $\alpha \in (0,1)$ nicht-negativ sind und positive Diagonalelemente haben. Damit ist $\operatorname{struct}\left(M^{\mathsf{DE}}\right) \supseteq \operatorname{struct}\left(M_i\right)$ für $i = 1, \ldots, c$ und es folgt

$$\begin{aligned}\operatorname{struct}\left(M^{\mathsf{DE}}\right) &\supseteq \operatorname{struct}\left(M_c + \cdots + M_1\right)\\ &= \operatorname{struct}\left(L_c + \cdots + L_1\right)\\ &= \operatorname{struct}(L)\\ &= \operatorname{struct}(M)\,.\end{aligned}$$

Es ist M^{DE} also irreduzibel, sofern der zugrunde liegende Graph zusammenhängend ist. Wegen $M^{\mathsf{SDE}} = \left(M^{\mathsf{DE}}\right)^T M^{\mathsf{DE}}$ ist diese Matrix ebenfalls irreduzibel.

Das nachfolgende Lemma stellt sicher, dass die Iteration mit der Matrix M^{DE}, obwohl diese unsymmetrisch ist, wie im Diffusionsfall nur auf dem orthogonalen Komplement von $\overline{w}$ arbeitet.

Lemma 3.13. *Es sei $z_1 = e = (1, \ldots, 1)^T$ und z ein weiterer Eigenvektor von M^{DE} zum Eigenwert $\mu \neq 1$. Dann stehen z_1 und z senkrecht aufeinander.*

Beweis. Da M^{DE} doppelt stochastisch ist, ist ${M^{\mathsf{DE}}}^H z_1 = z_1$. Dann gilt:

$$z^H z_1 = z^H \left({M^{\mathsf{DE}}}^H z_1\right) = \left(M^{\mathsf{DE}} z\right)^H z_1 = \overline{\mu} z^H z_1$$

Wegen $\mu \neq 1$ ist also $\langle z_1, z\rangle = 0$. □

Damit lässt sich Lemma 2.14 ($\overline{w} = z_1$) auf die Matrix M^{DE} übertragen und folgender Satz wird exakt so bewiesen wie Satz 2.18.

Satz 3.14. *Das DE-OPT-Verfahren endet nach $m-1$ Iterationsschritten mit $w^{m-1} = \overline{w}$, wobei m die Anzahl paarweise verschiedener Eigenwerte von M^{DE} ist.*

Als nächstes Verfahren wird DE-OPS betrachtet, vergleiche Algorithmus 3.7. Als Innenprodukt wird analog zum Diffusions-OPS

$$\langle p, q\rangle = \sum_{j=2}^{m} \omega_j p(\mu_j^{\mathsf{DE}}) q(\mu_j^{\mathsf{DE}})$$

verwendet mit $\omega_j = 1 - \mu_j^{\mathsf{DE}}$. Die Berechnung der α_k, β_k und γ_k wird ohne Änderung aus den Gleichungen (2.2) bis (2.4) übernommen.

Satz 3.15. *Das DE-OPS-Verfahren endet nach $m-1$ Iterationsschritten mit $w^{m-1} = \overline{w}$, wobei m die Anzahl paarweise verschiedener Eigenwerte von M^{DE} ist; es kann jedoch vorkommen, dass das Verfahren in der Vorbereitungsphase bei der Berechnung eines α_k mit einer Division durch 0 abbricht.*

$\hat{w}^0 = w^0$
$\hat{x}^0 = 0$
for $j = 1, \ldots, c$ **do** *{Schleife über die Farben}*
  $\hat{w}^j = M_j \hat{w}^{j-1}$
  $\hat{x}^j = \hat{x}^j + \alpha A_j^T \hat{w}^{j-1}$
end for
$w^1 = \frac{1}{\gamma_1}\left[\alpha_1 w^0 - \hat{w}^c\right]$
$x^1 = -\frac{1}{\gamma_1}\hat{x}^c$
for $k = 2, \ldots, m-1$ **do**
  $\hat{w}^0 = w^{k-1}$
  $\hat{x}^0 = x^{k-1}$
  for $j = 1, \ldots, c$ **do** *{Schleife über die Farben}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$
    $\hat{x}^j = \hat{x}^j + \alpha A_j^T \hat{w}^{j-1}$
  end for
  $w^k = \frac{1}{\gamma_k}\left[\alpha_k w^{k-1} - \hat{w}^c - \beta_k w^{k-2}\right]$
  $x^k = \frac{1}{\gamma_k}\left[\alpha_k x^{k-1} - \hat{x}^c - \beta_k x^{k-2}\right]$
end for

Algorithmus 3.7: DE-OPS-Verfahren

Beweis. Zu zeigen ist, dass $p_{m-1}(\mu_i^{\mathsf{DE}}) = 0$ ist für $i = 2, \ldots, m$. Denn mit der üblichen Zerlegung der Anfangslast w^0 in Eigenvektoren z_i von M^{DE} erhält man dann

$$w^{m-1} = p_{m-1}(M^{\mathsf{DE}})w^0 = p_{m-1}(M^{\mathsf{DE}})\sum_{i=1}^{m} z_i = \sum_{i=1}^{m} p_{m-1}(\mu_i)z_i = z_1 = \overline{w}\,.$$

Zunächst werden die Polynome mit dem Faktor γ_k skaliert, was auf folgende neue Rekursion führt.

$$\begin{aligned}
&\hat{p}_0(t) = 1 \\
&\hat{p}_1(t) = (\alpha_1 - t)\,\hat{p}_0(t) \\
&\hat{p}_k(t) = (\alpha_k - t)\,\hat{p}_{k-1}(t) - \hat{\beta}_k \hat{p}_{k-2}(t), \qquad\qquad k = 2, \ldots, m-1
\end{aligned}$$

mit

$$\begin{aligned}
\alpha_k &= \frac{\langle t\hat{p}_{k-1}, \hat{p}_{k-1}\rangle}{\langle \hat{p}_{k-1}, \hat{p}_{k-1}\rangle}, && k = 1, \ldots, m-1 \\
\hat{\beta}_k &= \frac{\beta_k}{\gamma_{k-1}} = \frac{\langle \hat{p}_{k-1}, \hat{p}_{k-1}\rangle}{\langle \hat{p}_{k-2}, \hat{p}_{k-2}\rangle}, && k = 2, \ldots, m-1\,.
\end{aligned}$$

Ersetze nun die Polynome $\hat{p}_k$ durch Vektoren $f_k \in \mathbb{R}^{m-1}$, $f_k = \left(p_k(\mu_2^{\mathsf{DE}}), \ldots, p_k(\mu_m^{\mathsf{DE}})\right)^T$. Definiere außerdem $\mathcal{M} = \operatorname{diag}(\mu_2^{\mathsf{DE}}, \ldots, \mu_m^{\mathsf{DE}})$, $\Omega = I - \mathcal{M} = \operatorname{diag}(\omega_2, \ldots, \omega_m)$ und die

Bilinearform $\langle f, g\rangle = \sum_{j=2}^{m} \omega_j f_j g_j = f^T \Omega g$. Damit erhalten wir eine neue Rekursion für die Vektoren f_k:

$$\begin{aligned} f_0 &= (1, \ldots, 1)^T \\ f_1 &= (\alpha_1 I - \mathcal{M})\, f_0 \\ f_k &= (\alpha_k I - \mathcal{M})\, f_{k-1} - \hat{\beta}_k f_{k-2}, \qquad\qquad k = 2, \ldots, m-1 \end{aligned}$$

mit

$$\begin{aligned} \alpha_k &= \frac{\langle \mathcal{M} f_{k-1}, f_{k-1}\rangle}{\langle f_{k-1}, f_{k-1}\rangle} = \frac{f_{k-1}^T \mathcal{M}\Omega f_{k-1}}{f_{k-1}^T \Omega f_{k-1}}, \qquad k = 1, \ldots, m-1 \\ \hat{\beta}_k &= \frac{\langle f_{k-1}, f_{k-1}\rangle}{\langle f_{k-2}, f_{k-2}\rangle} = \frac{f_{k-1}^T \Omega f_{k-1}}{f_{k-2}^T \Omega f_{k-2}}, \qquad k = 2, \ldots, m-1 \end{aligned}$$

Da M^{DE} eine reelle Matrix ist, sind die Eigenwerte μ_i^{DE} und die ω_i entweder reell oder sie treten in komplex konjugierten Paaren auf. Folglich sind die Produkte $\langle f, f\rangle$ und $\langle \mathcal{M} f, f\rangle$ reell – allerdings können sie negativ oder null werden. Im Falle $\langle f_{k-1}, f_{k-1}\rangle = 0$ bricht der Algorithmus ab. Im folgenden wird davon ausgegangen, dass kein solcher Break-down auftritt. Dann kann man leicht nachweisen, dass $\langle f_k, f_l\rangle = 0$ ist für $k \neq l$. Außerdem ist keiner der Vektoren $f_0, \ldots, f_{m-2}$ der Nullvektor, denn sonst gäbe es ein Polynom vom Grad $k \leq m-2$ mit $m-1$ Nullstellen. Also muss $f_{m-1} = 0$ sein, was den Beweis abschließt. □

Bemerkung 3.16. Bei der Rekursion für die Vektoren f_k im letzten Beweis handelt es sich um nichts anderes als das komplex symmetrische Lanczos-Verfahren [CW85]. Es berechnet $\mathcal{M}F = FJ$ mit $F = (f_0 \cdots f_{m-1})$ und $J = \mathrm{tridiag}(-1, \alpha_i, -\hat{\beta}_{i+1})$.

Auf den Abdruck der Verfahren SDE-OPT sowie SDE-OPS wird hier verzichtet, die algorithmischen Unterschiede zu den unsymmetrischen Varianten ähneln denen zwischen DE-FOS (Alg. 3.3) und SDE-FOS (Alg. 3.4). Da die Matrix M^{SDE} symmetrisch ist, ist sie im Gegensatz zu M^{DE} auf jeden Fall diagonalisierbar und es treten keinerlei komplexe Zahlen auf. Die Beweise für die Korrektheit von SDE-OPT und SDE-OPS stimmen exakt mit denen für OPT und OPS überein, da auch hier in beiden Fällen eine symmetrische Matrix verwendet wird.

Wie bei den Diffusionsverfahren auch wird zukünftig die Abkürzung DE-OPX für ein beliebiges der beiden Verfahren DE-OPS bzw. DE-OPT benutzt, entsprechendes gilt für SDE-OPX.

3.5.1 DE-OPT im nicht-diagonalisierbaren Fall

Bisher ist nicht bekannt, ob überhaupt Graphen und Kantengewichte existieren, so dass M^{DE} nicht diagonalisierbar ist. Dennoch soll in diesem Abschnitt gezeigt werden, dass das DE-OPT-Verfahren so modifiziert werden kann, dass es auch dann noch eine Gleichverteilung berechnet. Allerdings muss dann statt der Eigenwerte von M^{DE} deren Jordan-Form

bekannt sein. Da diese numerisch jedoch kaum zu berechnen ist, ist dieser Abschnitt eher von theoretischem Wert.

Sei nun $X \in \mathbb{R}^{n \times n}$ die Matrix, die M^{DE} auf Jordan-Gestalt transformiert, d. h.

$$X^{-1} M^{\mathsf{DE}} X = \begin{pmatrix} J_{r_1}(\mu_1) & & \\ & \ddots & \\ & & J_{r_l}(\mu_l) \end{pmatrix} \text{ mit } J_{r_k}(\mu_k) = \begin{pmatrix} \mu_k & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \mu_k \end{pmatrix} \in \mathbb{C}^{r_k \times r_k}$$

mit $\mu_1 = 1$ und $r_1 = 1$. Analog zu den Eigenvektoren im diagonalisierbaren Fall existieren nun verallgemeinerte Eigenvektoren $z_{i,j}$ mit $\left(M^{\mathsf{DE}} - \mu_i^{\mathsf{DE}} I\right)^j z_{i,j} = 0$ für $i = 1, \ldots, l$ und $j = 1, \ldots, r_i$. Diese stehen senkrecht auf der Gleichverteilung, wie folgende Verallgemeinerung von Lemma 3.13 zeigt:

Lemma 3.17. *Es sei $z_{1,1} = e = (1, \ldots, 1)^T$ und z ein weiterer verallgemeinerter Eigenvektor von M^{DE} zum Eigenwert $\mu \neq 1$. Dann stehen z_1 und z senkrecht aufeinander.*

Beweis. Da M^{DE} doppelt stochastisch ist, gilt wieder ${M^{\mathsf{DE}}}^H z_{1,1} = z_{1,1}$.

$$\begin{aligned} z^H z_{1,1} &= z^H \left[\frac{1}{(1-\overline{\mu})^j} \left({M^{\mathsf{DE}}}^H - \overline{\mu} I \right)^j z_{1,1} \right] \\ &= \left[\frac{1}{(1-\mu)^j} \left(M^{\mathsf{DE}} - \mu I \right)^j z \right]^H z_{1,1} \\ &= 0 \end{aligned}$$

□

Es sei nun wie üblich m die Anzahl verschiedener Eigenwerte. Zu jedem Eigenwert μ_k^{DE}, $k = 1, \ldots, m$ bezeichne s_k die Dimension des größten Jordanblock zu μ_k^{DE}. Der DE-OPT-Algorithmus muss nun so verändert werden, dass jeder Eigenwert μ_k^{DE} bzw. λ_k^{DE} mehrfach entsprechend seiner maximalen Vielfachheit s_k in einem Jordanblock angewandt wird, vergleiche Algorithmus 3.8.

---

for $k = 1, \ldots, m-1$ **do**
  $w^{k,0} = w^{k-1}$
  for $j = 1, \ldots, s_k$ **do**
    $w^{k,j} = \left(I - \frac{1}{\lambda_{k+1}^{\mathsf{DE}}} L^{\mathsf{DE}} \right) w^{k,j-1}$
  end for
  $w^k = w^{k,s_k}$
end for

Algorithmus 3.8: DE-OPT-Verfahren für den nicht-diagonalisierbaren Fall

Zur Korrektheit des Algorithmus ist zunächst anzumerken, dass ein verallgemeinerter Eigenvektor z von M^{DE} zum Eigenwert μ auch ein verallgemeinerter Eigenvektor von L^{DE} zum Eigenwert λ mit $\mu = 1 - \alpha\lambda$ ist, denn

$$\begin{aligned}\left(L^{\mathsf{DE}} - \lambda I\right)^j z &= \left[\frac{1}{\alpha}\left(I - M^{\mathsf{DE}}\right) - \frac{1}{\alpha}(1-\mu)\, I\right]^j z \\ &= \left(-\frac{1}{\alpha}\right)^j \left(M^{\mathsf{DE}} - \mu I\right)^j z \\ &= 0 \qquad\qquad \text{falls } \left(M^{\mathsf{DE}} - \mu I\right)^j z = 0\,.\end{aligned}$$

Schreibt man nun die Anfangsverteilung in der Form $w^0 = \sum_{i=1}^{m}\sum_{j=1}^{s_i} z_{i,j}$ mit $z_{1,1} = \overline{w}$ dann berechnet Algorithmus 3.8

$$\begin{aligned}w^{m-1} &= \prod_{k=m-1}^{1}\left(I - \frac{1}{\lambda_{k+1}^{\mathsf{DE}}} L^{\mathsf{DE}}\right)^{s_k} w^0 \\ &= \prod_{k=m-1}^{1}\left(I - \frac{1}{\lambda_{k+1}^{\mathsf{DE}}} L^{\mathsf{DE}}\right)^{s_k} \sum_{i=1}^{m}\sum_{j=1}^{s_i} z_{i,j} \\ &= \sum_{i=1}^{m}\sum_{j=1}^{s_i} \prod_{k=m-1}^{1}\left(I - \frac{1}{\lambda_{k+1}^{\mathsf{DE}}} L^{\mathsf{DE}}\right)^{s_k} z_{i,j} \\ &= z_{1,1}\,,\end{aligned}$$

also die Gleichverteilung.

3.6 Eigenwerte gefärbter Graphen

Der Aufwand für die im vorigen Abschnitt beschriebenen Verfahren hängt ab von der Anzahl der verschiedenen Eigenwerte. Diese Anzahl wiederum ist abhängig von der konkret gewählten Einfärbung. Bei den Diffusionsvarianten war von Vorteil, dass die Eigenwerte vieler Standardgraphen bekannt sind und nicht aufwändig berechnet werden müssen. Ziel dieses Abschnittes ist es, nicht nur die Anzahl der Eigenwerte gefärbter Graphen zu ermitteln sondern nach Möglichkeit auch die Eigenwerte selbst zu bestimmen.

3.6.1 Mathematische Grundlagen

Zur Untersuchung der Eigenwerte benötigt man *Kroneckerprodukte* und *zirkulante Matrizen*. Deren Verwendung im Zusammenhang mit Matrizen gefärbter Graphen wurde zuerst vorgeschlagen in [XL95].

Definition 3.18. Es seien $A_1, A_2, \ldots, A_m$ quadratische Matrizen der Größe $r \times r$. Eine *blockzirkulante Matrix* hat dann die Form

$$\Phi(A_1, \ldots, A_m) = \begin{pmatrix} A_1 & A_2 & \cdots & A_m \\ A_m & A_1 & \ldots & A_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_2 & A_3 & \cdots & A_1 \end{pmatrix}.$$

Die Eigenwerte einer blockzirkulanten Matrix lassen sich, falls die Blockgröße r klein genug ist, relativ leicht berechnen.

Lemma 3.19 ([XL95, Lemma 3.1]). *Es sei $A = \Phi(A_1, \ldots, A_m)$ eine blockzirkulante Matrix. Die Eigenwerte von A sind gleich der Vereinigung aller Eigenwerte der folgenden Matrizen $A^{(0)}, \ldots, A^{(m-1)}$. Hierbei ist für $j = 0, \ldots, m-1$:*

$$A^{(j)} = \sum_{k=1}^{m} \omega^{(k-1)j} A_k \quad \textit{mit} \quad \omega = e^{i\frac{2\pi}{m}} = \cos\frac{2\pi}{m} + i \sin\frac{2\pi}{m}$$

Außerdem wird noch eine Aussage über die Eigenwerte von Kroneckerprodukten (siehe Definition 1.11) benötigt.

Lemma 3.20 ([HJ91, Theorem 4.2.12]). *Es sei $A \in \mathbb{C}^{p\times p}$ und $B \in \mathbb{C}^{r\times r}$. Sind λ und μ Eigenwerte von A bzw. B mit zugehörigen Eigenvektoren x und y, dann ist $\lambda\mu$ Eigenwert von $A \otimes B$ mit Eigenvektor $x \otimes y$. Jeder Eigenwert von $A \otimes B$ lässt sich als ein solches Produkt schreiben.*

3.6.2 Die Bedeutung von $\alpha = \frac{1}{2}$

Die Anzahl der Eigenwerte und damit die Laufzeit der Verfahren hängt entscheidend davon ab, wie der Wert α gewählt wird. Für $\alpha = \frac{1}{2}$ ist die Anzahl der Eigenwerte Beobachtungen zufolge im Allgemeinen geringer als für alle anderen Werte. Während dies in den nächsten Abschnitten für bestimmte Graphen näher betrachtet wird, soll hier zunächst eine allgemeine Erklärung gegeben werden.

Die zu den Teilgraphen $G_i = (V, E_i)$ gehörenden Matrizen M_i haben die Gestalt

$$M_i = P \begin{pmatrix} 1-\alpha & \alpha & & & & & & \\ \alpha & 1-\alpha & & & & & & \\ & & \ddots & & & & & \\ & & & 1-\alpha & \alpha & & & \\ & & & \alpha & 1-\alpha & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} P^T,$$

mit $N_i = |E_i|$ Blöcken $\left(\begin{smallmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{smallmatrix}\right)$, wobei P irgendeine Permutation ist. Setzt man $\alpha = \frac{1}{2}$, wird aus diesen Blöcken $\left(\begin{smallmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{smallmatrix}\right)$, so dass M_i damit N_i-mal den Eigenwert 0 hat. Insgesamt haben M^{DE} und M^{SDE} also den Eigenwert 0 mindestens mit Vielfachheit $N^{max} := \max_{i=1,\ldots,c} N_i$. Die Gesamtzahl der Eigenwerte ist demzufolge $m \leq n - N^{max} + 1$.

3.6.3 Zyklus C_n mit geradem n

Der am leichtesten zu untersuchende Fall ist der Zyklus C_n mit geradem n. Hier genügen zwei Farben. Die Kante $\{1, 2\}$ soll die Farbe 1 erhalten, vgl. Abb. 3.2.

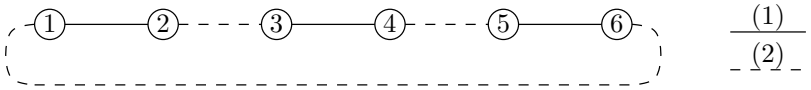


Abbildung 3.2: Einfärbung des Zyklus C_{2k}

Die zu den Teilgraphen G_1 und G_2 gehörenden Diffusionsmatrizen haben die Gestalt

$$M_1 = \begin{pmatrix} 1-\alpha & \alpha & & & & & & & & \\ \alpha & 1-\alpha & & & & & & & & \\ & & 1-\alpha & \alpha & & & & & & \\ & & \alpha & 1-\alpha & & & & & & \\ & & & & \ddots & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1-\alpha & \alpha & & \\ & & & & & & \alpha & 1-\alpha & & \\ & & & & & & & & 1-\alpha & \alpha \\ & & & & & & & & \alpha & 1-\alpha \end{pmatrix},$$

$$M_2 = \begin{pmatrix} 1-\alpha & & & & & & & & & \alpha \\ & 1-\alpha & \alpha & & & & & & & \\ & \alpha & 1-\alpha & & & & & & & \\ & & & \ddots & & & & & & \\ & & & & \ddots & & & & & \\ & & & & & 1-\alpha & \alpha & & & \\ & & & & & \alpha & 1-\alpha & & & \\ & & & & & & & 1-\alpha & \alpha & \\ & & & & & & & \alpha & 1-\alpha & \\ \alpha & & & & & & & & & 1-\alpha \end{pmatrix}.$$

Dann hat die Produktmatrix $M^{\mathsf{DE}} = M_2 \cdot M_1$ die Form

$$M^{\mathsf{DE}} = \begin{pmatrix} A & B & & & & & & & C & B \\ B & A & B & C & & & & & & \\ C & B & A & B & & & & & & \\ & & B & A & & & & & & \\ & & C & B & \ddots & & & & & \\ & & & & & \ddots & B & C & & \\ & & & & & & A & B & & \\ & & & & & & B & A & B & C \\ & & & & & & C & B & A & B \\ B & C & & & & & & & B & A \end{pmatrix}$$

mit $A = (1-\alpha)^2$, $B = \alpha\,(1-\alpha)$, $C = \alpha^2$. Man sieht, dass M^{DE} eine blockzirkulante Matrix ist. Es gilt nämlich

$$M^{\mathsf{DE}} = \Phi\left(A_1, \ldots, A_{\frac{n}{2}}\right)$$

mit

$$A_1 = \begin{pmatrix} A & B \\ B & A \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 \\ B & C \end{pmatrix}, \quad A_3 = \ldots = A_{\frac{n}{2}-1} = 0, \quad A_{\frac{n}{2}} = \begin{pmatrix} C & B \\ 0 & 0 \end{pmatrix}.$$

Für die Matrizen $A^{(j)}$, $j = 0, \ldots, \frac{n}{2} - 1$ aus Lemma 3.19 erhält man damit

$$A^{(j)} = A_1 + \omega^j A_2 + \omega^{-j} A_{\frac{n}{2}} = \begin{pmatrix} A + C\left(c^{(j)} - is^{(j)}\right) & B\left(1 + c^{(j)} - is^{(j)}\right) \\ B\left(1 + c^{(j)} + is^{(j)}\right) & A + C\left(c^{(j)} + is^{(j)}\right) \end{pmatrix}$$

mit $c^{(j)} = \cos\frac{4\pi j}{n}$ und $s^{(j)} = \sin\frac{4\pi j}{n}$. Das charakteristische Polynom dieser Matrix lautet

$$\chi^{(j)}(t) = t^2 - \left(2(1-\alpha)^2 + 2\alpha^2 c^{(j)}\right) t + (1-2\alpha)^2 .$$

Dessen Nullstellen und damit die Eigenwerte von M^{DE} sind:

$$t_{1,2}^{(j)} = (1-\alpha)^2 + \alpha^2 c^{(j)} \pm \alpha\sqrt{\alpha^2 - 4\alpha + 2 + 2(\alpha-1)^2 c^{(j)} + \alpha^2\left(c^{(j)}\right)^2}, \quad j = 0, \ldots, \frac{n}{2} - 1 \tag{3.1}$$

Die Anzahl verschiedener Nullstellen m hängt also ab von den $c^{(j)}$ und von α. Es wird im Folgenden nicht berücksichtigt, dass für bestimmte Werte von α zufällig Eigenwerte zusammenfallen können. Zunächst sei $\alpha \neq \frac{1}{2}$.

1. Fall: n ist durch 4 teilbar: Dann gilt

$$c^{(j)} = \cos\frac{4\pi j}{n} = \cos\frac{4\pi\left(\frac{n}{2} - j\right)}{n} = c^{(\frac{n}{2}-j)}$$

für $j = 1, \ldots, \frac{n}{4} - 1$. Zu $c^{(\frac{n}{4})} = -1$ gibt es einen doppelten Eigenwert $t_{1,2}^{(\frac{n}{4})} = 1 - 2\alpha$. Da es für jedes $c^{(j)}$, $j = 1, \ldots, \frac{n}{4} - 1$ zwei Nullstellen in (3.1) gibt, erhält man insgesamt $m = n - 2\left(\frac{n}{4} - 1\right) - 1 = \frac{n}{2} + 1$.

2. Fall: n nicht durch 4 teilbar: In diesem Fall ist $c^{(j)} = c^{(\frac{n}{2}-j)}$ für $j = 1, \ldots, \frac{n-2}{4}$ und damit $m = n - 2\left(\frac{n-2}{4}\right) = \frac{n}{2} + 1$.

Im Falle $\alpha = \frac{1}{2}$ wird aus Gleichung (3.1):

$$t_1^{(j)} = \frac{1}{2} + \frac{1}{2}c^{(j)}, \qquad t_2^{(j)} = 0$$

Hier fällt auf, dass die Diffusionsmatrix M^{Diff} für einen Zyklus der Länge $\frac{n}{2}$ mit $\alpha = \frac{1}{4}$ genau dieselben Eigenwerte hat, ggf. ohne die 0. Man kann sogar nachrechnen, dass $M^{\mathsf{Diff}} = \left(\mathcal{L}_1^T M_2 \mathcal{L}_1\right) \otimes \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ gilt, wobei $M_1 = \mathcal{L}_1\mathcal{L}_1^T$ die Cholesky-Zerlegung von M_1 ist. Die Anzahl m der Eigenwerte halbiert sich für $\alpha = \frac{1}{2}$ also ungefähr gegenüber dem Fall $\alpha \neq \frac{1}{2}$. Für $4 \mid n$ ist zu berücksichtigen, dass 0 für $\alpha \neq \frac{1}{2}$ auch schon ein Eigenwert war, für $4 \nmid n$ kommt 0 als zusätzlicher Eigenwert hinzu. Damit ist $m = \frac{1}{2} \cdot \left(\frac{n}{2} + 1 - 1\right) + 1 = \frac{n}{4} + 1$ für durch 4 teilbares n und $m = \frac{1}{2}\left(\frac{n}{2} + 1\right) + 1 = \frac{n}{4} + \frac{3}{2}$ im andern Fall.

Nachdem die Eigenwerte der Matrix M^{DE} nun bekannt sind, soll als nächstes die Matrix $M^{\mathsf{SDE}} = \left(M^{\mathsf{DE}}\right)^T M^{\mathsf{DE}}$ näher untersucht werden. Als Produkt blockzirkulanter Matrizen ist M^{SDE} selbst wieder blockzirkulant, nämlich:

$$M^{\mathsf{SDE}} = \Phi\left(A_1, \ldots, A_{\frac{n}{2}}\right)$$

mit

$$A_1 = \begin{pmatrix} \mathfrak{A} & \mathfrak{B} \\ \mathfrak{B} & \mathfrak{A} \end{pmatrix}, \quad A_2 = \begin{pmatrix} \mathfrak{D} & \mathfrak{E} \\ \mathfrak{C} & \mathfrak{D} \end{pmatrix}, \quad A_3 = \ldots = A_{\frac{n}{2}-1} = 0\,, \quad A_{\frac{n}{2}} = \begin{pmatrix} \mathfrak{D} & \mathfrak{C} \\ \mathfrak{E} & \mathfrak{D} \end{pmatrix}.$$

Hierbei sind wie oben $A = (1-\alpha)^2$, $B = \alpha(1-\alpha)$, $C = \alpha^2$ und

$$\begin{aligned}
\mathfrak{A} &= A^2 + 2B^2 + C^2 &&= 4\left(\alpha^2 - \alpha + \frac{1}{2}\right)^2 \\
\mathfrak{B} &= 2AB + 2BC &&= 2\alpha(1-\alpha)^3 + 2\alpha^3(1-\alpha) \\
\mathfrak{C} &= 2AB &&= 2\alpha(1-\alpha)^3 \\
\mathfrak{D} &= AC + B^2 &&= 2\alpha^2(1-\alpha)^2 \\
\mathfrak{E} &= 2BC &&= 2\alpha^3(1-\alpha)\,.
\end{aligned}$$

Ähnlich wie für M^{DE} oben muss man auch hier Matrizen $A^{(j)}$ berechnen. Deren charakteristisches Polynom ist

$$\chi^{(j)}(t) = t^2 - 2\left(\mathfrak{A} + 2\mathfrak{D}c^{(j)}\right)t + \mathfrak{F}$$

mit $\mathfrak{F} = \mathfrak{A}^2 - \mathfrak{B}^2 - (\mathfrak{C} - \mathfrak{E})^2 = 16\left(\alpha - \frac{1}{2}\right)^4$. Damit erhält man die Eigenwerte

$$t_{1,2}^{(j)} = \mathfrak{A} + 2\mathfrak{D}c^{(j)} \pm \sqrt{\mathfrak{I} + \mathfrak{J}c^{(j)} + \mathfrak{K}\left(c^{(j)}\right)^2}$$

mit

$$\begin{aligned}
\mathfrak{I} &= \mathfrak{A}^2 - \mathfrak{F} &&= 16\alpha^2(1-\alpha)^2\left(\alpha^4 - 2\alpha^3 + 3\alpha^2 - 2\alpha + \frac{1}{2}\right) \\
\mathfrak{J} &= 4\mathfrak{A}\mathfrak{D} &&= -32\alpha^2(1-\alpha)^2\left(\alpha^2 - \alpha + \frac{1}{2}\right) \\
\mathfrak{K} &= 4\mathfrak{D}^2 &&= 4\alpha^4(1-\alpha)^4\,.
\end{aligned}$$

Hieraus ergibt sich, dass M^{SDE} für alle Werte von α dieselbe Anzahl verschiedener Eigenwerte hat wie M^{DE}.

Ist speziell $\alpha = \frac{1}{2}$, so vereinfachen sich diese Formeln erheblich. Statt mit obigen Formeln kann man die Eigenwerte in diesem Fall jedoch auch anders herleiten. Es ist nämlich $M^{\mathsf{SDE}} = \frac{1}{2}M \otimes \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, wobei M wieder die Diffusionsmatrix des Zyklus $C_{\frac{n}{2}}$ zu $\alpha = \frac{1}{4}$ ist. M^{DE} und M^{SDE} haben also für $\alpha = \frac{1}{2}$ dieselben Eigenwerte.

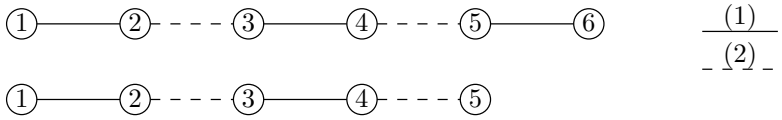


Abbildung 3.3: Einfärbung des Pfades P_n für gerades und ungerades n

3.6.4 Pfad P_n

Als nächster Graph soll der Pfad P_n untersucht werden. Dieser kann wie in Abbildung 3.3 gezeigt immer mit zwei Farben eingefärbt werden.

Im allgemeinen Fall $\alpha \neq \frac{1}{2}$ konnten bislang keine Formeln für die Eigenwerte gefunden werden. Experimente zeigen, dass es immer n paarweise verschiedene Eigenwerte gibt.

Wählt man $\alpha = \frac{1}{2}$, so lassen sich auch für den Pfad Formeln herleiten. Begonnen wird mit dem Fall, dass n gerade ist.

n gerade

Beim Pfad geradzahliger Länge ist für $\alpha = \frac{1}{2}$

$$M_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & & & & & \\ \frac{1}{2} & \frac{1}{2} & & & & & \\ & & \frac{1}{2} & \frac{1}{2} & & & \\ & & \frac{1}{2} & \frac{1}{2} & & & \\ & & & & \ddots & & \\ & & & & & \frac{1}{2} & \frac{1}{2} \\ & & & & & \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & & & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & & \\ & & & \ddots & & & \\ & & & & \frac{1}{2} & \frac{1}{2} & \\ & & & & \frac{1}{2} & \frac{1}{2} & \\ & & & & & & 1 \end{pmatrix}.$$

Bestimme nun die Cholesky-Zerlegung $M_1 = \mathcal{L}_1 \mathcal{L}_1^T$:

$$\mathcal{L}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & & & \\ 1 & 0 & & & \\ & & \ddots & & \\ & & & 1 & 0 \\ & & & 1 & 0 \end{pmatrix},$$

Dann ist M^{DE} ähnlich zu $\mathcal{L}_1^T M_2 \mathcal{L}_1 = M \otimes \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, wobei

$$M = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & & & \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & \\ & \ddots & \ddots & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ & & & \frac{1}{4} & \frac{3}{4} \end{pmatrix}$$

die Diffusionsmatrix des Pfades der Länge $\frac{n}{2}$ zu $\alpha = \frac{1}{4}$ ist. Die gesuchten Eigenwerte sind daher 0 sowie (vgl. Tabelle 2.1)

$$1 - \frac{1}{4}\left(2 - 2\cos\left(\frac{\pi}{\frac{n}{2}} j\right)\right) = \frac{1}{2} + \frac{1}{2}\cos\left(\frac{2\pi}{n} j\right), \quad j = 0, \ldots, \frac{n}{2} - 1.$$

Ähnlich wie beim Zyklus oben gilt für M^{SDE}

$$M^{\mathsf{SDE}} = M_1 M_2 M_2 M_1 = \frac{1}{2} M \otimes \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

so dass M^{SDE} und M^{DE} wieder dieselben Eigenwerte besitzen.

n ungerade

Das Vorgehen für ungerades n entspricht exakt dem für gerades n mit der Einschränkung, dass man hier keine bekannte Matrix erhält. Für die Matrix $\mathcal{L}_1$ aus der Cholesky-Zerlegung erhält man

$$\mathcal{L}_1 = \begin{pmatrix} l & 0 & & & & \\ l & 0 & & & & \\ & & \ddots & & & \\ & & & l & 0 & \\ & & & l & 0 & \\ & & & & & 1 \end{pmatrix}$$

mit $l = \frac{\sqrt{2}}{2}$, und hiermit

$$\mathcal{L}_1^T M_2 \mathcal{L}_1 = \begin{pmatrix} \frac{3}{4} & & \frac{1}{4} & & & & & & \\ & 0 & & & & & & & \\ \frac{1}{4} & & \frac{1}{2} & & \frac{1}{4} & & & & \\ & & & 0 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 0 & & & \\ & & & & \frac{1}{4} & & \frac{1}{2} & & \frac{1}{4} \\ & & & & & & & 0 & \\ & & & & & & \frac{1}{4} & & \frac{1}{2} & & \frac{\sqrt{2}}{4} \\ & & & & & & & 0 & \\ & & & & & & & & \frac{\sqrt{2}}{4} & & \frac{1}{2} \end{pmatrix}.$$

Behauptung 3.21. *Obige Matrix hat die Eigenwerte*

$$\frac{1}{2} + \frac{1}{2} \cos\left(\frac{2\pi}{n} j\right), \qquad j = 0, \ldots, \frac{n-1}{2}$$

und 0.

Beweis. Offensichtlich ist 0 ein $\frac{n-1}{2}$-facher Eigenwert. Durch Streichen der Nullzeilen und -spalten entsteht folgende Matrix der Dimension $p = \frac{n+1}{2}$:

$$\begin{pmatrix} \frac{3}{4} & \frac{1}{4} & & & & \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & \\ & \ddots & & & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \\ & & & \frac{1}{4} & \frac{1}{2} & \frac{\sqrt{2}}{4} \\ & & & & \frac{\sqrt{2}}{4} & \frac{1}{2} \end{pmatrix},$$

wobei alle Eigenwerte bis auf 0 erhalten bleiben. Wie man leicht nachrechnet, gehört zum Eigenwert $\frac{1}{2} + \frac{1}{2}\cos\left(\frac{2\pi}{n}j\right)$ der Eigenvektor $v^j = \left(v_1^j, \ldots, v_p^j\right)^T$ mit

$$v_l^j = (-1)^{p-l} \cdot 2 \cdot \cos\left(\frac{(p-l)\,(2p-2j-1)\,\pi}{n}\right), \qquad l = 1, \ldots, p-1$$
$$v_p^j = \sqrt{2}\,.$$

□

Nun fehlt noch die Bestimmung der Eigenwerte von M^{SDE}. Hier ergibt sich:

$$M^{\mathsf{SDE}} = \frac{1}{8}\begin{pmatrix} 3 & 3 & 1 & 1 & & & & & \\ 3 & 3 & 1 & 1 & & & & & \\ 1 & 1 & 2 & 2 & 1 & 1 & & & \\ 1 & 1 & 2 & 2 & 1 & 1 & & & \\ & & & & \ddots & & & & \\ & & & 1 & 1 & 2 & 2 & 1 & 1 \\ & & & 1 & 1 & 2 & 2 & 1 & 1 \\ & & & & & 1 & 1 & 2 & 2 & 2 \\ & & & & & 1 & 1 & 2 & 2 & 2 \\ & & & & & & & 2 & 2 & 4 \end{pmatrix}$$

Es sei $\tilde{M}$ die Matrix, die aus M^{SDE} durch Anhängen einer Nullzeile und -spalte entsteht. Hierdurch vergrößert sich lediglich die Vielfachheit des schon vorhandenen Eigenwertes 0 um 1. Dann gilt:

$$\tilde{M} = \frac{1}{2}Q^T\left[\begin{pmatrix} \frac{3}{4} & \frac{1}{4} & & & & & \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & \\ & \ddots & & & & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & \\ & & & \frac{1}{4} & \frac{1}{2} & \frac{\sqrt{2}}{4} & \\ & & & & \frac{\sqrt{2}}{4} & \frac{1}{2} & \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\right] Q$$

mit

$$Q = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ & & & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Es folgt wieder das Ergebnis, dass die Eigenwerte von M^{SDE} und M^{DE} übereinstimmen.

3.6.5 Zyklus C_n mit ungeradem n

Bei den bisher betrachteten Pfaden und Zyklen genügten immer zwei verschiedene Farben. Zur Einfärbung von Zyklen ungerader Länge werden mindestens drei Farben benötigt. Werden keine weiteren Voraussetzungen an n gemacht, bietet sich die erste Einfärbung aus Abbildung 3.4 an. Die Kanten $\{1,2\},\ldots,\{n-1,n\}$ werden abwechselnd mit den Farben 1 und 2 belegt und Kante $\{1,n\}$ bekommt Farbe 3. Leider ist es für diese Einfärbung bisher nicht gelungen, Aussagen über die Eigenwerte zu beweisen.

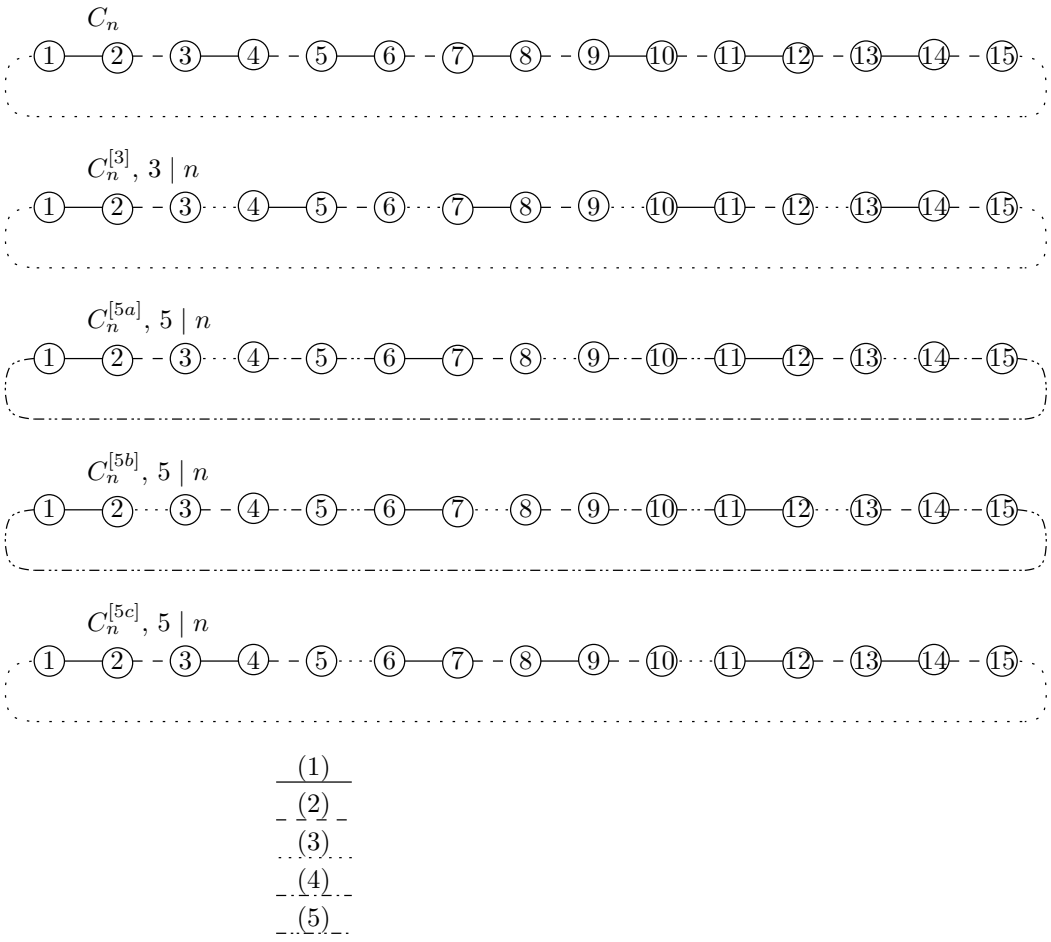


Abbildung 3.4: Einfärbung von Zyklen C_n mit ungeradem n

Erfreulicher gestaltet sich der Spezialfall, dass n durch 3 teilbar ist. Dann kann man nämlich den Kanten zyklisch drei Farben zuweisen, wie dies in der Abbildung unter $C_n^{[3]}$ dargestellt ist. Die zu den drei Farben gehörenden Matrizen M_1, M_2 und M_3 sind blockzirkulant mit Blockgröße 3. Entsprechend sind auch die Produkte hiervon, M^{DE} und M^{SDE}, blockzirkulant. Mit den im Abschnitt 3.6.3 vorgeführten Methoden könnte man hier also wieder alle Eigenwerte berechnen. Im Falle der unsymmetrischen Matrix

M^{DE} ergeben sich diesmal allerdings komplexe Eigenwerte; da die Anzahl der Eigenwerte für den wichtigen Fall $\alpha = \frac{1}{2}$ gegenüber der ersten Einfärbung aus Abbildung 3.4 Experimenten zufolge sogar zunimmt, wird auf die Berechnung hier verzichtet.

Auch im symmetrischen Fall soll hier nur auf den Fall $\alpha = \frac{1}{2}$ etwas näher eingegangen werden. Es wird die Matrix $Q^T M^{\mathsf{SDE}} Q$ mit $Q = I_{\frac{n}{3}} \otimes \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$ betrachtet. Nach Streichen der 1., 4., 7. usw. Spalte und Zeile (diese enthalten nur Nullen) hat diese Matrix die Gestalt

$$\frac{1}{32} \begin{pmatrix} 14 & 6\sqrt{2} & 4 & & & & 4 & 4\sqrt{2} \\ 6\sqrt{2} & 12 & 4\sqrt{2} & & & & & \\ 4 & 4\sqrt{2} & 14 & 6\sqrt{2} & 4 & & & \\ & & & \ddots & & & & \\ & & & & \ddots & & & \\ & & & & 6\sqrt{2} & 12 & 4\sqrt{2} & \\ 4 & & & & 4 & 4\sqrt{2} & 14 & 6\sqrt{2} \\ 4\sqrt{2} & & & & & & 6\sqrt{2} & 12 \end{pmatrix}.$$

Die gesuchten Eigenwerte sind 0 sowie für $j = 1, \ldots, \frac{n}{3}$

$$t_{1,2}^{(j)} = \frac{1}{32}\left[13 + 4c^{(j)} \pm 4\sqrt{\left(\frac{21}{4} + c^{(j)}\right)\left(\frac{5}{4} + c^{(j)}\right)}\right]$$

mit $c^{(j)} = \cos\frac{6\pi j}{n}$.

Als letzter Spezialfall soll hier kurz betrachtet werden, dass n durch 5 teilbar ist. Hierzu existieren nur experimentell ermittelte Resultate. Analog zu $C_n^{[3]}$ kann man nun 5 verschiedene Farben im Wechsel verwenden, siehe $C_n^{[5a]}$ in Abb. 3.4. Bei Verfahren, die die symmetrische Matrix M^{SDE} verwenden, reduziert sich dann die Anzahl der Eigenwerte und damit der Schritte, vgl. Tabelle 3.2. Andererseits erhöht sich die Anzahl der Teilschritte von 3 auf 5. Vertauscht man die Farben 2 und 3 ($C_n^{[5b]}$), dann sind die zu den neuen Farben gehörenden Teilschritte 1 und 2 bzw. 3 und 4 unabhängig voneinander und können somit gleichzeitig ausgeführt werden. Fasst man die gleichzeitig ausgeführten Teilschritte jeweils zu einem Teilschritt zusammen, so gelangt man zu der Einfärbung $C_n^{[5c]}$, die mit drei Farben auskommt. Experimente zeigen, dass die Matrizen zu $C_n^{[5b]}$ und $C_n^{[5c]}$ genau dieselben Eigenwerte besitzen.

3.6.6 Produktgraphen

Es sei $G = G^{(1)} \times G^{(2)}$ ein Produktgraph mit $\left|V^{(1)}\right| = p$ und $\left|V^{(2)}\right| = q$. Der Graph $G^{(i)}$ sei mit $c^{(i)}$ Farben gefärbt ($i = 1, 2$). Die Verfahrensmatrizen seien $M^{\mathsf{DE}(i)} = M_{c^{(i)}}^{(i)} \cdots M_1^{(i)}$ und $M^{\mathsf{SDE}(i)}$ analog. Dann lässt sich der Produktgraph G mit $c = c^{(1)} + c^{(2)}$ Farben einfärben, wobei die Farben $1, \ldots, c^{(1)}$ für die Kanten in Richtung von $G^{(1)}$ verwendet

werden und die übrigen für die Kanten in Richtung von $G^{(2)}$. Für die Matrizen M^{DE} und M^{SDE} zu G folgt:

$$\begin{aligned}
M^{\mathsf{DE}} &= M_c \cdots M_{c^{(1)}+1} \cdot M_{c^{(1)}} \cdots M_1 \\
&= \left(M^{(2)}_{c^{(2)}} \otimes I_p\right) \cdots \left(M^{(2)}_1 \otimes I_p\right) \cdot \left(I_q \otimes M^{(1)}_{c^{(1)}}\right) \cdots \left(I_q \otimes M^{(1)}_1\right) \\
&= \left(M^{\mathsf{DE}(2)} \otimes I_p\right) \cdot \left(I_q \otimes M^{\mathsf{DE}(1)}\right) \\
&= M^{\mathsf{DE}(2)} \otimes M^{\mathsf{DE}(1)} \\
M^{\mathsf{SDE}} &= \left(M^{\mathsf{DE}}\right)^T M^{\mathsf{DE}} \\
&= \left(M^{\mathsf{DE}(2)} \otimes M^{\mathsf{DE}(1)}\right)^T \left(M^{\mathsf{DE}(2)} \otimes M^{\mathsf{DE}(1)}\right) \\
&= \left(\left(M^{\mathsf{DE}(2)}\right)^T \otimes \left(M^{\mathsf{DE}(1)}\right)^T\right) \left(M^{\mathsf{DE}(2)} \otimes M^{\mathsf{DE}(1)}\right) \\
&= \left(\left(M^{\mathsf{DE}(2)}\right)^T M^{\mathsf{DE}(2)}\right) \otimes \left(\left(M^{\mathsf{DE}(1)}\right)^T M^{\mathsf{DE}(1)}\right) \\
&= M^{\mathsf{SDE}(2)} \otimes M^{\mathsf{SDE}(1)}
\end{aligned}$$

Es müssen also zunächst die Eigenwerte der beiden Faktorgraphen bestimmt werden. Anschließend müssen alle möglichen Produkte je zweier Eigenwerte gebildet werden (Aufwand $\mathcal{O}(pq)$). In der Regel treten hierbei einige Produkte mehrfach auf, und zwar auch dann, wenn man aus Symmetriegründen offensichtlich doppelte Produkte erst gar nicht mit berechnet. Mehrfache Eigenwerte lassen sich zum Beispiel dadurch entfernen, dass man die Werte sortiert, diese sortierte Folge einmal durchläuft und dabei alle mehrfachen Werte streicht. Der Aufwand hierfür beträgt $\mathcal{O}(pq\log(pq))$. Selbst in den Fällen, in denen keine Formeln für die Eigenwerte des eindimensionalen Graphen bekannt sind, liegt der Gesamtaufwand so nur bei $\mathcal{O}\left(p^3+q^3\right)$ verglichen mit $\mathcal{O}\left((pq)^3\right)$, würde man die Eigenwerte von $M^{(\mathsf{S})\mathsf{DE}}$ direkt ausrechnen.

Der Graph $G^{(i)}$ habe nun $m^{(i)}$ verschiedene Eigenwerte. Dann ergibt sich für G eine Anzahl

$$m \leq m^{(1)}m^{(2)}\,.$$

Ist $G^{(1)} = G^{(2)}$, so erhält man als maximale Anzahl verschiedener Produkte von $m^{(1)}$ Werten

$$m \leq \sum_{i=1}^{m^{(1)}} i = \frac{1}{2}m^{(1)}\left(m^{(1)}+1\right)\,.$$

Im Fall $\alpha = \frac{1}{2}$ ist einer der Eigenwerte der Faktorgraphen 0, und damit sind auch alle Produkte mit diesem Eigenwert gleich 0. Die Zahl m reduziert sich hierdurch auf

$$m \leq \left(m^{(1)}-1\right)\left(m^{(2)}-1\right)+1 = m^{(1)}m^{(2)} - \left(m^{(1)}+m^{(2)}\right)$$

bzw. im Fall $G^{(1)} = G^{(2)}$ auf

$$m \leq 1 + \sum_{i=1}^{m^{(1)}-1} i = \frac{1}{2} m^{(1)} \left(m^{(1)} - 1 \right) + 1 \,.$$

3.6.7 Gitter und Tori

Gitter und Tori sind nichts anderes als Produkte von Pfaden bzw. Zyklen. Entsprechend gelten für die Eigenwertanzahlen die Abschätzungen aus dem letzten Abschnitt. Bei einzelnen Graphen und Verfahren liegt die tatsächliche Eigenwertanzahl etwas unterhalb der so berechneten Schranken, zumindest der Koeffizient vor n^2 stimmt aber Experimenten zufolge exakt. Vergleiche hierzu auch Tabelle 3.2.

Ist beim Torus mindestens eine Dimension ungerade, so benötigt man zur Einfärbung mindestens fünf Farben. Falls sogar beide Dimensionen ungerade sind, gibt es keine Einfärbung mit minimaler Farbanzahl mehr, die die Produktstruktur respektiert, vgl. Abbildung 3.5. Allerdings hat diese Einfärbung zwei gravierende Nachteile: Der Aufwand zur Berechnung der Eigenwerte ist (im quadratischen Fall) $\mathcal{O}\left(n^3\right)$ und die Anzahl der Eigenwerte ist verglichen mit der der Diffusionsmatrix zu hoch. Wesentlich günstiger

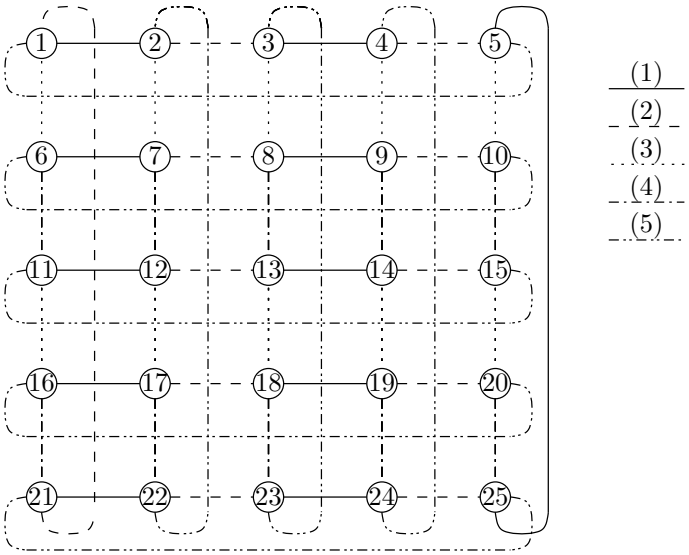


Abbildung 3.5: Einfärbung des Torus T_{2k+1}

ist es, sechs Farben zu verwenden. Zur Unterscheidung der verschiedenen Einfärbungen werden die folgenden Bezeichnungen benutzt: $T_k^{[6]} = C_k \times C_k$ und $T_k^{[C3]} = C_k^{[3]} \times C_k^{[3]}$.

3.6.8 Hypercube H_d

Der Hypercube H_d der Dimension d wird anschaulich so nummeriert und mit d Farben eingefärbt, dass parallele Kanten dieselbe Farbe besitzen, vgl. Abbildung 3.6. Verringerte

man alle Knotennummern um eins, so unterschieden sich die Endknoten von Kanten der Farbe j in der Binärdarstellung nur im j-ten Bit.

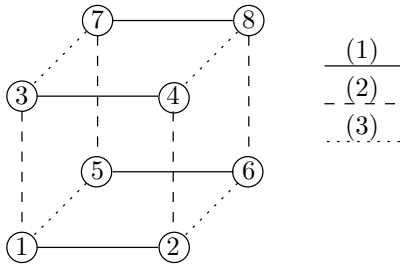


Abbildung 3.6: Einfärbung des Hypercubes H_d

Dieselbe Einfärbung ergibt sich, wenn man den Hypercube H_d auffasst als d-faches Produkt von Pfaden P_2 der Länge 2. Damit ist

$$M^{\mathsf{DE}} = \bigotimes_{i=1}^{d} \begin{pmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{pmatrix}$$

und hat als Eigenwerte $1, 1-2\alpha, \ldots, (1-2\alpha)^d$. Wählt man speziell $\alpha = \frac{1}{2}$, so existieren nur zwei Eigenwerte, nämlich 0 und 1. Dies stimmt mit den Ergebnissen in [Cyb89] überein.

Da die Matrix M^{DE} symmetrisch ist, ist $M^{\mathsf{SDE}} = \left(M^{\mathsf{DE}}\right)^2$ und hat folglich die Eigenwerte $1, (1-2\alpha)^2, \ldots, (1-2\alpha)^{2d}$.

3.6.9 Zusammenfassung

In Tabelle 3.2 sind für alle bisher betrachteten Einfärbungen und Verfahren die Anzahlen der Eigenwerte der zugehörigen Matrizen zusammengefasst. Die Zahl der Schritte, die die Verfahren benötigen ist jeweils um eins geringer. Um genaue Aussagen über das Laufzeitverhalten zu erlangen, muss aber jeweils noch die Anzahl der Teilschritte pro Schritt mit berücksichtigt werden, vergleiche hierzu Abschnitt 3.9. Zusätzlich ist der Aufwand zur Bestimmung der Eigenwerte für die Dimension-Exchange-Verfahren angegeben. Dort, wo für den Aufwand zwei verschiedene Werte angegeben sind, gilt der kleinere Wert jeweils für den Fall $\alpha = \frac{1}{2}$ und der größere für $\alpha \neq \frac{1}{2}$.

3.7 Flussminimierung

Bei den Diffusionsverfahren ist bekannt, dass die berechneten Flüsse in der l_2-Norm minimal sind [HB99, DFM99]. Dies ist beim Dimension-Exchange, wie Experimente zeigen, in der Regel nicht der Fall. Die einzigen Graphen, für die diese Beobachtung nicht zutrifft, sind Bäume (also z. B. auch Pfade), bei denen der Fluss nämlich eindeutig bestimmt ist. Bei allen übrigen Graphen, also Graphen mit Zyklen, deuten Messungen darauf hin, dass

Graph		OPT	SDE-OPT, $\alpha \neq \frac{1}{2}$	DE-OPT, $\alpha \neq \frac{1}{2}$	SDE-OPT, $\alpha = \frac{1}{2}$	DE-OPT, $\alpha = \frac{1}{2}$	Aufwand
P_n	$2 \mid n$	n	(n)		$\frac{1}{2}n+1$		$\mathcal{O}(n)$ / $\mathcal{O}(n^3)$
	$2 \nmid n$	n	(n)		$\frac{1}{2}n+\frac{3}{2}$		$\mathcal{O}(n)$ / $\mathcal{O}(n^3)$
C_n	$4 \mid n$	$\frac{1}{2}n+1$	$\frac{1}{2}n+1$		$\frac{1}{4}n+1$		$\mathcal{O}(n)$
	$2 \mid n, 4 \nmid n$	$\frac{1}{2}n+1$	$\frac{1}{2}n+1$		$\frac{1}{4}n+\frac{3}{2}$		$\mathcal{O}(n)$
	$2 \nmid n$	$\frac{1}{2}n+\frac{1}{2}$	(n)		$(\frac{1}{2}n+\frac{3}{2})$		$\mathcal{O}(n^3)$
$C_n^{[3]}$	$3 \mid n, 2 \nmid n$	$\frac{1}{2}n+\frac{1}{2}$	$(\frac{1}{2}n+\frac{3}{2})$	(n)	$\frac{1}{3}n+2$	$(\frac{2}{3}n+1)$	$\mathcal{O}(n)$
$C_n^{[5a]}$	$5 \mid n, 2 \nmid n$	$\frac{1}{2}n+\frac{1}{2}$	$(\frac{1}{2}n+\frac{5}{2})$	(n)	$(\frac{2}{5}n+3)$	$(\frac{4}{5}n+1)$	$\mathcal{O}(n^3)$
$C_n^{[5b/c]}$	$5 \mid n, 2 \nmid n$	$\frac{1}{2}n+\frac{1}{2}$	$(\frac{1}{2}n+\frac{5}{2})$	(n)	$(\frac{3}{10}n+\frac{5}{2})$	$(\frac{3}{5}n+1)$	$\mathcal{O}(n^3)$
G_k	$2 \mid k$	$\leq \frac{1}{2}k^2+1$	$(\frac{1}{2}k^2+1)$		$\frac{1}{8}k^2+\frac{1}{4}k+1$		$\mathcal{O}(k^2 \log k)$ / $\mathcal{O}(k^3)$
	$2 \nmid k$	$\leq \frac{1}{2}k^2+\frac{3}{2}$	$(\frac{1}{2}k^2+\frac{3}{2})$		$\frac{1}{8}k^2+\frac{1}{2}k+\frac{11}{8}$		$\mathcal{O}(k^2 \log k)$ / $\mathcal{O}(k^3)$
T_k	$4 \mid k$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+1$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+1$		$\frac{1}{32}k^2+\frac{1}{8}k+1$		$\mathcal{O}(k^2 \log k)$
	$2 \mid k, 4 \nmid k$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{2}$	$\leq \frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{2}$		$\frac{1}{32}k^2+\frac{1}{4}k+\frac{11}{8}$		$\mathcal{O}(k^2 \log k)$
	$2 \nmid k$	$\frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{8}$	(k^2)		$(\frac{1}{4}k^2+\frac{1}{2}k+\frac{5}{4})$		$\mathcal{O}(k^6)$
$T_k^{[6]}$	$2 \nmid k$	$\frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{8}$	$\frac{1}{2}k^2+\frac{1}{2}k$		$\frac{1}{8}k^2+\frac{1}{2}k+\frac{11}{8}$		$\mathcal{O}(k^3)$
$T_k^{[C3]}$	$3 \mid k, 2 \nmid k$	$\frac{1}{8}k^2+\frac{1}{2}k+\frac{3}{8}$	$\frac{1}{8}k^2+k+\frac{15}{8}$	$\frac{1}{2}k^2+\frac{1}{2}k$	$\frac{1}{18}k^2+\frac{1}{3}k+\frac{5}{2}$	$\frac{2}{9}k^2+\frac{1}{3}k+1$	$\mathcal{O}(k^3)$
H_d		$d+1$	$d+1$		2		$\mathcal{O}(d)$ / $\mathcal{O}(1)$
S_n		3	n		n		$\mathcal{O}(n)$
K_n	$2 \mid n$	2	$\leq n$		$(\leq \sqrt{n+1}+1)$	$(\leq 2\sqrt{2n+1}-2)$	$\mathcal{O}(n^3)$
	$2 \nmid n$	2	$\leq n$		$(\leq 2\sqrt{n+1})$	$(\leq 2\sqrt{2n+2}-2)$	$\mathcal{O}(n^3)$

Tabelle 3.2: Anzahl der verschiedenen Eigenwerte bei Standardgraphen und verschiedenen Verfahren; experimentell ermittelte Werte sind in Klammern notiert; wo für den Aufwand zwei verschiedene Werte angegeben sind, gilt der kleinere für den Fall $\alpha = \frac{1}{2}$ und der größere für $\alpha \neq \frac{1}{2}$

die l_2-Normen der Flüsse monoton mit α wachsen. Wie folgendes Lemma zeigt, ergeben sich zumindest für α nahe Null annähernd l_2-minimale Flüsse.

Lemma 3.22. *Lässt man bei Dimension-Exchange-Verfahren den Wert α gegen Null laufen, dann konvergieren die von den Verfahren berechneten Flüsse gegen den l_2-minimalen Fluss.*

Beweis. Wegen der Äquivalenz von OPS und OPT (siehe Bemerkung 2.19) wird nur letzteres Verfahren betrachtet. Der berechnete Fluss hängt bei diesem endlichen Verfahren stetig von α ab. Für die Matrix L^{DE} gilt:

$$\begin{aligned} L^{\mathsf{DE}} &= \frac{1}{\alpha}\left(I - M^{\mathsf{DE}}\right) \\ &= \frac{1}{\alpha}\left(I - M_c \cdots M_1\right) \\ &= \frac{1}{\alpha}\left(I - (I - \alpha L_c) \cdots (I - \alpha L_1)\right) \\ &= \frac{1}{\alpha}\left(\alpha\left(L_c + \ldots + L_1\right) + \alpha^2\left(\ldots\right)\right) \\ &\xrightarrow{\alpha \to 0} L_c + \ldots + L_1 = L^{\mathsf{Diff}} \end{aligned}$$

Bis auf Terme höherer Ordnung stimmt L^{DE} also mit L^{Diff} überein. Analog erhält man $L^{\mathsf{SDE}} \approx 2L$, wobei sich der Faktor 2 bei der Division durch die Eigenwerte in den Verfahren herauskürzt. Für α nahe 0 stimmen also beide Dimension-Exchange-Varianten mit dem Diffusionsverfahren (nahezu) überein. Von letzterem ist aber bereits bekannt, dass es minimale Flüsse erzeugt. □

Bemerkung 3.23. Der letzte Beweis zeigt, dass für α nahe 0 bei den Dimension-Exchange-Varianten aus Stetigkeitsgründen nicht weniger Eigenwerte auftreten können als beim Diffusionsverfahren.

Wie der letzte Abschnitt über die Eigenwerte gezeigt hat, sind die Dimension-Exchange-Verfahren aber nicht für solch kleine α-Werte sondern nur für $\alpha = \frac{1}{2}$ besonders schnell. Es wäre daher wünschenswert, Abschätzungen dafür zu erhalten, um wie viel die Flüsse für $\alpha = \frac{1}{2}$ über dem Minimum liegen.

Abbildung 3.7 zeigt am Beispiel des Torus T_8, wie die Flusswerte sich mit $\alpha \in (0, 1)$ ändern. Während die Normen bei beiden Varianten für $\alpha = \frac{1}{2}$ nahezu identisch sind, sind für $\alpha \to 1$ große Unterschiede zu beobachten. Während die Werte beim SDE-OPX beschränkt bleiben, gehen sie beim DE-OPX gegen Unendlich. Wie man solche Flussnormen berechnen kann, ist Thema des nächsten Abschnitts 3.8. Hier werden zunächst Techniken vorgestellt, mit denen die Norm der Flüsse gesenkt oder in bestimmten Fällen sogar minimiert werden kann. Deren Korrektheit wird durch folgenden (unmittelbar einsichtigen) Satz sichergestellt.

Satz 3.24. *Sei $G = (V, E)$ ein ungerichteter zusammenhängender Graph mit Inzidenzmatrix A; w^0 und $\overline{w}$ seien wie gehabt. Es mögen verschiedene ausgleichende Flüsse $x_1, \ldots, x_r$ existieren, d. h. $Ax^i = w^0 - \overline{w}$, $i = 1, \ldots, r$. Ferner seien $\delta_1, \ldots, \delta_r \in \mathbb{R}$ mit $\sum_{i=1}^{r} \delta_i = 1$. Dann ist auch $\overline{x} = \sum_{i=1}^{r} \delta_i x^i$ ein ausgleichender Fluss, also $A\overline{x} = w^0 - \overline{w}$.*

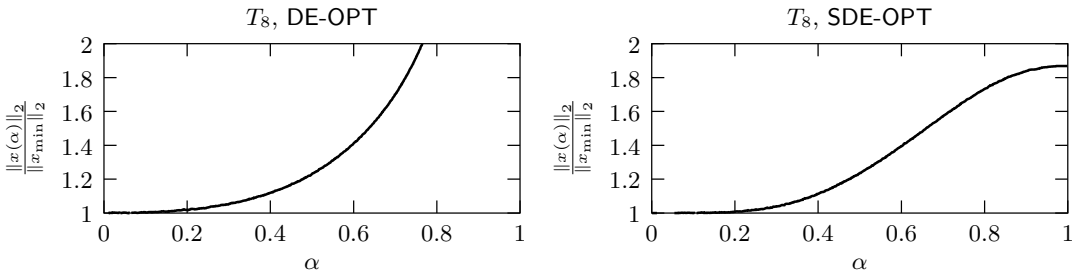


Abbildung 3.7: l_2-Norm des Flusses in Abhängigkeit von α am Beispiel eines 8×8-Torus mit zufälligem w^0 (minimaler Fluss = 1)

3.7.1 Vorwärts und rückwärts

Es ist zu beobachten, dass die einzelnen Komponenten des Flusses davon abhängig sind, in welcher Reihenfolge die einzelnen Teilschritte und damit die Farben abgearbeitet werden.

Als einfaches Beispiel hierzu wird der Hypercube H_2 betrachtet, bei dem anfangs ein Knoten die Last 4 und die drei übrigen die Last 0 haben. Abhängig davon, ob man mit den waagerechten oder den senkrechten Kanten beginnt, erhält man zwei verschiedene Flüsse x^1 und x^2, beide mit Norm $\sqrt{6}$. Der Fluss $x = \frac{1}{2}\left(x^1 + x^2\right)$, der sich als Mittelwert der anderen beiden ergibt, hat eine geringere und in diesem Beispiel sogar minimale Norm, nämlich $\sqrt{5}$. Vergleiche hierzu Abbildung 3.8.

Dieses bislang für einen Spezialfall beschriebene Vorgehen lässt sich problemlos auf beliebige Graphen übertragen. Das DE-OPS- bzw. DE-OPT-Verfahren wird zweimal durchgeführt, wobei die Farben einmal immer vorwärts und einmal rückwärts durchlaufen werden, anschließend wird der Mittelwert der beiden so bestimmten Flüsse berechnet. Die Korrektheit dieses neuen, mit DE-OPSfb bzw. DE-OPTfb *(forward-backward)* bezeichneten Verfahrens ist durch Satz 3.24 sichergestellt. Die Matrizen $M_c \cdots M_1 = M^{\mathsf{DE}}$ und $M_1 \cdots M_c = M^{\mathsf{DE}^T}$ haben dieselben Eigenwerte, denn die Diffusionsmatrizen M_i sind symmetrisch. Der Kommunikationsaufwand ist sicherlich nicht höher als für SDE-OPX, wo ja pro Schritt auch alle Farben zweimal (vorwärts und rückwärts) durchlaufen werden. Weitere Details zur Implementierung werden in Abschnitt 5.3 gegeben.

3.7.2 Zyklisches Durchlaufen der Farben

Die zweite Variante nutzt aus, dass alle Matrizen der Form $M_{j-1} \cdots M_1 \cdot M_c \cdots M_j$ für $j \in \{2, \ldots, c\}$ dieselben Eigenwerte haben wie $M^{\mathsf{DE}} = M_c \cdots M_1$. Das DE-OPX-Verfahren wird nun für jede dieser insgesamt c Matrizen einmal durchgeführt, anschließend wird der Mittelwert aller Flüsse bestimmt. Das gesamte, mit DE-OPXcc *(color cycling)* bezeichnete Verfahren hat zwar den c-fachen Rechenaufwand verglichen mit DE-OPX, der Kommunikationsaufwand erhöht sich dagegen nur um $c - 1$ Schritte. Erreicht wird dies dadurch, dass die Einzelrechnungen jeweils um einen Teilschritt versetzt gestartet werden, vergleiche Abbildung 3.9. Da die Anzahl der Rechen- und Kommunikationsopera-

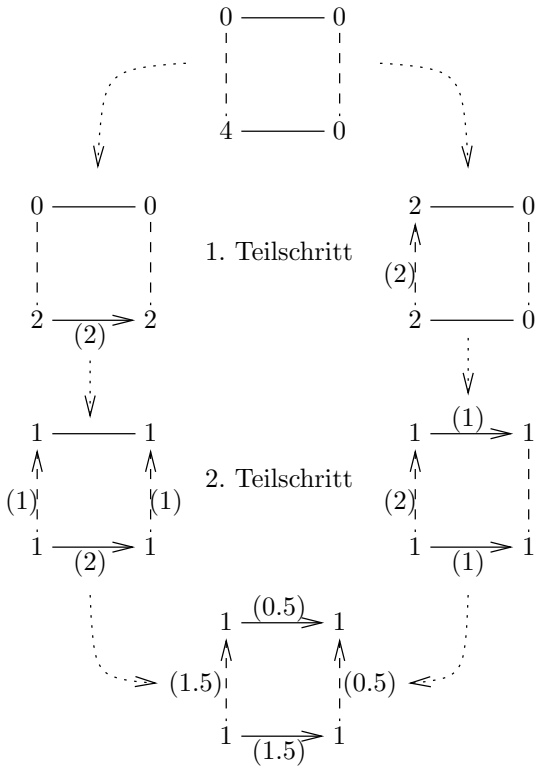


Abbildung 3.8: DE-OPSfb für den Hypercube H_2

tionen die gleiche Größenordnung haben, spielt der erhöhte Rechenaufwand keine Rolle. Dass die Menge der mit jeder Kommunikation auszutauschenden Daten sich ver-c-facht ist ebenfalls eher bedeutungslos, da die Datenmengen trotzdem noch sehr klein sind und daher die meiste Zeit für den Aufbau der Kommunikation verwandt wird. Details zur Implementierung dieser Variante werden später in Algorithmus 5.1 erläutert.

Abbildung 3.10 zeigt, wie sich beide Varianten auf das Beispiel des T_8 aus Abb. 3.7 auswirken.

3.8 Abschätzungen für Flüsse

In diesem Abschnitt werden Sätze hergeleitet, die es erlauben, zu gegebenen Verfahren und Graphen auszurechnen, um welchen Faktor die l_2-Normen der Flüsse die minimale Norm höchstens übersteigen. Diese Faktoren sind als Normen gewisser Matrizen gegeben und lassen sich in einigen Fällen explizit zahlenmäßig angeben, in den anderen Fällen lassen sie sich experimentell bestimmen.

Für die folgende Definition der Pseudoinversen sowie die beiden sich daran anschließenden Sätze vergleiche zum Beispiel [GL96, Dav79].

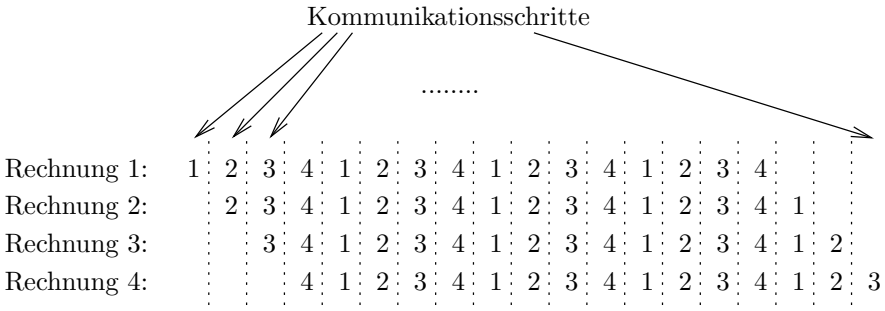


Abbildung 3.9: Ablauf der Rechnung beim DE-OPXcc am Beispiel von 4 Schritten und 4 Farben

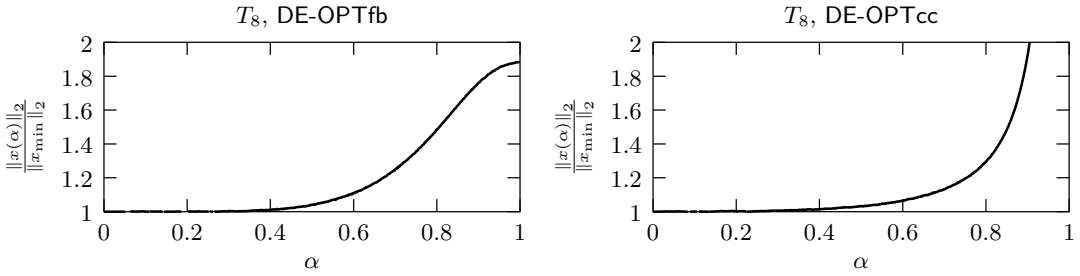


Abbildung 3.10: Fortsetzung des Beispiels aus Abbildung 3.7 für die beiden Flussreduzierungsvarianten

Definition 3.25. Sei $A \in \mathbb{R}^{n\times q}$. Eine Matrix $X \in \mathbb{R}^{q\times n}$ heißt *Pseudoinverse* oder *Moore-Penrose-Inverse* zu A genau dann, wenn folgende vier Bedingungen erfüllt sind:

$$AXA = A \tag{3.2}$$
$$XAX = X \tag{3.3}$$
$$(AX)^T = AX \tag{3.4}$$
$$(XA)^T = XA \tag{3.5}$$

Die Pseudoinverse von A ist hierdurch eindeutig bestimmt und wird mit A^+ bezeichnet. Ist A quadratisch und regulär, so ist $A^+ = A^{-1}$.

Satz 3.26. *Es sei* $D = \operatorname{diag}(d_1, \ldots, d_n) \in \mathbb{R}^{n\times n}$ *eine Diagonalmatrix. Dann ist* $D^+ = \operatorname{diag}\left(d_1^+, \ldots, d_n^+\right)$ *gegeben durch*

$$d_i^+ = \begin{cases} \frac{1}{d_i} & \text{falls } d_i \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

Satz 3.27. *Es sei* $A \in \mathbb{R}^{n\times q}$ *und* $A = U\Sigma V^T$ *deren Singulärwertzerlegung. Dann gilt*

$$A^+ = V\Sigma^+ U^T.$$

Nach [Iji65, Theorem 1] gilt für Adjazenzmatrizen $A \in \mathbb{R}^{n \times N}$, die zu zusammenhängenden Graphen gehören, $AA^+ = I - \frac{1}{n}J$. Dabei ist J die Matrix, deren Einträge alle gleich 1 sind. Auf ähnliche Weise wie dieses Resultat lässt sich folgendes zeigen.

Satz 3.28. *Es sei L^{ALG} die Laplace-Matrix des Diffusionsverfahrens oder eines der entsprechenden Gegenstücke eines Dimension-Exchange-Verfahrens. Dann gilt:*

$$L^{\mathsf{ALG}} L^{\mathsf{ALG}^+} = I - \frac{1}{n}J$$

und

$$L^{\mathsf{ALG}^+} L^{\mathsf{ALG}} = I - \frac{1}{n}J$$

Dabei ist J die Matrix, die überall Einsen enthält.

Beweis. Nach Definition der Pseudoinversen ist $L^{\mathsf{ALG}} L^{\mathsf{ALG}^+} L^{\mathsf{ALG}} = L^{\mathsf{ALG}}$ und damit

$$\left(I - L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}\right) L^{\mathsf{ALG}} = 0\,.$$

Für $e = (1, \ldots, 1)^T \in \mathbb{R}^n$ gilt $e^T L^{\mathsf{ALG}} = e^T A A^{\mathsf{ALG}^T} = 0$. Da L^{ALG} Rang $n-1$ hat, müssen folglich alle Zeilen von $I - L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}$ Vielfache von e^T sein. Da $L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}$ nach der Moore-Penrose-Bedingung (3.4) außerdem symmetrisch ist, ist $I - L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}$ ein Vielfaches von J. Wegen

$$\left(I - L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}\right)^2 = I - 2L^{\mathsf{ALG}} L^{\mathsf{ALG}^+} + \underbrace{L^{\mathsf{ALG}} L^{\mathsf{ALG}^+} L^{\mathsf{ALG}}}_{L^{\mathsf{ALG}}} L^{\mathsf{ALG}^+} = I - L^{\mathsf{ALG}} L^{\mathsf{ALG}^+}$$

folgt schließlich die erste Behauptung. Die zweite Gleichheit zeigt man völlig analog. □

Folgendes Lemma stammt ebenfalls aus [Iji65].

Lemma 3.29. *Es seien $A \in \mathbb{R}^{m \times n}$ und $x \in \mathbb{R}^n$. Dann ist $Ax = 0$ genau dann, wenn $x^T A^+ = 0$.*

In Analogie zur Matrix $A^{\mathsf{Diff}} = A$ bei den Diffusionsverfahren werden nun ähnliche Matrizen für die übrigen Verfahren benötigt.

Definition 3.30. Gegeben sei ein gefärbter Graph G, M_i und A_i seien wie bisher die Diffusionsmatrizen und Adjazenzmatrizen der Teilgraphen zur Farbe i. Dann werden

folgende neue Matrizen definiert:

$$
\begin{aligned}
A^{\mathsf{DE}} &= \sum_{j=1}^{c} M_1 \cdots M_{j-1} A_j \\
A^{\mathsf{SDE}} &= \sum_{j=1}^{c} M_1 \cdots M_{j-1} A_j + \sum_{j=c}^{1} M_1 \cdots M_c M_c \cdots M_{j+1} A_j \\
&= \sum_{j=1}^{c} \left(M_1 \cdots M_{j-1} + M_1 \cdots M_c M_c \cdots M_{j+1} \right) A_j \\
A^{\mathsf{DEfb}} &= \frac{1}{2} \sum_{j=1}^{c} \left(M_1 \cdots M_{j-1} + M_c \cdots M_{j+1} \right) A_j \\
A^{\mathsf{DEcc}} &= \frac{1}{c} \sum_{j=1}^{c} \left(\sum_{l=1}^{j} M_l \cdots M_{j-1} + \sum_{l=j+1}^{c} M_l \cdots M_c M_1 \cdots M_{j-1} \right) A_j
\end{aligned}
$$

Graph-Beispiel 3.31.

$$
\begin{aligned}
A^{\mathsf{DE}} &= A_1 + M_1 A_2 &&= \begin{pmatrix} 1 & \alpha \\ -1 & 1-\alpha \\ 0 & -1 \end{pmatrix} \\
A^{\mathsf{SDE}} &= A_1 + M_1 A_2 + M_1 M_2 A_2 + M_1 M_2 M_2 A_1 &&= 2 \begin{pmatrix} 1-\alpha+\alpha^2-\alpha^3 & \alpha(1-\alpha) \\ -1+2\alpha-2\alpha^2+\alpha^3 & (1-\alpha)^2 \\ \alpha(1-\alpha) & \alpha-1 \end{pmatrix} \\
A^{\mathsf{DEfb}} &= \frac{1}{2}\left(A_1 + M_1 A_2 + A_2 + M_2 A_1 \right) &&= \begin{pmatrix} 1 & \frac{\alpha}{2} \\ \frac{\alpha}{2}-1 & 1-\frac{\alpha}{2} \\ -\frac{\alpha}{2} & -1 \end{pmatrix} \\
A^{\mathsf{DEcc}} &= \frac{1}{2}\left(A_1 + M_1 A_2 + A_2 + M_2 A_1 \right) &&= \begin{pmatrix} 1 & \frac{\alpha}{2} \\ \frac{\alpha}{2}-1 & 1-\frac{\alpha}{2} \\ -\frac{\alpha}{2} & -1 \end{pmatrix}
\end{aligned}
$$

Der folgende Satz zeigt, dass sich die Eigenschaft $L^{\mathsf{Diff}} = AA^{\mathsf{Diff}^T}$ auf die neuen Matrizen überträgt.

Satz 3.32. *Für alle Loadbalancing-Verfahren* $\mathsf{ALG} \in \{\mathsf{Diff}, \mathsf{DE}, \mathsf{SDE}, \mathsf{DEcc}, \mathsf{DEfb}\}$ *gilt*

$$L^{\mathsf{ALG}} = AA^{\mathsf{ALG}^T}.$$

Beweis. Bewiesen wird nur den Fall $\mathsf{ALG} = \mathsf{DE}$. Alle übrigen Fälle werden ähnlich ge-

zeigt.

$$
\begin{aligned}
\alpha L^{\mathsf{DE}} &= I - M^{\mathsf{DE}} \\
&= I - M_c \cdots M_1 \\
&= I - \left(I - \alpha A_c A_c^T\right) M_{c-1} \cdots M_1 \\
&= I - M_{c-1} \cdots M_1 + \alpha A_c A_c^T M_{c-1} \cdots M_1 \\
&\vdots \\
&= I - M_1 + \alpha A_2 A_2^T M_1 + \cdots + \alpha A_c A_c^T M_{c-1} \cdots M_1 \\
&= \alpha A_1 A_1^T + \cdots + \alpha A_c A_c^T M_{c-1} \cdots M_1 \\
&= \alpha \sum_{j=1}^{c} A_j A_j^T M_{j-1} \cdots M_1 \\
&= \alpha \left(\sum_{j=1}^{c} A_j \right) \left(\sum_{j=1}^{c} A_j^T M_{j-1} \cdots M_1 \right) \quad \text{da } A_k A_l^T = 0 \text{ für } k \neq l \text{ nach Lemma 3.6} \\
&= \alpha A A^{\mathsf{DE}^T}
\end{aligned}
$$

□

Mit Hilfe der Matrizen aus Definition 3.30 lässt sich nun ein erstes Resultat für die Flüsse bestimmter Verfahren herleiten.

Satz 3.33. *Es sei* $\mathsf{ALG} \in \{\mathsf{Diff}, \mathsf{DE}, \mathsf{SDE}, \mathsf{DEcc}, \mathsf{DEfb}\}$ *eine der Verfahrensklassen beim Loadbalancing. Betrachtet werden Verfahren der Form*

$$w^k = p_k(M^{\mathsf{ALG}}) w^0$$

wobei p_k *ein Polynom vom Grad* k *ist mit* $p_k(1) = 1$*, so dass*

$$\lim_{k \to \infty} p_k(\mu_i^{\mathsf{ALG}}) = 0 \quad \text{für } i = 2, \ldots, m\,. \tag{3.6}$$

Schreibe die Anfangslastverteilung w^0 *als Summe von Eigenvektoren von* L^{ALG}*, also* $w^0 = \sum_{i=1}^{m} z_i$ *mit* $L^{\mathsf{ALG}} z_i = \lambda_i^{\mathsf{ALG}} z_i$ *und* $\lambda_1^{\mathsf{ALG}} = 0$*. Dann berechnet das Verfahren den folgenden ausgleichenden Fluss:*

$$x^{\mathsf{ALG}} = A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} z_i$$

Bemerkung 3.34. Bedingung (3.6) ist für alle betrachteten Verfahren erfüllt. Insbesondere gilt für die OPS-Varianten $p_{k-1}(\mu_i^{\mathsf{ALG}}) = 0$ und nach Gleichung (2.5) ist auch OPT von obiger polynomialer Form. Für das unsymmetrische Dimension-Exchange (DE) und dessen Varianten muss wieder die Diagonalisierbarkeit von L^{DE} vorausgesetzt werden.

Beweis von Satz 3.33. Wir beweisen den Satz zunächst für die einfachste Algorithmenklasse, nämlich ALG-FOS. Für den Fluss $x_{\mathsf{FOS}}^{\mathsf{ALG}^k}$ nach k Schritten gilt:

$$\begin{aligned}
x_{\mathsf{FOS}}^{\mathsf{ALG}^k} &= x_{\mathsf{FOS}}^{\mathsf{ALG}^{k-1}} + \alpha A^{\mathsf{ALG}^T} w^{k-1} \\
&= \alpha \sum_{l=0}^{k-1} A^{\mathsf{ALG}^T} M^{\mathsf{ALG}^l} w^0 \\
&= \alpha \sum_{l=0}^{k-1} A^{\mathsf{ALG}^T} M^{\mathsf{ALG}^l} \sum_{i=2}^{m} z_i \\
&= \alpha \sum_{l=0}^{k-1} A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \mu_i^{\mathsf{ALG}^l} z_i \\
&= \alpha A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \sum_{l=0}^{k-1} \mu_i^{\mathsf{ALG}^l} z_i \\
&= \alpha A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1-\mu_i^{\mathsf{ALG}^k}}{1-\mu_i^{\mathsf{ALG}}} z_i \\
&= A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1-\mu_i^{\mathsf{ALG}^k}}{\lambda_i^{\mathsf{ALG}}} z_i \qquad \text{da } \mu_i^{\mathsf{ALG}} = 1 - \alpha\lambda_i^{\mathsf{ALG}}
\end{aligned}$$

Im diesem einfachen Fall ist die Aussage wegen $\lim_{k\to\infty} \mu_i^{\mathsf{ALG}^k} = 0$, $i = 2, \ldots, m$ hiermit bereits bewiesen. Nun wird der allgemeine Fall $p_k(t) = \sum_{r=0}^{k} \varphi_r t^r$ betrachtet, also

$$w^k = \sum_{r=0}^{k} \varphi_r M^{\mathsf{ALG}^r} w^0.$$

Jeder Summand kann aufgefasst werden als ein nach r Schritten abgebrochenes FOS-Verfahren. Damit erhält man:

$$\begin{aligned}
x^{\mathsf{ALG}^k} &= \sum_{r=0}^{k} \varphi_r x_{\mathsf{FOS}}^{\mathsf{ALG}^r} \\
&= A^{\mathsf{ALG}^T} \sum_{r=0}^{k} \varphi_r \sum_{i=2}^{m} \frac{1-\mu_i^{\mathsf{ALG}^r}}{\lambda_i^{\mathsf{ALG}}} z_i \\
&= A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} \sum_{r=0}^{k} \left(\varphi_r \cdot 1 - \varphi_r \mu_i^{\mathsf{ALG}^r} \right) z_i \\
&= A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} \left(p_k(1) - p_k(\mu_i^{\mathsf{ALG}}) \right) z_i \\
&= A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} \left(1 - p_k(\mu_i^{\mathsf{ALG}}) \right) z_i
\end{aligned}$$

$$\begin{aligned} x^{\mathsf{ALG}} &= \lim_{k\to\infty} x^{\mathsf{ALG}^k} \\ &= A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} z_i \end{aligned}$$

□

Satz 3.33 hat den Nachteil, dass er die üblicherweise unbekannte Zerlegung von w^0 in Eigenvektoren benutzt. Der folgende Satz vermeidet dies.

Satz 3.35. *Es sei* $\mathsf{ALG} \in \{\mathsf{Diff}, \mathsf{DE}, \mathsf{SDE}, \mathsf{DEcc}, \mathsf{DEfb}\}$ *eine Loadbalancing-Verfahrensklasse. Ferner sei* w^0 *die Ausgangslastverteilung. Dann wird von dem Verfahren folgender Fluss* x^{ALG} *bestimmt:*

$$x^{\mathsf{ALG}} = A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^0$$

Beweis. Zunächst ist festzustellen, dass das angegebene x^{ALG} tatsächlich ein ausgleichender Fluss ist:

$$Ax^{\mathsf{ALG}} = AA^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^0 = L^{\mathsf{ALG}} L^{\mathsf{ALG}^+} w^0 = \left(I - \frac{1}{n} J\right) w^0 = w^0 - \overline{w}$$

Nun wird wieder die Zerlegung $w^0 = \sum_{i=1}^{m} z_i$ von w^0 in Eigenvektoren von L^{ALG} aus Satz 3.33 betrachtet. Für $i \neq 1$ erhält man mit Hilfe von Satz 3.28

$$L^{\mathsf{ALG}^+} L^{\mathsf{ALG}} z_i = \left(I - \frac{1}{n} J\right) z_i = z_i \,,$$

denn wegen $z_1 \perp z_i$ ist $Jz_i = 0$. Also ist

$$L^{\mathsf{ALG}^+} z_i = \frac{1}{\lambda_i^{\mathsf{ALG}}} z_i$$

und schließlich

$$A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^0 = A^{\mathsf{ALG}^T} \sum_{i=2}^{m} \frac{1}{\lambda_i^{\mathsf{ALG}}} z_i \,,$$

was mit der bekannten Formel aus Satz 3.33 übereinstimmt. □

Damit ist es nun möglich, den Fluss für ein bestimmtes Verfahren auf einem bestimmten Graphen unabhängig von der Anfangslast w^0 abzuschätzen.

Satz 3.36. *Es sei* $\mathsf{ALG} \in \{\mathsf{Diff}, \mathsf{DE}, \mathsf{SDE}, \mathsf{DEcc}, \mathsf{DEfb}\}$ *eine Loadbalancing-Verfahrensklasse.*

1. *Ist* $\tilde{x}$ *ein beliebiger ausgleichender Fluss, so erhält man den mit dem speziellen Verfahren berechneten Fluss* x^{ALG} *durch*

$$x^{\mathsf{ALG}} = A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A\tilde{x} \,. \tag{3.7}$$

   Insbesondere ist x^{ALG} *unabhängig von der Wahl von* $\tilde{x}$*.*

2. *Die l_2-Norm des Flusses x^{ALG} liegt maximal um den Faktor $\left\|A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A\right\|_2$ über der Norm des minimalen Flusses. Insbesondere ist also $\left\|A^T L^{\mathsf{Diff}^+} A\right\|_2 = 1$, da Diffusionsverfahren minimale Flüsse bestimmen.*

3. *Für jeden Graphen und jedes Verfahren lässt sich eine Ausgangslastverteilung w^0 finden, so dass die Schranke aus Teil 2 genau angenommen wird.*

Beweis.

1. Nach Satz 3.35 und mit $w^0 = A\tilde{x} + \overline{w}$ ist

$$x^{\mathsf{ALG}} = A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} (A\tilde{x} + \overline{w}) \,.$$

   Aus $\overline{w}^T L^{\mathsf{ALG}} = \overline{w}^T A A^{\mathsf{ALG}^T} = 0$ folgt mit Lemma 3.29 aber

$$L^{\mathsf{ALG}^+} \overline{w} = 0 \,.$$

2. Nimmt man für $\tilde{x}$ den minimalen Fluss x^{Diff}, so folgt aus Gleichung (3.7) unmittelbar

$$\left\|x^{\mathsf{ALG}}\right\|_2 \leq \left\|A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A\right\|_2 \left\|x^{\mathsf{Diff}}\right\|_2 \,.$$

3. Sei $N = A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A$ und $\nu = \|N\|_2$. Das heißt, ν^2 ist der größte Eigenwert von $N^T N$. Ferner sei y ein zugehörige Eigenvektor, also $N^T N y = \nu^2 y$. Der Vektor y lässt sich auffassen als Fluss auf dem Graphen. Eine passende Anfangslastverteilung erhält man aus $Ay = w^0 - \overline{w}$, indem man $\overline{w}$ zunächst beliebig wählt. Erhöhe dann w^0 und $\overline{w}$ um Vielfache von $\overline{w}$, bis alle Komponenten von w^0 nicht-negativ sind. Für den mit dem speziellen Verfahren berechneten Fluss x gilt dann:

$$\|x\|_2 = \|Ny\|_2 = \left(y^T N^T N y\right)^{\frac{1}{2}} = \nu \|y\|_2$$

   (Es ist also y ein minimaler Fluss.)

□

Aus Teil 1 des Satzes folgt unmittelbar:

Korollar 3.37. *Alle Flüsse, die von beliebigen Dimension-Exchange-Verfahren berechnet werden, sind trotz eventuell komplexer Eigenwerte reell.*

Tabelle 3.3 enthält experimentell ermittelte Werte für $\left\|A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A\right\|_2$ für verschiedene Verfahren und Graphen. XXX steht hierbei für einen beliebigen Verfahrenstyp, also FOS, OPS oder OPT.

Einige wenige dieser Resultate lassen sich zudem theoretisch bestätigen, vergleiche hierzu die nachfolgenden Sätze. Zunächst aber wird noch ein allgemeinerer Satz zum Nachweis minimaler Flüsse bewiesen.

		SDE-XXX	DE-XXX	DE-XXXfb	DE-XXXcc
P_n		1	1	1	1
C_n	$2 \mid n$	$\sqrt{2} \approx 1,414$	$\sqrt{2} \approx 1,414$	1	1
	$2 \nmid n$	$< \sqrt{2} \approx 1,414$	$< \sqrt{2} \approx 1,414$	$< \frac{3\sqrt{11}}{19} \approx 1,00277$	$< \frac{\sqrt{10}}{3} \approx 1,0541$
$C_n^{[3]}$	$3 \mid n$	$\frac{\sqrt{39}}{5} \approx 1,249$	$\frac{\sqrt{15}}{3} \approx 1,291$	$\frac{3\sqrt{11}}{19} \approx 1,00277$	1
$C_n^{[5]}$	$5 \mid n$	$\frac{5\sqrt{3}}{7} \approx 1,237$	$\frac{\sqrt{35}}{5} \approx 1,183$	$\frac{\sqrt{12335}}{111} \approx 1,000568$	1
$C_n^{[5b]}$	$5 \mid n$	$\frac{\sqrt{145}}{9} \approx 1,338$	$\frac{3\sqrt{5}}{5} \approx 1,342$	$\frac{\sqrt{15245}}{123} \approx 1,00383$	$\approx 1,0275$
$C_n^{[5c]}$	$5 \mid n$	$\frac{\sqrt{145}}{9} \approx 1,338$	$\frac{3\sqrt{5}}{5} \approx 1,342$	$\frac{\sqrt{15245}}{123} \approx 1,00383$	$\approx 1,0170$
G_k		< 2	< 2	$< \sqrt{2} \approx 1,414$	$< \frac{\sqrt{5}}{2} \approx 1,118$
T_k	$2 \mid k$	2	2	$\sqrt{2} \approx 1,414$	$\frac{\sqrt{5}}{2} \approx 1,118$
	$2 \nmid k$	< 2	$< 2,002$	$< \sqrt{2}$	$< 1,183$
$T_k^{[6]}$	$2 \nmid k$	< 2	$< 2,016$	$< \sqrt{2}$	$< 1,22$?
$T_k^{[3C]}$	$2 \nmid k$	$< 1,93$?	$< 2,078$?	$< 1,307$?	$< 1,18$?
H_d		$\sqrt{d}$	$\sqrt{d}$	$\sqrt{\frac{d}{2}}$, (1 für $d = 1$)	*

Tabelle 3.3: Die Werte geben an, um welchen Faktor die l_2-Norm der von den Verfahren berechneten Flüsse für $\alpha = \frac{1}{2}$ im schlimmsten Fall über der Norm des minimalen Flusses liegen kann. Die Werte für die ungeraden Tori sind numerisch bestätigt bis $k = 45$. Theoretisch nachgewiesen sind die Ergebnisse für den Pfad und den Zyklus gerader Länge sowie die ersten beiden Werte für den Hypercube. Bei DE-XXXfb lässt sich für H_d nur der Wert $\sqrt{\frac{d+1}{2}}$ beweisen. Vergleiche hierzu die nachfolgenden Sätze. *$1, 1, \frac{\sqrt{10}}{3}, \frac{\sqrt{20}}{4}, \frac{\sqrt{34}}{5}, \frac{\sqrt{54}}{6}, \frac{\sqrt{81}}{7}, \frac{\sqrt{114}}{8}, \frac{\sqrt{156}}{9}, \frac{\sqrt{208}}{10}, \frac{\sqrt{268}}{11}$ für $d = 1, \ldots, 11$

Satz 3.38. *Ein Loadbalancing-Verfahren erzeugt minimale Flüsse für jede Ausgangslastverteilung w^0 genau dann, wenn es minimale Flüsse für alle Peak-Ausgangsverteilung $w^{0,i,p_i} = (0, \ldots, 0, p_i, 0, \ldots, 0)$, $i = 1, \ldots, n$, $p_i \in \mathbb{N}_0$ erzeugt. Hierbei hat Prozessor i anfänglich p_i Lasteinheiten, alle anderen Prozessoren haben keine Last.*

Beweis. Sei $w^0 = \left(w_1^0, \ldots, w_n^0\right) = \sum_{i=1}^n w^{0,i,w_i^0}$ eine beliebige Ausgangslastverteilung und sei x^i der Fluss zu w^{0,i,p_i}, also $x^i = A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^{0,i,p_i}$, vergleiche Satz 3.35. Dann ist der Fluss x zu w^0

$$\begin{aligned} x &= A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^0 \\ &= A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} \sum_{i=1}^n w^{0,i,w_i^0} \\ &= \sum_{i=1}^n A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} w^{0,i,w_i^0} = \sum_{i=1}^n x^i \,. \end{aligned}$$

Nach Voraussetzung sind die x^i minimal, was insbesondere bei Diffusionsverfahren immer erfüllt ist. Da Diffusionsverfahren sogar für jede Ausgangsverteilung minimale Flüsse erzeugen und darüber hinaus minimalen Flüsse eindeutig bestimmt sind, ist auch x minimal. □

Jetzt werden wie angekündigt einige der Resultate aus Tabelle 3.3 theoretisch nachgerechnet. Bei Zyklen gerader Länge erhält man die Flussabschätzung durch pures Nachrechnen der entsprechenden Formeln.

Satz 3.39. *Bei Zyklen C_n gerader Länge n ist der von DE-XXX und SDE-XXX mit $\alpha = \frac{1}{2}$ berechnete Fluss in der l_2-Norm höchstens um den Faktor $\sqrt{2}$ höher als der minimale Fluss.*

Beweis. Nach Satz 3.36 ist zum Beweis die Norm von $X^{(\mathsf{S})\mathsf{DE}} = A^{(\mathsf{S})\mathsf{DE}^T} L^{(\mathsf{S})\mathsf{DE}^+} A$ nachzurechnen. Diese Berechnung ist nicht allzu schwierig, dafür etwas langwierig. Aus diesem Grund werden nur die wichtigsten Zwischenergebnisse angegeben. Dabei wird intensiv die in Definition 3.18 eingeführte Notation für blockzirkulante Matrizen verwendet, die hier auf nicht-quadratische Blöcke ausgedehnt wird.

$$
\begin{aligned}
A_1 &= \begin{pmatrix} \Phi\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, 0, \ldots, 0\right) & 0 \end{pmatrix} \\
A_2 &= \begin{pmatrix} 0 & \Phi\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} -1 \\ 0 \end{pmatrix}\right) \end{pmatrix} \\
A &= A_1 + A_2 \\
M_1 &= \Phi\left(\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}, 0, \ldots, 0\right) \\
M_2 &= \Phi\left(\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 0 \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix}\right) \\
A^{\mathsf{DE}} &= A_1 + M_1 A_2 \\
&= \begin{pmatrix} \Phi\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, 0, \ldots, 0\right) & \Phi\left(\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}\right) \end{pmatrix} \\
A^{\mathsf{SDE}} &= A_1 + M_1 A_2 + M_1 M_2 A_2 + M_1 M_2 M_2 A_1 \\
&= \begin{pmatrix} \Phi\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} -\frac{1}{4} \\ -\frac{1}{4} \end{pmatrix}\right) & \Phi\left(\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}\right) \end{pmatrix} \\
L^{\mathsf{DE}} &= \Phi\left(\begin{pmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} \end{pmatrix}, 0, \ldots, 0, \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 \end{pmatrix}\right) \\
L^{\mathsf{SDE}} &= \Phi\left(\Phi\left(\frac{3}{2}, -\frac{1}{2}\right), \Phi\left(-\frac{1}{4}, -\frac{1}{4}\right), 0, \ldots, 0, \Phi\left(-\frac{1}{4}, -\frac{1}{4}\right)\right)
\end{aligned}
$$

Für die benötigten Pseudoinversen $P^{(\mathsf{S})\mathsf{DE}} = L^{(\mathsf{S})\mathsf{DE}^+}$ findet man folgende Formeln:

$$
P^{(\mathsf{S})\mathsf{DE}} = \frac{1}{n} \Phi\left(P_1^{(\mathsf{S})\mathsf{DE}}, \ldots, P_{\frac{n}{2}}^{(\mathsf{S})\mathsf{DE}}\right)
$$

mit

$$
\begin{aligned}
P_1^{\mathsf{DE}} &= \Phi\left(\frac{1}{24}n^2 + \frac{1}{4}n - \frac{1}{6}, \frac{1}{24}n^2 - \frac{1}{4}n - \frac{1}{6}\right) \\
P_l^{\mathsf{DE}} &= \begin{pmatrix} l^2 - l - \frac{1}{2}ln + \frac{1}{24}n^2 + \frac{1}{4}n - \frac{1}{6} \\ l^2 - 3l - \frac{1}{2}ln + \frac{1}{24}n^2 + \frac{3}{4}n + \frac{11}{6} \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 \end{pmatrix} \qquad l = 2, \ldots, \frac{n}{2} \\
P_1^{\mathsf{SDE}} &= \Phi\left(\frac{1}{24}n^2 + \frac{1}{4}n - \frac{1}{6}, \frac{1}{24}n^2 - \frac{1}{4}n - \frac{1}{6}\right) \\
P_l^{(1)} &= \left(l^2 - 2l - \frac{1}{2}ln + \frac{1}{24}n^2 + \frac{1}{2}n + \frac{5}{6}\right) \cdot J_2 \qquad l = 2, \ldots, \frac{n}{2}
\end{aligned}
$$

Die Korrektheit der angegebenen Pseudoinversen lässt sich durch Nachrechnen der vier Moore-Penrose-Bedingungen (3.2)-(3.5) nachweisen; dabei ist

$$
P^{(\mathsf{S})\mathsf{DE}} L^{(\mathsf{S})\mathsf{DE}^+} = L^{(\mathsf{S})\mathsf{DE}^+} P^{(\mathsf{S})\mathsf{DE}} = \frac{1}{n}\,\Phi\,(n-1, -1 \ldots, -1)\ .
$$

Damit erhält man schließlich

$$
X^{\mathsf{DE}} = X^{\mathsf{SDE}} = \begin{pmatrix} I & -\frac{2}{n}J \\ 0 & I - \frac{2}{n}J \end{pmatrix}
$$

und

$$
X^{(\mathsf{S})\mathsf{DE}} X^{(\mathsf{S})\mathsf{DE}^T} = \begin{pmatrix} I + \frac{2}{n}J & 0 \\ 0 & I - \frac{2}{n}J \end{pmatrix}\ .
$$

Die letzte Matrix hat nur die drei Eigenwerte 0, 1 und 2 und somit folgt das Ergebnis

$$
\left\| X^{(\mathsf{S})\mathsf{DE}} \right\|_2 = \sqrt{2}\,.
$$

□

Bemerkung 3.40. Der Faktor $\sqrt{2}$ aus dem letzten Satz wird in folgender Situation erreicht: $w^0 = (4, 0, 4, 0, \ldots, 4, 0)$, $\overline{w} = (2, 2, \ldots, 2)$. Sortiert man die Einträge in den Flussvektoren nach Farben, so erhält man als minimalen Fluss

$$
x^{\mathsf{Diff}} = (1, \ldots, 1, -1, \ldots, -1)\ , \qquad \left\| x^{\mathsf{Diff}} \right\|_2 = \sqrt{n}
$$

und als Dimension-Exchange-Fluss

$$
x^{\mathsf{DE}} = (2, \ldots, 2, 0, \ldots, 0)\ , \qquad \left\| x^{\mathsf{DE}} \right\|_2 = \sqrt{\frac{n}{2} \cdot 4} = \sqrt{2} \cdot \sqrt{n}\,.
$$

Der Nachweis, dass die Flüsse auf dem Zyklus durch Flussreduktion sogar minimal werden, ist dank Satz 3.38 sehr einfach.

Satz 3.41. *Bei Zyklen C_n gerader Länge n ist der von DE-XXXfb und DE-XXXcc mit beliebigem α berechnete Fluss in der l_2-Norm minimal.*

Beweis. Nach Satz 3.38 genügt es, die Behauptung für Peakverteilungen zu zeigen. Außerdem stimmen beide Verfahren bei nur zwei Farben überein. O. B. d. A. habe anfänglich nur Knoten 1 eine von 0 verschiedene Last. Ein minimaler Fluss ist dadurch gekennzeichnet, dass der Flusswert für die beiden Kanten, die jeweils denselben Abstand vom Knoten 1 haben, gleich ist. Dies wird durch die Mittelwertbildung der Verfahren genau erreicht. □

Die Berechnung der Flüsse beim Hypercube vereinfacht sich dadurch ein wenig, dass man die benötigte Pseudoinverse leicht angegeben kann.

Satz 3.42. *Beim Hypercube H_d ist der von DE-XXX und SDE-XXX mit $\alpha = \frac{1}{2}$ berechnete Fluss in der l_2-Norm höchstens um den Faktor $\sqrt{d}$ höher als der minimale Fluss. Diese Grenze ist scharf, d. h. es gibt Fälle, in denen dieser Faktor erreicht wird.*

Beweis. Zunächst ist festzustellen, dass für $\alpha = \frac{1}{2}$ beide Verfahren dasselbe Ergebnis liefern. Das SDE-Verfahren hat nämlich bereits nach dem ersten Halbschritt (der mit einem DE-Schritt übereinstimmt) die Gleichverteilung der Last erzielt. Die Behauptung wird nun durch Berechnung der Norm von $A^{\mathsf{DE}^T} L^{\mathsf{DE}^+} A$ bewiesen, vergleiche Satz 3.36.

$$
\begin{aligned}
A_i &= \begin{pmatrix} 0_{2^d,(i-1)2^{d-1}} & I_{2^{d-i}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{i-1}} & 0_{2^d,(d-i)2^{d-1}} \end{pmatrix} \\
A &= \sum_{i=1}^{d} A_i \\
M_i &= \frac{1}{2} I_{2^{d-i}} \otimes J_2 \otimes I_{2^{i-1}} \\
A_i^{\mathsf{DE}} &= M_1 \cdots M_{i-1} A_i \\
&= \begin{pmatrix} 0_{2^d,(i-1)2^{d-1}} & \frac{1}{2^{i-1}} I_{2^{d-i}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes J_{2^{i-1}} & 0_{2^d,(d-i)2^{d-1}} \end{pmatrix} \\
A^{\mathsf{DE}} &= \sum_{i=1}^{d} A_i^{\mathsf{DE}} \\
M^{\mathsf{DE}} &= \frac{1}{2^d} J_{2^d} \\
L^{\mathsf{DE}} &= 2 \left(I_{2^d} - \frac{1}{2^d} J_{2^d} \right)
\end{aligned}
$$

Da die Matrix L^{DE} symmetrisch ist und als Eigenwerte nur 0 und 2 besitzt, hat sie als Pseudoinverse $L^{\mathsf{DE}^+} = \frac{1}{4} L^{\mathsf{DE}}$ (verwende $L^{\mathsf{DE}} = Q \Sigma Q^T$ mit $\Sigma = \operatorname{diag}(0, 2, \ldots, 2)$ und Satz 3.27). Damit ist

$$
A^{\mathsf{DE}^T} L^{\mathsf{DE}^+} A = \frac{1}{4} A^{\mathsf{DE}^T} L^{\mathsf{DE}} A = \frac{1}{4} A^{\mathsf{DE}^T} A A^{\mathsf{DE}^T} A = \frac{1}{4} \left(A^{\mathsf{DE}^T} A \right)^2 .
$$

Berechne nun $A^{\mathsf{DE}^T} A =: N = (N_{ij})_{i,j\in\{1,\dots,d\}}$:

$$N_{ij} = \begin{cases} \frac{1}{2^{i-2}} I_{2^{d-i}} \otimes J_{2^{i-1}} & i = j \\ 0 & i > j \\ \frac{1}{2^{i-1}} I_{2^{d-j}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-i-1}} \otimes J_{2^{i-1}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} & i < j \end{cases}$$

Weiterhin stellt man fest, dass $N^2 = 2N$ ist. Zusammengefasst ergibt sich, dass die gesuchte Größe

$$\left\| A^{\mathsf{DE}^T} L^{\mathsf{DE}^+} A \right\|_2 = \frac{1}{4} \left\| N^2 \right\|_2 = \frac{1}{4} \left\| 2N \right\|_2 = \frac{1}{2} \left\| N \right\|_2 = \frac{1}{2} \varrho \left(NN^T \right)^{\frac{1}{2}}$$

ist. NN^T ist eine Blockdiagonalmatrix mit Diagonalblöcken

$$\left(NN^T \right)_{ii} = \frac{1}{2^{i-3}} I_{2^{i-1}} \otimes J_{2^{i-1}} + \frac{1}{2^{i-2}} \sum_{k=i+1}^{d} I_{2^{d-k}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes I_{2^{k-i-1}} \otimes J_{2^{i-1}} \,.$$

Mit Hilfe des Satzes von Geršgorin folgt nun endlich

$$\left\| N \right\|_2^2 \leq \max_{i=1,\dots,d} \max_{r=1,\dots,2^{d-1}} \sum_{s=1}^{2^{d-1}} \left| \left(\left(NN^T \right)_{ii} \right)_{rs} \right| \leq \max_{i=1,\dots,d} 4 + 4\,(d-i) = 4d \,.$$

Beide Abschätzungen sind sogar Gleichheiten, wie sich durch Konstruktion einer Ausgangsverteilung zeigen lässt, bei der der Faktor $\sqrt{d}$ erreicht wird. Wähle hierzu w^0 so, dass für die beiden Endknoten jeder Kante gilt, dass einer die Last $2d$ hat, der andere 0 (zum Beispiel indem die Knoten i die Last $2d$ erhalten, für die die Binärdarstellung von $i-1$ eine gerade Anzahl Einsen enthält). Dann sind alle Komponenten des minimalen Flusses x^{Diff} betragsmäßig 1 und dessen Norm ist daher

$$\left\| x^{\mathsf{Diff}} \right\|_2 = \sqrt{N} = \sqrt{d \cdot 2^{d-1}} \,.$$

Der Dimension-Exchange-Fluss ist auf allen 2^{d-1} Kanten der ersten Farbe d und auf allen anderen 0 und seine Norm somit

$$\left\| x^{\mathsf{DE}} \right\|_2 = \sqrt{2^{d-1} \cdot d^2} = \sqrt{d} \cdot \sqrt{d \cdot 2^{d-1}} \,.$$

□

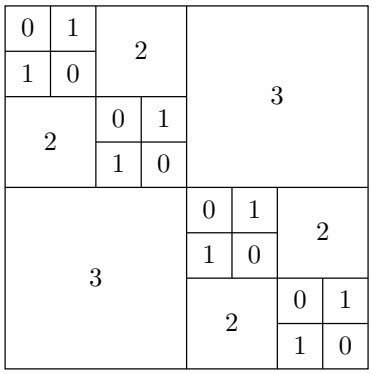
Das folgende Lemma wird benötigt für Hypercubeflüsse, die mit Vorwärts-rückwärts-Technik generiert werden.

Lemma 3.43. *Seien $i, d \in \mathbb{N}$ mit $1 \leq i \leq d$. Dann gelten folgende Gleichheiten:*

$$\sum_{k=1}^{i-1} \frac{1}{2^k} I_{2^{i-k-1}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes J_{2^{k-1}} + \frac{1}{2^{i-1}} J_{2^{i-1}} = I_{2^{i-1}}$$

$$\sum_{k=i+1}^{d} \frac{1}{2^{d-k+1}} J_{2^{d-k}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes I_{2^{k-i-1}} + \frac{1}{2^{d-i}} J_{2^{d-i}} = I_{2^{d-i}}$$

Beweis. Da beide Gleichungen ähnlich bewiesen werden, beschränkt sich der Beweis auf die erste Gleichung. Die Matrix auf deren linker Seite werde mit H bezeichnet. Diese werde entsprechend der folgenden Skizze in Gebiete zerlegt:



Auf der Diagonalen (Gebiet 0) gilt:

$$\mathit{H}_{rr} = \sum_{k=1}^{i-1} \frac{1}{2^k} + \frac{1}{2^{i-1}} = 1\,.$$

Sei nun H_{rs} ein Element aus Gebiet $l \in \{1, \ldots, i-1\}$. Für dieses Element folgt:

$$\mathit{H}_{rs} = -\frac{1}{2^l} + \sum_{k=l+1}^{i-1} \frac{1}{2^k} + \frac{1}{2^{i-1}} = 0\,.$$

□

Satz 3.44. *Beim Hypercube H_d ist der von* DE-XXXfb *mit $\alpha = \frac{1}{2}$ berechnete Fluss in der l_2-Norm höchstens um den Faktor $\sqrt{\frac{d+1}{2}}$ höher als der minimale Fluss.*

Beweis. Der Beweis dieses Satzes erfolgt nach dem selben Schema wie beim letzten Satz. Insbesondere ist $L^{\mathsf{DEfb}} = L^{\mathsf{DE}} = 2\left(I_{2^d} - \frac{1}{2^d} J_{2^d}\right)$. Für die zu diesem Verfahren gehörende Matrix A^{DEfb} gilt:

$$\begin{aligned}
A_i^{\mathsf{DEfb}} &= \frac{1}{2}\left(M_1 \cdots M_{i-1} A_i + M_d \cdots M_{i+1} A_i\right) \\
&= \begin{pmatrix} 0 & \frac{1}{2^i} I_{2^{d-i}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes J_{2^{i-1}} + \frac{1}{2^{d-i+1}} J_{2^{d-i}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{i-1}} & 0 \end{pmatrix} \\
A^{\mathsf{DEfb}} &= \sum_{i=1}^{d} A_i^{\mathsf{DEfb}}
\end{aligned}$$

Berechne nun die Matrix $N := A^{\mathsf{DEfb}^T} A$:

$$N_{ij} = \begin{cases} \frac{1}{2^{i-1}} I_{2^{d-i}} \otimes J_{2^{i-1}} + \frac{1}{2^{d-i}} J_{2^{d-i}} \otimes I_{2^{i-1}} & i = j \\ \frac{1}{2^i} I_{2^{d-j}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-i-1}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes J_{2^{i-1}} & i < j \\ \frac{1}{2^{d-i+1}} J_{2^{d-i}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-1}} & i > j \end{cases}$$

Weiterhin lässt sich nachrechnen, dass $N^2 = 2N$ ist. Im Gegensatz zum letzten Beweis muss man hier statt NN^T die Matrix $\mathfrak{N} := N^T N$ berechnen. Es werden nun zunächst die Diagonalblöcke dieser Matrix betrachtet.

$$\begin{aligned} \mathfrak{N}_{ii} &= \sum_{k=1}^{i-1} N_{ki}^T N_{ki} + N_{ii}^2 + \sum_{k=i+1}^{d} N_{ki}^T N_{ki} \\ &= \sum_{k=1}^{i-1} \frac{1}{2^k} I_{2^{d-k-1}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes J_{2^{k-1}} \\ &\quad + \frac{1}{2^{i-1}} I_{2^{d-i}} \otimes J_{2^{i-1}} + \frac{1}{2^{d-i}} J_{2^{d-i}} \otimes I_{2^{i-1}} + \frac{1}{2^{d-2}} J_{2^{d-1}} \\ &\quad + \sum_{k=i+1}^{d} \frac{1}{2^{d-k+1}} J_{2^{d-k}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes I_{2^{k-2}} \\ &= 2 I_{2^{d-1}} + \frac{1}{2^{d-2}} J_{2^{d-1}} \end{aligned}$$

Für die letzte Gleichheit ist Lemma 3.43 benutzt worden. Sei nun $i > j$:

$$\begin{aligned} \mathfrak{N}_{ij} &= \sum_{k=1}^{j-1} N_{ki}^T N_{kj} + N_{ji}^T N_{jj} + \sum_{k=j+1}^{i-1} N_{ki}^T N_{kj} + N_{ii}^T N_{ij} + \sum_{k=i+1}^{d} N_{ki}^T N_{kj} \\ &= I_{2^{d-i}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes \sum_{k=1}^{j-1} \frac{1}{2^{k+1}} I_{2^{j-k-1}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes J_{2^{k-1}} \\ &\quad + \frac{1}{2^j} I_{2^{d-i}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes J_{2^{j-1}} \\ &\quad + 0 \\ &\quad + \frac{1}{2^{d-i+1}} J_{2^{d-i}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-1}} \\ &\quad + \sum_{k=i+1}^{d} \frac{1}{2^{d-k+2}} J_{2^{d-k}} \otimes \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \otimes I_{2^{k-i-1}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-1}} \\ &= I_{2^{d-i}} \otimes \begin{pmatrix} 1 & -1 \end{pmatrix} \otimes I_{2^{i-j-1}} \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \otimes I_{2^{j-1}} \end{aligned}$$

Die letzte Umformung gilt wiederum nach Lemma 3.43. Die Summe der Beträge der Elemente jeder Zeile von $\mathfrak{N}$ ist damit

$$2 + \frac{1}{2^{d-2}} \cdot 2^{d-1} + (d-1) \cdot 2 = 2d + 2\,.$$

Nach Satz von Geršgorin ist damit gezeigt:

$$\left\| A^{\mathsf{DE}^T} L^{\mathsf{DE}^+} A \right\|_2 = \frac{1}{4} \left\| N^2 \right\|_2 = \frac{1}{4} \left\| 2N \right\|_2 = \frac{1}{2} \left\| N \right\|_2 = \frac{1}{2} \varrho\left(\mathfrak{N}\right)^{\frac{1}{2}} \leq \frac{1}{2}\sqrt{2d+2} = \sqrt{\frac{d+1}{2}}$$

□

Bemerkung 3.45. Man beachte, dass die experimentell ermittelte Schranke bei $\sqrt{\frac{d}{2}}$ statt $\sqrt{\frac{d+1}{2}}$ aus letztem Satz liegt. Im Gegensatz zu den vorangegangenen Sätze erhält man hier also nur eine Abschätzung der Flussnorm. Dennoch ist hiermit bewiesen, dass das Vorwärts-und Rückwärts-Durchlaufen der Farben eine Verbesserung des Verfahrens bewirkt.

Alle bisher angegebenen Schranken für Flüsse bezogen sich auf endliche Verfahren und das Kantengewicht $\alpha = \frac{1}{2}$. Beim DE-FOS und dessen Abwandlungen hängt das laufzeitoptimale α aber von der Größe des Graphen ab und liegt zwischen $\frac{1}{2}$ und 1, vergleiche Satz 3.11. Die in Abbildung 3.11 dargestellten Schranken für Gitter und Tori der Größen 2 bzw. 4 bis 24 verdeutlichen, dass hierdurch höhere Flüsse entstehen als bei OPT und OPS.

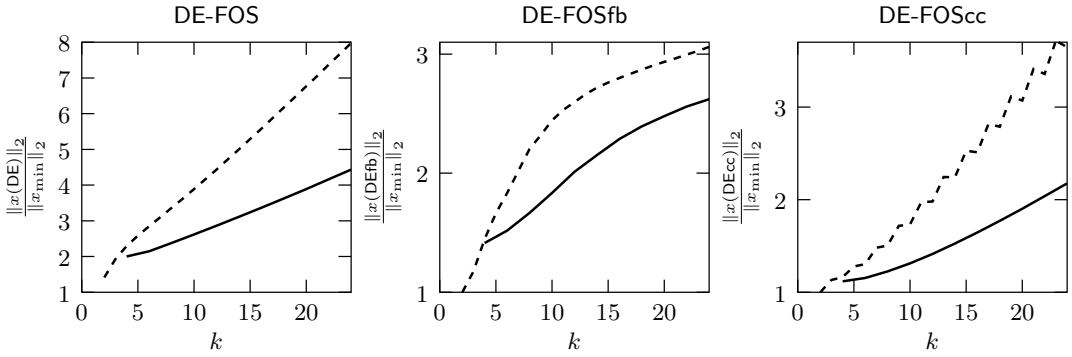


Abbildung 3.11: Schranken für die Normen der Flüsse bei DE-FOS, DE-FOSfb und DE-FOScc für Gitter G_k (gestrichelt) und Tori T_k (durchgezogene Linie) verschiedener Größe mit laufzeitoptimiertem α

3.9 Aufwand der Verfahren

Zur Beurteilung, welches Verfahren bei einem gegebenem Graphen das schnellste ist, reicht es nicht aus, die Zahl m der verschiedenen Eigenwerte zu betrachten. Vielmehr

muss man zusätzlich die Anzahl c der Farben berücksichtigen, da hierdurch (im wesentlichen) die Anzahl der Teilschritte pro Schritt bestimmt wird. Jeder Teilschritt besteht aus einer Kommunikationsoperation und einer festen Anzahl von Rechenoperationen. Da darüber hinaus die Größe der kommunizierten Daten recht klein ist, wird der Zeitaufwand praktisch ausschließlich durch die Zahl der Kommunikationen bestimmt.

Die DE-OPX-Verfahren benötigen $c(m-1)$ Kommunikationsschritte. Beim SDE-OPX werden pro Eigenwert alle Farben zweimal verwendet, was auf $2c(m-1)$ Schritte führt. Um Zeit zu sparen, kann man jedoch aufeinander folgende Teilschritte mit der gleichen Farbe zusammenfassen, vergleiche Abbildung 3.12. Dann genügen $(2c-2)(m-1)+1$ Kommunikationsschritte.

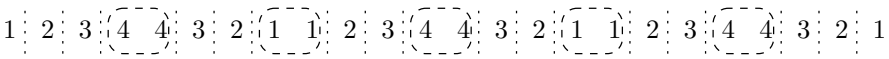


Abbildung 3.12: Kommunikationsschritte beim SDE-OPX am Beispiel $c = 4$, $m = 4$

Hierzu bedarf es folgender kleiner Abänderung im Verfahren: Zwischen den Knoten k und l gebe es eine Kante der Farbe c. Zu Beginn des c-ten Teilschritts müssen k und l ihre aktuellen Lastwerte w_k und w_l austauschen. Danach berechnen beide Prozessoren nicht nur ihren eigenen Lastwert neu sondern auch den ihres Nachbarn. Dadurch entfällt die Kommunikation zu Beginn des nächsten Teilschrittes, dem wiederum die Farbe c zugeordnet ist. Analog wird für Farbe 1 vorgegangen. Der Vorteil dieser Technik macht sich am stärksten bei Graphen mit nur zwei Farben bemerkbar, da in diesem Fall nur ein einziger Kommunikationsschritt mehr gebraucht wird verglichen mit DE-OPX.

Wie in den Abschnitten 3.7.1 und 3.7.2 bereits angedeutet, stimmt der Aufwand von DE-OPXfb mit SDE-OPX überein und für DE-OPXcc ergeben sich $c(m-1)+c-1$ Schritte.

Die direkte Zuordnung der Kommunikationsschritte zu den Farben ergibt in einigen Fällen bloß eine obere Schranke für den tatsächlichen Zeitbedarf. Dies betrifft solche Graphen bei denen die Anzahl der Farben größer ist als der maximale Knotengrad. Betrachte als Beispiel hierzu Abbildung 3.13. Durch Vermischung der zu den Farben gehörenden Teilschritte sind für 5 Schritte (Eigenwerte) nicht $5 \cdot 3 = 15$ sonder nur 11 Kommunikationsschritte notwendig. Diese Beobachtung wurde bereits in [XL95] beschrieben.

Nun wird die Situation für beliebiges n betrachtet Jeder Prozessor muss pro Eigenwert mit zwei Nachbarn kommunizieren; insgesamt führt dies auf $n(m-1)$ Kommunikationen. Wie die Abbildung verdeutlicht, finden in jedem Kommunikationsschritt $\frac{n-1}{2}$ Kommunikationen statt. Die Gesamtzahl der Kommunikationsschritte ist daher $\left\lceil \frac{2n}{n-1}(m-1) \right\rceil$ und nicht $3(m-1)$, wie obige Formel vermuten ließe. Ähnliche Überlegungen ergeben für den Torus $T_n^{[6]}$, dass nicht $6(m-1)$ sondern nur $\left\lceil \frac{4n}{n-1}(m-1) \right\rceil$ Kommunikationsschritte anfallen.

Dieselben Überlegungen gelten für die Diffusionsverfahren im One-Port-Kommunikationsmodell.

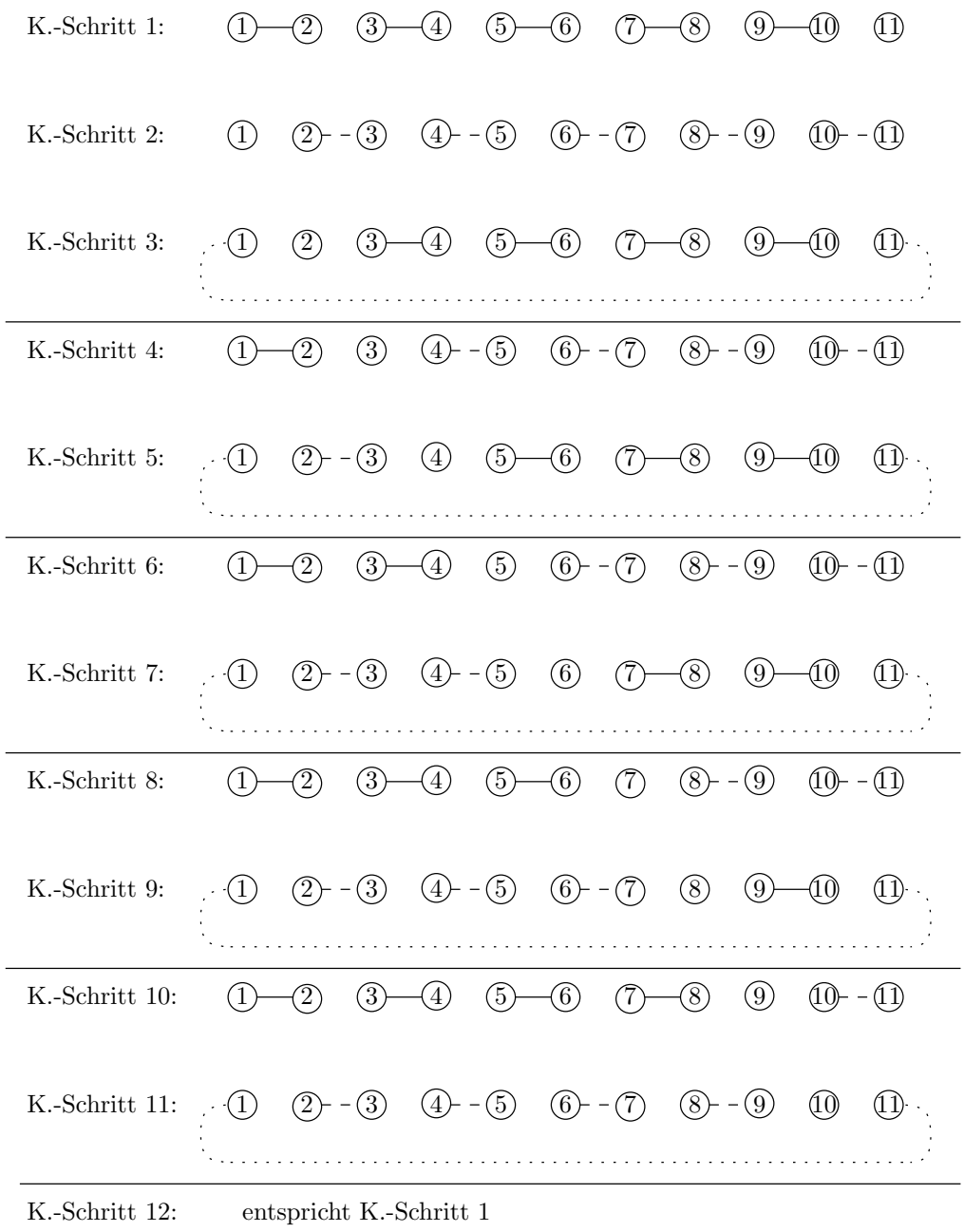


Abbildung 3.13: Ablauf des DE-OPX beim Zyklus C_{11}; dargestellt sind jeweils nur die Kanten, über die im jeweiligen Kommunikationsschritt Daten ausgetauscht werden

3.9.1 Eine Verbesserung bei Diffusionsverfahren: FB-OPT

Zerlegt man bei Diffusionsverfahren die kompletten Schritte in Teilschritte, so lässt sich ausnutzen, dass die Reihenfolge der Teilschritte hier völlig beliebig ist. Setzt man voraus, dass überhaupt eine Einfärbung für den Graphen bestimmt ist, dann kann man die Farben, ähnlich wie beim SDE-OPT, abwechselnd vorwärts und rückwärts durchlaufen. Auf diese Weise lassen sich wiederum aufeinander folgende Kommunikationsoperationen über Kanten der selben Farbe zu einer Kommunikation zusammenfassen. Diese effizienzsteigernde Abwandlung wird im folgenden mit FB-OPT usw. bezeichnet.

3.9.2 Zusammenfassender Vergleich

In Tabelle 3.4 werden die Laufzeiten der Diffusions- und Dimension-Exchange-Verfahren verglichen. Hierbei wird davon ausgegangen, dass der verwendete Parallelrechner nur *One-Port-Kommunikation* zulässt. Die einzelnen Schritte der Diffusionsverfahren müssen dann ebenfalls in Teilschritte zerlegt werden, wobei hierfür stets die Einfärbung verwendet wird, die auf die geringste Laufzeit führt. Bei den Dimension-Exchange-Verfahren wird jeweils nur der Fall $\alpha = \frac{1}{2}$ betrachtet. Die Tabelle zeigt, dass DE-OPX und auch DE-OPXcc für viele, wenn auch nicht alle Graphen, schneller sind als OPX. Mit der neuen Diffusionsart FB-OPX erreicht man zumindest bei den eindimensionalen Graphen vergleichbar gute Ergebnisse. Der folgende Satz gibt eine untere Schranke für die Anzahl der Schritte.

Satz 3.46. *Die Anzahl der Kommunikationsschritte bei One-Port-Kommunikation eines beliebigen Loadbalancing-Verfahrens auf einem Graphen G ist im Worst Case mindestens so groß wie der Durchmesser* $\operatorname{diam}(G)$ *des Graphen.*

Beweis. Es seien k und l zwei Knoten des Graphen mit Abstand $\operatorname{diam}(G)$. Betrachte den Fall, dass anfänglich die gesamte Last ausschließlich auf Knoten k vorliegt. Dann sind mindestens $\operatorname{diam}(G)$ Kommunikationsschritte nötig, bis sich der Lastwert des Knotens l zum ersten Mal ändert. □

Im Sinne obigen Satzes sind die DE-OPX-Verfahren (ggf. bis auf eine konstante Zahl von ein oder zwei Kommunikationsschritten) optimal für den Pfad, den Ring gerader Länge und den Hypercube.

3.10 Stabilität der Dimension-Exchange-Verfahren

Die Dimension-Exchange-Verfahren sind den Diffusionsverfahren nicht nur im Hinblick auf die Laufzeiten überlegen sondern auch bezüglich der numerischen Stabilität. Die Beobachtungen zum Einfluss der Reihenfolge der Eigenwerte beim OPT in Abschnitt 2.5 übertragen sich direkt auf die DE- und SDE-Varianten. Kleine Fehler am Ende der Rechnung ergeben sich nur unter Verwendung der Leja-Sortierungen. Komplexe Eigenwerte werden hierbei bezüglich ihrer Beträge sortiert. Wählt man α nahe 0, dann sind Diffusions- und Dimension-Exchange-Verfahren nahezu identisch, vgl. Lemma 3.22. Entsprechend unterscheiden sich die Diagramme zur Konvergenz auch nicht wesentlich. So

Graph		diam	OPX	FB-OPX	SDE-OPX	DE-OPX	DE-OPXcc
P_n	$2 \mid n$	$n-1$	$2n-2$	n	$n+1$	n	$n+1$
	$2 \nmid n$	$n-1$	$2n-2$	n	$n+2$	$n+1$	$n+2$
C_n	$4 \mid n$	$\frac{1}{2}n$	n	$\frac{1}{2}n+1$	$\frac{1}{2}n+1$	$\frac{1}{2}n$	$\frac{1}{2}n+1$
	$2 \mid n, 4 \nmid n$	$\frac{1}{2}n$	n	$\frac{1}{2}n+1$	$\frac{1}{2}n+2$	$\frac{1}{2}n+1$	$\frac{1}{2}n+2$
	$2 \nmid n$	$\frac{1}{2}n-\frac{1}{2}$	n	n	$2n+3$	$n+2+\left\lceil\frac{2}{n-1}\right\rceil$	$n+4+\left\lceil\frac{2}{n-1}\right\rceil$
$C_n^{[3]}$	$3 \mid n, 2 \nmid n$	$\frac{1}{2}n-\frac{1}{2}$	n	n	$\frac{4}{3}n+5$	$2n$	$2n+2$
$C_n^{[5]}$	$5 \mid n, 2 \nmid n$	$\frac{1}{2}n-\frac{1}{2}$	n	n	$\frac{16}{5}n+17$	$4n$	$4n+4$
$C_n^{[5b,c]}$	$5 \mid n, 2 \nmid n$	$\frac{1}{2}n-\frac{1}{2}$	n	n	$\frac{6}{5}n+10$	$\frac{9}{5}n$	$\frac{9}{5}n+2$
G_k	$2 \mid k$	$2k-2$	$2k^2$	$\frac{3}{2}k^2+1$	$\frac{3}{4}k^2+\frac{3}{2}k+1$	$\frac{1}{2}k^2+k$	$\frac{1}{2}k^2+k+3$
	$2 \nmid k$	$2k-2$	$2k^2+2$	$\frac{3}{2}k^2+\frac{5}{2}$	$\frac{3}{4}k^2+3k+\frac{13}{4}$	$\frac{1}{2}k^2+2k+\frac{3}{2}$	$\frac{1}{2}k^2+2k+\frac{9}{2}$
T_k	$4 \mid k$	k	$\frac{1}{2}k^2+2k$	$\frac{3}{8}k^2+\frac{3}{2}k+1$	$\frac{3}{16}k^2+\frac{3}{4}k+1$	$\frac{1}{8}k^2+\frac{1}{2}k$	$\frac{1}{8}k^2+\frac{1}{2}k+3$
	$2 \mid k, 4 \nmid k$	k	$\frac{1}{2}k^2+2k+2$	$\frac{3}{8}k^2+\frac{3}{2}k+\frac{5}{2}$	$\frac{3}{16}k^2+\frac{3}{2}k+\frac{13}{4}$	$\frac{1}{8}k^2+k+\frac{3}{2}$	$\frac{1}{8}k^2+k+\frac{9}{2}$
	$2 \nmid k$	$k-1$	$f(k)$	$\frac{1}{2}k^2+2k-\frac{3}{2}$	$2k^2+4k+3$	$\frac{5}{4}k^2+\frac{5}{2}k+\frac{5}{4}$	$\frac{5}{4}k^2+\frac{5}{2}k+\frac{21}{4}$
$T_k^{[6]}$	$2 \nmid k$	$k-1$	$f(k)$	$\frac{1}{2}k^2+2k-\frac{3}{2}$	$\frac{5}{4}k^2+5k+\frac{19}{4}$	$\frac{1}{2}k^2+\frac{5}{2}k+8+\left\lceil\frac{8}{k-1}\right\rceil$	$\frac{1}{2}k^2+\frac{5}{2}k+13+\left\lceil\frac{8}{k-1}\right\rceil$
$T_k^{[C3]}$	$2 \nmid k$	$k-1$	$f(k)$	$\frac{1}{2}k^2+2k-\frac{3}{2}$	$\frac{5}{9}k^2+\frac{10}{3}k+16$	$\frac{4}{3}k^2+2k$	$\frac{4}{3}k^2+2k+5$
H_d		d	d^2	d^2-d+1	$2d-1$	d	$2d-1$
S_n		2	$2n-2$	$2n-3$	$2n^2-6n+4$	n^2-2n+1	n^2-n-1

Tabelle 3.4: Anzahl der Kommunikationsschritte bei Standardgraphen und verschiedenen Verfahren; bei allen Dimension-Exchange-Verfahren wird $\alpha = \frac{1}{2}$ angenommen; zum Vergleich ist jeweils der Durchmesser diam der Graphen mit angegeben; $(f(k) = \min\left\{\frac{5}{8}k^2+\frac{5}{2}k-\frac{25}{8}, \frac{1}{2}k^2+\frac{5}{2}k\right\})$

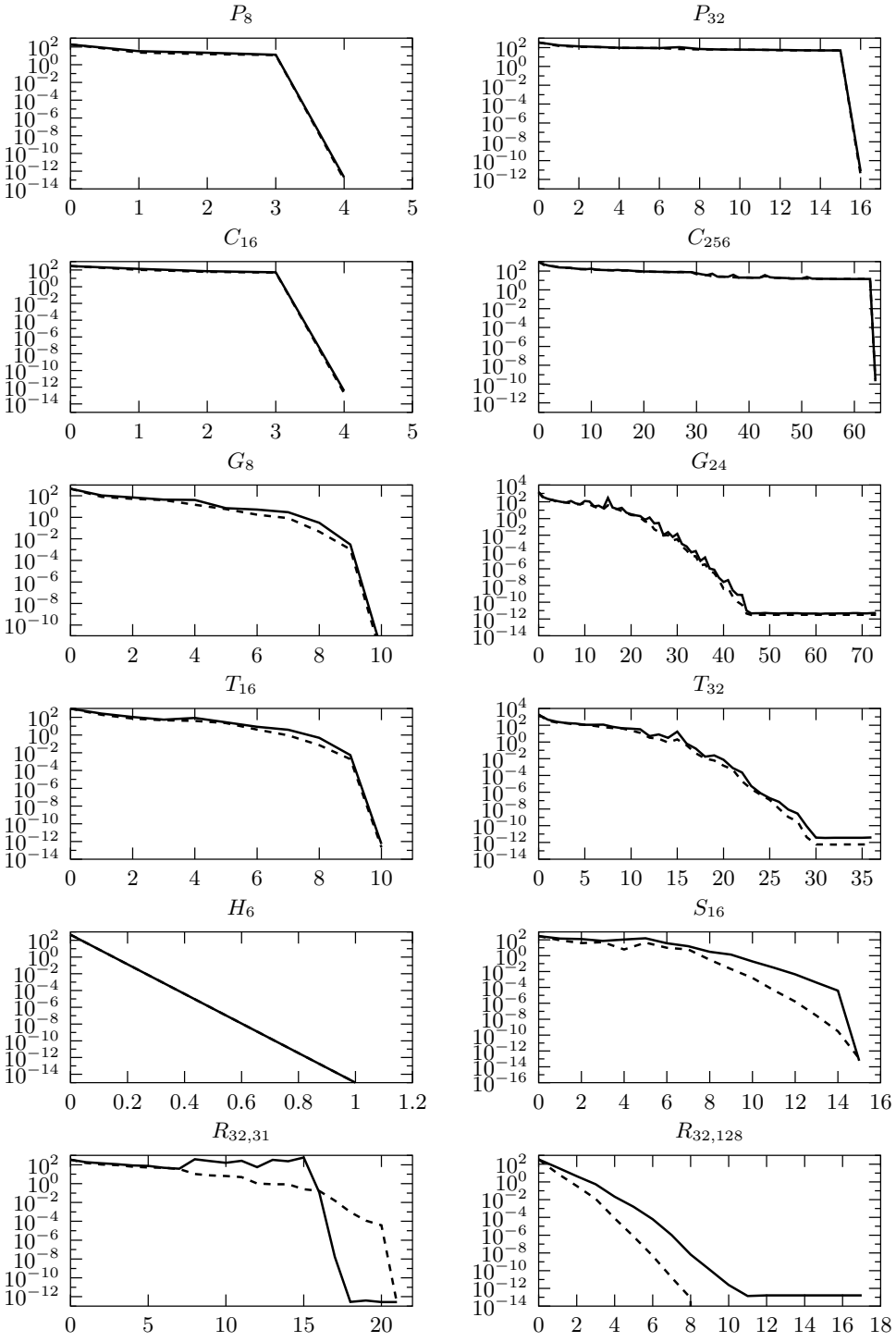


Abbildung 3.14: Vergleich von DE-OPT (durchgezogene Linie) und SDE-OPT (gestrichelt); es werden dieselben Graphen verwendet wie für die Diffusionsverfahren in Abbildung 2.4

verbessern sich die Ergebnisse für den Zufallsgraphen $R_{32,31}$ aus Abbildung 2.4 erst ungefähr ab $\alpha = 0{,}17$.

Das Konvergenzverhalten für $\alpha = \frac{1}{2}$ ist in der Abbildung 3.14 dargestellt. Bei keinem der zwölf betrachteten Graphen und zwei Verfahren sind Konvergenzprobleme festzustellen.

4 Verfahren für Produktgraphen

Das letzte Kapitel hat gezeigt, dass für einige Graphen wie Pfade und Zyklen zeitoptimale (oder annähernd optimale) Verfahren gefunden sind. Bei Gittern und Tori dagegen unterscheiden sich der Durchmesser des Graphen ($\mathcal{O}(n)$) und die Laufzeit ($\mathcal{O}(n^2)$) um eine ganze Größenordnung. In diesem Kapitel werden Verfahren vorgestellt werden, die die spezielle Struktur der Graphen besser ausnutzen und die Laufzeiten damit tatsächlich in die Nähe des Durchmessers bringen. Als eine Schwierigkeit dabei wird sich das Problem herausstellen, dass die Flüsse relativ groß werden können.

Der Begriff des Produktgraphen, der hier wieder benötigt wird, ist bereits in Kapitel 1.8 eingeführt worden.

4.1 ADI-FOS

Das einfachste, wenn auch nicht sehr effiziente Diffusionsverfahren ist das FOS aus Kapitel 2.2. Hat man nun einen Produktgraphen $G = G^{(1)} \times G^{(2)}$ vorliegen, so kann man FOS abwechselnd auf den Kanten in Richtung von $G^{(1)}$ und $G^{(2)}$ ausführen. Dieses Verfahren stammt aus [EFMP99] und wird mit *ADI-FOS* bezeichnet. ADI steht hierbei für *alternating direction iteration*.

Es seien $M^{\mathsf{Diff}(1)} = I - \alpha^{(1)} L^{\mathsf{Diff}(1)}$ und $M^{\mathsf{Diff}(2)} = I - \alpha^{(2)} L^{\mathsf{Diff}(2)}$ die Diffusionsmatrizen von $G^{(1)}$ bzw. $G^{(2)}$. Ein Schritt von ADI-FOS hat für Knoten (i,j) die Form aus Algorithmus 4.1.

---

for all $e = \{(i,j),(l,j)\} \in E$ **do**
$\quad y_e^{k-1} = \alpha^{(1)} \left(w_{(i,j)}^{k-1} - w_{(l,j)}^{k-1} \right)$
end for
$w_{(i,j)}^{k-\frac{1}{2}} = w_{(i,j)}^{k-1} - \sum_{e=\{(i,j),(l,j)\} \in E} y_e^{k-1}$
for all $e = \{(i,j),(i,l)\} \in E$ **do**
$\quad y_e^{k-1} = \alpha^{(2)} \left(w_{(i,j)}^{k-\frac{1}{2}} - w_{(i,l)}^{k-\frac{1}{2}} \right)$
end for
$w_{(i,j)}^{k} = w_{(i,j)}^{k-\frac{1}{2}} - \sum_{e=\{(i,j),(i,l)\} \in E} y_e^{k-1}$

Algorithmus 4.1: Schritt k des ADI-FOS-Verfahrens für Prozessor i

In Matrixschreibweise ergibt sich der sehr kurze Algorithmus 4.2.

for $k = 1, 2, \ldots$ **do**
  $w^k = M^{\mathsf{Diff}(2)} \otimes M^{\mathsf{Diff}(1)} w^{k-1}$
end for

Algorithmus 4.2: ADI-FOS-Verfahren in Matrixnotation

In [EFMP99] wird gezeigt, dass im Falle $G^{(1)} = G^{(2)}$ ADI-FOS nur halb so viele Iterationen wie FOS benötigt, um die selbe Fehlerschranke zu erreichen. Die Anzahl der benötigten Iterationen und die Höhe des Flusses hängen beide vom Parameter α ab. Minimale Flüsse ergeben sich, wenn man α gegen Null laufen lässt, wobei die Anzahl der Iteratiosschritte sehr groß werden kann. Wählt man α so, dass die Laufzeit minimiert wird, können sich andererseits recht hohe Flüsse ergeben.

Das ADI-FOS kann man auch auffassen als eine Kombination von Diffusions- und Dimension-Exchange-Verfahren, also von FOS und DE-FOS [EMP00]. Zur Vereinfachung sei $M^{\mathsf{Diff}(2)} = M^{\mathsf{Diff}(1)}$. Dann ist $M^{\mathsf{Diff}(1)} = I - \alpha L^{\mathsf{Diff}(1)}$, $L^{\mathsf{Diff}(1)} = A^{(1)} A^{(1)^T}$ und

$$\begin{aligned} M^{\mathsf{ADI}} &= M^{\mathsf{Diff}(1)} \otimes M^{\mathsf{Diff}(1)} = \left(M^{\mathsf{Diff}(1)} \otimes I\right)\left(I \otimes M^{\mathsf{Diff}(1)}\right) \\ &= \left(I - \alpha L^{\mathsf{Diff}(1)} \otimes I\right)\left(I - \alpha I \otimes L^{\mathsf{Diff}(1)}\right). \end{aligned}$$

Färbe die Kanten nun so mit 2 Farben, dass alle Kanten einer Richtung dieselbe Farbe besitzen. (Dies ist keine Einfärbung im bisherigen Sinne!) Bezeichnet man die beiden Inzidenzmatrizen der Teilgraphen einer Farbe mit A_1 bzw. A_2, so ist $A_1 = \left(I \otimes A^{(1)} \mid 0\right)$ und $A_2 = \left(0 \mid A^{(1)} \otimes I\right)$. Mit $L_1 = A_1 A_1^T = I \otimes L^{\mathsf{Diff}(1)}$ und $L_2 = L^{\mathsf{Diff}(1)} \otimes I$ sowie $M_i = I - \alpha L_i$ erhält man schließlich

$$M^{\mathsf{Diff/DE}} = M_2 M_1 = \left(I - \alpha L^{\mathsf{Diff}(1)} \otimes I\right)\left(I - \alpha I \otimes L^{\mathsf{Diff}(1)}\right) = M^{\mathsf{ADI}}.$$

Diese Gleichheit kann man nun ausnutzen, um mit Hilfe von Satz 3.35 eine Formel für den Fluss des Verfahrens zu bestimmen.

Korollar 4.1. *Von ADI-Verfahren wird bei Anfangslast w^0 folgender Fluss x berechnet:*

$$x = A^{\mathsf{ADI}^T} L^{\mathsf{ADI}^+} w^0$$

mit $M^{\mathsf{ADI}} = I - L^{\mathsf{ADI}}$, $L^{\mathsf{ADI}} = A A^{\mathsf{ADI}^T}$ und $A^{\mathsf{ADI}} = A_1 + M_1 A_2$.

MDI-FOS

Zur Verkleinerung der Flüsse wird in [EFMP99] als Variante des Verfahrens *MDI-FOS (mixed direction iteration)* eingeführt. Hierbei bleiben alle Schritte mit ungeradem k unverändert, in den anderen wird mit den Kanten der zweiten Richtung begonnen und die anderen werden danach benutzt. Formal verwendet man im Wechsel die Matrizen $M^{\mathsf{Diff}(2)} \otimes M^{\mathsf{Diff}(1)}$ und $M^{\mathsf{Diff}(1)} \otimes M^{\mathsf{Diff}(2)}$. Diese Variante kann man Auffassen als Mischform von Diffusion und symmetrischem Dimension-Exchange (SDE-FOS). Aus Satz 3.35 ergibt sich hiermit das folgende Korollar.

Korollar 4.2. *Von MDI-Verfahren wird bei Anfangslast w^0 folgender Fluss x berechnet:*

$$x = A^{\mathsf{MDI}^T} L^{\mathsf{MDI}^+} w^0$$

mit $A^{\mathsf{MDI}} = A_1 + M_1 A_2 + M_1 M_2 A_2 + M_1 M_2 M_2 A_1$ *und* $L^{\mathsf{MDI}} = A A^{\mathsf{MDI}^T}$.

ADC-FOS

Als weitere Variante kann man das zyklische Durchlaufen der Farben des DE-OPTcc aus Abschnitt 3.7.2 auf die verschiedenen Richtungen übertragen. Man beginnt hier also einmal mit Richtung eins und einmal mit Richtung zwei und bildet am Ende den Mittelwert beider errechneter Flüsse. Dieses neue Verfahren wird mit ADC-FOS bezeichnet *(alternating direction cycling)*. Ähnlich wie beim DE-OPTcc können auch hier beide Rechnungen gleichzeitig, nur um einen Halbschritt versetzt stattfinden, wodurch sich der Kommunikationsaufwand im Vergleich zu ADI-FOS nur wenig erhöht. Analog zum DE-OPTcc verwendet man nun die Matrizen $A^{\mathsf{ADC}} = \frac{1}{2}(A_1 + M_1 A_2 + A_2 + M_2 A_1)$ und $L^{\mathsf{ADC}} = A A^{\mathsf{ADC}^T}$ zur Darstellung der Flüsse gemäß Satz 3.35.

Korollar 4.3. *Von ADC-Verfahren wird bei Anfangslast w^0 folgender Fluss x berechnet:*

$$x = A^{\mathsf{ADC}^T} L^{\mathsf{ADC}^+} w^0$$

Experimenteller Vergleich der Varianten

Eine experimentelle Untersuchung zeigt, dass die Normen $\left\| A^{\mathsf{XXX}^T} L^{\mathsf{XXX}^+} A \right\|_2$, $\mathsf{XXX} \in \{\mathsf{ADI}, \mathsf{MDI}, \mathsf{ADC}\}$ und damit die Schranken für die Flüsse bei (quadratischen) Tori für festes α unabhängig von der Größe sind; die Schranken für Gitter nähern sich denen des Torus mit wachsender Dimension an. Leider wächst aber das optimale α mit der Größe des Graphen, so dass die Flüsse im jeweils laufzeitoptimalen Fall doch deutlich anwachsen, bei MDI allerdings schwächer als bei ADI, vergleiche hierzu auch Abbildung 4.1. Es fällt auf, dass sich hier recht hohe Schranken ergeben, obwohl es sich um eine Mischung aus normalem Diffusionsverfahren und Dimension-Exchange handelt, wobei das eine minimale Flüsse erzeugt und das andere zumindest moderatere Normen produziert, vergleiche Abbildung 3.11. In der Praxis bleiben die Flüsse meist weit unter diesen Schranken. Erst bei der ADC-Variante ergeben sich Schranken, die in ähnlichen Bereichen liegen wie die der normalen Diffusionsverfahren.

Das ADI-FOS-Verfahren ist natürlich nicht auf zwei Dimensionen beschränkt. Daher sei nun der d-dimensionale Graph $G = G^{(1)} \times \cdots \times G^{(d)}$ gegeben mit $G^{(i)} = \left(V^{(i)}, E^{(i)}\right)$, $\left|V^{(i)}\right| = n^{(i)}$ und $\left|E^{(i)}\right| = N^{(i)}$. Der zugehörige Algorithmus 4.3 zeigt zusätzlich die Aufdatierung des Flusses. Beim MDI-FOS werden alle Dimensionen abwechselnd vorwärts und rückwärts durchlaufen.

4.2 SDI-Verfahren

In den nachfolgenden Abschnitten werden weitere Verfahren besprochen, die das Prinzip der alternierenden Richtungen mehr oder weniger aufwändig mit einfachen Loadbalan-

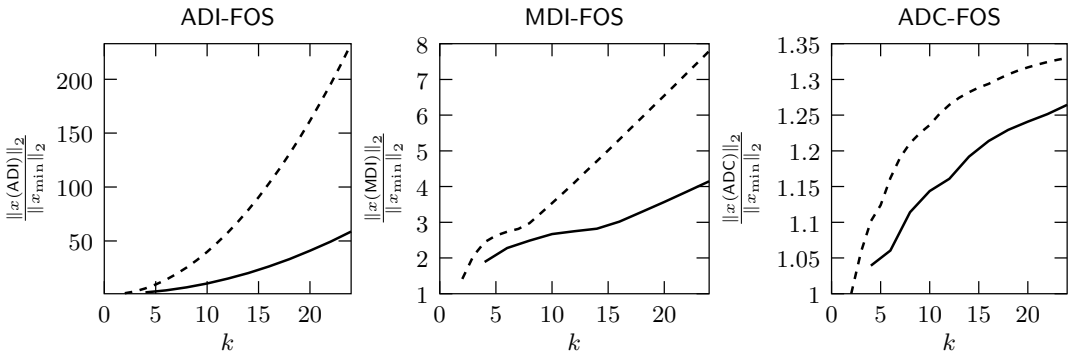


Abbildung 4.1: Schranken $\left\|A^{\mathsf{XXX}^T} L^{\mathsf{XXX}^+} A\right\|_2$ für die Normen der Flüsse bei ADI-FOS, MDI-FOS und ADC-FOS für Gitter G_k (gestrichelt) und Tori T_k (durchgezogene Linie) verschiedener Größe mit laufzeitoptimiertem α

$x = 0$
for $k = 1, 2, \ldots$ **do**
  $\tilde{w}^0 = w^{k-1}$
  for $l = 1, \ldots, d$ **do** *{Dimensionsschleife}*
    $\tilde{w}^l = \left(I_{n^{(d)}\ldots n^{(l+1)}} \otimes M^{\mathsf{Diff}^{(l)}} \otimes I_{n^{(l-1)}\ldots n^{(1)}}\right) \tilde{w}^{l-1}$
    $x = x + \alpha \left(I_{n^{(d)}\ldots n^{(l+1)}} \otimes {A^{(l)}}^T \otimes I_{n^{(l-1)}\ldots n^{(1)}}\right) \tilde{w}^{l-1}$
  end for
  $w^k = \tilde{w}^d$
end for

Algorithmus 4.3: ADI-FOS-Verfahren für mehrdimensionale Graphen

cing-Verfahren kombinieren. Eine Möglichkeit, die grundsätzlich sehr leicht zu implementieren ist, ist die einzelnen Richtungen komplett hintereinander zu durchlaufen. Erst wenn die Last in allen parallelen Graphen der ersten Richtung ausgeglichen ist, wird mit der nächsten Richtung begonnen, usw. Hierbei macht es weder Probleme, dass Verfahren wie SOS oder OPS auf die letzten *beiden* Iterierten zugreifen, noch die einzelnen Faktorgraphen verschiedene und unterschiedlich viele Eigenwerte haben können. Nachteil ist, dass die so produzierten Flüsse meist deutlich höher sind als die alternierend berechneten. Diese Verfahrensklasse wird als weniger wichtig angesehen. Wenn auf sie zurückgegriffen wird, wird das Präfix SDI- (*sequential directions*) verwendet.

4.3 ADI-OPT

Kombiniert man die Technik der alternierenden Richtungen mit dem OPT-Verfahren, so erhält man das in [EFMP99] vorgestellte ADI-OPT. Es sei $G = G^{(1)} \times G^{(1)}$, $L^{\mathsf{Diff}^{(1)}}$ die Laplace-Matrix von $G^{(1)}$, und mit $\lambda_1^{(1)}, \ldots, \lambda_m^{(1)}$ werden die Eigenwerte von $L^{\mathsf{Diff}^{(1)}}$

bezeichnet. Wie in Algorithmus 4.4 zu sehen ist, wird das auf den Eigenwerten des Graphen $G^{(1)}$ basierende OPT abwechselnd auf die beiden Richtungen von G angewandt. So erhält man nach $m-1$ Iterationsschritten die Gleichverteilung $\overline{w} = w^{m-1}$. Das einfache OPT-Verfahren würde dagegen deutlich länger dauern, da die Laplace-Matrix von G bis zu $\frac{m(m+1)}{2}$ verschiedene Eigenwerte haben kann.

---

for $k = 1, \ldots, m-1$ **do**

$$w^k = \left(I - \frac{1}{\lambda_{k+1}^{(1)}} L^{\mathsf{Diff}^{(1)}} \otimes I\right)\left(I - \frac{1}{\lambda_{k+1}^{(1)}} I \otimes L^{\mathsf{Diff}^{(1)}}\right) w^{k-1}$$

end for

Algorithmus 4.4: ADI-OPT-Verfahren, zweidimensional

Natürlich ist auch ADI-OPT nicht auf zwei Dimensionen beschränkt und die Graphen der verschiedenen Richtungen müssen auch nicht identisch sein. Bei verschiedenen Graphen $G^{(i)}$ muss man lediglich vorab d Sätze von Eigenwerten bestimmen. In Algorithmus 4.5 wird im ersten Schritt mit allen Richtungen begonnen, Richtungen mit weniger Eigenwerten sind daher früher beendet als andere. Als Bezeichnung hierfür wird VDI-OPT benutzt *(variable directions)*.

---

for $k = 1, \ldots, \max\left\{m^{(1)}, \ldots, m^{(d)}\right\} - 1$ **do**

$\tilde{w}^0 = w^{k-1}$

for $l = 1, \ldots, c$ **do** *{Dimensionsschleife}*

if $k < m^{(l)}$ **then**

$$\tilde{w}^l = \left(I_n - \frac{1}{\lambda_{k+1}^{(l)}} I_{n^{(d)}\ldots n^{(l+1)}} \otimes L^{\mathsf{Diff}^{(l)}} \otimes I_{n^{(l-1)}\ldots n^{(1)}}\right) \tilde{w}^{l-1}$$

$$x = x + \frac{1}{\lambda_{k+1}^{(l)}} \left(I_{n^{(d)}\ldots n^{(l+1)}} \otimes {A^{(l)}}^T \otimes I_{n^{(l-1)}\ldots n^{(1)}}\right) \tilde{w}^{j-1}$$

else

$\tilde{w}^l = \tilde{w}^{l-1}$

end if

end for

$w^k = \tilde{w}^d$

end for

Algorithmus 4.5: VDI-OPT-Verfahren, d-dimensional

In Kapitel 2.5 wurde festgestellt, dass die numerische Stabilität des OPT-Verfahrens von der Reihenfolge der Eigenwerte λ_i abhängt. Diese Eigenschaft überträgt sich direkt auf ADI-OPT, wobei die Fehler dank der geringeren Anzahl von Rechenschritten weniger groß werden, vgl. Abbildung 4.2. Verwendet man die auf Leja-Punkten beruhende Sortierung, so läuft ADI-OPT selbst auf den großen Gittern G_{32} und G_{64} noch stabil.

Wie beim ADI-FOS sind auch hier die berechneten Flüsse nicht mehr l_2-minimal. Zusätzlich hängen sie nun noch entscheidend von der Reihenfolge der Eigenwerte ab. Wie die experimentellen Beispiele in Tabelle 4.1 zeigen, erhält man die geringsten Flusswerte bei absteigender Sortierung. Aus Gründen der besseren numerischen Stabilität

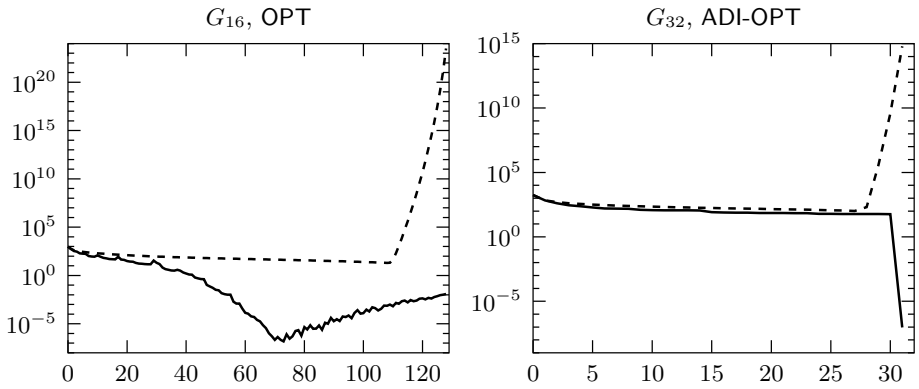


Abbildung 4.2: Stabilitätsvergleich von OPT (links, für G_{16}) und ADI-OPT (rechts, für G_{32}) bei aufsteigender (gestrichelt) und Leja$^{(1)}$-Sortierung (durchgezogene Linie)

ist eine Leja-Sortierung mit passender Gewichtsfunktion aber wiederum vorzuziehen. Wie Abbildung 4.3 zeigt, hat der Exponent g der Gewichtsfunktion $\omega(z) = |z|^g$ bei Leja-Sortierungen zweierlei Auswirkungen. Mit steigendem g verringert sich der Fluss, während der abschließende Fehler ansteigt. Werte um $1,5$ haben sich praktisch als guter Kompromiss erwiesen. Zur weiteren Flussreduzierung empfehlen sich die schon vom ADI-FOS bekannten Varianten MDI-OPT (durchlaufe die Richtungen abwechselnd vorwärts und rückwärts [EFMP99]) und ADC-OPT (mehrere jeweils mit einer anderen Richtung startende Durchläufe und Mittelwertbildung).

		$P_6 \times P_6 = G_6$	$C_4 \times C_4 = T_4$	$C_8 \times C_8 = T_8$	$C_{32} \times C_{32} = T_{32}$	$C_8 \times C_8 \times C_8$
ADI-OPT	aufst.	527	1,48	34,2	–	82,3
ADI-OPT	abst.	1,012	1,049	1,046	1,024	1,13
ADI-OPT	Young	6,61	1,48	1,62	1,89	2,08
ADI-OPT	Leja$^{(0)}$	3,83	1,049	1,98	353	2,18
ADI-OPT	Leja$^{(1)}$	1,013	1,049	1,056	1,041	1,14
MDI-OPT	Leja$^{(1)}$	1,008	1,031	1,033	1,025	1,10
ADC-OPT	Leja$^{(1)}$	0,996	0,998	0,998	0,999	1,0003
SDI-OPT	Leja$^{(1)}$	1,12	1,16	1,39	2,05	1,84

Tabelle 4.1: Die Zahlen geben an, um welchen Faktor die l_2-Normen der Flüsse bei einigen Beispielen über dem Minimum liegen. (Werte < 1 sind auf die Rundung auf ganze Zahlen zurückzuführen, siehe auch Abschnitt 5.4.)

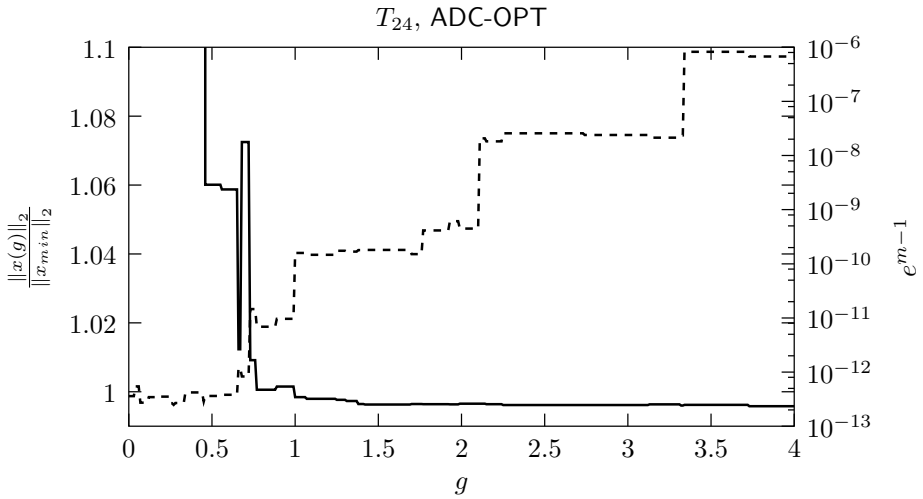


Abbildung 4.3: Abhängigkeit des ADC-OPT-Verfahrens mit Leja$^{(g)}$-Sortierung vom Exponenten g der Gewichtsfunktion $\omega(z) = |z|^g$. Dargestellt ist, um welchen Faktor der Fluss über dem Minimum liegt (durchgezogene Linie, linke Skala) und der abschließende Fehler e^{m-1} (gestrichelte Linie, rechte Skala)

4.4 ADI-SOS und ADI-Čebyšev

Als erstes Verfahren, das in jedem Schritt auf die zwei letzten Iterierten zurückgreift, wird das ADI-SOS betrachtet. Hierzu werden zwei verschiedene Möglichkeiten der Umsetzung betrachtet. Die erste stammt aus [Els02], die zweite und neue, hier zur Unterscheidung mit ADI-SOS2 bezeichnete, ist etwas aufwändiger, wird sich dafür aber im nächsten Abschnitt auf das OPS übertragen lassen. Zur Vereinfachung der Darstellung wird hier nur ein einfaches „quadratisches“ Produkt $G = G^{(1)} \times G^{(1)}$ betrachtet. Eine Übertragung auf verschiedene Faktorgraphen in $G = G^{(1)} \times G^{(2)}$ ist aber möglich. Zur Vorabbestimmung der Anzahl benötigter Schritte muss man dann die „langsamste“ Richtung betrachten. Weitere Hinweise zum allgemeinen Fall werden an den jeweiligen Stellen gegeben.

ADI-SOS

Das ADI-SOS aus [Els02] beruht auf der Iteration

$$w^1 = \left(M^{(1)} \otimes M^{(1)}\right) w^0$$
$$w^k = \beta \left(M^{(1)} \otimes M^{(1)}\right) w^{k-1} + (1-\beta)\, w^{k-2} \qquad k = 2, 3, \ldots$$

und ist ausführlicher in Algorithmus 4.6 dargestellt.

In unserem Spezialfall entspricht der optimale Wert für β dem Wert $\beta_{opt}^{(1)}$ des Faktorgraphen und die Anzahl der Iterationen verringert sich gegenüber dem normalen SOS um den Faktor $\sqrt{2}$ [Els02]. (Bei nicht-quadratischen Produkten nimmt man $\beta_{opt}^{(i)}$, wobei

$x = 0$
$\tilde{w}^1 = \left(I \otimes M^{(1)}\right) w^0$
$x = x + \alpha \left(I \otimes A^{(1)^T}\right) w^0$
$w^1 = \left(M^{(1)} \otimes I\right) \tilde{w}^1$
$x = x + \alpha \left(M^{(1)} \otimes I\right) \tilde{w}^1$
for $k = 2, 3, \ldots$ **do**
  $\tilde{w}^0 = w^{k-1}$
  $\tilde{x}^0 = x^{k-1}$
  $\tilde{w}^1 = \left(I \otimes M^{(1)}\right) \tilde{w}^0$
  $\tilde{x}^1 = \tilde{x}^0 + \alpha \left(I \otimes A^{(1)^T}\right) \tilde{w}^0$
  $\tilde{w}^2 = \left(M^{(1)} \otimes I\right) \tilde{w}^1$
  $\tilde{x}^2 = \tilde{x}^1 + \alpha \left(M^{(1)} \otimes I\right) \tilde{w}^1$
  $w^k = \beta \tilde{w}^2 + (1 - \beta) w^{k-2}$
  $x^k = \beta \tilde{x}^2 + (1 - \beta) x^{k-2}$
end for

Algorithmus 4.6: ADI-SOS-Verfahren

i die Richtung ist, deren zweitgrößter Eigenwert γ maximal ist.)

Wie im Abschnitt 4.1 zu ADI-FOS beschrieben, kann man auch hier die Varianten MDI-SOS und ADC-SOS konstruieren. Da die jeweiligen FOS- und SOS-Varianten polynomial in derselben Iterationsmatrix ($M^{(1)} \otimes M^{(1)}$ für ADI) sind erzeugen sie auch jeweils dieselben Flüsse und es gelten auch für diese Verfahren zweiter Ordnung die Flussabschätzungen aus Abschnitt 4.1.

ADI-SOS2

Das nun einzuführende ADI-SOS2 ist ausschließlich für zwei Richtungen konzipiert und benötigt zusätzliche Zwischenwerte. Zu diesem Zweck werden die verschiedenen zu berechnenden Lastvektoren mit Doppelindizes bezeichnet. Zur Unterscheidung, aus welcher Richtung ein Wert berechnet wird, werden Indizes z und s (Zeile / Spalte) verwendet. Definiere nun die folgenden Lastvektoren:

$$
\begin{aligned}
w^{0,0} &= w^0 && \\
w_z^{k,1} &= \left(M^{(1)} \otimes I\right) w^{k,0} && k = 0, 1, \ldots \\
w_s^{1,l} &= \left(I \otimes M^{(1)}\right) w^{0,l} && l = 0, 1, \ldots \\
w_z^{k,l} &= \beta \left(M^{(1)} \otimes I\right) w^{k,l-1} + (1 - \beta) w^{k,l-2} && k = 0, 1, \ldots;\ l = 2, 3, \ldots \\
w_s^{k,l} &= \beta \left(I \otimes M^{(1)}\right) w^{k-1,l} + (1 - \beta) w^{k-2,l} && k = 2, 3, \ldots;\ l = 0, 1, \ldots
\end{aligned}
$$

Der folgende Satz garantiert, dass $w_z^{k,l} = w_s^{k,l}$ ist für alle k und l. Im Vorgriff auf dieses Resultat wurden in der obigen Formel auf der rechten Seite die Indizes z und s bereits weggelassen.

Satz 4.4. *Für obige Lastvektoren gilt* $w_z^{k,l} = w_s^{k,l} =: w^{k,l}$ *für* $k, l \geq 1$.

Beweis. Der Beweis wird per Induktion über $k + l$ geführt.

Induktionsanfang: $k = l = 1$:

$$\begin{aligned}
w^{1,0} &= \left(I \otimes M^{(1)}\right) w^{0,0} \\
w^{0,1} &= \left(M^{(1)} \otimes I\right) w^{0,0} \\
\Rightarrow w_z^{1,1} &= \left(M^{(1)} \otimes I\right) w^{1,0} = \left(M^{(1)} \otimes I\right)\left(I \otimes M^{(1)}\right) w^{0,0} \\
\Rightarrow w_s^{1,1} &= \left(I \otimes M^{(1)}\right) w^{0,1} = \left(I \otimes M^{(1)}\right)\left(M^{(1)} \otimes I\right) w^{0,0}
\end{aligned}$$

Wegen $\left(I \otimes M^{(1)}\right)\left(M^{(1)} \otimes I\right) = M^{(1)} \otimes M^{(1)} = \left(M^{(1)} \otimes I\right)\left(I \otimes M^{(1)}\right)$ gilt also $w_z^{1,1} = w_s^{1,1}$.

Induktionsschritt: Die Behauptung sei nun korrekt für alle k, l mit $k + l \leq c$ für ein festes c. Sei jetzt $k + l = c + 1$.

1. Fall: $k, l \geq 2$:

$$\begin{aligned}
w^{k,l-1} = w_s^{k,l-1} &= \beta \left(I \otimes M^{(1)}\right) w^{k-1,l-1} + (1-\beta)\, w^{k-2,l-1} \\
w^{k,l-2} = w_s^{k,l-2} &= \beta \left(I \otimes M^{(1)}\right) w^{k-1,l-2} + (1-\beta)\, w^{k-2,l-2} \\
\Rightarrow w_z^{k,l} &= \beta^2 \left(M^{(1)} \otimes M^{(1)}\right) w^{k-1,k-1} \\
&\quad + \beta\,(1-\beta) \left[\left(M^{(1)} \otimes I\right) w^{k-2,l-1} + \left(I \otimes M^{(1)}\right) w^{k-1,l-2}\right] \\
&\quad + (1-\beta)^2\, w^{k-2,l-2}
\end{aligned}$$

Wie man leicht nachrechnet, erhält man für $w_s^{k,l}$ dasselbe Ergebnis.

2. und 3. Fall: $k = 1$, $l \geq 2$ bzw. $k \geq 2$, $l = 1$: Ähnlich.

□

Korollar 4.5. *Die Vektoren* $w^{k,k}$ *konvergieren gegen die Gleichverteilung* $\overline{w}$.

Beweis. Dem letzten Satz zufolge kommt es nicht darauf an, über welche Zwischenwerte $w^{i,j}$ ein Vektor $w^{k,k}$ berechnet wird. Insbesondere kann man den folgenden „Weg“ wählen, der exakt dem Verfahren SDI-SOS entspricht: $w^{0,0}, w^{0,1}, \ldots, w^{0,k}, w^{1,k}, \ldots, w^{k,k}$. Für genügend großes k wird in diesem Fall zuerst eine Gleichverteilung innerhalb aller Kopien von $G^{(1)}$ in einer Richtung mit dem gewöhnlichen SOS-Verfahren hergestellt; danach wird SOS in der anderen Richtung angewandt, was zur globalen Gleichverteilung führt. □

Wählt man den Weg aus dem letzten Beweis, so braucht man zwar keine „überflüssigen“ Zwischenwerte zu berechnen, die zugehörigen Flüsse liegen aber Experimenten zufolge ähnlich wie beim SDI-OPT weit entfernt vom Minimum. Die geringsten Flüsse erhält man bei einem „Weg durch die Mitte“. In jedem bis auf den ersten Iterationsschritt werden aus vier vorhandenen Werten $w^{k-2,k-2}$, $w^{k-1,k-2}$, $w^{k-2,k-1}$ und $w^{k-1,k-1}$ neue Werte $w^{k,k-2}$, $w^{k,k-1}$, $w^{k-1,k}$ und $w^{k,k}$ berechnet, vergleiche Abbildung 4.4. Die Anzahl der Kommunikationsoperationen erhöht sich durch die Berechnung zusätzlicher Zwischenwerte jedoch nicht. Die Daten, die zur Berechnung von $w^{k,k-2}$ und $w^{k,k-1}$ benötigt werden, können gemeinsam verschickt werden, ebenso die Daten zur Berechnung von $w^{k-1,k}$ und $w^{k,k}$. Das gesamte Verfahren ist Algorithmus 4.7 zu entnehmen.

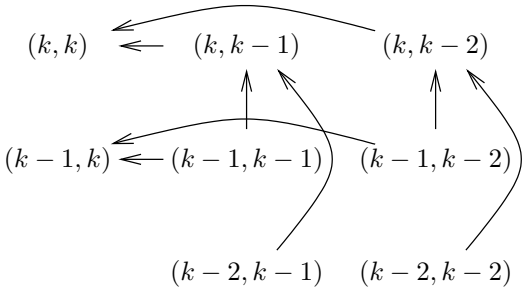


Abbildung 4.4: Ein Iterationsschritt beim ADI-SOS2

$w11 = w^0$
$w01 = \left(I \otimes M^{(1)}\right) w11$
$w10 = \left(M^{(1)} \otimes I\right) w11$
$w = \left(M^{(1)} \otimes I\right) w01$
for $k = 2, 3, \ldots$ **do**
  $w22 = w11$
  $w12 = w01$
  $w21 = w10$
  $w11 = w$
  $w02 = \beta \left(I \otimes M^{(1)}\right) w12 - (1 - \beta)\, w22$
  $w01 = \beta \left(I \otimes M^{(1)}\right) w11 - (1 - \beta)\, w21$
  $w10 = \beta \left(M^{(1)} \otimes I\right) w11 - (1 - \beta)\, w12$
  $w = \beta \left(M^{(1)} \otimes I\right) w01 - (1 - \beta)\, w02$
end for

Algorithmus 4.7: ADI-SOS2-Verfahren

Genau wie bei ADI-OPT und ADC-OPT ergeben sich auch hier kleinere Flüsse, falls das Verfahren ein zweites Mal angewandt wird und der Mittelwert beider Flüsse genommen wird. Die Iterationsvorschrift für den zweiten Durchlauf erhält man, indem man in Algorithmus 4.7 bei den Größen wXY die Indizes X und Y vertauscht sowie $M^{(1)} \otimes I$

und $I \otimes M^{(1)}$ vertauscht. Das Verfahren wird entsprechend mit ADC-SOS2 bezeichnet.

Satz 4.4 und Korollar 4.5 gelten ebenfalls für $G^{(1)} \neq G^{(2)}$. Prinzipiell ist sogar eine Übertragung auf $d > 2$ Faktoren möglich. Die Lastvektoren erhalten dann d-fache Indizes und pro Schritt müssen aus 2^d alten $d \cdot 2^{d-1}$ neue Lastwerte berechnet werden. Die Anwendung der Verfahren auf höherdimensionale Graphen ist daher nicht sinnvoll.

Vergleich

Nun soll untersucht werden, welche der beiden SOS-Versionen für Produktgraphen bessere Resultate erzielt. Da das ADI-SOS2 auf Grund der Berechnung von Zwischenwerten nicht wie die anderen Verfahren durch einfache Matrixoperationen ausgedrückt werden kann, lassen sich die üblichen Sätze zur Berechnung des Flusses nicht mehr anwenden, so dass im Gegensatz zum ADI-SOS keine theoretischen Resultate angegeben werden können. Experimente zeigen jedoch, dass die Flüsse von ADI-SOS2 nicht nur höher sind als die von ADI-SOS erzeugten, sondern sogar ganz erheblich über den schon relativ hohen Flüssen von SDI-SOS liegen. In der Konvergenzgeschwindigkeit unterscheiden sich beide Verfahren dagegen nicht, wie nicht nur Experimente belegen sondern im Folgenden auch gezeigt wird. Der Grund dafür, dass das insgesamt schlechtere ADI-SOS2 hier überhaupt betrachtet wird, ist der, dass sich nur dieses effizient auf OPS übertragen lässt.

Bevor die Konvergenz von ADI-SOS und ADI-SOS2 verglichen wird, werden neue Matrizen S^1 und S^2 zur Beschreibung der Verfahren eingeführt. Vernachlässigt man jeweils den ersten Schritt der Verfahren, so kann man die beiden Verfahren so formulieren, dass jeder Schritt nur aus einer Multiplikation mit einer (vergrößerten) Matrix besteht:

$$\begin{pmatrix} w^{k-1} \\ w^k \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & I \\ (1-\beta)\, I & \beta M^{(1)} \otimes M^{(1)} \end{pmatrix}}_{=:S^1} \begin{pmatrix} w^{k-2} \\ w^{k-1} \end{pmatrix}$$

bei ADI-SOS und

$$\begin{pmatrix} w^{k-1,k-1} \\ w^{k-1,k} \\ w^{k,k-1} \\ w^{k,k} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & I \\ 0 & 0 & (1-\beta)I & \beta M^{(1)} \otimes I \\ 0 & (1-\beta)I & 0 & \beta I \otimes M^{(1)} \\ (1-\beta)^2 & \beta(1-\beta) M^{(1)} \otimes I & \beta(1-\beta) I \otimes M^{(1)} & \beta^2 M^{(1)} \otimes M^{(1)} \end{pmatrix}}_{=:S^2} \begin{pmatrix} w^{k-2,k-2} \\ w^{k-2,k-1} \\ w^{k-1,k-2} \\ w^{k-1,k-1} \end{pmatrix}$$

bei ADI-SOS2.

Satz 4.6. *Die Parameter α und β seien optimal gewählt, γ bezeichne den zweitgrößten Eigenwert von $M^{(1)}$. Dann stimmt der zweitgrößte Eigenwert von S^1 mit dem von S^2 überein und für diesen gilt $\gamma_S = \frac{1-\sqrt{1-\gamma^2}}{\gamma}$.*

Beweis. Die Eigenwerte und Eigenvektoren von $M^{(1)}$ werden mit μ_i und z_i bezeichnet, also $M^{(1)} z_i = \mu_i z_i$. Wie üblich sei $\mu_1 = 1 > \mu_2 \geq \cdots \geq \mu_n$ und $z_1 = \overline{w}$. Sind α und β optimal gewählt, dann ist $\gamma = \mu_2 = -\mu_n$ und $\beta = \frac{2}{1+\sqrt{1-\gamma^2}}$.

Das Eigensystem von S^1 lässt sich leicht berechnen. Es ist

$$\varphi_{i,j,\pm} = \frac{\beta \mu_i \mu_j}{2} \pm \sqrt{\frac{\beta^2 \mu_i^2 \mu_j^2}{4} + 1 - \beta}$$

ein Eigenwert zum Eigenvektor $v_{i,j,\pm} = \begin{pmatrix} z_i \otimes z_j \\ \varphi_{i,j,\pm} z_i \otimes z_j \end{pmatrix}$. Insbesondere ist $v_{1,1,+} = \begin{pmatrix} \overline{w} \\ \overline{w} \end{pmatrix}$ Eigenvektor zum Eigenwert $\varphi_{1,1,+} = 1$. Der zweitgrößte Eigenwert ist

$$\begin{aligned}
\gamma_{S^1} &= \varphi_{1,2,+} \\
&= \frac{\beta\gamma}{2} + \sqrt{\frac{\beta^2\gamma^2}{4} + 1 - \beta} \\
&= \frac{\gamma}{1+\sqrt{1-\gamma^2}} + \sqrt{\frac{\gamma^2}{\left(1+\sqrt{1-\gamma^2}\right)^2} + 1 - \frac{2}{1+\sqrt{1-\gamma^2}}} \\
&= \frac{\gamma\left(1-\sqrt{1-\gamma^2}\right)}{1-(1-\gamma^2)} + 0 \\
&= \frac{1-\sqrt{1-\gamma^2}}{\gamma}.
\end{aligned}$$

Für die Matrix S^2 erhält man ähnliche Resultate. Zum Eigenwert $(\sigma_{i,\pm}\sigma_{j,\pm})$ gehört der Eigenvektor

$$\mathfrak{v}_{i,\pm,j,\pm} = \begin{pmatrix} z_i \otimes z_j \\ \sigma_{j,\pm} z_i \otimes z_j \\ \sigma_{i,\pm} z_i \otimes z_j \\ \sigma_{i,\pm}\sigma_{j,\pm} z_i \otimes z_j \end{pmatrix}$$

mit

$$\sigma_{\iota,\pm} = \frac{\beta\mu_\iota}{2} \pm \sqrt{\frac{\beta^2\mu_\iota^2}{4} + 1 - \beta}.$$

Der zweitgrößte Eigenwert ist $\gamma_{S^2} = (\sigma_{1,+}\sigma_{2,+})$ und stimmt mit γ_{S^1} überein. □

Korollar 4.7. *Für optimal gewähltes α und β konvergieren ADI-SOS und ADI-SOS2 gleich schnell.*

Beweis. Die Konvergenzgeschwindigkeit wird jeweils bestimmt durch den zweitgrößten Eigenwert der Matrizen S^1 bzw. S^2, die nach Satz 4.6 übereinstimmen. Die Iteration wirkt nämlich nur auf das orthogonale Komplement des Vektors der Gleichverteilung. Im Falle des ADI-SOS2 steht der Vektor $v_{1,1,+} = \begin{pmatrix} \overline{w} \\ \overline{w} \end{pmatrix}$ senkrecht auf allen übrigen Eigenvektoren mit Ausnahme von $v_{1,1,-} = \begin{pmatrix} \overline{w} \\ (\beta-1)\,\overline{w} \end{pmatrix}$. Aufgrund der Tatsache, dass die Gesamtlast erhalten bleibt, enthalten die Iterierten $\begin{pmatrix} w^{k-1} \\ w^k \end{pmatrix}$ aber keinen Anteil in Richtung von $v_{1,1,-}$. Für ADI-SOS2 gilt entsprechendes. □

Die Aussagen über die vergleichbare Konvergenz beider Verfahren lassen sich wegen der geforderten Optimalität der Parameter für beide Richtungen nicht auf den Fall verschiedener Faktoren übertragen.

ADI-Čebyšev
Alle Konstruktionen aus diesem Abschnitt lassen sich auf die ähnlich zu konstruierenden Verfahren ADI-Čebyšev und ADI-Čebyšev2 übertragen. Ein Analogon zu Satz 4.6 lässt sich allerdings nicht zeigen, da sich die Iterationsmatrizen hier in jedem Schritt ändern. Experimenten zufolge konvergieren aber auch diese beiden Verfahren etwa gleich schnell und es gelten dieselben Beobachtungen zu den Flüssen.

4.5 ADI-OPS

Will man eine OPS-Version für Produktgraphen erzeugen, so hat man prinzipiell dieselben beiden Möglichkeiten wie im letzten Abschnitt für SOS. Der erste Ansatz, OPS für die Matrix $M^{(1)} \otimes M^{(1)}$ ist jedoch nicht sinnvoll, da diese in etwa genauso viele Eigenwerte besitzt, wie die Diffusionsmatrix des kompletten Produktgraphen. Dagegen lässt sich der für ADI-SOS2 verwendete Ansatz problemlos auf OPS übertragen, das resultierende Verfahren wird mit ADI-OPS bezeichnet und ist in Algorithmus 4.8 dargestellt. Zur Be-

$w11 = w^0$
$w01 = \frac{1}{\gamma_1}\left(\alpha_1 I - I \otimes M^{(1)}\right) w11$
$w10 = \frac{1}{\gamma_1}\left(\alpha_1 I - M^{(1)} \otimes I\right) w11$
$w = \frac{1}{\gamma_1}\left(\alpha_1 I - M^{(1)} \otimes I\right) w01$
for $k = 2, \ldots, m-1$ **do**
  $w22 = w11$
  $w12 = w01$
  $w21 = w10$
  $w11 = w$
  $w02 = \frac{1}{\gamma_k}\left(\left(\alpha_k I - I \otimes M\right) w12 - \beta_k w22\right)$
  $w01 = \frac{1}{\gamma_k}\left(\left(\alpha_k I - I \otimes M\right) w11 - \beta_k w21\right)$
  $w10 = \frac{1}{\gamma_k}\left(\left(\alpha_k I - M \otimes I\right) w11 - \beta_k w12\right)$
  $w = \frac{1}{\gamma_k}\left(\left(\alpha_k I - M \otimes I\right) w01 - \beta_k w02\right)$
end for

Algorithmus 4.8: ADI-OPS-Verfahren

rechnung der für die Dreitermrekursion des OPS benötigten Koeffizienten α_k, β_k, γ_k wird das Innenprodukt (2.1) für Polynome verwendet ($\langle p, q\rangle = \sum_{j=2}^{m} \omega_j p(\mu_j^{\mathsf{Diff}}) q(\mu_j^{\mathsf{Diff}})$). Dabei war ω_j so gewählt, dass der Fehler nach jedem einzelnen Schritt möglichst klein wird. Darauf, dass das Verfahren nach m Schritten mit Fehler 0 endet, hat diese Wahl jedoch keinen Einfluss. Während beim OPS grundsätzlich minimale Flüsse berechnet werden und eine Änderung des Produktes damit keinen Sinn macht, lässt sich beim ADI-OPS der Fluss dadurch verbessern, dass ω_j durch Potenzen ω_j^η ersetzt wird, also

$$\langle p, q\rangle = \sum_{j=2}^{m} \omega_j^\eta p(\mu_j^{\mathsf{Diff}}) q(\mu_j^{\mathsf{Diff}}) .$$

Experimente zeigen, dass die Flüsse mit steigendem η kleiner werden, während der abschließende Rundungsfehler ansteigt. Eine zusätzliche Verringerung der Flüsse wird auch hier wieder erreicht, indem zwei Rechnungen, einmal mit der ersten und einmal mit der zweiten Richtung beginnend, durchgeführt und die Flüsse gemittelt werden. Die Auswirkungen von η auf dieses ADC-OPS genannte Verfahren ist am Beispiel des T_{16} in Abbildung 4.5 dargestellt. Werte für η zwischen 3 und 4 haben sich praktisch als gut erwiesen.

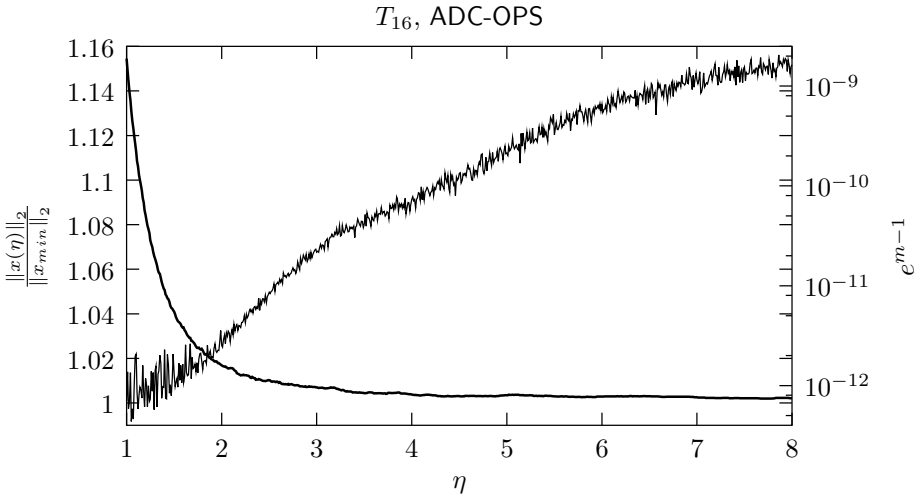


Abbildung 4.5: Abhängigkeit des ADC-OPS-Verfahrens von η: Dargestellt ist, um wie viel der Fluss über dem Minimum liegt (dicke Linie, linke Skala) und der abschließende Fehler e^{m-1} (dünne Linie, rechte Skala)

Eine Übertragung des quadratischen Falles auf verschiedene Faktorgraphen $G^{(1)} \neq G^{(2)}$ ist wiederum möglich; die Eigenwerte sowie die für die Dreitermrekursion benötigten Parameter α_k, β_k, γ_k müssen dann vorab für jede Richtung separat berechnet werden. Die Anzahl verschiedener Eigenwerte der Faktorgraphen werde mit $m^{(1)}$ und $m^{(2)}$ bezeichnet, wobei die erste Zahl größer sein möge. Dann wird Algorithmus 4.8 bis zur Berechnung von $w^{m^{(2)}-1,m^{(2)}-1}$ angewandt und anschließend auf den eindimensionalen OPS-Algorithmus bis zur Berechnung von $w^{m^{(1)}-1,m^{(2)}-1}$ gewechselt.

4.6 Dimension Exchange für Produktgraphen

Die Übertragung der Diffusionsverfahren auf das Dimension-Exchange-Modell ist im Falle der Verfahren für Produktgraphen genauso einfach wie im Abschnitt 3.3. Es muss wiederum nur die Diffusionsmatrix M bzw. die Laplace-Matrix L durch die entsprechende Matrix für gefärbte Graphen ersetzt werden. Die daraus resultierenden Verfahren heißen z. B. DE-ADC-OPS oder SDE-ADI-OPT. Die Schnelligkeit dieser Verfahren verdeutlicht das Beispiel in Abbildung 4.6.

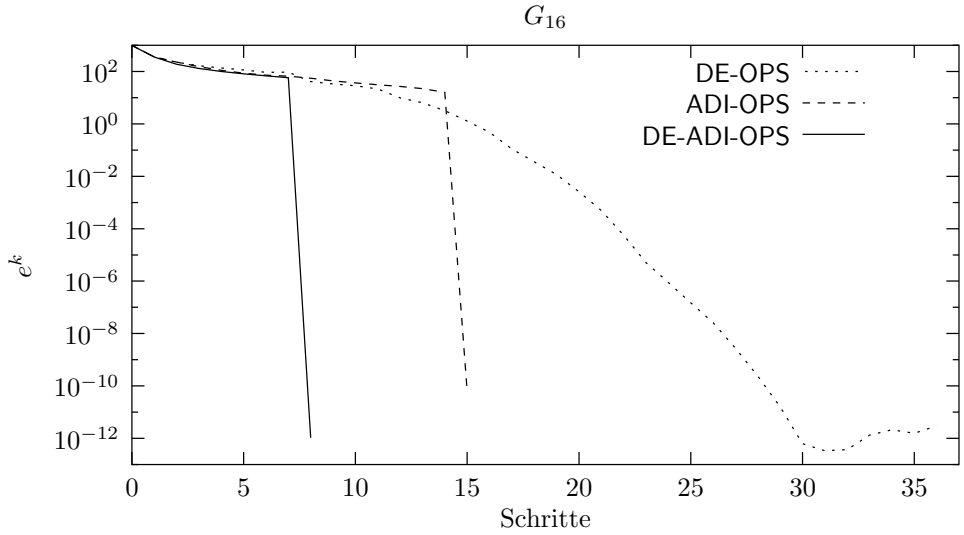


Abbildung 4.6: Konvergenz von ADI- und nicht-ADI-Verfahren am Beispiel des Gitters G_{16}

Die Methoden zur Flussreduktion aus dem letztem Abschnitt (ADC-..., $\omega(z) = |z|^g, \eta$) lassen sich ebenfalls direkt auf die Dimension-Exchange-Verfahren anwenden, vergleiche hierzu die Abbildungen 4.7 und 4.8.

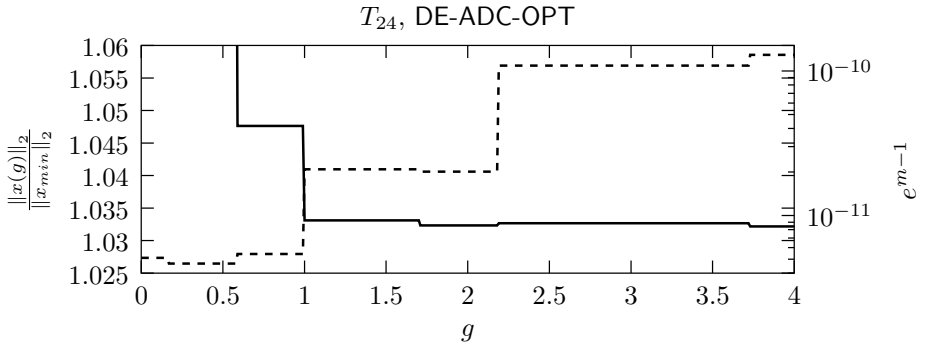


Abbildung 4.7: Abhängigkeit des DE-ADC-OPT-Verfahrens mit Leja$^{(g)}$-Sortierung vom Exponenten g der Gewichtsfunktion $\omega(z) = |z|^g$: Fluss (durchgezogene Linie, linke Skala) und abschließender Fehler (gestrichelte Linie, rechte Skala),vgl. Abb. 4.3.

Als weitere Variante kann man auch das schon aus Abschnitt 3.7.2 bekannte zyklische Durchlaufen der Farben auf die Verfahren DE-ADI-OPX für Produktgraphen (mit zwei Richtungen) übertragen, siehe Abbildung 4.9. Zu beachten ist hierbei, dass die zu den beiden Richtungen gehörenden Halbschritte nicht vermischt werden dürfen; das zyklische

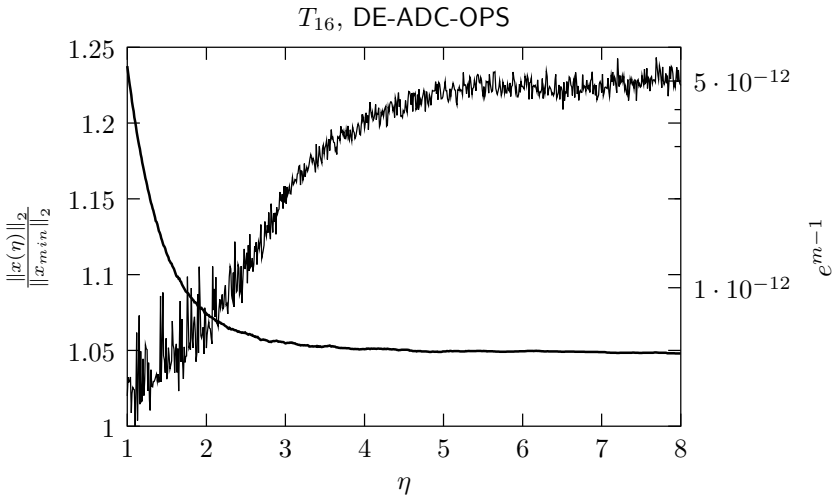


Abbildung 4.8: Abhängigkeit des DE-ADC-OPS-Verfahrens von η: Dargestellt ist, um wie viel der Fluss über dem Minimum liegt (dicke Linie, linke Skala) und der abschließende Fehler e^{m-1} (dünne Linie, rechte Skala), vgl. Abb. 4.5

Rechnung 1:	1	2	3		4	5	6		
Rechnung 2:	1	2	3			5	6	4	
Rechnung 3:	1	2	3				6	4	5
Rechnung 4:		2	3	1	4	5	6		
Rechnung 5:		2	3	1		5	6	4	
Rechnung 6:		2	3	1			6	4	5
Rechnung 7:			3	1	2	4	5	6	
Rechnung 8:			3	1	2		5	6	4
Rechnung 9:			3	1	2			6	4	5

Abbildung 4.9: Ablauf eines Schrittes des DE-ADI-OPTcc-Verfahrens am Beispiel eines Graphen, der in zwei Richtungen mit 3 je Farben gefärbt ist; jede Spalte steht für einen Kommunikationsschritt

Durchlaufen ist hier also auf jeden einzelnen Halbschritt anzuwenden. Ist der Graph in beiden Richtungen mit jeweils c Farben gefärbt, so erhöht sich der Aufwand für einen kompletten Schritt von $2c$ auf $4c - 2$ Kommunikationsoperationen. Der Rechenaufwand erhöht sich gemäß der Abbildung um den Faktor c^2. (Bei $d > 2$ Dimensionen ist dieser Ansatz nicht mehr vernünftig, da sich der Rechenaufwand um den Faktor c^d erhöht.)

Flüsse

Für die Verfahren (S)DE-OPX konnte gezeigt werden, dass der Fluss für α gegen 0 minimal wird. Auf die ADI-Varianten trifft dies nicht mehr zu, wie die Experimente in Abbildung 4.10 zeigen. An den Unstetigkeitsstellen der beiden rechten Bilder ändert sich

jeweils die durch Leja-Sortierung bestimmte Reihenfolge der Eigenwerte.

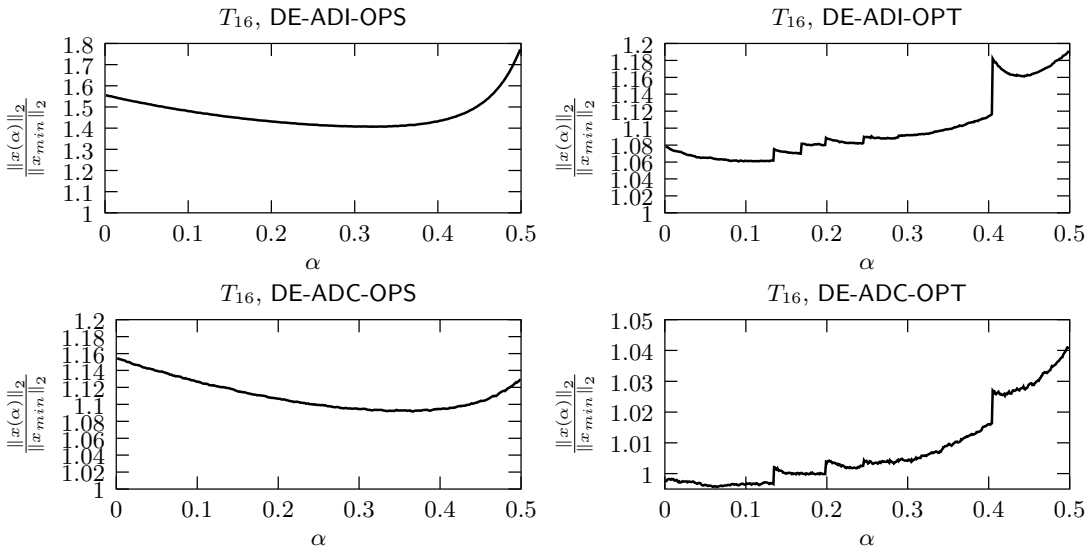


Abbildung 4.10: l_2-Norm des Flusses in Abhängigkeit von α am Beispiel eines 16×16-Torus mit zufälligem w^0 (minimaler Fluss = 1)

4.7 Laufzeiten

Die Laufzeiten der verschiedenen Verfahren sind in Tabelle 4.2 zusammengefasst. Hierbei ist jeweils berücksichtigt, dass zwei aufeinander folgende Kommunikationen entlang Kanten der selben Farbe durch eine Kommunikation zusammengefasst werden können. Wie die Laufzeiten für konkrete Graphen und Einfärbungen bei ausgesuchten Verfahren aussehen, ist Tabelle 4.3 zu entnehmen. Es ist festzustellen, dass für Gitter und für Tori mit geraden Dimensionen Verfahren mit optimaler oder zumindest annähernd optimaler Laufzeit gefunden sind. Zum einen erreicht man diese Laufzeiten durch die Kombination von alternierenden Richtungen und Dimension Exchange. Die zweite Möglichkeit ist, in beiden Richtungen hintereinander zu balancieren und für jede Richtung Vorwärts-rückwärts-Diffusion zu verwenden. Hierbei ist zu beachten, dass die zweite Alternative Experimenten zufolge erheblich höhere Flüsse erzeugt. Die Zeiten für Vorwärts-rückwärts-Diffusion mit alternierenden Richtungen liegen zwischen den optimalen und den „Standardverfahren“ wie ADI-OPT.

Verfahren	Anzahl Kommunikationsschritte
ADI-OPX	$2c\,(m-1)$
MDI-OPX	$2c\,(m-1)$
SDI-OPX	$2c\,(m-1)$
ADC-OPX	$2c\,(m-1)+c$
FB-SDI-OPX	$(2c-2)\,(m-1)+2$
FB-MDI-OPX	$(2c-1)\,(m-1)+1$
DE-XYZ-OPX	$\widehat{=}\ XYZ$-OPX
DE-ADI-OPXcc	$(4c-2)\,(m-1)$
DE-ADC-OPXcc	$(4c-2)\,(m-1)+2c-1$
SDE-ADI-OPX	$(4c-2)\,(m-1)$
SDE-MDC-OPX	$(4c-3)\,(m-1)+4c-2$

Tabelle 4.2: Anzahl der Kommunikationsschritte verschiedener Verfahren für Produktgraphen $G = G^{(1)} \times G^{(1)}$, c ist die Anzahl der Farben von $G^{(1)}$ und m die Anzahl der Eigenwerte von $G^{(1)}$

Graph G		diam	ADI-OPX	ADC-OPX	FB-SDI-OPX	FB-MDI-OPX
$G_k = P_k \times P_k$		$2k-2$	$4k-4$	$4k-2$	$2k$	$3k-2$
$T_k = C_k \times C_k$	$2 \mid k$	k	$2k$	$2k+2$	$k+2$	$\frac{3}{2}k+1$
	$2 \nmid k$	$k-1$	$2k$	$2k+3$	$2k$	$\frac{5}{2}k-\frac{3}{2}$

Graph G		diam	DE-ADI-OPX	SDE-MDC-OPX	DE-ADC-OPX	DE-ADC-OPXcc
$G_k = P_k \times P_k$	$2 \mid k$	$2k-2$	$2k$	$\frac{5}{2}k+6$	$2k+2$	$3k+3$
	$2 \nmid k$	$2k-2$	$2k+2$	$\frac{5}{2}k+\frac{17}{2}$	$2k+4$	$3k+6$
$T_k = C_k \times C_k$	$4 \mid k$	k	k	$\frac{5}{4}k+6$	$k+2$	$\frac{3}{2}k+3$
	$2 \mid k, 4 \nmid k$	k	$k+2$	$\frac{5}{4}k+\frac{17}{2}$	$k+4$	$\frac{3}{2}k+6$
	$2 \nmid k$	$k-1$	$2k+4+\left\lceil\frac{4}{k-1}\right\rceil$	$\frac{9}{2}k+\frac{29}{2}$	$2k+7+\left\lceil\frac{4}{k-1}\right\rceil$	$5k+10$
$T_k^{[C3]} = C_k^{[3]} \times C_k^{[3]}$	$3 \mid k, 2 \nmid k$	$k-1$	$4k$	$3k+19$	$4k+3$	$\frac{20}{3}k+5$

Tabelle 4.3: Anzahl der Kommunikationsschritte bei Produktgraphen $G = G^{(1)} \times G^{(1)}$ mit verschiedenen Verfahren; bei allen Dimension-Exchange-Verfahren wird $\alpha = \frac{1}{2}$ angenommen; zum Vergleich ist jeweils der Durchmesser $\mathrm{diam}(G)$ der Graphen mit angegeben

5 Details zur Implementierung und Messergebnisse

In diesem Kapitel werden einige Detailinformationen zur Implementierung der besprochenen Verfahren gegeben. Sie sollen helfen, die damit durchgeführten Konvergenz- und Zeitmessungen besser zu verstehen und diese Messungen besser mit denen anderen Implementationen vergleichen zu können. Außerdem werden einige speziellere Algorithmen (zum Beispiel zum zyklischen Durchlaufen der Farben) angegeben, die in den vorigen Kapiteln aus Gründen der Übersichtlichkeit weggelassen worden sind.

5.1 Allgemeines

Die Implementierung des Testprogramms erfolgte komplett objektorientiert in C++. Für den Datenaustausch zwischen den Prozessen wird MPI verwendet.

5.2 Verwaltung der Last

Jeder Prozessor hat eine variable Anzahl von Lasteinheiten (auch *Token* genannt) zu verwalten. Alle Tokens haben grundsätzlich eine feste Größe. Im Testprogramm besteht jedes Token aus einem dynamisch beschafften Speicherbereich, der mit einer Zeichenkette belegt wird. Die einzelnen Speicherbereiche werden über einen Vektor aus Zeigern hierauf verwaltet.

Zur Erzeugung der Ausgangslastverteilung stehen zwei verschiedene Initialisierungen zur Verfügung: zum einen eine zufällige Verteilung, die so gewählt ist, das eine vorgegebene Gesamtlast erreicht wird *(Rand)* und zum anderen eine *Peak*-Verteilung, bei der ein Prozessor die gesamte Last erhält und alle übrigen keine Last haben. Der ausgewählte Knoten bei Peak-Verteilungen ist immer Knoten Nummer 1, d. h. bei Pfaden ein Endknoten und bei Gittern ein Eckknoten.

5.3 Speicherung der Graphen

Jeder Knoten kennt seine eigene Nummer, die Größe des Graphen und die Anzahl der benachbarten Knoten sowie deren Nummern. Je nach Graph kommen hierzu weitere Informationen wie zum Beispiel die beiden Dimensionen bei einem Gitter oder eine komplette Kantenliste für zufällige Graphen. Dies wird benötigt, um die Eigenwerte während

des Loadbalancings ohne Kommunikation lokal berechnen zu können. Eine parallele Berechnung der Eigenwerte würde sich nur für große unstrukturierte Graphen lohnen.

Ergänzungen für Verfahren, die die Kanten vorwärts und rückwärts durchlaufen
Bei einigen Verfahren wie DE-OPTfb oder SDE-OPT werden die Kanten des Graphen im Wechsel vorwärts und rückwärts durchlaufen. Hat ein Knoten A als ersten Nachbarn den Knoten B und ist *gleichzeitig* A erster Nachbar von B, dann gibt es für beide zwei aufeinander folgende Kommunikationsoperationen über die gemeinsame Kante. Diese zwei Kommunikationen lassen sich zu einer vereinen. Dasselbe gilt für den jeweils letzten Nachbarn eines Knotens. So haben die Knoten 1 und 2 des Graphen in Abbildung 5.2 a) z. B. eine solche spezielle erste Kante, Knoten 3 dagegen nicht. Ergänzend muss also gespeichert werden, ob die erste und die letzte Kante eine solch spezielle Kante ist oder nicht.

Ergänzungen für das zyklische Durchlaufen der Farben
Gegeben sei ein mit c Farben gefärbter Graph. Im Gegensatz zu früher startet die erste Rechnung nun mit Farbe zwei, die letzte (c-te) Rechnung entspricht dem Ablauf des einfachen Verfahrens. Gemäß Abbildung 5.1 wird die gesamte Rechnung aufgeteilt in einen ersten Halbschritt, $m-2$ reguläre Schritte und einen abschließenden Halbschritt. Die dicken grauen Linien deuten an, wo innerhalb der Einzelrechnung eine Schleife über alle Farben komplett ist. Gibt es Knoten, deren Grad kleiner ist als c, so muss diese Infor-

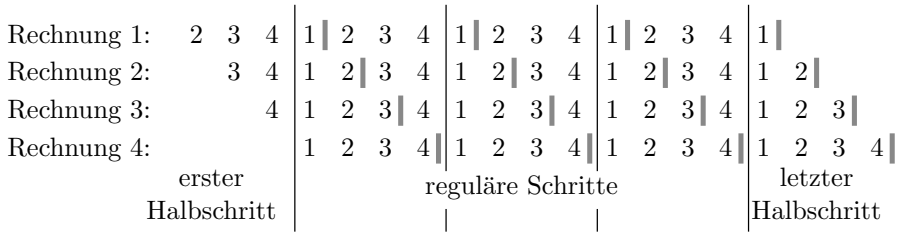


Abbildung 5.1: Ablauf von DE-OPTcc

mation extra gespeichert werden. Als Beispiel hierzu wird der C_3 aus Abbildung 5.2 a) betrachtet. Kursiv neben jedem Knoten eingetragen ist die lokale Nummerierung der Nachbarn. Teil b) der Abbildung enthält wiederum den Ablauf der Rechnung inklusive der Markierungen, wo jeweils eine Schleife über alle Nachbarn beendet ist. Diese Information muss in einem separaten Vektor gespeichert werden; für den Prozessor 3 aus dem Beispiel ergibt sich also $(0, 1, 2)$. Auf den Prozessoren 1 und 2 lautet die Vektoren $(1, 1, 2)$ und $(1, 2, 2)$; sie tragen im Algorithmus 5.1 zum DE-OPTcc die Bezeichnung *ccend*.

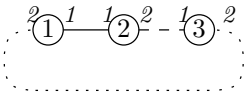
5.4 Berechnung und Speicherung der Flüsse

Die Flüsse auf jedem Prozessor werden in einem Vektor gespeichert, dessen Länge der Anzahl der Nachbarn entspricht. Während des Loadbalancings wird mit Gleitkommazahlen gerechnet, die am Ende auf die nächste ganze Zahl gerundet werden, denn es

```
{erster Halbschritt}
ŵ = w
for j = ccend(1) + 1, ..., Anzahl der Nachbarn do {Farbschleife}
    Datenaustausch, ŵ an Nachbar j, ŵ_j von Nachbar j
    for r = 1, ..., Anzahl der Farben and while ccend(r) < j do
        {Schleife über die einzelnen Rechnungen}
        y = α (ŵ(r) − ŵ_j(r))
        ŵ(r) = ŵ(r) − y
        x_j(r) = x_j(r) + 1/(α λ_1^DE) y
    end for
end for
{m − 2 reguläre Schritte}
for k = 1, ..., m − 2 do
    for r = 1, ..., Anzahl der Farben and while ccend(r) = 0 do
        w^k(r) = (1 − 1/(α λ_{k+1}^DE)) w^{k−1} + 1/(α λ_{k+1}^DE) ŵ
        ŵ(r) = w^k(r)
    end for
    for j = 1, ..., Anzahl der Nachbarn do {Farbschleife}
        Datenaustausch, ŵ an Nachbar j, ŵ_j von Nachbar j
        for r = 1, ..., Anzahl der Farben do {Rechnungsschleife}
            y = α (ŵ(r) − ŵ_j(r))
            ŵ(r) = ŵ(r) − y
            if ccend(r) ≥ j then
                x_j(r) = x_j(r) + 1/(α λ_{k+1}^DE) y
            else
                x_j(r) = x_j(r) + 1/(α λ_{k+2}^DE) y
            end if
            if ccend(r) = j then
                {ein Schritt in Rechnung r ist beendet: neues w}
                w^k(r) = (1 − 1/(α λ_{k+1}^DE)) w^{k−1} + 1/(α λ_{k+1}^DE) ŵ
                ŵ(r) = w^k(r)
            end if
        end for
    end for
end for
{abschließender Halbschritt}
for j = 1, ..., ccend(c) do {Farbschleife}
    Datenaustausch, ŵ an Nachbar j, ŵ_j von Nachbar j
    for r = Anzahl der Farben, ..., 1 and while ccend(r) ≥ j do
        y = α (ŵ(r) − ŵ_j(r))
        ŵ(r) = ŵ(r) − y
        x_j(r) = x_j(r) + 1/(α λ_m^DE) y
    end for
end for
for r = 1, ..., Anzahl der Farben do
    w^{m−1}(r) = (1 − 1/(α λ_m^DE)) w^{m−2} + 1/(α λ_m^DE) ŵ
end for
{Mittelwert der r Flüsse berechnen}
for j = 1, ..., Anzahl der Nachbarn do
    x_j = Mittelwert{x_j(r) | r = 1, ..., Anzahl der Farben}
end for
```

Algorithmus 5.1: DE-OPTcc-Verfahren für Prozessor i

a)



b)

Rechnung 1:	1	2	1	2	1	2	1	2					
Rechnung 2:		2	1	2	1	2	1	2	1				
Rechnung 3:			1	2	1	2	1	2	1	2			

Abbildung 5.2: Ablauf von DE-OPTcc für den Zyklus C_3: a) Skizze des Graphen mit lokaler Nummerierung der Nachbarn, b) Ablauf des Verfahrens aus Sicht von Prozessor 3

können nur ganze Lasten verschoben werden. Diese Rundung kann dazu führen, dass die tatsächliche Last nach Ende des Schedulings bei einem Knoten vom Grad δ um maximal $\frac{\delta}{2}$ vom exakten Mittelwert ($\overline{w}_i$) abweicht. Außerdem kann es vorkommen, dass der bei exakter Rechnung l_2-minimale Fluss nach Rundung unter allen ganzzahligen Flüssen nicht mehr die kleinste Norm aufweist.

Haben zwei Prozessoren A und B eine gemeinsame Kante, so wird der Fluss über diese Kante auf beiden Prozessoren, aber mit unterschiedlichem Vorzeichen, gespeichert.

$$\begin{array}{cc} \text{Prozessor } A & \text{Prozessor } B \\ y = \alpha\,(w_A - w_B) & y = \alpha\,(w_B - w_A) \\ x_{AB} = x_{AB} + y & x_{BA} = x_{BA} + y \end{array}$$

In obigem Ausschnitt aus dem FOS muss sichergestellt sein, dass anschließend exakt $x_{AB} = -x_{BA}$ gilt. Schon bei kleinen Rundungsfehlern können sich bei der abschließenden Rundung der Flüsse auf ganze Zahlen Abweichungen von einer ganzen Lasteinheit ergeben. Bei x86-Prozessoren tritt genau dieses Problem auf, da Zwischenergebnisse in Registern mit höherer Genauigkeit (80 Bit) gespeichert werden können. Werden Registerinhalte zwischenzeitlich in den Hauptspeicher ausgelagert, so wird auf 64 Bit gerundet. IEEE-Konformität erreicht man mit dem `g++`-Compiler durch die Option `-ffloat-store`. Die damit verbundenen Performance-Einbußen bei den Berechnungen fallen wegen der Dominanz der Kommunikationszeiten kaum auf.

5.5 Grundlegende Voraussetzungen für alle Verfahren

Es wird bei allen Verfahren verlangt, dass sie ohne globale Kommunikation während des Loadbalancings auskommen. Bei allen iterativen, nicht-endlichen Verfahren (z. B. FOS, SOS) wird jedoch zur Bestimmung der notwendigen Anzahl der Iterationsschritte einmal zu Beginn der anfängliche Fehler $\left\|e^0\right\|_2 = \left\|w^0 - \overline{w}\right\|_2$ bestimmt. Mit Hilfe der Fehlerabschätzungen aus Kapitel 2.2 wird garantiert, dass der abschließende Fehler in der l_2-Norm kleiner als $\frac{1}{2}$ wird. Beim DE-FOS findet sich keine vergleichbar einfache Abschätzung, da die Iterationsmatrix M^{DE} unsymmetrisch ist. Ungeachtet dessen wird

hier wie beim FOS der zweitgrößte Eigenwert zu Rate gezogen. Die Praxis zeigt, dass die abschließenden Fehler auf diese Weise zumindest in der Größenordnung von $\frac{1}{2}$ bleiben.

Endliche Verfahren wie OPS oder OPT werden nicht vorzeitig abgebrochen, auch wenn dies manchmal sinnvoll wäre; vergleiche hierzu zum Beispiel die Konvergenz für das Gitter G_{24} in Abbildung 2.4. Stattdessen werden immer $m-1$ Schritte durchgeführt, wobei m die Anzahl verschiedener Eigenwerte ist, inklusive 1. Allerdings werden bei der Berechnung der Eigenwerte solche Eigenwerte als gleich angesehen und damit nur einmal gezählt, die absolut näher als 10^{-12} beieinander liegen. (Eine vergleichbare relative Genauigkeit kann mit den üblichen Berechnungsmethoden für Eigenwerte nicht garantiert werden.)

5.6 Synchron oder Asynchron?

Bei Dimension-Exchange-Verfahren ist die Reihenfolge der Kommunikation eines Prozessors mit seinen Nachbarn durch die Einfärbung des Graphen strikt festgelegt. Der Datenaustausch erfolgt paarweise durch eine Sende-Empfangs-Operation (`Sendrecv` in MPI).

Da der gleichzeitige Datenaustausch mit mehreren Nachbarknoten, wie er für Diffusionsverfahren optimal wäre, auf den meisten Systemen nicht durchführbar ist, wird auch bei diesen Verfahren einzeln mit allen Nachbarn kommuniziert. Am einfachsten ist es, hierzu die sowieso vorhandene Einfärbung zu benutzen und genauso zu verfahren, wie beim Dimension-Exchange auch. Alternativ hierzu wurde folgende asynchrone Alternative untersucht: Zu Beginn eines Schrittes wird für alle Nachbarn ein nicht-blockierendes Senden (`Isend`) des eigenen Lastwertes initialisiert. In der darauffolgenden Schleife werden Lastinformationen von den Nachbarn nacheinander, aber in beliebiger Reihenfolge, empfangen und weiterverarbeitet. Erst danach werden die Sendeoperationen abgeschlossen. Verfahren, die nach diesem Prinzip arbeiten, bekommen den Zusatz Async-.

Die schnellsten Diffusionszeiten erreicht man durch eine Kombination von Vorwärts-Rückwärts-Iteration und asynchroner Kommunikation; bezeichnet wird dies mit FB-Async-. Zwei aufeinander folgende Kommunikationen über eine spezielle erste oder letzte Kante werden zusammengefasst und nur für die übrigen „mittleren“ Kanten wird asynchrone Kommunikation verwendet.

5.7 Verwendete Rechner und Software

Die Zeitmessungen wurden auf drei verschiedenen Parallelrechnern bzw. Rechnerclustern durchgeführt.

ALiCE

ALiCE (Alpha-Linux-Cluster-Engine,
`www.theorie.physik.uni-wuppertal.de/Computerlabor/ALiCE.phtml`) ist der Parallelrechner der Universität Wuppertal. Er besteht aus 128 Compaq DS10 Workstations. Jeder einzelne Rechner ist mit einem Alpha 21264 EV67 Prozessor mit 616 MHz

Taktfrequenz sowie 256 MB Speicher ausgestattet. Als Kommunikationshardware kommt ein Myrinet-Netzwerk zum Einsatz. Als darauf aufsetzende MPI-Implementierung wird Parastation3 (`www.par-tec.com`) verwendet. Übersetzt wird das Programm mit dem Compaq-C++-Compiler `cxx` in Version 6.3.9.6.

PSC

Der zweite zum Testen verwendete Rechner ist der PSC am Paderborn Center for Parallel Computing (PC^2, `www.upb.de/pc2/index.html`). Dieser Rechner vom Typ Siemens hpcLine besteht aus 96 Knoten mit je zwei Pentium-III-Prozessoren mit 850 MHz und 512 MB Speicher. Zwar enthalten die Knoten schnelle Dolphin-PCI/SCI-Karten zur Kommunikation, für die Tests wurden jedoch nur die zusätzlich vorhandenen 100-MBit-Ethernet-Karten benutzt. Auf diese Weise stehen die Resultate für einen „typischen“ PC-Cluster. Als Kommunikationssoftware wird MPICH in der Version 1.2.4 (`www.mcs.anl.gov/mpi/mpich`) verwendet, als Compiler `g++` Version 2.95.3. Von den zwei Prozessoren pro Knoten wurde für die Messungen jeweils nur einer verwendet.

PC-Cluster

Als dritte Testplattform diente ein kleiner PC-Cluster am Fachbereich Mathematik der Universität Wuppertal, bestehend aus 16 Pentium-III-PCs mit 800 MHz und 384 MB RAM sowie 100-MBit-Ethernet-Kommunikation. Bei Topologien über 16 Knoten wurden mehrere Prozesse pro Rechner gestartet. Die verwendete MPI-Implementierung ist LAM/MPI Version 6.5.6 (`www.lam-mpi.org`) in Kombination mit dem `g++` in Version 3.0.3. Statt des Client-to-Client-Mode (c2c) wurde die prinzipiell langsamere Kommunikation über den LAM-Dämon verwendet. Der Grund hierfür sind Performance-Einbrüche im c2c-Mode, wenn mehrere Prozesse auf dem selben Rechner asynchron kommunizieren.

Eigenwertberechnung

In Fällen, wo keine expliziten Formeln zur Eigenwertberechnung bekannt sind, also vor allem bei unstrukturierten Graphen, wird zur effizienten Berechnung der Eigenwerte LAPACK (*Linear Algebra Package*, `www.netlib.org/lapack`) verwendet in Kombination mit einer für den jeweiligen Rechner optimierten BLAS-Bibliothek (*Basic Linear Algebra Subprograms*). Während für ALiCE mit der `cxml` eine Compaq-eigene BLAS-Version existiert, sollte auf PCs ATLAS (*Automatically Tuned Linear Algebra Software*, `math-atlas.sourceforge.net`) verwendet werden.

5.8 Ergebnisse der Zeit- und Flussmessungen

Für die Zeit- und Flussmessungen wurden insgesamt sechs Topologien betrachtet: Zyklus, kompletter Graph, Gitter, Torus, Hypercube und zufälliger Graph. Die angegebenen Zeiten beinhalten nur das eigentliche Loadbalancing, nicht aber die Berechnung der Eigenwerte oder zum Beispiel die Bestimmung des optimalen α beim FOS. Geht man davon aus, dass sich die Topologie nicht ändert, muss die Eigenwertberechnung ja auch nur einmal durchgeführt werden, während das Loadbalancing wiederholt angewandt wird. Außerdem haben die Messwerte ergeben, dass diese zusätzlichen Zeiten im Vergleich zum gesamten Loadbalancing keine entscheidende Rolle spielen. Die einzige

Ausnahme hiervon stellt vielleicht das DE-FOS bei unstrukturierten Graphen dar, wo zur Bestimmung des optimalen α eine Art binäre Suche mit mehreren Eigenwertberechnungen durchgeführt wird. Um die Zeiten der verschieden schnellen Rechner besser vergleichen zu können, sind skalierte statt absoluter Zeiten abgedruckt. Als Referenz dient das einfachste (d. h. am leichtesten zu implementierende) endliche Verfahren, nämlich OPT; dessen Zeiten werden auf 100 gesetzt. Die realen Zeiten für dieses Verfahren sind in Tabelle 5.1 ersichtlich. Sie liegen in der Regel im Bereich weniger Milli- bis Zentisekunden.

Graph	ALiCE	PSC	PC-Cluster
C_{32}	1,41	4,26	16,83
K_{16}	0,751	2,26	10,2
G_8	5,23	14,6	191
T_8	2,14	6,88	94,8
H_6	1,65	9,85	92,0
$R_{32,96}$	10,7	29,3	110

Tabelle 5.1: Zeiten für OPT auf den Graphen der nachfolgenden Tabellen in Millisekunden (im Falle des PC-Clusters sind die gemessenen Zeiten dividiert durch die Anzahl der Prozesse pro Prozessor)

Im Falle des PC-Clusters sind die gemessen Zeiten durch die Anzahl der Prozesse pro Rechner dividiert. Mit Ausnahme der Hypercubes und vielleicht des zufälligen Graphen war hier sichergestellt, dass keine zwei zu benachbarten Knoten gehörenden Prozesse auf dem selben Prozessor liefen. Als Referenz für den minimalen Fluss wurden die Resultate des OPS-Verfahrens verwendet, da dieses ein besonders stabiles Konvergenzverhalten zeigt.

Von maximal 28 verschiedenen Verfahren wurden jeweils die durchgeführt, die auf den jeweiligen Graphen anwendbar sind.

Der Wert für das konstante Kantengewicht α wurde für alle nicht-endlichen Verfahren jeweils optimal gewählt, bei allen Dimension-Exchange-Verfahren gleich $\frac{1}{2}$, und beim OPS so, dass alle Diagonalelemente von M^{Diff} positiv bleiben (0,4 / 0,06 / 0,2 / 0,2 / 0,15 / 0,1 für C / K / G / T / H / R). Bei den endlichen Verfahren wurden die Eigenwerte mittels Leja-Sortierung angeordnet; als Exponent g für die Gewichtsfunktion wurde 1 bei Nicht-ADI- und 1,5 bei ADI-Verfahren benutzt. Der Wert für η bei ADI-OPS und dessen Varianten wurde auf 4 gesetzt.

Um die Zeiten im Millisekundenbereich exakt genug messen zu können, wurden erstens mindestens 50 Messungen hintereinander ausgeführt und zweitens wurde mindestens 20 Sekunden lang gemessen. Diese Prozedur wurde sechsmal wiederholt. Nach Streichung des kleinsten und größten Wertes wurde über die vier übrigen Werte gemittelt.

Die erste Tabelle 5.2 zum Zyklus C_{32} zeigt, dass sich die OPT-Zeiten, in Übereinstimmung mit den Vorhersagen in Tabelle 3.4, etwa auf die Hälfte reduzieren lassen. Es ist zu erkennen, dass die Verfahren DE-OPTfb und SDE-OPT bei einem Graphen mit nur zwei Farben wegen der Zusammenfassung von je zwei Teilschritten tatsächlich kaum länger dauern als DE-OPT. Das Problem der nicht-minimalen Flüsse kann bei Zyklen

weitestgehend vernachlässigt werden.

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	341	1866	1333	1787	0,999	1
Async-SOS	45	298	225	256	1	1
Async-Čebyšev	37	255	196	215	1	1
OPT	16	100	100	100	1,001	1
FB-Async-OPT	16	51	60	57	1,001	1
Async-OPS	16	95	74	90	1	1
DE-FOS	17	153	152	123	1,02	1,037
DE-OPT	8	39	56	54	1,002	1,001
DE-OPTfb	8	42	60	57	1	1
DE-OPTcc	8	42	62	57	1	1
SDE-OPT	8	52	58	57	1,002	1,001
DE-OPS	8	39	56	54	1,002	1,001

Tabelle 5.2: Ergebnisse für den Zyklus C_{32}

Der komplette Graph aus Tabelle 5.3 entspricht am ehesten der Topologie vieler Parallelrechner, bei denen mit Hilfe von Switches oder Crossbars jeder Prozessor mit jedem anderen gleichberechtigt kommunizieren kann. Allerdings ist es technisch ausgeschlossen, alle möglichen Verbindungen gleichzeitig zu benutzen. Da dieser Graph nur einen von Null verschiedenen Eigenwert hat, sind Diffusionsverfahren hierfür relativ schnell. Dennoch steigt der Aufwand mit der Gesamtzahl der Prozessoren, da mit allen Nachbarn einmal kommuniziert werden muss. Für das nachfolgende Scheduling bedeutet dies, dass die Lasten über wesentlich mehr Kanten verschoben werden als bei anderen Graphen. Nichtsdestotrotz ist Dimension-Exchange im Falle dieses speziellen Graphen den Diffusionsverfahren sowohl in Bezug auf die Laufzeiten als auch auf die Flüsse klar unterlegen.

Die beiden wichtigsten Graphen sind sicherlich das Gitter G_8 in Tabelle 5.4 sowie der Torus T_8 in Tabelle 5.5, da man an diesen Beispielen (fast) alle Verfahren vergleichen kann und gerade der Torus in der Praxis als Topologie einiger Parallelrechner verwendet wird. Unter den Verfahren, die die Produktgraphstruktur nicht ausnutzen, sticht vor allem das DE-OPTcc hervor, das in vergleichsweise kurzer Zeit Flüsse berechnet, die nur wenig über dem Minimum liegen. Unter den ADI-Verfahren ist DE-ADC-OPT dasjenige, das geringe Zeiten und kleine Flüsse am besten vereint. Legt man ausschließlich Wert auf eine möglichst geringe Laufzeit, kommen vor allem FB-SDI-OPT und DE-ADI-OPT in Betracht; sollen die Flüsse möglichst klein gehalten werden, ist eher Async-ADC-OPS mit genügend großem η das Verfahren der Wahl, oder auch das wegen der komplexeren Implementierung nur für den Torus getestete DE-ADC-OPTcc. Die recht hohen Zeiten auf der PSC für Tori scheinen nicht repräsentativ zu sein und sollten nicht überbewertet werden.

Der Hypercube aus Tabelle 5.6 verhält sich ähnlich wie Gitter und Tori, auch hier ist DE-OPTcc das zu bevorzugende Verfahren. Die aufgeführten ADI-Verfahren sind zwar

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	2	200	153	191	0,997	1
Async-SOS	2	202	153	194	1	1
Async-Čebyšev	2	201	152	195	1	1
OPT	1	100	100	100	1	1
FB-Async-OPT	1	76	60	91	1	1
Async-OPS	1	72	46	91	1	1
DE-FOS	6	577	577	545	2,83	3,00
DE-OPT	7	405	590	568	2,83	3,00
DE-OPTfb	7	754	1107	1039	1,91	1,90
DE-OPTcc	7	573	920	699	1,10	1,06
SDE-OPT	4	616	635	646	2,78	2,76
DE-OPS	7	411	590	572	2,82	3,00

Tabelle 5.3: Ergebnisse für den kompletten Graphen K_{16}

durchführbar, indem man den Hypercube als Produkt von P_2-Pfaden auffasst, die resultierenden Verfahren entsprechen aber alle entweder DE-OPT oder DE-OPTcc, bringen dafür aber einen zusätzlichen Verwaltungsaufwand mit sich. Die Zeiten auf dem PC-Cluster sind für den Hypercube dadurch etwas verfälscht, dass eine von sechs Kommunikationen lokal erfolgte.

Der „zufällige“ Graph (Tabelle 5.7) ist für alle Testfälle identisch. Er besteht aus 32 Knoten, 96 Kanten, hat einen Knotengrad zwischen 2 und 9 und lässt sich mit 9 Farben einfärben. Hier lieferte DE-FOS die Gleichverteilung am schnellsten, aber auf Kosten recht hoher Flüsse; am zweitschnellsten war das asynchrone Čebyšev-Verfahren, das sogar minimale Flüsse erzeugt. Sinnvoll könnte noch das hier nicht getestete DE-FOScc sein. Der Grund für die Überlegenheit gegenüber eigenwertbasierten Verfahren dürfte sein, dass der hier gewählte Zufallsgraph einen recht kleinen Durchmesser hatte. Bei unregelmäßigen Gittern, wie sie bei der Aufteilung von Teilgebieten auf Prozessoren in Finite-Elemente-Anwendungen entstehen, lohnen sich endliche Verfahren eher.

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	173	439	388	398	1	1
Async-SOS	33	98	89	84	1	1
Async-Čebyšev	27	84	77	69	1	1
OPT	32	100	100	100	1	1
FB-Async-OPT	32	67	69	82	1	1
Async-OPS	32	81	71	79	1	1
DE-FOS	9	46	49	32	1,53	1,20
DE-OPT	10	24	35	34	1,14	1,041
DE-OPTfb	10	37	57	50	1,017	1,003
DE-OPTcc	10	27	40	36	1,020	1,012
SDE-OPT	10	49	56	51	1,13	1,031
DE-OPS	10	24	35	33	1,14	1,041
Async-MDI-FOS	87	272	221	204	1,32	1,041
Async-ADI-Čebyšev	19	74	65	50	2,14	1,041
Async-ADC-Čebyšev	19	58	67	51	1,009	1,003
Async-ADI-Čebyšev2	19	87	73	54	9,47	2,93
Async-ADC-Čebyšev2	19	121	83	57	3,51	1,99
FB-SDI-OPT	7	12	18	12	1,51	1,47
Async-MDI-OPT	7	24	22	23	1,020	1,010
FB-MDI-OPT	7	19	23	21	1,020	1,010
Async-ADC-OPT	7	18	23	22	0,996	1,001
Async-ADI-OPS	7	37	26	23	1,037	1,016
Async-ADC-OPS	7	54	35	25	1,006	1,002
DE-ADI-OPT	4	12	18	16	1,16	1,060
DE-ADC-OPT	4	13	20	16	1,037	1,012
DE-ADI-OPS	4	26	21	16	1,20	1,061
DE-ADC-OPS	4	43	28	19	1,047	1,012

Tabelle 5.4: Ergebnisse für das Gitter $G_8 = P_8 \times P_8$

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	47	319	276	319	1	1
Async-SOS	17	145	129	124	1	1
Async-Čebyšev	14	127	115	104	1	1
OPT	12	100	100	100	1	1
FB-Async-OPT	12	69	67	86	1	1
Async-OPS	12	78	69	90	1	1
DE-FOS	4	77	77	39	1,38	1,38
DE-OPT	3	24	34	25	1,23	1,23
DE-OPTfb	3	34	48	40	1,042	1,039
DE-OPTcc	3	30	44	35	1,033	1,031
SDE-OPT	3	43	47	40	1,23	1,23
DE-OPS	3	24	33	25	1,23	1,23
Async-MDI-FOS	24	216	184	166	1,29	1,26
Async-ADI-Čebyšev	10	117	103	74	1,37	1,44
Async-ADC-Čebyšev	10	99	130	81	1,006	1,014
Async-ADI-Čebyšev2	10	136	192	80	2,32	2,47
Async-ADC-Čebyšev2	10	177	220	84	1,20	1,21
FB-SDI-OPT	4	24	60	22	1,38	1,39
Async-MDI-OPT	4	36	77	37	1,03	1,033
FB-MDI-OPT	4	31	70	27	1,03	1,033
Async-ADC-OPT	4	30	77	39	0,998	1,003
Async-ADI-OPS	4	55	82	37	1,058	1,059
Async-ADC-OPS	4	76	104	40	1,005	1,005
DE-ADI-OPT	2	19	50	21	1,24	1,24
DE-ADC-OPT	2	23	58	22	1,059	1,069
DE-ADC-OPTcc	2	32	81	37	0,992	1,004
DE-ADI-OPS	2	31	55	21	1,28	1,28
DE-ADC-OPS	2	51	76	24	1,070	1,073

Tabelle 5.5: Ergebnisse für den Torus $T_8 = C_8 \times C_8$

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	21	281	266	70	0,999	1
Async-SOS	11	176	156	39	1	1
Async-Čebyšev	9	155	134	33	1	1
OPT	6	100	100	100	1	1
FB-Async-OPT	6	69	83	55	1	1
Async-OPS	6	73	80	22	1	1
DE-FOS	1	73	35	17	1,51	1,58
DE-OPT	1	21	29	16	1,51	1,58
DE-OPTfb	1	30	46	33	1,12	1,17
DE-OPTcc	1	32	43	23	1,070	1,080
SDE-OPT	1	38	45	33	1,51	1,58
DE-OPS	1	21	28	16	1,51	1,58
Async-MDI-FOS	1	–	–	11	1,51	1,58
FB-SDI-OPT	1	26	29	16	1,51	1,58
Async-MDI-OPT	1	29	30	8	1,51	1,58
FB-MDI-OPT	1	26	29	15	1,51	1,58
Async-ADC-OPT	1	34	45	16	1,044	1,080
DE-ADI-OPT	1	22	29	15	1,51	1,58
DE-ADC-OPT	1	33	44	23	1,044	1,080

Tabelle 5.6: Ergebnisse für den Hypercube $H_6 = P_2 \times P_2 \times P_2 \times P_2 \times P_2 \times P_2$

Verfahren	Schritte	Zeiten			Fluss: $\frac{\|x\|_2}{\|x_{\min}\|_2}$	
		ALiCE	PSC	PC-Cluster	Rand	Peak
Async-FOS	23	59	41	71	0,999	1
Async-SOS	11	32	24	36	0,999	1
Async-Čebyšev	10	30	23	35	0,999	1
OPT	31	100	100	100	1	1
FB-Async-OPT	31	60	45	81	1	1
Async-OPS	31	71	48	92	1	1
DE-FOS	4	21	22	20	1,84	1,87
DE-OPT	17	37	57	55	1,58	1,60
DE-OPTfb	17	68	103	96	1,20	1,20
DE-OPTcc	17	44	74	58	1,14	1,15
SDE-OPT	17	95	101	95	1,60	1,62
DE-OPS	17	37	57	54	1,58	1,60

Tabelle 5.7: Ergebnisse für den zufälligen Graphen $R_{32,96}$

6 Scheduling-Verfahren

In den vorangegangenen Kapiteln ist eine Vielzahl von Verfahren vorgestellt worden, die alle — ausgehend von einer anfänglichen Lastverteilung w^0 — einen Fluss x berechnen, der diese Lasten ausgleichen soll. Obwohl man hier von Loadbalancing-Verfahren spricht, ist bisher noch keine einzige Lasteinheit von einem Prozessor zu einem anderen bewegt worden — dies ist Aufgabe des sich nun anschließenden Schedulings. Im Idealfall muss jeder Prozessor weniger Last an seine Nachbarn abgeben, als er anfänglich besitzt; dann kann der Fluss in einem einzigen Schritt erfüllt werden. Anderenfalls muss eine der unten beschriebenen Strategien angewandt werden.

6.1 Das Scheduling-Problem

Die folgende formale Beschreibung des Scheduling-Problems ist angelehnt an die in [DFM99].

Gegeben sei ein Graph G mit Adjazenzmatrix A, eine Anfangslastverteilung w^0 sowie ein ausgleichender Fluss x, also $Ax = w^0 - \overline{w}$. Es wird zur Vereinfachung angenommen, dass die durch A festgelegte Richtung der Kanten jeweils der Flussrichtung entspricht, d. h. $x_i \geq 0$ für alle i. Es sei $A = A^+ + A^-$ eine Zerlegung von A in ihren positiven und negativen Anteil, also $a_{ij}^+ = \max\{a_{ij}, 0\}$ und $a_{ij}^- = \min\{a_{ij}, 0\}$. Gesucht ist eine Zerlegung des Flusses $S(x) = \left(\tilde{x}^0, \ldots, \tilde{x}^{k-1}\right)$ mit $x = \sum_{j=0}^{k-1} \tilde{x}^j$ und

$$A\tilde{x}^j = \underbrace{A^+\tilde{x}^j}_{\text{senden}} + \underbrace{A^-\tilde{x}^j}_{\text{empfangen}} = w^j - w^{j+1},$$

so dass

$$w^j \geq A^+\tilde{x}^j \text{ für alle } j \in \{0, \ldots, k-1\}$$

ist, wobei die Ungleichung komponentenweise erfüllt sein muss. Die Zerlegung S heißt zeitoptimal, falls k minimal ist unter all solchen Zerlegungen.

6.2 Scheduling-Verfahren

Ein zeitoptimales Scheduling ist schwer zu finden. Stattdessen werden in der Praxis heuristische Verfahren verwendet; ein allgemeiner Rahmen für ein solches Verfahren ist in Algorithmus 6.1 angegeben. Schon für die Loadbalancing-Verfahren galt die Forderung, dass sie ohne globale Kommunikation auskommen sollen. Entsprechend soll auch hier die Bestimmung des Teilflusses $\tilde{x}^k$ in jedem Schritt nur auf lokalen Informationen beruhen.

Gemäß [DFM99] spricht man von einem lokalen Greedy-Scheduling, wenn zusätzlich folgendes gilt: Hat ein Knoten in einem Schritt genügend Lasteinheiten, um die Flüsse über alle ausgehenden Kanten zu sättigen, so wird dies erfüllt; hat er keine ausreichende Zahl an Lasteinheiten, so verteilt er all seine Last nach einer bestimmten Regel (Heuristik) auf einen Teil oder auch alle ausgehenden Kanten.

$x^0 = x$; $k = 0$
while $x^k \neq 0$ **do**
  bestimme $\tilde{x}^k \neq 0$ **mit** $w^k \geq A^+ \tilde{x}^k$
  $x^{k+1} = x^k - \tilde{x}^k$
  $w^{k+1} = w^k - A\tilde{x}^k$
  $k = k + 1$
end while

Algorithmus 6.1: allgemeiner Ablauf des Scheduling

RRG und PPG

Zwei Beispiele für solche Greedy-Verfahren werden in [DFM99] beschrieben. Bei *Round-Robin Greedy (RRG)* wird erst eine Kante komplett aufgefüllt, dann die nächste und so weiter. Sind in einem Schritt nicht genügend Lasteinheiten vorhanden, so werden einige (eventuell auch gar keine) Kanten komplett aufgefüllt, eine Kante zum Teil und der Rest gar nicht. Im Gegensatz dazu werden bei *Proportional Parallel Greedy (PPG)* in jedem Schritt alle Kanten bedacht. Dabei ist die in einem Schritt über eine Kante verschobene Last proportional zur Flussanforderung über diese Kante.

Verbesserung des RRG

Die bisherige Beschreibung des RRG macht keine Aussage darüber, in welcher Reihenfolge die Kanten durchlaufen werden; sie könnte bei einer konkreten Implementierung zum Beispiel von der Speicherung des Graphen und nicht vom jeweiligen Fluss abhängen. Eine Verbesserung lässt sich dadurch erzielen, dass die Kanten vorher nach der Höhe der Flussanforderungen sortiert werden, beginnend bei der größten. Schließlich ist bei großen Flüssen über eine Kante die Wahrscheinlichkeit relativ hoch, dass die Lasten nicht komplett im Zielknoten verbleiben, sondern von dort zum Teil noch weiter versandt werden müssen, eventuell sogar über mehrere Stationen. Eine Bevorzugung solcher Kanten kann die Gesamtzahl der Schritte reduzieren. Diese Modifikation wird mit *SRRG (Sorted RRG)* bezeichnet.

Scheduling mit gefärbten Graphen

Bei den Loadbalancing-Verfahren lag der Schwerpunkt dieser Arbeit auf Dimension-Exchange-Verfahren. Die hierfür benötigte Einfärbung des Graphen kann auch für das anschließende Scheduling verwendet werden. Außerdem sind RRG und insbesondere PPG wiederum für das All-Port-Modell entworfen. Sie arbeiten am besten, wenn Lasten an mehrere bzw. alle Nachbarn zugleich versandt werden können. Das in Algorithmus 6.2 präsentierte *DE-Sched* ist mit seiner Schleife über die Farben dagegen bestens für das One-Port-Modell geeignet. Allerdings entspricht das Verfahren insofern nicht mehr exakt

obiger Definition lokaler Greedy-Scheduler, als dass in einem Schritt Lasteinheiten von einem Knoten aus versandt werden können, die zuvor im selben Schritt erst eingetroffen sind.

```
while x_e ≠ 0 für mindestens eine inzidente Kante e do
    for j = 1,...,c do {Farbschleife}
        if es gibt eine Kante e der Farbe j mit x_e > 0 then
            sende max{x_e, w} Lasteinheiten über Kante e, wobei
                   w die aktuelle Last ist
        else if es gibt ein e der Farbe j mit x_e < 0 then
            versuche, Last über x_e zu empfangen
        end if
    end for
end while
```

Algorithmus 6.2: DE-Sched aus Sicht eines Prozessors

6.3 Messergebnisse

Vergleich der Konvergenz

Bevor die Laufzeiten verglichen werden, wird die Konvergenz der vier Verfahren untersucht. Bei einer zufälligen Ausgangslastverteilung und einem Loadbalancing-Verfahren, das (annähernd) minimale Flüsse erzeugt, sind alle Verfahren sehr schnell und unterscheiden sich kaum: In der Regel werden hier nur ein bis drei Schritte benötigt. Erst bei künstlich erhöhten Flüssen, wie in Abbildung 6.1 werden Unterschiede deutlich. In allen gezeigten Beispielen erhält man die typische Reihenfolge, dass DE-Sched am schnellsten konvergiert, gefolgt von PPG, SRRG und RRG.

Implementierungsdetails

Wie schon in Abschnitt 5.2 beschrieben, werden die Lasten mit Hilfe eines Vektors von Zeigern auf Speicherbereiche fester Größe (die eigentlichen Lasteinheiten) verwaltet. Zu Beginn des Schedulings wird auf jedem Knoten i, der insgesamt mehr Last erhält als er abgeben muss, Speicher für $\overline{w}_i - w_i^0$ Lasteinheiten beschafft. Unter Umständen muss aber bei solchen Knoten, die Lasten von einem Nachbarn zu einem anderen „durchleiten“, während des Schedulings weiterer Speicher alloziert werden. Die Freigabe nicht mehr benötigten Speichers erfolgt erst nach Herstellung der Gleichverteilung. Bevor jeweils Lasten verschickt werden können, müssen sich beide beteiligten Prozessoren über die Anzahl der Einheiten verständigen, wozu vorher eine Nachricht in Richtung des Flusses gesandt wird. Gegebenenfalls wird hier eine Null versandt, falls ein Prozessor, der Daten senden müsste, gerade keine Last hat.

Die Ergebnisse der Zeitmessungen auf ALiCE (vergleiche Abschnitt 5.7) sind in Tabelle 6.1 zusammengefasst. Die Zeiten sind so skaliert, dass 100 dem Loadbalancing mit DE-OPT und $\alpha = \frac{1}{2}$ entspricht. Der betrachtete Graph ist stets ein 8×8-Torus, die

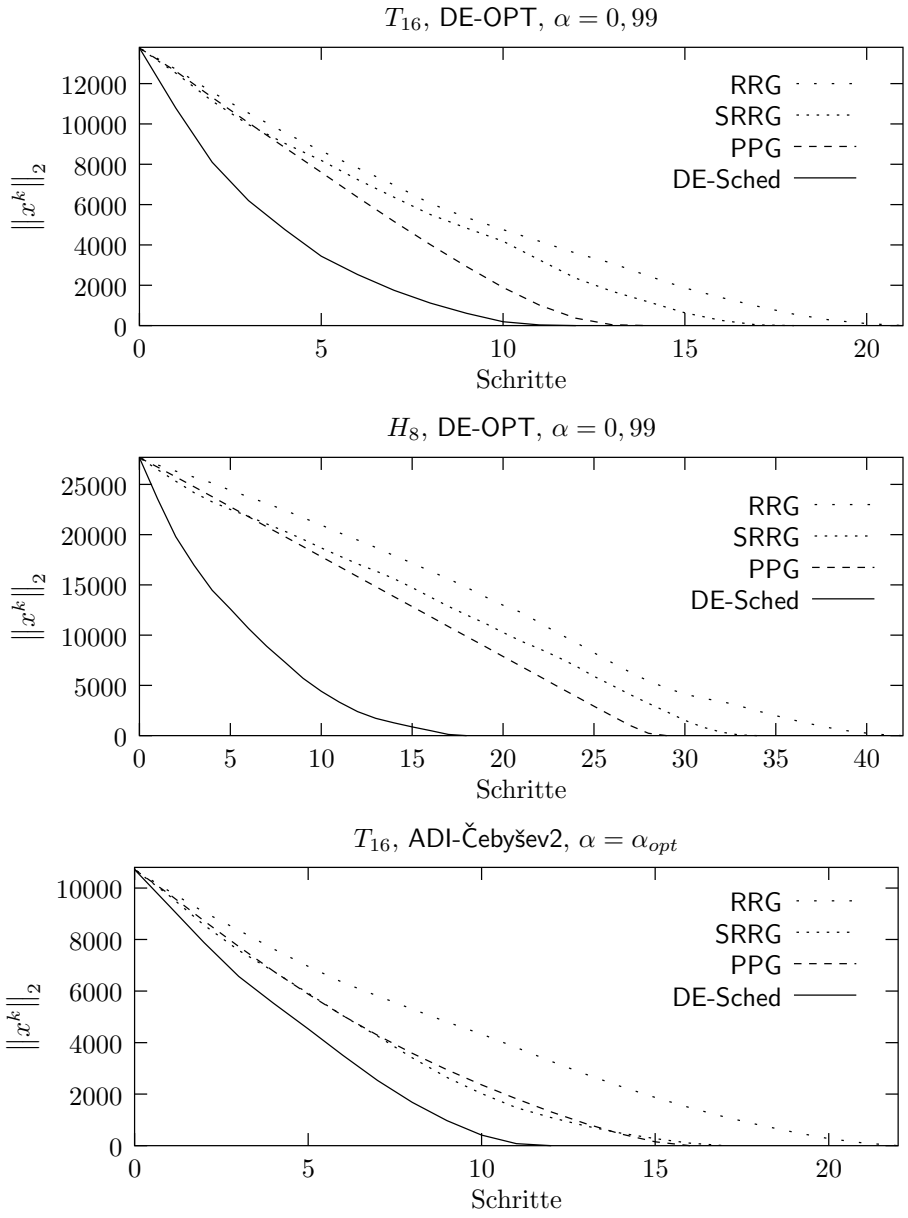


Abbildung 6.1: Vergleich der Konvergenz der vier Scheduling-Verfahren, bei den ersten beiden Beispielen sind durch die Wahl $\alpha = 0{,}99$ künstlich hohe Flüsse erzeugt worden, ADI-Čebyšev2 erzeugt schon bei optimalem α hohe Flüsse

Flüsse wurden mit DE-OPT generiert. Die kleinen Zahlen am linken Spaltenrand geben die Anzahl der Iterationsschritte beim Scheduling an.

Im ersten Abschnitt der Tabelle werden verschiedene Initialisierungen der Lasten verglichen. Bei der zufälligen Verteilung Rand unterscheiden sich alle vier Verfahren kaum. Bei den anderen beiden hat DE-Sched leichte Vorteile. Peak bedeutet, dass zu Beginn alle Last auf einem Knoten konzentriert ist. Da hierbei sehr spezielle Flüsse erzeugt werden und bei Rand sehr kleine, wird für alle weiteren Vergleiche die mit spec bezeichnete spezielle Verteilung verwendet, bei der die meisten Knoten eine zufällige kleine Last haben und 6 ausgewählte Knoten eine im Mittel fünfzigfach höhere Last.

Im zweiten Tabellenabschnitt werden die Flussnormen durch Erhöhung von α schrittweise vergrößert. Hierbei entstehen spezielle Kreisflüsse, mit denen PPG klar am besten zurechtkommt. Insbesondere stellt man hier fest, dass die Anzahl der Schritte alleine kein gutes Maß für die Schnelligkeit der Verfahren ist.

Bei den weiteren Messungen wurde die Gesamtzahl der Lasteinheiten von 64 bis auf 640000 heraufgesetzt bzw. die Lastgröße von einem Byte bis auf 100 kB. Ab einer bestimmten Größe nimmt die Zeit jeweils linear in Abhängigkeit von der Gesamtlast und der Lastgröße zu. Dann sind die Datenpakete so groß, dass die Bandbreite der Kommunikationshardware die Zeiten bestimmt. Es ist zu beobachten, dass DE-Sched in der Regel etwas schneller ist als die anderen drei, die sich untereinander kaum unterscheiden. Vor allem aber ist festzustellen, dass bei größeren oder vielen Lasten ein gutes Scheduling-Verfahren wichtiger ist als ein schnelles Loadbalancing-Verfahren.

α	Ges.-last	Lastgr.	Init.	RRG	SRRG	PPG	DE-Sched
0,5	$100n$	1000	Rand	$_{2}$ 55	$_{2}$ 57	$_{2}$ 56	$_{1}$ 54
0,5	$100n$	1000	spec	$_{4}$ 266	$_{4}$ 264	$_{4}$ 328	$_{2}$ 240
0,5	$100n$	1000	Peak	$_{8}$ 2360	$_{8}$ 2369	$_{8}$ 2366	$_{3}$ 2156
0,01	$100n$	1000	spec	$_{3}$ 221	$_{3}$ 236	$_{3}$ 226	$_{2}$ 207
0,5	$100n$	1000	spec	$_{4}$ 266	$_{4}$ 264	$_{4}$ 328	$_{2}$ 240
0,9	$100n$	1000	spec	$_{10}$ 1414	$_{9}$ 1350	$_{8}$ 1405	$_{4}$ 1256
0,99	$100n$	1000	spec	$_{75}$ 17364	$_{71}$ 19203	$_{56}$ 7993	$_{25}$ 14866
0,999	$100n$	1000	spec	$_{691}$ 235490	$_{540}$ 181411	$_{533}$ 104068	$_{229}$ 190194
0,5	$1n$	1000	spec	$_{4}$ 13	$_{4}$ 13	$_{4}$ 13	$_{2}$ 11
0,5	$10n$	1000	spec	$_{4}$ 37	$_{4}$ 38	$_{4}$ 38	$_{2}$ 34
0,5	$100n$	1000	spec	$_{4}$ 266	$_{4}$ 264	$_{4}$ 328	$_{2}$ 240
0,5	$1000n$	1000	spec	$_{4}$ 2971	$_{4}$ 2974	$_{4}$ 2974	$_{2}$ 2576
0,5	$10000n$	1000	spec	$_{4}$ 29953	$_{4}$ 30140	$_{4}$ 30054	$_{2}$ 25962
0,5	$100n$	1	spec	$_{4}$ 26	$_{4}$ 28	$_{4}$ 28	$_{2}$ 25
0,5	$100n$	10	spec	$_{4}$ 29	$_{4}$ 30	$_{4}$ 31	$_{2}$ 28
0,5	$100n$	100	spec	$_{4}$ 54	$_{4}$ 57	$_{4}$ 57	$_{2}$ 52
0,5	$100n$	1000	spec	$_{4}$ 266	$_{4}$ 264	$_{4}$ 328	$_{2}$ 240
0,5	$100n$	10000	spec	$_{4}$ 2771	$_{4}$ 2769	$_{4}$ 2770	$_{2}$ 2416
0,5	$100n$	100000	spec	$_{4}$ 27665	$_{4}$ 27691	$_{4}$ 27618	$_{2}$ 24590

Tabelle 6.1: Scheduling-Ergebnisse für den Torus T_8 auf ALiCE, Zeiten und Iterationsschritte (kleine Zahlen)

7 Kurze Ausblicke

Vor einer Zusammenfassung der Ergebnisse werden noch kurz einige Aspekte des Loadbalancings betrachtet, die in dieser Arbeit bisher unbeachtet geblieben sind.

7.1 Unterschiedliche Kantengewichte

Bisher wurde das Kantengewicht α immer konstant gehalten. Möchte man bestimmte Kanten für den Lastaustausch bevorzugen, so muss man zu individuellen Gewichten $\alpha_1, \ldots, \alpha_N$ übergehen. Gründe hierfür können verschieden leistungsfähige Netzwerkverbindungen sein oder geometrische Gründe bei Gebietszerlegungen. Die Ergebnisse in [DFM99] zu Diffusionsverfahren lassen sich folgendermaßen verallgemeinern. Die bisherigen Matrixdefinitionen werden ein wenig abgeändert:

$$\begin{aligned} \mathfrak{A} &= \operatorname{diag}(\alpha_1, \ldots, \alpha_N) \\ \tilde{L}_i &= A_i \mathfrak{A} A_i^T & \tilde{M}_i &= I - \tilde{L}_i \\ \tilde{A}^{\mathsf{ALG}} &\text{ wie bisher mit } \tilde{M}_j \text{ statt } M_j \\ \tilde{L}^{\mathsf{ALG}} &= A \mathfrak{A} \tilde{A}^{\mathsf{ALG}^T} & \tilde{M}^{\mathsf{ALG}} &= I - \tilde{L}^{\mathsf{ALG}} \end{aligned}$$

Das kantengewichtete ALG-FOS berechnet die Werte

$$w^k = \tilde{M}^{\mathsf{ALG}} w^{k-1}, \qquad x^k = x^{k-1} + \mathfrak{A} \tilde{A}^{\mathsf{ALG}^T} w^{k-1}, \qquad k = 1, 2, \ldots .$$

Alle anderen Verfahren werden entsprechend angepasst. In Analogie zu Satz 3.36 gilt für die Flüsse

$$\begin{aligned} x^{\mathsf{ALG}} &= \mathfrak{A} \tilde{A}^{\mathsf{ALG}^T} \tilde{L}^{\mathsf{ALG}^+} A \tilde{x}, \qquad \tilde{x} \text{ irgendein ausgleichender Fluss} \\ \left\| x^{\mathsf{ALG}} \right\|_{\mathfrak{A}^{-1}} &\leq \left\| \mathfrak{A} \tilde{A}^{\mathsf{ALG}^T} \tilde{L}^{\mathsf{ALG}^+} A \right\|_{\mathfrak{A}^{-1}} \left\| x^{\mathsf{Diff}} \right\|_{\mathfrak{A}^{-1}} \\ &= \left\| \mathfrak{A}^{\frac{1}{2}} \tilde{A}^{\mathsf{ALG}^T} \tilde{L}^{\mathsf{ALG}^+} A \mathfrak{A}^{-\frac{1}{2}} \right\|_2 \left\| x^{\mathsf{Diff}} \right\|_{\mathfrak{A}^{-1}} \end{aligned}$$

Hierbei ist

$$\|x\|_{\mathfrak{A}^{-1}} = \left\| \mathfrak{A}^{-\frac{1}{2}} x \right\|_2 = \left(\sum_{i=1}^{N} \frac{1}{\alpha_i} x_i^2 \right)^{\frac{1}{2}},$$

und der Diffusionsfluss ist nach [DFM99] in dieser Norm minimal.

Weitere, insbesondere numerische Ergebnisse hierzu liegen nicht vor. Der Geschwindigkeitsvorteil der Dimension-Exchange-Verfahren für $\alpha = \frac{1}{2}$ geht aber sicherlich verloren. Die Stabilitätsvorteile dürften dagegen für moderate Kantengewichte (α_i nicht zu nahe bei 0) erhalten bleiben.

7.2 Knotengewichte

Nicht nur die Kanten, sondern auch die Knoten lassen sich mit Gewichten versehen. Hiermit lässt sich die Situation verschieden leistungsfähiger Prozessoren innerhalb eines Clusters beschreiben. Die Gewichte werden mit $c_1, \ldots, c_n$ bezeichnet. Statt der üblichen Gleichverteilung $\overline{w}$ wird dann eine gewichtete Zielverteilung $\overline{\overline{w}}$ berechnet: $\overline{\overline{w}}_i = \frac{c_i}{\overline{c}}\overline{w}_i$ mit $\overline{c} = \sum_{i=1}^{n} c_i$. Die Ergebnisse in [EMP00] hierzu lassen sich ebenfalls problemlos auf den Dimension-Exchange-Fall verallgemeinern.

$$C = \operatorname{diag}(c_1, \ldots, c_n) \qquad \hat{A}^{\mathsf{ALG}} = C^{-1}A$$
$$\hat{L}^{\mathsf{ALG}} = A\hat{A}^{\mathsf{ALG}^T} = L^{\mathsf{ALG}}C^{-1} \quad \hat{M}^{\mathsf{ALG}} = I - \alpha\hat{L}^{\mathsf{ALG}}$$

Das knotengewichtete ALG-FOS berechnet damit die Werte

$$w^k = \hat{M}^{\mathsf{ALG}}w^{k-1}, \qquad x^k = x^{k-1} + \alpha\hat{A}^{\mathsf{ALG}^T}w^{k-1}, \qquad k = 1, 2, \ldots .$$

Weitere Verfahren lassen sich ähnlich konstruieren. Allerdings liegen zur Konvergenz dieser Dimension-Exchange-Arten bisher weder theoretische noch experimentelle Ergebnisse vor. Die Untersuchung wird hier dadurch erschwert, dass $\hat{M}^{\mathsf{ALG}}$ nicht mehr doppelt stochastisch ist und der Eigenvektor $\overline{\overline{w}}$ zum Eigenwert 1 damit auch nicht mehr senkrecht auf den übrigen Eigenvektoren steht. Der Geschwindigkeitsvorteil durch viele zusammenfallende Eigenwerte wird durch die Knotengewichtung genauso zunichte gemacht wie bei gewichteten Kanten.

8 Zusammenfassung der Ergebnisse

Zur Lösung des Loadbalancing-Problems wurden zahlreiche Verfahren aus den zwei Klassen Diffusion und Dimension-Exchange untersucht und verglichen; dabei wurden sowohl endliche als auch nicht-endliche Varianten betrachtet. Das Hauptaugenmerk lag hierbei auf den in dieser Arbeit neu eingeführten endlichen Dimension-Exchange-Verfahren.

Die wichtigsten neu eingeführten Loadbalancing-Verfahren sind das auf einer Dreitermrekursion basierende DE-OPS und das einfachere DE-OPT aus Abschnitt 3.5. Zusätzlich wurden mit SDE-OPS und SDE-OPT symmetrische Varianten konstruiert.

In Abschnitt 3.6 konnten für fast alle Standardgraphen Formeln für die Eigenwerte der Iterationsmatrizen M^{DE} und M^{SDE} bestimmt werden, wenn auch zum Teil nur für den wichtigen Fall des Kantengewichtes $\alpha = \frac{1}{2}$. Die Anzahl der von den Verfahren benötigten Kommunikationsschritte und damit die Laufzeit der Verfahren hängt direkt ab von der Anzahl verschiedener Eigenwerte. Über die Berechnung der Eigenwerte konnte so gezeigt werden, dass die Laufzeiten der Dimension-Exchange-Verfahren in vielen Fällen unter denen der Diffusionsvarianten liegen. Insbesondere bei sehr regelmäßig strukturierten Graphen ergeben sich hier oft Verbesserungen um Faktoren zwei oder vier wie bei Zyklen oder Gittern und Tori; aber auch bei unstrukturierten Graphen hat Dimension-Exchange kleinere Vorteile. Nicht zu empfehlen ist Dimension-Exchange lediglich bei solchen stark strukturierten Graphen, die sich nur unregelmäßig einfärben lassen, wie Tori ungerader Dimension, oder die zu große Knotengrade aufweisen wie der komplette Graph oder der Stern. In einigen konkreten Fällen kann man die Laufzeit mit Dimension-Exchange sogar minimieren, zum Teil gelingt dies auch mit der hier ebenfalls neu vorgestellten Vorwärts-Rückwärts-Implementierung von Diffusionsverfahren (FB-OPX), vergleiche Tabelle 3.4.

Während Diffusionsverfahren vereinzelt numerische Probleme aufweisen, ist die Konvergenz bei Dimension-Exchange praktisch nie gefährdet, sofern man zur Anordnung der Eigenwerte des Graphen gewichtete Leja-Punkte verwendet.

Als grundsätzlicher Nachteil von Dimension-Exchange sind die im Allgemeinen nichtminimalen Flüsse zu nennen, so dass im Vergleich zur Diffusion mehr Lasten über weitere Wege verschoben werden. Durch Anwendung der vorgestellten Flussreduktionstechniken (DE-OPTfb, DE-OPTcc) lässt sich dieser Effekt jedoch deutlich reduzieren. Das zyklische Durchlaufen der Farben aus Abschnitt 3.7.2 schafft dies sogar, ohne die Laufzeit des Verfahrens nennenswert zu verlängern. Eine genaue Analyse der produzierten Flüsse wurde in Abschnitt 3.8 vorgenommen. Für einen gegebenen Graphen und eine bestimmte Verfahrensklasse ALG (Diffusion, symmetrisches / unsymmetrisches Dimension-Exchange oder eine der Varianten zur Flussreduktion) lässt sich die Höhe des Flusses im Worst Case direkt berechnen. Gemäß Satz 3.36 liegt die l_2-Norm des Flusses im schlimms-

ten Fall genau um den Faktor $\left\| A^{\mathsf{ALG}^T} L^{\mathsf{ALG}^+} A \right\|_2$ über dem Minimum. Diese Normen konnten für alle wichtigen Graphen unabhängig von deren Größe und den wichtigen Fall $\alpha = \frac{1}{2}$ bestimmt werden, teils wurden die Werte theoretisch berechnet und teils numerisch ermittelt; aufgelistet sind diese in Tabelle 3.3.

Speziell angepasst an die Struktur von Produktgraphen wie Gitter und Tori sind die ADI-Verfahren aus Kapitel 4. Es wurden nicht nur erstmals Dimension-Exchange-Verfahren für Produktgraphen konstruiert, sondern auch ein neues Diffusionsverfahren. Durch die Berechnung zusätzlicher Zwischenwerte ließ sich auch aus dem schnellen und ziemlich stabilen OPS ein ADI-Algorithmus konstruieren, ohne dass die Laufzeiten spürbar über denen von ADI-OPT liegen. Der Aufwand der DE-ADI-Verfahren ist bei Gittern und Tori gerader Dimension (ggf. bis auf eine von der Größe des Graphen unabhängige Zahl von zwei oder vier Schritten) optimal (siehe Tabelle 4.3). Das schon für das DE-OPT-Verfahren erfolgreiche zyklische Durchlaufen der Farben zur Reduzierung der Flüsse lässt sich bei ADI-Verfahren auf die Richtungen übertragen (ADC-Verfahren). Durch die Wahl der Gewichtsfunktion der Leja-Sortierung bei (DE-)ADI-OPT und den Exponenten η im Innenprodukt des (DE-)ADI-OPS sind weitere Verringerungen der Flüsse möglich.

Für die zweite Phase des Loadbalancings, das Scheduling, wurde ein neues Verfahren vorgestellt, das auf der für Dimension-Exchange benötigten Einfärbung des Graphen beruht. Experimente haben gezeigt, dass sich hiermit bei höheren Flüssen mit vielen oder sehr großen Lasten Zeitvorteile erzielen lassen; allerdings gilt dies nicht in allen Situationen. In praktisch relevanten Fällen unterscheiden sich alle vier getesteten Verfahren kaum. Die Ergebnisse haben jedoch auch gezeigt, dass das Scheduling bei sehr vielen oder sehr großen Lasten wesentlich länger dauern kann als das Loadbalancing-Verfahren davor; Versuche, den gesamten Prozess weiter zu beschleunigen, müssen also hier ansetzen.

Literaturverzeichnis

[ALO02] Götz Alefeld, Ingrid Lenhardt und Holger Obermaier. *Parallele numerische Verfahren*. Springer, Berlin, 2002.

[Arj82] Eshrat Arjomandi. An Efficient Algorithm for Colouring the Edges of a Graph with $\Delta + 1$ Colours. *INFOR*, 20(2):82–101, Mai 1982.

[CDS95] Dragoš M. Cvetković, Michael Doob und Horst Sachs. *Spectra of Graphs: Theory and Applications*. Johann Ambrosius Barth Verlag, Heidelberg, Leipzig, 3. Auflage, 1995.

[Cro97] Thomas W. Crockett. An introduction to parallel rendering. *Parallel Computing*, 23:819–843, 1997.

[CW85] Jane K. Cullum und Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol. I Theory*. Progress in Scientific Computing Vol. 3. Birkhäuser, Boston, Basel, Stuttgart, 1985.

[Cyb89] George Cybenko. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[Dav79] Philip J. Davis. *Circulant Matrices*. Pure and Applied Mathematics. John Wiley & Sons, New York, Chichester, Brisbane, Toronto, 1979.

[DDNR96] Ralf Diekmann, Uwe Dralle, Friedhelm Neugebauer und Thomas Römke. PadFEM: A Portable Parallel FEM-Tool. In *HPCN Europe 96*, Brüssel, April 1996.

[DFM98] Ralf Diekmann, Andreas Frommer und Burkhard Monien. Nearest Neighbor Load Balancing on Graphs. In G. Bilardi, G. Italiano, A. Piertracaprina und G. Pucci, Hrsg., *6th European Symposium on Algorithms (ESA '98)*, Lecture Notes in Computer Science, No. 1461, Seiten 429–440. Springer-Verlag, 1998.

[DFM99] Ralf Diekmann, Andreas Frommer und Burkhard Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25:789–812, 1999.

[DMP99] Ralf Diekmann, Burkhard Monien und Robert Preis. Load Balancing Strategies for Distributed Memory Machines. In B.H.V. Topping, Hrsg., *Parallel and Distributed Processing for Computational Mechanics: Systems and Tools*, Seiten 124–157. Saxe-Coburg Publications, 1999.

[EFMP99] Robert Elsässer, Andreas Frommer, Burkhard Monien und Robert Preis. Optimal and Alternating-Direction Loadbalancing Schemes. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud und D. Ruiz, Hrsg., *Euro-Par'99 Parallel Processing*, Lecture Notes in Computer Science, No. 1685, Seiten 280–290. Springer-Verlag, 1999.

[Els02] Robert Elsässer. *Spectral Methods for Efficient Load Balancing Strategies*. Dissertation, Universität Paderborn, Fachbereich Mathematik / Informatik, Februar 2002.

[EMP00] Robert Elsässer, Burkhard Monien und Robert Preis. Diffusive Load Balancing Schemes on Heterogeneous Networks. In *12th ACM Symposium on Parallel Algorithms and Architectures (SPAA) 2000*, Seiten 30–38, 2000.

[Fel97] Rainer Feldmann. Computer Chess: Algorithms and Heuristics for a Deep Look into the Future. In F. Plasil und K. G. Jeffrey, Hrsg., *SOFSEM'97: Theory and Practice of Informatics*, Lecture Notes in Computer Science, No. 1338, Seiten 1–18. Springer-Verlag, 1997.

[FM98] Rainer Feldmann und Burkhard Monien. Selective Game Tree Search on a Cray T3E. Technical report, University of Paderborn, November 1998.

[FW78] Stanley Fiorini und Robin J. Wilson. Edge-Colorings of Graphs. In Lowell W. Beineke und Robin J. Wilson, Hrsg., *Selected Topics in Graph Theory*, Kapitel 5, Seiten 103–126. Academic Press, London, 1978.

[GL96] Gene H. Golub und Charles F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, London, 3. Auflage, 1996.

[HB95] Y. F. Hu und R. J. Blake. An Optimal Dynamic Load Balancing Algorithm. Daresbury Laboratory Report DLP 95010, Daresbury Laboratory, 1995.

[HB99] Y. F. Hu und R. J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25:417–444, 1999.

[HJ85] Roger A. Horn und Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[HJ91] Roger A. Horn und Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[HLM+90] Seyed H. Hosseini, Bruce Litow, Mohammad I. Malkawi, Joseph McPherson und K. Vairavan. Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm. *Journal of Parallel and Distributed Computing*, 10:160–166, 1990.

[Iji65] Yuji Ijiri. On the Generalized Inverse of an Incidence Matrix. *Journal of the Society for Industrial and Applied Mathematics*, 13(3):827–836, September 1965.

[LM92] Reinhard Lüling und Burkhard Monien. Load Balancing for Distributed Branch & Bound Algorithms. In *Proceedings of the 6th International Processing Symposium (IPPS '92)*, Seiten 543–549, 1992.

[MGS96] S. Muthukrishnan, Bhaskar Ghosh und Martin H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *SPAA '96. Proceedings of the 8th annual ACM symposium on Parallel algorithms and architectures*, Seiten 72–81, 1996.

[Pad] PadFEM: Parallel Adaptive FEM. `www.upb.de/pc2/projects/padfem`.

[Rei91] Lothar Reichel. The Application of Leja Points to Richardson Iteration and Polynomial Preconditioning. *Linear Algebra and its Applications*, 154–156:389–414, 1991.

[SR99] Olaf Schmidt und Ludger Reeker. New Dynamic Load Balancing Strategy for Efficient Data-Parallel Radiosity Calculations. In H. R. Arabnia, Hrsg., *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Seiten 532–538. CSREA Press, 1999.

[vdSD02] Aad J. van der Steen und Jack J. Dongarra. Overview of Recent Supercomputers, 2002. `www.top500.org/ORSC/2002`.

[XL92] Chengzhong Xu und Francis C. M. Lau. Analysis of the Generalized Dimension Exchange Method for Dynamic Load Balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, Dezember 1992.

[XL95] Chengzhong Xu und Francis C. M. Lau. The Generalized Dimension Exchange Method for Load Balancing in k-ary n-cubes and Variants. *Journal of Parallel and Distributed Computing*, 24(1):72–85, Januar 1995.