Berichts-Nr.: **DLR-IB 224-2011 A 60**

Verfasser: Barbara Brandfass

Titel: **Optimierung der MPI-Prozess-Kommunikation bei Multicore-Rechnerarchitekturen mit Hilfe des quadratischen Zuordnungsproblems**

Diplomarbeit

Datum: November 2011

Auftraggeber:

Auftrags-Nr.:

Vorgesehen für:

Der Bericht umfasst:

77 Seiten
19 Bilder
16 Tabellen
46 Literaturstellen

D L R

Institut für Aerodynamik und Strömungstechnik
Abteilung $C^2A^2S^2E$
Bunsenstr. 10
37073 Göttingen
Germany

Kurzfassung:

Diese Arbeit beschäftigt sich mit der Optimierung der MPI-Prozess-Kommunikation bei Multicore-Rechnerarchitekturen. Ziel ist es, die Gesamtrechenzeit einer parallelen Anwendung dadurch zu reduzieren, dass die Prozesse so den einzelnen Prozessoren zugeordnet werden, dass die Kommunikation zwischen den Prozessen mit möglichst wenig Aufwand durchgeführt werden kann. Diese Aufgabe lässt sich mathematisch als quadratisches Zuordnungsproblem modellieren. Die vorliegende Arbeit setzt sich mit den theoretischen Grundlagen des quadratischen Zuordnungsproblems auseinander und stellt anhand von verschiedenen Heuristiken eine mögliche Lösungsvariante für die oben beschriebene Problematik dar. Anhand des DLR TAU-Codes und einigen ausgewählten aerodynamischen Konfigurationen wird die praktische Anwendbarkeit der Heuristiken untersucht.

**DEUTSCHES ZENTRUM FÜR LUFT-
UND RAUMFAHRT E.V.**

Institut für Aerodynamik und Strömungstechnik

Institutsleiter:

(Prof. Dr. Andreas Dillmann)

Verfasser:

(Barbara Brandfass)

Abteilungsleiter:

(Dr. Dieter Schwamborn)

Datum: 22.11.2011	**Abteilung:**	**Bericht:**
Bearbeitet: Barbara Brandfass	$C^2A^2S^2E$	DLR-IB 224-2011 A 60

Optimierung der MPI-Prozess-Kommunikation bei Multicore-Rechnerarchitekturen mit Hilfe des quadratischen Zuordnungsproblems

Diplomarbeit

vorgelegt von

Barbara Brandfass

aus Göttingen

angefertigt im

Institut für Numerische und Angewandte Mathematik

der Georg-August-Universität zu Göttingen

2010

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Problematik der optimalen Platzierung von Prozessen auf verfügbaren Prozessoren. Ziel ist es, die Gesamtrechenzeit einer parallelen Anwendung dadurch zu reduzieren, dass die Prozesse so den einzelnen Prozessoren zugeordnet werden, dass die Kommunikation zwischen den Prozessen mit möglichst wenig Aufwand durchgeführt werden kann. Diese Aufgabe lässt sich mathematisch als *quadratisches Zuordnungsproblem* modellieren und stellt den Kern dieser Arbeit dar.

Das quadratische Zuordnungsproblem ist ein stark NP-schweres kombinatorisches Optimierungsproblem. Im Allgemeinen können schon Probleme der Größe $n > 20$ nicht mehr in akzeptabler Zeit exakt gelöst werden [6]. Bei parallelen Anwendungen kommen jedoch meist deutlich mehr als 64 CPUs zum Einsatz. Für eine Optimierung der Prozess-Kommunikation in Form eines quadratischen Zuordnungsproblems muss daher auf Heuristiken zurückgegriffen werden, mit denen sich suboptimale Lösungen in deutlich kürzerer Zeit berechnen lassen.

Diese Arbeit setzt sich mit den theoretischen Grundlagen des quadratischen Zuordnungsproblems auseinander und stellt anhand von verschiedenen Heuristiken eine mögliche Lösungsvariante für die oben beschriebene Problematik dar. Anhand des DLR TAU-Codes und einigen ausgewählten aerodynamischen Konfigurationen wird die praktische Anwendbarkeit der Heuristiken untersucht.

Die Arbeit ist wie folgt gegliedert: In Kapitel 2 wird eine Beschreibung des Problems aus technischer Sicht gegeben sowie eine grundlegende Beschreibung des DLR TAU-Codes. Kapitel 3 beinhaltet die mathematische Beschreibung sowie Komplexitätsuntersuchungen des quadratischen Zuordnungsproblems. Weiterhin werden alternative Formulierungen und untere Schranken für das Problem vorgestellt. In Kapitel 4 wird gezeigt, wie sich die Optimierung der Prozess-Kommunikation als quadratisches Zuordnungsproblem modellieren lässt und die resultierende Problemstruktur untersucht. Eine Beschreibung der Heuristiken, welche bezüglich Lösungsqualität und Laufzeit untersucht wurden, erfolgt in Kapitel 5. Die Ergebnisse dieser Untersuchungen als auch die Ergebnisse der Prozess-Kommunikations-Optimierung für den DLR TAU-Code finden sich in Kapitel 6.

2 Motivation

2.1 Problemstellung

Die in der Einleitung kurz beschriebene Problematik der optimalen Platzierung von Prozessen auf verfügbaren Prozessoren taucht fast immer bei Anwendungen aus dem Bereich des wissenschaftlichen Rechnens auf, die auf eine möglichst große Rechenkapazität angewiesen sind und aus diesem Grund auf sehr vielen Prozessoren laufen. Hierzu zählen z. B. Simulationsanwendungen aus dem Bereich der Klimaforschung oder auch der Flugzeug- bzw. der Automobilentwicklung [3].

Damit diese Anwendungen auch effizient auf hunderten von CPU-Cores rechnen können, sind die jeweiligen Programmpakete parallelisiert. Die Parallelisierung erfolgt in der Regel mittels klassischer Gebietszerlegung der Rechennetze, wobei zur Kommunikation zwischen den Prozessen das Message Passing Interface (MPI) [34] zum Einsatz kommt. Für eine effiziente Ausnutzung der vorhandenen Rechenkerne ist es erforderlich, dass pro CPU-Rechenkern auch ein Rechennetz zur Verfügung steht.

Obwohl durch die Parallelisierung von Programmabläufen deren Rechenzeit deutlich verringert werden kann, bedeutet die Hinzunahme von weiteren Prozessoren auch immer einen Zuwachs an Kommikationsaufwand zwischen den einzelnen Prozessoren (vgl. [1]).

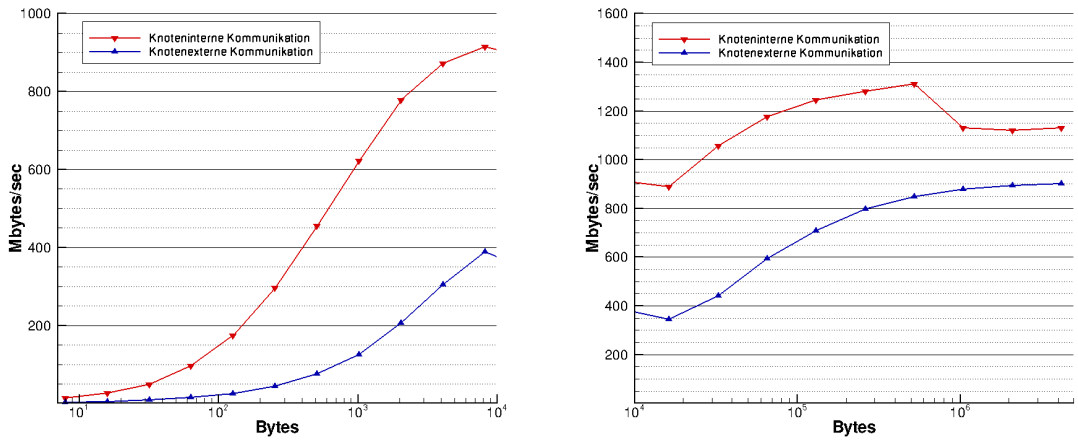
Ziel ist es, diesen Kommunikationsaufwand und somit auch die Gesamtrechenzeit einer Anwendung dadurch zu reduzieren, dass die MPI-Prozesse so auf die einzelnen CPU-Cores verteilt werden, dass die größte Kommunikationslast der Prozessoren innerhalb ihrer eigenen Rechenknoten liegt. Die Annahme ist hier, dass bei Rechenclustern mit Multicore-Architektur die knoteninterne Kommunikation schneller ist als die knotenexterne über das Hochgeschwindigkeitsnetzwerk. Diese Annahme wird unterstützt durch die in Tabelle 2.1 angegebenen Kommunikationszeiten für den MPI-PingPong Benchmark [18] auf einem typischen Rechencluster (vgl. auch Abbildung 2.1). Es wird ersichtlich, dass die knoteninterne Kommunikation um den Faktor 2–8 schneller als die knotenexterne über das Hochgeschwindigkeitsnetzwerk (in diesem Fall Infiniband Single Data Rate (SDR) [23]) ist.

Aufgrund der technischen Entwicklung findet heutzutage keine nennenswerte Steigerung des CPU-Taktes (und damit der Rechenleistung) mehr statt, sondern es wird vielmehr in jeder neuen CPU-Generation die Anzahl der verfügbaren CPU-Cores pro Rechenknoten

Größe	Kommunikationszeit intern μs	extern μs	Faktor
64 B	0,62	5,12	8,25
1 kB	1,51	9,10	6,03
32 kB	28,44	88,81	3,12
512 kB	365,62	961,11	2,63

Tabelle 2.1: Ergebnisse für MPI-PingPong Benchmark auf dem DLR Enigma Cluster [13] (AMD-Barcelona 1,9 GHz, 8 Cores per Node, Infiniband SDR).

gesteigert. Dieser Trend in der Chipherstellung existiert seit 2005 und setzt sich mittlerweile in fast allen Einsatzgebieten fort. So wurden im Jahr 2005 noch überwiegend Single-Core-Prozessoren [12] mit x86 Technologie ausgeliefert. Diese wurden 2006 durch Dual-Cores [13, 26] abgelöst und seit 2008 liegt die übliche Core-Anzahl neuer Prozessoren bei 4 Cores [14, 42] bzw. seit 2009 bei 6 Cores [36, 11]. Seit dem 1. Quartal 2010 sind sogar schon CPU-Chips mit bis zu 12 Cores pro Prozessor verfügbar [37].



(a) Kleinere Nachrichten bis 10 KB

(b) Größere Nachrichten bis 4 MB

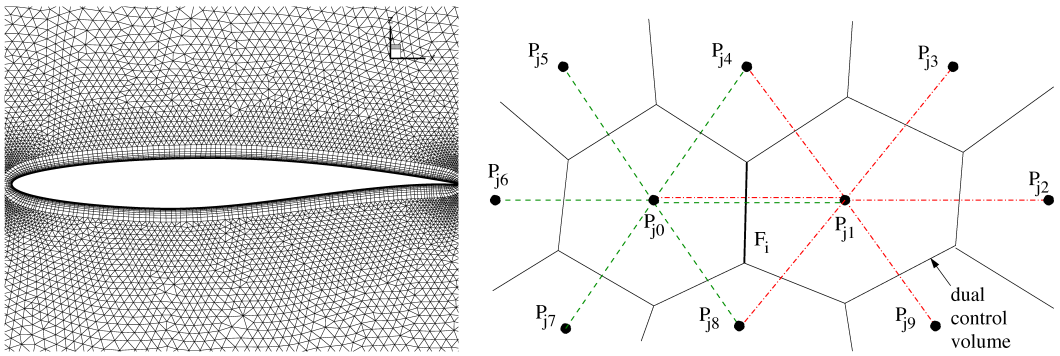
Abbildung 2.1: Vergleich der Bandbreite für die knoteninterne und knotenexterne Kommunikation. Ergebnisse für MPI-PingPong Benchmark auf dem Enigma Cluster [13].

Es ist anzunehmen, dass die Problematik der optimalen Prozesszuordnung mit steigender Prozessoranzahl zunehmend wichtiger wird. Anhand des DLR TAU-Codes und entsprechender aerodynamischer Konfigurationen wird sich diese Arbeit mit einer möglichen Lösung dieser Problematik befassen.

2.2 Der DLR TAU-Code

Der DLR TAU-Code ist ein modernes Software-System für die Simulation von komplexen Geometrien in reibungsfreien bzw. reibungsbehafteten Strömungen. Der Anwendungsbereich erstreckt sich von kleinen Mach-Zahlen bis hin zum Hyperschallbereich und deckt damit den überwiegenden Teil kompressibler Außenumströmungen ab. Der TAU-Code kann nicht nur mit komplett unstrukturierten Netzen umgehen, sondern auch mit sogenannten hybriden Netzen, die eine Mischung aus strukturierten und unstrukturierten Elementen darstellen (siehe Abbildung 2.2(a)). Der Einsatz von hybriden Netzen und den damit verbundenen semi-unstrukturierten Elementen wie Prismen oder Hexaedern ist überall dort sinnvoll, wo eine gleichmäßige höhere Auflösung bzw. Diskretisierung über reibungsbehafteten Wänden gefordert ist (z. B. im Grenzschichtbereich). Anders als bei einigen anderen Strömungssimulationscodes verfügt der TAU-Code nicht über ein eigenes Netzgenerierungswerkzeug. Es ist daher erforderlich, Programme anderer Anbieter (z. B. Centaur [7]) zu nutzen.

Wie viele andere unstrukturierte Simulationscodes auch (z. B. der NSU3D von Mavriplis [33]), basiert der TAU-Code auf einem Finite-Volumen-Ansatz. Die räumliche Diskretisierung erfolgt knotenbasiert (cell-vertex) und für die Berechnung der Kontrollvolumen eines jeden Netzpunktes ist die Anwendung des TAU-Preprocessings notwendig. Hierbei werden für jeden Primärgitterknoten die zugehörigen Kontrollvolumina, auch duale Zellen genannt, berechnet (siehe Abbildung 2.2(b)). Durch die Verwendung des Ansatzes eines dualen Gitters, bezüglich dessen das Verfahren zellzentriert arbeitet, ist der Code unabhängig von den unterschiedlichen Elementen eines Netzes einsatzfähig (siehe auch Beschreibung in [30] Kapitel 5, Seite 81 ff.).



(a) Hybrides Netz

(b) Duale Metrik zweier Netzpunkte

Abbildung 2.2: (a) RAE-2822 Flügelprofil und (b) Skizze der Kontrollvolumina

Die Entkopplung und Linearisierung der nichtlinearen gekoppelten Gleichungen für die Strömungsgrößen (und gegebenenfalls für die Turbulenzgrößen) kann für stationäre Probleme durch Iteration in einer Pseudo-Zeit mittels eines expliziten K-stufigen Runge-Kutta-Verfahrens erfolgen. Alternativ ist es möglich, ein semi-implizites LU-SGS-Verfahren zu verwenden, basierend auf der Arbeit von Yoon und Jameson [46].

Für die zeitgenaue Simulation steht ein duales Zeitschritt-Verfahren nach Jameson zur Verfügung [24], bei dem die Semidiskretisierung in der Zeit mittels einer *Backward-Differencing Formula* höherer Ordnung durchgeführt wird. In jedem Zeitschritt werden die entstandenen stationären Probleme mittels expliziten K-stufigen Runge-Kutta-Verfahren oder semi-impliziten LU-SGS-Verfahren gelöst.

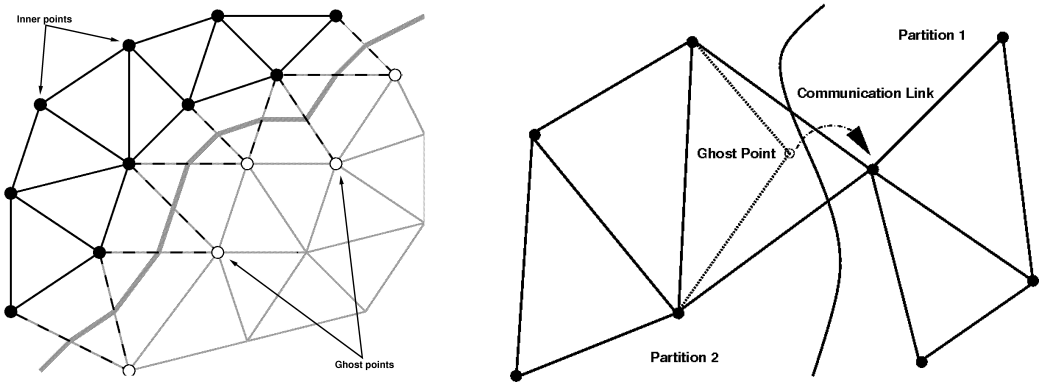
Einen guten Überblick über den aktuellen Stand der Entwicklung sowie den Anwendungsbereich des TAU-Codes gibt die Dissertation von Alrutz [2] und der Übersichtsvortrag von Schwamborn et al. [41]. Weitere Details zum TAU-Code finden sich in [15, 16, 28].

2.3 Kommunikationsverhalten des TAU-Strömungslösers

Die Parallelisierung des TAU-Strömungslösers basiert auf einer Gebietszerlegung des Rechennetzes. Jedem Prozess wird ein Teilnetz zugeordnet, welches aufgrund des expliziten Zeitschrittverfahrens jeweils als komplettes Netz behandelt werden kann. Während einer Flussintegration müssen jedoch wiederholt Daten zwischen verschiedenen Netzpartitionen ausgetauscht, d. h. zwischen den zugehörigen Prozessen kommuniziert werden. Dabei können zwei Arten von Kommunikation unterschieden werden: Kommunikation über alle Prozesse (globale Kommunikation) und Kommunikation zwischen einzelnen Prozessen (Punkt-zu-Punkt Kommunikation).

Die globale Kommunikation findet nur zu Monitoringzwecken statt. Hierbei kommuniziert jeder Prozess mit jedem Prozess, und sowohl die Anzahl als auch die Größe der einzelnen Nachrichten ist dabei für alle Prozesse gleich.

Eine Punkt-zu-Punkt Kommunikation zwischen den Prozessen ist immer dann notwendig, wenn bei Berechnungen für einen Netzpunkt Daten von benachbarten Punkten benötigt werden, die sich durch die Partitionierung auf einem anderen Teilnetz befinden. Um die Daten dieser Nachbarpunkte zugänglich zu machen, werden zusätzliche Netzpunkte, die sogenannten *ghost-points* eingeführt (siehe Abbildung 2.3). Die gefüllten Netzpunkte (•) sind dabei *innere* Punkte der linken Partition. Jeder innere Punkt der rechten Partition, welcher durch eine Kante mit einem der inneren Punkte der linken Partition verbunden ist (◦), ist ein ghost-point der linken Partition. Ist für eine Berechnung eine Aktualisierung der Daten der ghost-points erforderlich, müssen die jeweiligen



(a) Skizze Partitionierung (b) Skizze Erzeugung von *ghost-points*

Abbildung 2.3: Parallelisierung des TAU-Strömungslösers

Variablen der zugrundeliegenden Punkte der Nachbarpartition mittels Punkt-zu-Punkt Kommunikation der entsprechenden Prozesse übertragen werden. Die Anzahl der zu übermittelnden Variablen kann dabei je nach verwendetem Turbulenzmodell variieren.

Da nicht nur die globalen Kommunikationen sondern auch die Punkt-zu-Punkt Kommunikationen immer für alle Prozesse gleichzeitig durchgeführt werden, ist die Gesamtanzahl der Kommunikationsaufrufe für jeden Prozess gleich. Allerdings kann bei einem Kommunikationsaufruf in der Punkt-zu-Punkt Kommunikation eine unterschiedliche Anzahl von Nachrichten übertragen werden. Die Anzahl der zu übertragenden Nachrichten entspricht dabei der Anzahl der Nachbarpartitionen.

3 Das quadratische Zuordnungsproblem

Das "quadratische Zuordnungsproblem" beschreibt die eineindeutige Zuordnung von n Objekten, zwischen denen bestimmte Flussbeziehungen bestehen, zu n Orten, die bestimmte Entfernungen voneinander haben. Ziel ist es, die Objekte so an den Orten zu platzieren, dass die Gesamtflusskosten, welche sich als Summe der Produkte der Intensitäten der Flussbeziehungen und der Entfernungen zwischen den einzelnen Objekten ergeben, minimiert werden.

3.1 Mathematische Formulierung des Problems

Sei a_{ij} die Größe des Flusses von Objekt i zu Objekt j und b_{kl} die Entfernung von Ort k zu Ort l. Wird Objekt i dem Ort k und Objekt j dem Ort l zugeordnet, so sind die Kosten für den Fluss von Objekt i zu Objekt j proportional zu $a_{ij} \cdot b_{kl}$.

Sei $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ eine bijektive Abbildung, welche die Zuordnung von Orten i zu Objekten $\pi(i)$ beschreibt. Dann betragen die Gesamtflusskosten, welche sich durch diese Zuordnung ergeben

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij}$$

Der Einfachheit halber wurde hierbei der Proportionalitätsfaktor auf 1 gesetzt.

Eine sehr umfassende Abhandlung über das quadratische Zuordnungsproblem stellt das Buch "The Quadratic Assignment Problem: Theory and Algorithms" von Çela [6] dar. Es diente als Vorlage für die folgenden Definitionen und Bezeichnungen.

Definition 3.1 (Quadratisches Zuordnungsproblem)
Gegeben seien eine Menge $N = \{1, 2, \ldots, n\}$ und zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ mit $A = (a_{ij})$ und $B = (b_{ij})$. Dann wird das *quadratische Zuordnungsproblem* mit Koeffizientenmatrizen A und B beschrieben durch

$$\min_{\pi \in \mathcal{S}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij} \tag{3.1}$$

wobei $\mathcal{S}_n$ die Menge aller Permutationen der Menge N sei.

Bemerkung 3.2 (QAP) Im Folgenden wird das quadratische Zuordnungsproblem der Einfachheit halber mit *QAP* (von englisch: Quadratic Assignment Problem) abgekürzt, das quadratische Zuordnungsproblem mit Koeffizientenmatrizen A und B analog mit *QAP(A,B)*. Die Matrix A wird auch als *Flussmatrix* und die Matrix B als *Distanzmatrix* bezeichnet.

Definition 3.3 (Zielfunktion für das QAP)
Die *Zielfunktion* Z für ein QAP(A,B) ist gegeben durch

$$Z(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij} \tag{3.2}$$

Um die Abhängigkeit des Zielfunktionswertes von den Eingaben A und B zu verdeutlichen, schreibt man auch $Z(A, B, \pi)$.

Der initiale Zielfunktionswert, welcher sich ohne Permutation der Matrix A ergibt, wird mit $Z(A, B)$ bezeichnet. Es ist $Z(A, B) := Z(A, B, id)$ für die identische Permutation id.

Bemerkung 3.4 (Symmetrisches QAP) Ist eine oder sind beide der Matrizen A, B symmetrisch, so nennt man QAP(A,B) ein *symmetrisches* quadratisches Zuordnungsproblem.

3.2 Alternative Formulierungen für das QAP

Für viele kombinatorische Optimierungsprobleme gibt es mehrere unterschiedliche aber äquivalente mathematische Beschreibungen. Da bei diesen verschiedenen Formulierungen oft auch unterschiedliche strukturelle Eigenschaften des Problems zutage treten, können sich daraus verschiedene Ansätze zur Problemlösung ergeben.

Die Äquivalenz der folgenden beiden Formulierungen des QAP zu der in Definition 3.1 gegebenen Formulierung beruht im Wesentlichen auf der Beziehung zwischen Permutationen und Permutationsmatrizen.

Definition 3.5 (Permutationsmatrix)
Eine $n \times n$-Matrix $X = (x_{ij})$ heißt *Permutationsmatrix*, wenn für die Einträge x_{ij} folgende Bedingungen gelten:

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1 \qquad \text{für } \ 1 \le j \le n \\ \sum_{j=1}^{n} x_{ij} &= 1 \qquad \text{für } \ 1 \le i \le n \\ x_{ij} &\in \{0, 1\} \qquad \text{für } \ 1 \le i, j \le n \end{aligned} \tag{3.3}$$

Die Menge aller $n \times n$-Permutationsmatrizen wird mit Π_n bezeichnet.

Beobachtung 3.6 Jede Permutation $\pi \in \mathcal{S}_n$ kann eindeutig mit der Permutationsmatrix $X \in \Pi_n$ assoziiert werden für deren Einträge x_{ij} gilt

$$x_{ij} = \begin{cases} 1 & \text{falls } \pi(i) = j \\ 0 & \text{sonst} \end{cases} \tag{3.4}$$

Umgekehrt lässt sich auch zu jeder Permutationsmatrix $X \in \Pi_n$ durch

$$\pi(i) = j \;\Leftrightarrow\; x_{ij} = 1 \tag{3.5}$$

eindeutig die assoziierte Permutation $\pi \in \mathcal{S}_n$ konstruieren.

Bemerkung 3.7 Sei $X \in \Pi_n$. Dann gilt:

1. X ist invertierbar.
2. X ist orthogonal, d.h. $X^{-1} = X^t$
3. Ist X assoziiert zu $\pi \in \mathcal{S}_n$, dann ist X^t assoziiert zu π^{-1}.

Beobachtung 3.8 Seien $v \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $\pi \in \mathcal{S}_n$ und $X \in \Pi_n$ die zu π assoziierte Permutationsmatrix. Bezeichne v^π den Vektor, welcher durch eine Umordnung der Elemente von v entsprechend der Permutation π entsteht. Dann ist

$$v^\pi := \left(v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}\right)^t = Xv \qquad \text{und} \qquad (v^\pi)^t = v^t X^t$$

Analog werden durch Bilden des Produkts XA die Zeilen und durch AX^t die Spalten von A entsprechend der Permutation π vertauscht und es ist

$$XAX^t = \left(a_{\pi(i)\pi(j)}\right) \tag{3.6}$$

3.2.1 Die Koopmans-Beckmann-Formulierung

In der Literatur findet sich die erste Beschreibung des QAP 1957 bei Koopmans und Beckmann [29]. Sie formulierten das QAP ursprünglich als 0-1-Optimierungsproblem mit quadratischer Zielfunktion.

Definition 3.9 (Koopmans-Beckmann-Formulierung des QAP)
Seien $A, B \in \mathbb{R}^n$ mit $A = (a_{ij})$ und $B = (b_{ij})$. Für $X = (x_{ij})$ ist dann QAP(A,B) gegeben durch

$$\min_{X\in\Pi_n} \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij}\cdot b_{kl}\cdot x_{ki}\cdot x_{lj} \tag{3.7}$$

In dieser Formulierung wird durch die Bedingungen (3.3), die eine Permutationsmatrix X erfüllen muss sichergestellt, dass jedem Objekt jeweils nur ein Ort zugeordnet wird. Das Objekt j wird genau dann dem Ort i zugeordnet, wenn der Wert $x_{ij} = 1$ ist. Damit geht der Term $a_{ij} \cdot b_{kl} \cdot x_{ki} \cdot x_{lj}$ genau dann mit dem Wert $a_{ij} \cdot b_{kl}$ in den Zielfunktionswert ein, wenn Objekt i dem Ort k und Objekt j dem Ort l zugeordnet ist, also wenn $x_{ki} = x_{lj} = 1$ ist. Für die zu X assoziierte Permutation π gilt dann $\pi(k) = i$ und $\pi(l) = j$. Damit ist $a_{ij} \cdot b_{kl} = a_{\pi(k)\pi(l)} \cdot b_{kl}$ und diese Darstellung entspricht genau den Summanden in (3.1), was die Äquivalenz der beiden Formulierungen zeigt.

3.2.2 Die Spur-Formulierung

Die Spur-Formulierung des QAP wurde zum ersten Mal 1977 von Edwards [9] benutzt.

Definition 3.10 (Spur einer Matrix)
Die *Spur* einer $n \times n$-Matrix $A = (a_{ij})$ ist als die Summe ihrer Diagonalelemente definiert und wird mit $\mathrm{tr}(A)$ (von engl. *trace*) bezeichnet. Es ist also

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii} \tag{3.8}$$

Definition 3.11 (Spur-Formulierung des QAP)
Seien $A, B \in \mathbb{R}^{n \times n}$. Dann ist QAP(A,B) gegeben durch

$$\min_{X \in \Pi_n} \mathrm{tr}(XAX^tB^t) \tag{3.9}$$

Die Äquivalenz dieser Formulierung zu Definition 3.1 des QAP ergibt sich folgendermaßen: Seien $A = (a_{ij})$, $B = (b_{ij})$. Dann ist $XAX^t = (a_{\pi(i)\pi(j)})$ für die zu $X \in \Pi_n$ assoziierte Permutation $\pi \in \mathcal{S}_n$ nach (3.6). Für die Einträge der Matrix XAX^tB^t gilt

$$(XAX^tB^t)_{ij} = \sum_{k=1}^{n} (XAX^t)_{ik} \cdot (B^t)_{kj} = \sum_{k=1}^{n} a_{\pi(i)\pi(k)} \cdot b_{jk}$$

Somit ist

$$\mathrm{tr}(XAX^tB^t) = \sum_{i=n}^{n} (XAX^tB^t)_{ii} = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{\pi(i)\pi(k)} \cdot b_{ik}$$

Aus der Spur-Formulierung ergibt sich mit Hilfe der folgenden Rechenregeln eine interessante Eigenschaft von symmetrischen QAPs (vgl. [10]).

Bemerkung 3.12 Seien $A, B \in \mathbb{R}^{n\times n}$ und $\alpha \in \mathbb{R}$. Dann gelten die folgenden Regeln:

$$\begin{aligned} (A^t)^t &= A & (AB)^t &= B^t A^t \\ \operatorname{tr}(A) &= \operatorname{tr}(A^t) & \operatorname{tr}(\alpha A) &= \alpha \operatorname{tr}(A) \\ \operatorname{tr}(AB) &= \operatorname{tr}(BA) & \operatorname{tr}(A+B) &= \operatorname{tr}(A) + \operatorname{tr}(B) \end{aligned} \tag{3.10}$$

Beobachtung 3.13 Sei QAP(A,B) gegeben und $X \in \Pi_n$. Ist die Matrix A symmetrisch, d.h. $A = A^t$, so ist nach (3.10)

$$\operatorname{tr}(XAX^tB^t) = \operatorname{tr}(XA^tX^tB^t) = \operatorname{tr}((XA^tX^tB^t)^t) = \operatorname{tr}(BXAX^t) = \operatorname{tr}(XAX^tB)$$

wobei B nicht symmetrisch sein muss.

Satz 3.14 *Seien $A, B \in \mathbb{R}^{n\times n}$ gegeben und A symmetrisch. Dann ist QAP(A,B) äquivalent zu QAP(A,E) mit symmetrischer Matrix $E = \frac{1}{2}(B + B^t)$.*

Beweis: Sei $X \in \Pi_n$. Da A symmetrisch ist, gilt nach Beobachtung 3.13

$$\operatorname{tr}(XAX^tB^t) = \operatorname{tr}(XAX^tB)$$

und mit (3.10) folgt

$$\begin{aligned} \operatorname{tr}(XAX^tE) &= \operatorname{tr}(XAX^t\tfrac{1}{2}(B+B^t)) \\ &= \tfrac{1}{2}\operatorname{tr}(XAX^tB + XAX^tB^t) \\ &= \tfrac{1}{2}\operatorname{tr}(XAX^tB) + \tfrac{1}{2}\operatorname{tr}(XAX^tB^t) \\ &= \operatorname{tr}(XAX^tB^t) \end{aligned}$$

Also ist auch

$$\min_{X\in\Pi_n} \operatorname{tr}(XAX^tE) = \min_{X\in\Pi_n} \operatorname{tr}(XAX^tB^t)$$

Damit ist die Äquivalenz von QAP(A,E) und QAP(A,B) gezeigt. □

3.3 Lösbarkeit und Komplexität von QAPs

Quadratische Zuordnungsprobleme gehören zu den schwierigsten Problemen der kombinatorischen Optimierung. Für n Objekte und n Orte gibt es $n!$ Möglichkeiten sie einander zuzuordnen. Durch Vollenumeration, d. h. dem Berechnen aller möglichen Lösungen und Auswahl der besten, können daher nur kleine Probleme gelöst werden.

Eine etwas effizientere Möglichkeit eine Optimallösung eines QAPs zu bestimmen bieten Branch-and-Bound Verfahren. Beim Branch-and-Bound wird die Menge der zulässigen Lösungen aufgeteilt (branching) und mithilfe von oberen und unteren Schranken (bounds) diejenigen Teilmengen ausgesondert, die keine optimale Lösung enthalten. Da es, insbesondere für größere Probleme, schwierig ist, gute untere Schranken für das QAP zu bestimmen, ist aber auch diese Methode nur für vergleichsweise kleine Probleme auch praktisch anwendbar. Mit dem besten heutzutage bekannten exakten Verfahren lassen sich im Allgemeinen schon Probleme der Größe $n > 20$ nicht mehr in akzeptabler Zeit lösen [6]. In der Praxis sind aber zumeist größere Probleme zu lösen, bei der Problematik der optimalen Prozesszuordnung sind insbesondere Probleme der Größe $n \geq 64$ interessant (vgl. auch [27]). Für diese Probleme kommen daher zur Zeit nur Heuristiken in Frage, mit deren Hilfe sich suboptimale Lösungen berechnen lassen.

3.3.1 Grundbegriffe der Komplexitätstheorie

In diesem Abschnitt sollen einige Grundbegriffe und Sätze aus der Komplexitätstheorie gegeben werden, welche für die anschließende Betrachtung der Komplexität von QAPs benötigt werden. Sofern nicht anders angegeben, sind diese aus dem Buch “Introduction to Algorithms” von Cormen et al. [8] entnommen.

Mit Hilfe der Komplexitätstheorie wird versucht, algorithmische Probleme anhand des Aufwands, der zu ihrer Lösung notwendig ist, zu klassifizieren. Formal sind dabei alle Aussagen für Entscheidungsprobleme bzw. formale Sprachen gegeben.

Definition 3.15 (Problem, Optimierungsproblem, Entscheidungsproblem)
Ein *Problem* P ist eine Relation zwischen einer Menge von *Probleminstanzen* oder *Eingaben* I und einer Menge von *Lösungen* S.

Ein *Optimierungsproblem*, bei welchem ein bestimmter Wert maximiert oder minimiert werden soll, ist weiterhin durch eine *Zielfunktion* $z : S(\mathrm{i}) \to \mathbb{R}$ gekennzeichnet, wobei $S(\mathrm{i})$ die Menge der zulässigen Lösungen für die Instanz $\mathrm{i} \in I$ bezeichnet.

Ein *Entscheidungsproblem* ist eine Abbildung $D : I \to \{0, 1\}$. In diesem Fall besteht die Lösungsmenge allein aus den Werten 1 (“ja”) und 0 (“nein”).

Zu jedem Optimierungsproblem P lässt sich ein Entscheidungsproblem D_P konstruieren, indem ein Grenzwert $k \in \mathbb{R}$ für den Zielfunktionswert z betrachtet wird. Sei P ein Minimierungsproblem und x_{i}^* eine Optimallösung für Probleminstanz $\mathrm{i} \in I$, d.h. für alle $x_{\mathrm{i}} \in S(\mathrm{i})$ ist $z(x_{\mathrm{i}}^*) \leq z(x_{\mathrm{i}})$. Dann ist das zu P *assoziierte Entscheidungsproblem* D_P gegeben durch die Abbildung $D_P : I \times \mathbb{R} \to \{0, 1\}$ mit

$$D_P(\mathrm{i}, k) := \begin{cases} 1 & \text{falls } z(x_{\mathrm{i}}^*) \leq k \\ 0 & \text{falls } z(x_{\mathrm{i}}^*) > k \end{cases} \tag{3.11}$$

Definition 3.16 (Alphabet, Wort, formale Sprache)
Ein *Alphabet* Σ ist eine endliche Menge von Zeichen. Die Hintereinanderreihung endlich vieler Zeichen aus Σ heißt *Wort*. Die Menge aller Wörter wird mit Σ^* bezeichnet. Eine Teilmenge $L \subseteq \Sigma^*$ heiß *formale Sprache* über Σ.

Bemerkung 3.17 (Formale Sprachen vs. Entscheidungsproblem) Betrachtet man die Menge Σ der einzelnen Zeichen, aus denen die Eingaben I für ein Problem zusammengesetzt sind, so kann man eine Instanz $\mathrm{i} \in I$ als Wort über dem Alphabet Σ auffassen. Da ein Entscheidungsproblem $D : I \to \{0, 1\}$ vollständig durch die Probleminstanzen $\mathrm{i} \in I$ mit $D(\mathrm{i}) = 1$ charakterisiert ist, kann die Menge der Eingaben auf alle Wörter $x \in \Sigma^*$ erweitert werden, indem $D(\mathrm{i}) = 0$ für alle $\mathrm{i} \in \Sigma^* \backslash I$ gesetzt wird. Damit lässt sich D als Sprache L über dem Alphabet Σ formulieren. Es ist $L = \{x \in \Sigma^* \mid D(x) = 1\}$.

Definition 3.18 (Polynomialzeitalgorithmus)
Ein Algorithmus, dessen maximale Laufzeit bei Eingaben der Größe[1] n in $\mathcal{O}(n^k)$ für eine ganze Zahl $k \geq 1$ liegt, heißt *Polynomialzeitalgorithmus*.

Definition 3.19 (Komplexitätsklasse $\mathcal{P}$)
Die *Komplexitätsklasse* $\mathcal{P}$ ist die Menge aller Entscheidungsprobleme, die in polynomieller Zeit gelöst werden können.

Eine Sprache $L \subseteq \Sigma^*$ gehört genau dann zur Klasse $\mathcal{P}$, wenn es einen Polynomialzeitalgorithmus A gibt, der für jedes Wort $x \in \Sigma^*$ entscheiden kann, ob es zu L gehört, d.h. für den gilt

$$A(x) = \begin{cases} 1 & \text{wenn } x \in L \\ 0 & \text{sonst} \end{cases}$$

Definition 3.20 (Komplexitätsklasse NP)
Die *Komplexitätsklasse* $\mathcal{NP}$ ist die Menge aller Entscheidungsprobleme, für die die Korrektheit einer Problemlösung in polynomieller Zeit verifiziert werden kann.

[1] Die Größe einer Eingabe bezeichnet z.B. die Anzahl der Bits, die für ihre Speicherung benötigt werden.

Eine formale Sprache $L \subseteq \Sigma^*$ gehört genau dann zur Klasse $\mathcal{NP}$, wenn es einen Polynomialzeitalgorithmus A und eine Konstante $k \geq 0$ gibt, so dass gilt

$$x \in L \;\Leftrightarrow\; \exists\, y \in \Sigma^* \text{ mit } |y| \leq |x|^k, \text{ so dass } A(x,y) = 1$$

Definition 3.21 (Polynomielle Reduzierbarkeit)
Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ formale Sprachen. L_1 heißt *polynomiell reduzierbar* auf L_2 (Schreibweise: $L_1 \preceq_p L_2$), wenn es eine in polynomieller Zeit berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $x \in \Sigma_1^*$ gilt

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Lemma 3.22 *Sei $L' \in \mathcal{P}$ und $L \preceq_p L'$. Dann ist auch $L \in \mathcal{P}$.*

Definition 3.23 (NP-schwer)
Eine formale Sprache $L \subseteq \Sigma^*$ heißt *NP-schwer*, wenn für alle $L' \in \mathcal{NP}$ gilt $L' \preceq_p L$.

Lemma 3.24 *Sei L' NP-schwer und $L' \preceq_p L$. Dann ist auch L NP-schwer.*

Definition 3.25 (NP-vollständig)
Eine formale Sprache L heißt NP-vollständig, wenn gilt

1. $L \in \mathcal{NP}$ und
2. L ist NP-schwer.

Beispiel 3.26 Das Hamiltonkreisproblem (HC) ist ein NP-vollständiges Problem.

(HC) Gegeben sei ein ungerichteter Graph $G = (E, V)$ mit Knotenmenge V und Kantenmenge E.
Enthält G einen Hamiltonkreis, also einen geschlossenen Weg, der jeden Knoten genau einmal berührt?

Bemerkung 3.27 (Das P-NP-Problem) Kann die Lösung für ein Problem in polynomieller Zeit gefunden werden, so kann sie auch in polynomieller Zeit verifiziert werden, d.h. es gilt $\mathcal{P} \subseteq \mathcal{NP}$. Die bisher ungelöste große Frage ist, ob auch $\mathcal{P} = \mathcal{NP}$ gilt.

Im Allgemeinen wird davon ausgegangen, dass $\mathcal{P} \neq \mathcal{NP}$ ist, u. a. da es bisher noch niemandem gelungen ist, für ein einziges der NP-vollständigen Probleme einen Polynomialzeitalgorithmus anzugeben, der das Problem löst. Dass dieses für die Beantwortung der Frage reichen würde, besagt das folgende Lemma.

Lemma 3.28 *Ist ein beliebiges NP-vollständiges Problem in polynomieller Zeit lösbar, so gilt $\mathcal{P} = \mathcal{NP}$.*

Definition 3.29 (stark NP-schwer)
Ein NP-schweres Problem heißt *stark NP-schwer*, falls es auch dann noch NP-schwer ist, wenn eine Eingabe i nur aus Zahlen besteht, deren Größe im Verhältnis zur Eingabelänge $L(\mathrm{i})$ polynomiell beschränkt ist. Es muss also gelten $MAX(\mathrm{i}) \leq p(L(\mathrm{i}))$ für ein Polynom p, wobei $MAX(\mathrm{i})$ die Größe der größten in i vorkommenden Zahl angibt (vgl. [45]).

Bemerkung 3.30 (Komplexität von Optimierungsproblemen) Alle hier getroffenen Aussagen zur Komplexität gelten für formale Sprachen bzw. Entscheidungsprobleme. Um diese Aussagen auch auf Optimierungsprobleme übertragen zu können, wird dazu jeweils das assoziierte Entscheidungsproblem D_P betrachtet, welches "höchstens so schwer" wie das zugehörige Optimierungsproblem P ist. Das heißt, ist D_P beispielsweise NP-schwer, so sagt man, dass auch P NP-schwer ist etc.

3.3.2 Über die Komplexität von QAPs

QAPs gehören zu den stark NP-schweren Problemen. Weiterhin konnte 1976 von Sahni und Gonzales [39] gezeigt werden, dass die Existenz eines Polynomialzeitalgorithmus, der eine Näherungslösung für QAPs innerhalb einer konstanten ε-Abweichung zur Optimallösung berechnet, $\mathcal{P} = \mathcal{NP}$ impliziert.

Satz 3.31 (angelehnt an Çela [6] und Sahni & Gonzales [39])
Das quadratische Zuordnungsproblem ist stark NP-schwer.

Beweis: Im Folgenden wird gezeigt, dass sich das NP-vollständige Hamiltonkreisproblem (HC) polynomiell auf die Entscheidungsvariante des QAP für Koeffizientenmatrizen mit Einträgen aus $\{0,1,2\}$ (hier mit $D_{\mathrm{QAP}}|_{\{0,1,2\}}$ bezeichnet) reduzieren lässt. Daraus lässt sich folgern, dass $D_{\mathrm{QAP}}|_{\{0,1,2\}}$ NP-schwer und D_{QAP} *stark* NP-schwer ist. Nach Bemerkung 3.30 folgt der Satz.

Sei G ein beliebiger Graph $G = (V, E)$ mit $|V| = n$ und $V = \{v_1, v_2, \ldots, v_n\}$. Betrachte das QAP mit $n \times n$-Koeffizientenmatrizen $A = (a_{ij})$ und $B = (b_{ij})$ mit

$$a_{ij} = \begin{cases} 1, & \text{falls } (v_i, v_j) \in E \\ 2 & \text{sonst} \end{cases} \qquad b_{ij} = \begin{cases} 1, & \text{falls } \begin{array}{r} j = i+1 \text{ für } i = 1, \ldots, n-1 \\ \text{oder } i = n \ \wedge \ j = 1 \end{array} \\ 0 & \text{sonst} \end{cases}$$

B entspricht in diesem Fall der Adjazenzmatrix eines (gerichteten) Hamiltonkreises durch n Knoten und für jede Permutation $\pi \in \mathcal{S}_n$ und Belegung von E bzw. A ist

$$n \leq Z(A, B, \pi) = \sum_{i=1}^{n} a_{\pi(i)\pi((i \mod n)+1)} \cdot b_{i((i \mod n)+1)} \leq 2n$$

Enthält G einen Hamiltonkreis $v_{k_1}, v_{k_2}, \ldots, v_{k_n}, v_{k_1}$, so ergibt sich durch die Zuordnung $\pi(k_i) = i$ der Zielfunktionswert $Z(A, B, \pi) = n$.

Enthält G dagegen keinen Hamiltonkreis, so muss mindestens einer der Werte

$$a_{\pi(i)\pi((i \mod n)+1)} = 2$$

sein, woraus sich für alle $\pi \in \mathcal{S}_n$ ergibt

$$Z(A, B, \pi) \geq n - 1 + 2 = n + 1 > n$$

Die o. a. Überführung einer Probleminstanz $G \in I_{\mathrm{HC}}$ in eine Instanz $(A, B, n) \in I_{D_{\mathrm{QAP}}}$ lässt sich in polynomieller Zeit durchführen und

$$\mathrm{HC}(G) = 1 \;\Leftrightarrow\; D_{\mathrm{QAP}}(A, B, n) = 1$$

Es gilt also $\mathrm{HC} \preceq_p D_{\mathrm{QAP}}|_{\{0,1,2\}}$. Nach Lemma 3.24 ist damit $D_{\mathrm{QAP}}|_{\{0,1,2\}}$ NP-schwer.

Für das Polynom $p = n$ ist weiterhin für jede Probleminstanz i von $D_{\mathrm{QAP}}(A, B, n)|_{\{0,1,2\}}$

$$MAX(\mathrm{i}) \leq n = p(L(\mathrm{i}))$$

Also ist D_{QAP} stark NP-schwer. □

Definition 3.32 (ε-Approximationsalgorithmus für das QAP)
Sei Υ ein Lösungsalgorithmus für das quadratische Zuordnungsproblem und $\varepsilon > 0$ eine reelle Zahl. Dann heißt Υ ein *ε-Approximationsalgorithmus für das QAP* genau dann, wenn für jede Probleminstanz QAP(A,B) die Abschätzung

$$\left| \frac{Z(A, B, \pi_\Upsilon) - Z(A, B, \pi_{opt})}{Z(A, B, \pi_{opt})} \right| \leq \varepsilon \tag{3.12}$$

gilt, wobei π_Υ die durch Algorithmus Υ berechnete Lösung und π_{opt} eine Optimallösung für QAP(A,B) sei. Die durch einen ε-Approximationsalgorithmus berechnete Lösung von QAP(A,B) heißt *ε-Approximationslösung.*

Satz 3.33 (Sahni & Gonzalez [39])
Für beliebiges $\varepsilon > 0$ impliziert die Existenz eines polynomiellen ε-Approximationsalgorithmus für das QAP, dass $\mathcal{P} = \mathcal{NP}$ gilt.

Beweis: Angenommen, für ein festes $\varepsilon > 0$ existiert ein ε-Approximationsalgorithmus Υ, mit dem in polynomieller Zeit eine Näherungslösung für QAPs bestimmt werden

kann. Sei π_Υ die durch Algorithmus Υ berechnete Lösung und π_{opt} eine Optimallösung für eine Instanz QAP(A,B). Nach Definition 3.32 gilt

$$\left|\frac{Z(A,B,\pi_\Upsilon) - Z(A,B,\pi_{opt})}{Z(A,B,\pi_{opt})}\right| \leq \varepsilon$$

$$\Leftrightarrow \quad \frac{Z(A,B,\pi_\Upsilon)}{1+\varepsilon} \leq Z(A,B,\pi_{opt}) \tag{3.13}$$

Das Hamiltonkreisproblem (HC) ist NP-vollständig, mit Hilfe von Υ lässt es sich jedoch in polynomieller Zeit lösen.

Betrachte dazu das Hamiltonkreisproblem für einen beliebigen Graphen $G = (E, V)$ mit $|V| = n$ und $V = \{v_1, v_2, \ldots, v_n\}$ und überführe die Probleminstanz G_{HC} in die Probleminstanz QAP(A,B) mit $n \times n$-Koeffizientenmatrizen $A = (a_{ij})$ und $B = (b_{ij})$ mit

$$a_{ij} = \begin{cases} 1, & \text{falls } (v_i, v_j) \in E \\ \omega & \text{sonst} \end{cases} \qquad b_{ij} = \begin{cases} 1, & \text{falls } \begin{array}{c} j = i+1 \text{ für } i = 1, \ldots, n-1 \\ \text{oder } i = n \ \wedge \ j = 1 \end{array} \\ 0 & \text{sonst} \end{cases}$$

und $\omega > 1 + n\varepsilon$.

Analog zum Beweis von Satz 3.31 gilt, dass G genau dann einen Hamiltonkreis enthält, wenn $Z(A, B, \pi_{opt}) = n$ ist.

Enthält G keinen Hamiltonkreis, so ist für alle $\pi \in \mathcal{S}_n$

$$Z(A,B,\pi) \geq n - 1 + \omega > n - 1 + 1 + n\varepsilon = n(1+\varepsilon)$$

Wendet man Algorithmus Υ auf QAP(A,B) an, so gilt andererseits für die Lösung π_Υ

$$\begin{aligned} & Z(A,B,\pi_\Upsilon) > n(1+\varepsilon) \quad \Leftrightarrow \quad \frac{Z(A,B,\pi_\Upsilon)}{1+\varepsilon} > n \\ \overset{(3.13)}{\Rightarrow} \quad & Z(A,B,\pi_{opt}) > n \quad \Rightarrow \quad \text{HC}(G) = 0 \end{aligned}$$

Das heißt, es ist $\text{HC}(G) = 0 \Leftrightarrow Z(A,B,\pi_\Upsilon) > n(1+\varepsilon)$ und im Umkehrschluss ist $\text{HC}(G) = 1 \Leftrightarrow Z(A,B,\pi_\Upsilon) \leq n(1+\varepsilon)$.

Die o.a. Überführung der Probleminstanzen, die Anwendung von Υ sowie der Test, ob für die Lösung π_Υ gilt $Z(A,B,\pi_\Upsilon) > n(1+\varepsilon)$, lassen sich in polynomieller Zeit durchführen. Somit ist (HC) $\in \mathcal{P}$, womit $\mathcal{P} = \mathcal{NP}$ folgt nach Lemma 3.28. □

3.4 Untere Schranken für das Quadratische Zuordnungsproblem

Obwohl die Bestimmung von unteren Schranken für das QAP ein seit langem intensiv untersuchtes Gebiet ist, konnte bis heute keine Schranke gefunden werden, die gleichzeitig "dicht" und effizient zu berechnen ist [6]. Je "dichter" eine Schranke ist, d.h. je kleiner die Differenz zum Zielfunktionswert einer optimalen Lösung ist, umso besser lässt sich mit ihr die Qualität einer suboptimalen Problemlösung bewerten. Weiterhin hängt auch die Effizienz von Branch-and-Bound Verfahren stark von der Güte der verwendeten Schranke ab.

Im Folgenden werden zwei untere Schranken für das QAP vorgestellt. Dabei verwendete Begriffe werden zunächst gegeben.

Bemerkung 3.34 Seien $u, v \in \mathbb{R}^n$. Dann wird mit $\langle u, v \rangle$ das Skalarprodukt der Vektoren u und v bezeichnet. Es ist

$$\langle u, v \rangle = \sum_{i=1}^{n} u_i \cdot v_i$$

Definition 3.35
Seien $u, v \in \mathbb{R}^n$ gegeben und $\phi, \psi \in \mathcal{S}_n$ so gewählt, dass gilt $u_{\phi(1)} \leq u_{\phi(2)} \leq \ldots \leq u_{\phi(n)}$ und $v_{\psi(1)} \geq v_{\psi(2)} \geq \ldots \geq v_{\psi(n)}$. Dann wird mit $\langle u, v \rangle_-$ das Skalarprodukt der Vektoren u^ϕ und v^ψ bezeichnet. Es ist

$$\langle u, v \rangle_- := \langle u^\phi, v^\psi \rangle = \sum_{i=1}^{n} u_{\phi(i)} \cdot v_{\psi(i)}$$

Lemma 3.36 (Hardy, Littlewood & Pòlya [21])
Seien $u, v \in \mathbb{R}^n$. Dann gilt für alle $\pi \in \mathcal{S}_n$ die Umordnungs-Ungleichung

$$\langle u, v \rangle_- \leq \langle u^\pi, v \rangle \tag{3.14}$$

3.4.1 Die Gilmore-Lawler-Schranke

Eine der ersten unteren Schranken für das QAP, die in der Literatur vorgestellt wurden, stammt von Gilmore [17] und Lawler [31]. Diese sogenannte *Gilmore-Lawler-Schranke* gehört zu den bekanntesten und am häufigsten verwendeten Schranken für das QAP [25].

Satz 3.37 (Gilmore-Lawler-Schranke)
Sei ein beliebiges QAP mit $A, B \in \mathbb{R}^{n\times n}$ gegeben. Sei $\widetilde{A} = (\tilde{a}_{ij})$ die Matrix, welche sich durch Entfernen der Diagonalelemente von A ergibt. Es ist $\widetilde{A} \in \mathbb{R}^{n\times n-1}$ mit

$$\tilde{a}_{ij} = \begin{cases} a_{ij} & \text{falls } i < j \\ a_{i(j+1)} & \text{falls } i \geq j \end{cases}$$

Bezeichne $\tilde{a}_k$ die k-te Zeile von $\widetilde{A}$ und seien weiter $\widetilde{B}$ sowie $\tilde{b}_k$ analog definiert. Betrachte die Matrix $C = (c_{ij}) \in \mathbb{R}^{n\times n}$ mit

$$c_{ij} = \langle \tilde{a}_i, \tilde{b}_j \rangle_{\_} + a_{ii} \cdot b_{jj}$$

Dann ist die Gilmore-Lawler-Schranke *GL für QAP(A,B) gegeben durch*

$$GL := \min_{\pi\in\mathcal{S}_n} \sum_{i=1}^{n} c_{\pi(i)i} \tag{3.15}$$

und es gilt

$$\min_{\pi\in\mathcal{S}_n} Z(A, B, \pi) \geq GL$$

d.h. GL ist eine untere Schranke für den optimalen Zielfunktionswert von QAP(A,B).

Beweis: Sei mit a_i und b_i jeweils die i-te Zeile von A bzw. B bezeichnet. Dann ist

$$\sum_{j=1}^{n} a_{ij} \cdot b_{ij} = \langle a_i, b_i \rangle = \langle \tilde{a}_i, \tilde{b}_i \rangle + a_{ii} \cdot b_{ii} \tag{3.16}$$

Damit gilt für den Zielfunktionswert einer beliebigen Permutation $\pi \in \mathcal{S}_n$ für QAP(A,B)

$$\begin{aligned} Z(A, B, \pi) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij} \\ &\overset{(3.16)}{=} \sum_{i=1}^{n} \left(\langle \tilde{a}^{\pi}_{\pi(i)}, \tilde{b}_i \rangle + a_{\pi(i)\pi(i)} \cdot b_{ii} \right) \\ &\overset{(3.14)}{\geq} \sum_{i=1}^{n} \left(\langle \tilde{a}_{\pi(i)}, \tilde{b}_i \rangle_{\_} + a_{\pi(i)\pi(i)} \cdot b_{ii} \right) \\ &= \sum_{i=1}^{n} c_{\pi(i)i} \end{aligned}$$

und es folgt

$$\min_{\pi\in\mathcal{S}_n} Z(A, B, \pi) \geq \min_{\pi\in\mathcal{S}_n} \sum_{i=1}^{n} c_{\pi(i)i} = GL$$ □

Die Gilmore-Lawler-Schranke lässt sich mit einem Zeitaufwand von $\mathcal{O}(n^3)$ berechnen. Dafür sortiert man im ersten Schritt die Zeilen der Matrix $\widetilde{A}$ aufsteigend und die Zeilen der Matrix $\widetilde{B}$ absteigend. Dies erfordert $\mathcal{O}(n^2 \log n)$ Rechenoperationen. Danach kann im zweiten Schritt die Matrix C in Zeit $\mathcal{O}(n^3)$ bestimmt werden. Für die Berechnung von GL ist im letzten Schritt ein *lineares Zuordnungsproblem*

$$\min_{\pi \in \mathcal{S}_n} \sum_{i=1}^{n} c_{\pi(i)i}$$

zu lösen. Mit den hierfür bekannten effizienten Algorithmen[2] lässt sich dies ebenfalls in Zeit $\mathcal{O}(n^3)$ durchführen.

Die Gilmore-Lawler-Schranke ist eine der am einfachsten und effizientesten zu berechnenden unteren Schranken für das QAP. Allerdings ist der Abstand zum optimalen Zielfunktionswert im Allgemeinen relativ groß und nimmt weiterhin mit steigender Problemgröße n stark zu [6]. Deshalb ist es sehr fraglich, ob für große n die Gilmore-Lawler-Schranke noch sinnvoll zur Beurteilung der Qualität einer suboptimalen Lösung für das QAP eingesetzt werden kann.

3.4.2 Eigenwertbasierte Schranken

Den eigenwertbasierten Schranken liegt die Beziehung zwischen der Spur-Formulierung des QAP (vgl. Kap. 3.2.2) und den Eigenwerten der Koeffizientenmatrizen zugrunde. Um reelle Eigenwerte zu garantieren, können diese Schranken nur für symmetrische Matrizen benutzt werden. Die erste Eigenwertschranke wurde 1978 von Finke et al. [10] vorgestellt. Diese wird im Folgenden beschrieben.

Satz 3.38 (Einfache Eigenwertschranke)
Sei ein QAP mit symmetrischen Koeffizientenmatrizen $A, B \in \mathbb{R}^{n \times n}$ gegeben und seien $\lambda_1, \lambda_2, \ldots, \lambda_n$ die Eigenwerte der Matrix A sowie $\mu_1, \mu_2, \ldots, \mu_n$ die Eigenwerte der Matrix B. Seien weiter $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)^t$ und $\mu = (\mu_1, \mu_2, \ldots, \mu_n)^t$. Dann ist die Eigenwertschranke *EV gegeben durch*

$$EV := \langle \lambda, \mu \rangle_- \tag{3.17}$$

und es gilt

$$\min_{\pi \in \mathcal{S}_n} Z(A, B, \pi) \geq EV$$

d.h. EV ist eine untere Schranke für den optimalen Zielfunktionswert von QAP(A,B).

[2] Zu diesen gehören beispielsweise Shortest Augmenting Path-Verfahren oder Varianten der Ungarischen Methode (vgl. [5]).

Beweis: Da A und B symmetrisch sind, sind sie orthogonal diagonalisierbar und alle Eigenwerte λ_i von A und μ_i von B sind reell. Es existieren also orthogonale Matrizen $U, V \in \mathbb{R}^{n\times n}$, so dass $A = ULU^t$ und $B = VMV^t$ ist, wobei $L = (l_{ij})$ und $M = (m_{ij})$ mit

$$l_{ij} = \begin{cases} \lambda_i & \text{für } i = j \\ 0 & \text{sonst} \end{cases} \qquad m_{ij} = \begin{cases} \mu_i & \text{für } i = j \\ 0 & \text{sonst} \end{cases}$$

Es wird zunächst gezeigt, dass gilt $\mathrm{tr}(AB) \geq \langle \lambda, \mu \rangle_-$. Sei $U^tV =: W = (w_{ij})$. Dann ist

$$\mathrm{tr}(AB) = \mathrm{tr}(ULU^tVMV^t) = \mathrm{tr}(LU^tVMV^tU) = \mathrm{tr}(LWMW^t) \tag{3.18}$$

Für die Einträge der Matrizen LW und MW^t gilt

$$(LW)_{ij} = \sum_{k=1}^{n} l_{ik}w_{kj} = l_{ii}w_{ij} = \lambda_i w_{ij}$$
$$(MW^t)_{ij} = \sum_{k=1}^{n} m_{ik}(W^t)_{kj} = \sum_{k=1}^{n} m_{ik}w_{jk} = m_{ii}w_{ji} = \mu_i w_{ji}$$

Damit ist die Matrix $C := LWMW^t$ gegeben durch

$$c_{ij} = \sum_{k=1}^{n} (LW)_{ik}(MW^t)_{kj} = \sum_{k=1}^{n} \lambda_i w_{ik}\mu_k w_{jk}$$

und daraus folgt mit (3.18)

$$\mathrm{tr}(AB) = \mathrm{tr}(C) = \sum_{i=1}^{n} c_{ii} = \sum_{i=1}^{n}\sum_{k=1}^{n} \lambda_i w_{ik}^2 \mu_k = \lambda^t S \mu \tag{3.19}$$

wobei $S = (s_{ij}) \in \mathbb{R}^{n\times n}$ mit $s_{ij} = w_{ij}^2$. Die Matrix W ist als Produkt orthogonaler Matrizen ebenfalls orthogonal, d.h. ihre Zeilen- bzw. Spaltenvektoren bilden eine Orthonormalbasis des $\mathbb{R}^n$, haben also insbesondere alle die Länge 1.

Sei W_{i*} die i-te Zeile und W_{*i} die i-te Spalte von W. Dann ist

$$1 = |W_{i*}| = \sqrt{\langle W_{i*}, W_{i*} \rangle}$$

und es folgt

$$1^2 = \langle W_{i*}, W_{i*} \rangle = \sum_{j=1}^{n} w_{ij}^2 = \sum_{j=1}^{n} s_{ij}$$

Das heißt, alle Zeilen von S summieren sich zu 1. Analog folgt aus $|W_{*i}| = 1$, dass die Spaltensummen von S alle gleich 1 sind. Da weiterhin alle Einträge von S nicht negativ sind, ist S eine doppelt stochastische Matrix. Diese Matrizen lassen sich nach einem Satz

von Birkhoff und von Neumann [44] als Konvexkombination von Permutationsmatrizen darstellen. Es ist also

$$S = \sum_{i \in I} \alpha_i \cdot X_i \qquad \text{mit} \quad X_i \in \Pi_n, \quad \alpha_i \in \mathbb{R}^n_{\geq 0} \quad \text{und} \quad \sum_{i \in I} \alpha_i = 1$$

und weiter

$$\lambda^t S \mu = \lambda^t (\sum_{i \in I} \alpha_i X_i) \mu = \sum_{i \in I} \alpha_i \lambda^t X_i \mu = \sum_{i \in I} \alpha_i \langle \lambda, X_i \mu \rangle \tag{3.20}$$

Sei $\pi_i \in \mathcal{S}_n$ die zu X_i assoziierte Permutation. Dann ist $X_i \mu = \mu^{\pi_i}$ und es folgt mit (3.19), (3.20) und Lemma 3.36

$$\operatorname{tr}(AB) = \sum_{i \in I} \alpha_i \langle \lambda, \mu^{\pi_i} \rangle \geq \sum_{i \in I} \alpha_i \langle \lambda, \mu \rangle_- = \langle \lambda, \mu \rangle_-$$

Da A und XAX^t für alle $X \in \Pi_n$ dieselben Eigenwerte besitzen, folgt hieraus

$$\operatorname{tr}(XAX^tB) \geq \langle \lambda, \mu \rangle_-$$

und damit die Behauptung, denn

$$\min_{\pi \in \mathcal{S}_n} Z(A, B, \pi) = \min_{X \in \Pi_n} \operatorname{tr}(XAX^tB) \geq \langle \lambda, \mu \rangle_- \qquad \square$$

Für die Berechnung dieser einfachen Eigenwertschranke müssen zunächst die Eigenwerte der Matrizen A und B bestimmt werden. Mit den üblichen Verfahren[3] beträgt der Zeitaufwand hierfür $\mathcal{O}(n^3)$. Die Eigenwertvektoren λ und μ können jeweils in Zeit $\mathcal{O}(n \log n)$ auf- bzw. absteigend sortiert werden und das anschließende Bilden des Skalarprodukts erfordert nur einen Zeitaufwand von $\mathcal{O}(n)$.

Insgesamt lässt sich EV also in Zeit $\mathcal{O}(n^3)$ und damit effizient berechnen. Durch die oft große Varianz der Eigenwerte ist aber im Allgemeinen auch die Differenz dieser Schranke zum optimalen Zielfunktionswert von QAP(A,B) sehr groß. Tatsächlich nimmt EV in den meisten Fällen sogar negative Werte an [6]. In dieser Hinsicht verbesserte eigenwertbasierte Schranken finden sich unter anderem in [4, 20].

[3] Dazu gehört beispielsweise der symmetrische QR-Algorithmus (vgl. [19]).

4 Modellierung der Optimierung der Prozess-Kommunikation als QAP

In diesem Kapitel wird beschrieben, wie sich eine optimale Zuordnung von MPI-Prozessen zu CPU-Cores zur Minimierung der Kommunikationslast des Netzwerks als quadratisches Zuordnungsproblem modellieren lässt. Einige der dabei verwendeten Begriffe werden zunächst kurz erklärt.

Nachricht	Daten, die zwischen zwei Prozessen bei einem Kommunikationsaufruf ausgetauscht werden.
Latenz	Zeit, die für einen Verbindungsaufbau zur Datenübertragung benötigt wird.
Bandbreite	Maximale Anzahl an Daten, die in einem bestimmten Zeitintervall übertragen werden kann.
Rechenknoten	Server bzw. Computer, der ausschließlich für Rechenaufgaben benutzt wird.
Rechencluster	Zusammenschluss mehrerer Rechenknoten zu einem Netzwerk.
Switch	Zentrale Netzwerkkomponente, die andere Komponenten[1] des Netzwerks miteinander verbindet.
Hop	Der Weg von einem Switch zu einem anderen Punkt[1] im Netzwerk.
Netzwerktopologie	Logische Struktur von Netzwerken. Beschreibt die Art und Weise, wie die einzelnen Netzwerkkomponenten miteinander verbunden sind.

4.1 Modellierung der Flussmatrix

Die Flussbeziehung zwischen den Objekten (MPI-Prozessen) ist in diesem Fall die Kommunikation, und die Flussmatrix A wird entsprechend als *Kommunikationsmatrix* bezeichnet. Der Wert des Eintrags a_{ij} gibt die Intensität der Kommunikation von MPI-Prozess i nach MPI-Prozess j an (a_{ji} entsprechend die Kommunikation von j nach i), welche direkt abhängig von der Gesamtanzahl der zwischen den beiden Prozessen zu

[1] Dies können Computer/Rechenknoten oder auch selbst wieder Switches sein.

übermittelnden Bytes ist. Diese wiederum bestimmt sich durch die Anzahl der von i nach j zu kommunizierenden Nachrichten sowie der Größe der einzelnen Nachrichten in Bytes.

Definition 4.1 (Kommunikationsmatrix)
Sei M die Gesamtanzahl der von Prozess i nach Prozess j zu übermittelnden Nachrichten und m_k die Anzahl der bei Nachricht k zu übertragenden Bytes. Dann lässt sich die *Kommunikationsmatrix* $A = (a_{ij})$ definieren durch

$$a_{ij} = \sum_{k=1}^{M} m_k \tag{4.1}$$

Für die Modellierung der Kommunikation ist es aber tatsächlich gar nicht notwendig, die exakte Anzahl der zu verschickenden Bytes zu benutzen, wie die folgenden Sätze zeigen. Wie mit Hilfe dieser Sätze die Aufstellung der Kommunikationsmatrix vereinfacht werden kann, wird im nächsten Abschnitt am Beispiel der Kommunikationsmatrix für den TAU-Strömungslöser demonstriert.

Satz 4.2 *Sei eine beliebige Instanz QAP(A,B) gegeben. Dann gilt:*

1. *Die Addition einer reellen Zahl zu jedem Element von A oder B entspricht der Addition einer reellen Zahl zum Zielfunktionswert $Z(A, B)$.*
2. *Die Multiplikation einer reellen Zahl mit A oder B entspricht der Multiplikation einer reellen Zahl mit dem Zielfunktionswert $Z(A, B)$.*

Insbesondere ist in beiden Fällen die Änderung im Zielfunktionswert unabhängig von der Wahl einer Permutation π.

Beweis: Seien $A, B \in \mathbb{R}^{n\times n}$ und $\pi \in \mathcal{S}_n$ gegeben mit $A = (a_{ij})$, $B = (b_{ij})$. Seien weiter $k, l \in \mathbb{R}$.

1. Betrachte A' mit $a'_{ij} = a_{ij} + k$ und B' mit $b'_{ij} = b_{ij} + l$. Dann ist

$$\begin{aligned}
Z(A', B', \pi) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a'_{\pi(i)\pi(j)} \cdot b'_{ij} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} (a_{\pi(i)\pi(j)} + k) \cdot (b_{ij} + l) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \Big(a_{\pi(i)\pi(j)} \cdot b_{ij} + a_{\pi(i)\pi(j)} \cdot l + k \cdot b_{ij} + k \cdot l \Big) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij} + l \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} a_{\pi(i)\pi(j)} + k \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} k \cdot l
\end{aligned}$$

$$= Z(A,B,\pi) + \underbrace{l \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} + k \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} + n^2 \cdot k \cdot l}_{=:\, x}$$

2. Betrachte $A'' = k \cdot A$ und $B'' = l \cdot B$. Dann ist

$$\begin{aligned} Z(A'', B'', \pi) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a''_{\pi(i)\pi(j)} \cdot b''_{ij} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} (k \cdot a_{\pi(i)\pi(j)})(l \cdot b_{ij}) \\ &= k \cdot l \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} a_{\pi(i)\pi(j)} \cdot b_{ij} \\ &= \underbrace{k \cdot l}_{=:\, y} \cdot Z(A, B, \pi) \end{aligned}$$

Offensichtlich sind x und y reell und nicht abhängig von π. □

Satz 4.3 *Seien $A, B \in \mathbb{R}^{n\times n}$ gegeben und sei π^* eine Optimallösung für QAP(A,B). Seien weiter $A_1, A_2, B_1, B_2 \in \mathbb{R}^{n\times n}$ und $x, y \in \mathbb{R}$ mit $y > 0$, so dass für alle $\pi \in \mathcal{S}_n$ gelte*

$$Z(A, B, \pi) = x + Z(A_1, B_1, \pi)$$

und

$$Z(A, B, \pi) = y \cdot Z(A_2, B_2, \pi)$$

Dann ist π^* ebenfalls eine Optimallösung für QAP(A_1,B_1) sowie für QAP(A_2,B_2).

Beweis: Für eine Optimallösung π^* von QAP(A,B) gilt für alle $\pi \in \mathcal{S}_n$

$$Z(A, B, \pi^*) \leq Z(A, B, \pi)$$

Damit folgt erstens

$$\begin{aligned} & x + Z(A_1, B_1, \pi^*) \leq x + Z(A_1, B_1, \pi) \\ \Leftrightarrow \quad & Z(A_1, B_1, \pi^*) \leq Z(A_1, B_1, \pi) \end{aligned} \tag{4.2}$$

und zweitens

$$\begin{aligned} & y \cdot Z(A_2, B_2, \pi^*) \leq y \cdot Z(A_2, B_2, \pi) \\ \Leftrightarrow \quad & Z(A_2, B_2, \pi^*) \leq Z(A_2, B_2, \pi) \end{aligned} \tag{4.3}$$

Ungleichungen (4.2) und (4.3) sind für alle $\pi \in \mathcal{S}_n$ erfüllt, also ist π^* auch Optimallösung für QAP(A_1,B_1) und QAP(A_2,B_2). □

Bemerkung 4.4 Wegen den Sätzen 4.2 und 4.3 kann ohne Beschränkung der Allgemeinheit angenommen werden, dass alle Elemente der Koeffizientenmatrizen eines QAPs nicht negativ sind.

4.1.1 Die Kommunikationsmatrix für den TAU-Strömungslöser

Beim TAU-Strömungslöser braucht man für die Modellierung der Kommunikationsmatrix nur die Punkt-zu-Punkt Kommunikationen zu berücksichtigen, da die globale Kommunikation für alle Prozesse immer mit der gleichen Anzahl an zu übertragenden Bytes stattfindet und somit als Addition eines konstanten Werts in den Zielfunktionswert eingehen würde (siehe Satz 4.2).

Bei der Punkt-zu-Punkt Kommunikation entspricht die Anzahl der zu übermittelnden Nachrichten genau der Anzahl der Nachbarpartitionen der einem Prozess zugrundeliegenden Netzpartition. Die Größe einer solchen Nachricht wird dabei von zwei Faktoren bestimmt. Das ist zum einen die Anzahl der ghost-points und zum anderen die Anzahl der Variablen, die pro Punkt übermittelt werden müssen (vgl. Kapitel 2.3). Welche Variablen übermittelt werden, kann bei den einzelnen Kommunikationsaufrufen zwar variieren, ist aber in jedem Fall für alle Prozesse gleich. Somit geht die Gesamtzahl der pro ghost-point zu übermittelnden Bytes als konstanter Faktor in den Zielfunktionswert ein (siehe Satz 4.2) und muss bei der Modellierung ebenfalls nicht berücksichtigt werden.

Um die Kommunikationsmatrix für den TAU-Strömungslöser aufzustellen, kann daher der Wert a_{ij} auf die Anzahl der zu Partition j gehörenden ghost-points der Partition i gesetzt werden, d.h. auf die Anzahl der Netzpunkte, für die Daten von Prozess j nach Prozess i kommuniziert werden müssen.

4.1.2 Beispiele für Kommunikationsmatrizen

Abbildungen 4.1 und 4.2 zeigen Kommunikationsmatrizen für den TAU-Strömungslöser. Der Matrix in Abbildung 4.1 liegt die aerodynamische Konfiguration DLR-F6 und der Matrix in Abbildung 4.2 die Konfiguration RAE-2822 zugrunde.

$$\begin{pmatrix}
0 & 3911 & 3 & 3 & 3619 & 155 & 0 & 0 & 11039 & 249 & 13 & 1 \\
4113 & 0 & 3925 & 852 & 25 & 3159 & 2458 & 1 & 3782 & 171 & 14 & 2 \\
5 & 3826 & 0 & 5 & 3008 & 14 & 18266 & 0 & 2613 & 1 & 21 & 0 \\
4 & 736 & 4 & 0 & 8 & 910 & 596 & 3440 & 0 & 4384 & 1174 & 305 \\
4042 & 21 & 3137 & 8 & 0 & 8 & 22 & 0 & 4115 & 10 & 20 & 2 \\
325 & 3211 & 9 & 1015 & 9 & 0 & 9 & 1 & 11 & 2690 & 10 & 1 \\
0 & 2397 & 18560 & 593 & 30 & 8 & 0 & 0 & 1 & 3 & 11775 & 0 \\
0 & 1 & 0 & 3564 & 0 & 1 & 0 & 0 & 0 & 270 & 102 & 3541 \\
13560 & 3823 & 2742 & 0 & 3860 & 18 & 1 & 0 & 0 & 1 & 0 & 0 \\
290 & 85 & 1 & 4327 & 8 & 2570 & 2 & 300 & 1 & 0 & 7 & 620 \\
12 & 13 & 15 & 1066 & 15 & 6 & 11081 & 34 & 0 & 6 & 0 & 119 \\
1 & 1 & 0 & 234 & 2 & 1 & 0 & 3399 & 0 & 615 & 149 & 0
\end{pmatrix}$$

Abbildung 4.1: TAU-Kommunikationsmatrix für DLR-F6

In beiden Fällen wurde das zugrundeliegende Netz in 12 Teilnetze partitioniert. Der Wert der Einträge der Matrizen entspricht jeweils der Anzahl der Netzpunkte, für die Daten kommuniziert werden müssen.

$$\begin{pmatrix}
0 & 0 & 1334 & 0 & 251 & 0 & 0 & 0 & 1093 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 458 & 602 & 0 & 0 & 0 & 41 & 0 & 0 \\
1327 & 0 & 0 & 0 & 0 & 0 & 1051 & 0 & 0 & 0 & 242 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 647 & 0 & 0 & 485 & 50 \\
248 & 459 & 0 & 0 & 0 & 0 & 0 & 0 & 555 & 0 & 0 & 0 \\
0 & 647 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1275 & 0 & 0 \\
0 & 0 & 1114 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 439 & 0 \\
0 & 0 & 0 & 678 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1304 \\
1129 & 0 & 0 & 0 & 431 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 39 & 0 & 0 & 0 & 1263 & 0 & 0 & 0 & 0 & 0 & 1240 \\
0 & 0 & 253 & 488 & 0 & 0 & 505 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 51 & 0 & 0 & 0 & 1248 & 0 & 1261 & 0 & 0
\end{pmatrix}$$

Abbildung 4.2: TAU-Kommunikationsmatrix für RAE-2822

4.2 Modellierung der Distanzmatrix

Die Distanz b_{ij} zwischen zwei CPU-Cores i und j gibt keine Entfernung im eigentlichen Sinne an, vielmehr ist sie ein Maß für den Zeitaufwand der benötigt wird, um eine Nachricht von i nach j zu schicken. Dieser ist allerdings nicht konstant, sondern abhängig von der jeweiligen Größe der Nachricht (siehe Tabelle 2.1 und Abbildung 2.1) bzw. Bandbreite und Latenz des zugrundeliegenden Netzwerks.

Die Übertragungszeit einer Nachricht lässt sich beispielsweise durch einen MPI-Ping-Pong-Test [18] ermitteln. Nur in dem (eher unwahrscheinlichen) Fall, dass alle Nachrichten dieselbe Größe haben, kann der Wert b_{ij} direkt durch den bei einem solchen Test ermittelten Wert ersetzt werden, anderenfalls sollten alle gemessenen Zeiten, die im Größenbereich der Nachrichten liegen, berücksichtigt werden. Eine genaue Modellierung des Kommunikationszeitaufwands ist daher, selbst wenn Bandbreite und Latenz des Netzwerks bekannt sind, schwierig.

Unter der Voraussetzung, dass die knoteninterne Kommunikation schneller als die Kommunikation über das Netzwerk ist, gilt aber generell: Die „Distanz“ zwischen zwei CPU-Cores (Orten) ist klein, wenn sie sich auf demselben Knoten befinden und groß, wenn sie sich auf verschiedenen Knoten befinden. Diese Tatsache ermöglicht es, die Distanzmatrix B auf einfache aber relativ allgemeingültige Weise zu definieren.

Definition 4.5 (Distanzmatrix, einfache Variante)
Bezeichne $K(i)$ den Rechenknoten, auf welchem sich CPU-Core i befindet und seien $w, \mathcal{W} \in \mathbb{R}_+$ mit $w << \mathcal{W}$. Eine *einfache Distanzmatrix* $B = (b_{ij})$ lässt sich dann definieren durch

$$b_{ij} = \begin{cases} 0, & \text{falls } i = j \\ w, & \text{falls } K(i) = K(j) \\ \mathcal{W}, & \text{falls } K(i) \neq K(j) \end{cases} \tag{4.4}$$

Wenn die genaue Topologie des Netzwerks des Rechenclusters bekannt ist, lässt sich diese einfache Modellierung der Distanz weiter verfeinern. Da das Passieren eines Switches (Hop) die Übertragungszeit einer Nachricht erhöht[2], kann die Kommunikationslast des Netzwerks weiter reduziert werden, wenn knotenexterne Kommunikation möglichst nur zwischen "nahe beieinander gelegenen" Knoten stattfindet. Diese "Knotendistanz" lässt sich modellieren, indem die Distanz zwischen CPU-Cores auf verschiedenen Knoten zusätzlich von der Anzahl der Hops zwischen den Prozessoren abhängig gemacht wird.

Definition 4.6 (Distanzmatrix, erweiterte Variante)
Bezeichne $K(i)$ den Rechenknoten, auf welchem sich CPU-Core i befindet und seien $w, \mathcal{W} \in \mathbb{R}_+$ mit $w << \mathcal{W}$. Gebe weiter h_{ij} die minimale Anzahl der Hops zwischen den Prozessoren i und j an. Eine *erweiterte Distanzmatrix* $B = (b_{ij})$ lässt sich dann definieren durch

$$b_{ij} = \begin{cases} 0, & \text{falls } i = j \\ w, & \text{falls } K(i) = K(j) \\ h_{ij} \cdot \mathcal{W}, & \text{falls } K(i) \neq K(j) \end{cases} \tag{4.5}$$

Beobachtung 4.7 Die nach den Definitionen 4.5 und 4.6 aufgestellten Distanzmatrizen sind immer symmetrisch.

4.2.1 Beispiele für Distanzmatrizen

Abbildung 4.3 zeigt zwei Distanzmatrizen für 12 CPU-Cores bei 4 Cores pro Rechenknoten. In beiden Fällen wurde der Wert für die Distanz zwischen CPU-Cores, die sich auf demselben Rechenknoten befinden, auf **1** gesetzt. Für die Distanz zwischen Prozessoren, die sich auf verschiedenen Knoten befinden, wurde der Wert **10** angenommen, welcher bei Matrix (b) zusätzlich mit der Anzahl der Hops zwischen den kommunizierenden Prozessoren multipliziert wurde.

[2] Dies macht sich insbesondere bei kleinen Nachrichten bemerkbar.

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
1 & 0 & 1 & 1 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
1 & 1 & 0 & 1 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
1 & 1 & 1 & 0 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 0 & 1 & 1 & 1 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 1 & 0 & 1 & 1 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 1 & 1 & 0 & 1 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 1 & 1 & 1 & 0 & 10 & 10 & 10 & 10 \\
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 0 & 1 & 1 & 1 \\
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 1 & 0 & 1 & 1 \\
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 1 & 1 & 0 & 1 \\
10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 1 & 1 & 1 & 0
\end{pmatrix}$$

(a) Einfache Distanzmatrix

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
1 & 0 & 1 & 1 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
1 & 1 & 0 & 1 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
1 & 1 & 1 & 0 & 10 & 10 & 10 & 10 & 30 & 30 & 30 & 30 \\
10 & 10 & 10 & 10 & 0 & 1 & 1 & 1 & 50 & 50 & 50 & 50 \\
10 & 10 & 10 & 10 & 1 & 0 & 1 & 1 & 50 & 50 & 50 & 50 \\
10 & 10 & 10 & 10 & 1 & 1 & 0 & 1 & 50 & 50 & 50 & 50 \\
10 & 10 & 10 & 10 & 1 & 1 & 1 & 0 & 50 & 50 & 50 & 50 \\
30 & 30 & 30 & 30 & 50 & 50 & 50 & 50 & 0 & 1 & 1 & 1 \\
30 & 30 & 30 & 30 & 50 & 50 & 50 & 50 & 1 & 0 & 1 & 1 \\
30 & 30 & 30 & 30 & 50 & 50 & 50 & 50 & 1 & 1 & 0 & 1 \\
30 & 30 & 30 & 30 & 50 & 50 & 50 & 50 & 1 & 1 & 1 & 0
\end{pmatrix}$$

(b) Erweiterte Distanzmatrix

Abbildung 4.3: Beispiele für Distanzmatrizen (a) ohne und (b) mit Berücksichtigung der genauen Netzwerktopologie

Abbildung 4.3(a) zeigt also eine einfache Distanzmatrix bei beliebiger Netzwerktopologie, die erweiterte Distanzmatrix in Abbildung 4.3(b) ergibt sich beispielsweise für eine Fat-Tree-Topologie (vgl. Abbildung 4.4). Deutlich lässt sich in beiden Fällen eine für diese Matrizen charakteristische Blockstruktur erkennen, welche im nächsten Abschnitt genauer beschrieben wird.

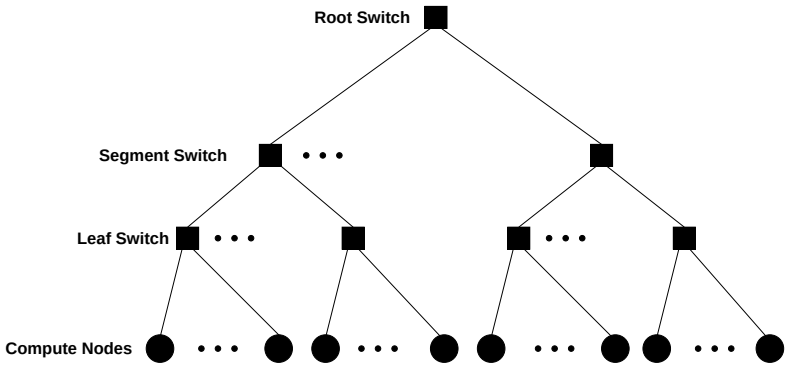


Abbildung 4.4: Schematische Darstellung eines Fat-Tree-Netzwerkes

4.2.2 Strukturelle Eigenschaften der Distanzmatrix

Bei der Optimierung der Prozess-Kommunikation wird der Einfachheit halber davon ausgegangen, dass der Kommunikationsaufwand für zwei Prozesse, die *auf demselben* Rechenknoten gestartet wurden, nur von der Nachrichtengröße abhängt, nicht aber davon auf welchem CPU-Core des Knotens sie laufen. Ebenso ist bei gleicher Nachrichtengröße

der Kommunikationsaufwand zwischen den Prozessen eines Rechenknotens und allen möglichen Prozessen eines anderen Knotens gleich. Für die Modellierung der Distanz bedeutet das, dass die Distanz zwischen CPU-Cores, die sich auf einem Knoten befinden immer gleich ist, und ebenso die Distanz zwischen allen CPU-Cores auf einem Knoten zu allen CPU-Cores auf einem anderen Knoten.

Dies spiegelt sich in der Struktur der nach den Definitionen 4.5 und 4.6 aufgestellten Distanzmatrizen wieder: Bis auf den Diagonaleintrag, der die Distanz eines CPU-Cores zu sich selbst angibt, sind jeweils alle Zeilen und Spalten, welche CPU-Cores auf demselben Knoten repräsentieren, gleich. Seien i und j zwei verschiedene CPU-Cores, die sich auf demselben Rechenknoten befinden und seien B_{i*} und B_{j*} die i-te bzw. j-te Zeile der Distanzmatrix B. Dann gilt für alle $k = 1, \ldots, n$

$$b_{ik} = \begin{cases} b_{jk} & \text{falls } k \neq i,\ k \neq j \\ b_{jj} & \text{falls } k = i \\ b_{ji} & \text{falls } k = j \end{cases}$$

Diese Beziehung gilt analog für die Spalten $B_{*i} = (B_{i*})^t$ und $B_{*j} = (B_{j*})^t$. Nach den o. a. Definitionen ist dabei $b_{ii} = b_{jj} = 0$ und $b_{ij} = b_{ji} = w$.

Bei einer parallelen Rechnung werden üblicherweise immer nur “ganze” Rechenknoten verwendet. Eine Teilung von Rechenknoten, beispielsweise für zwei unterschiedliche parallele Anwendungen, würde dazu führen, dass beide Anwendungen um die gemeinsamen Ressourcen wie Hauptspeicher und auch Kommunikationsbandbreite konkurrieren und sich schlimmstenfalls behindern würden. In der Praxis kann man also davon ausgehen, dass eine parallele Anwendung auch alle CPU-Cores eines zugewiesenen Rechenknotens exklusiv nutzt.

Sei also $k \in \mathbb{N}$ die Anzahl der Rechenknoten und $c \in \mathbb{N}$ die Anzahl der CPU-Cores pro Knoten, dann ist $n = k \cdot c$ die Anzahl der für die parallele Anwendung verwendeten CPU-Cores. Weiterhin seien die Prozessoren so durchnummeriert, dass der i-te Knoten die CPU-Cores $(i-1) \cdot c + 1$ bis $i \cdot c$ enthält. Dann ergibt sich für die nach Definitionen 4.5 und 4.6 aufgestellten Distanzmatrizen die in Abbildung 4.5 dargestellte Blockstruktur. Hierbei ist jeder Block D eine $c \times c$-Matrix mit Einträgen

$$d_{ij} = \begin{cases} 0 & \text{falls } \ i = j \\ w & \text{sonst} \end{cases}$$

Alle Einträge der $c \times c$-Matrix W_{ij} sind für die erweiterte Distanzmatrix gleich

$$h_{lm} \cdot \mathcal{W} \quad \text{mit} \quad \begin{aligned} l &= (i-1) \cdot c + 1 \\ m &= (j-i) \cdot c + 1 \end{aligned}$$

$$\begin{pmatrix} D & W_{12} & W_{13} & \cdots\cdots & W_{1k} \\ W_{12} & D & W_{23} & \cdots\cdots & W_{2k} \\ W_{13} & W_{23} & D & \cdots\cdots & W_{3k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_{1k} & W_{2k} & W_{3k} & \cdots\cdots & D \end{pmatrix}$$

Abbildung 4.5: Blockstruktur der Distanzmatrizen

Für die einfache Distanzmatrix sind alle Blöcke W_{ij} gleich und alle Einträge der Blöcke sind gleich $\mathcal{W}$.

Analog zur Einteilung der Distanzmatrizen in k^2 Blöcke der Größe $c \times c$, lässt sich eine Lösung $\pi \in \mathcal{S}_n$ für QAP(A,B) in k Blöcke bzw. Abschnitte K_i der Größe c einteilen. Diese Blöcke geben die Rechenknoten an, denen die Prozessoren zugeordnet werden. Hierfür betrachtet man die Permutation π als n-Tupel und fasst jeweils die ersten c Elemente von π zu Block K_1, die zweiten c Elemente zu Block K_2 etc. zusammen, d. h.

$$\pi = \Big(\underbrace{\pi(1), \pi(2), \ldots, \pi(c)}_{K_1}, \underbrace{\pi(c+1), \ldots, \pi(2c)}_{K_2}, \ldots, \underbrace{\pi((k-1)c+1), \ldots, \pi(kc)}_{K_k} \Big)$$

Dann ist klar, dass die Prozesse $\pi((i-1)c+1), \pi(i-1)c+2), \ldots, \pi((i-1)c+c)$ innerhalb des Blocks K_i jeweils dem Rechenknoten i zugeordnet werden. Vertauscht man nun jeweils die Elemente innerhalb eines Blockes von π beliebig, erhält man eine Permutation $\phi \in \mathcal{S}_n$ für die gilt $Z(A, B, \phi) = Z(A, B, \pi)$, denn die Prozesse werden durch diese Umordnung zwar möglicherweise anderen CPU-Cores zugeordnet, bleiben aber alle jeweils demselben Rechenknoten wie zuvor zugeordnet.

Es gibt $c!$ Möglichkeiten, jeden Block K_i von π umzuordnen. Damit gibt es insgesamt

$$k \cdot c! = \frac{n}{c} \cdot c! = n \cdot (c-1)!$$

Permutationen (inklusive π), die zum selben Zielfunktionswert führen. Insgesamt gibt es $n!$ Möglichkeiten π zu wählen. Die Anzahl der maximal möglichen verschiedenen Zielfunktionswerte für QAP(A,B) reduziert sich damit von $n!$ auf

$$\frac{n!}{n(c-1)!} = \frac{(n-1)!}{(c-1)!}$$

Die Anzahl der Lösungen mit unterschiedlichen Zielfunktionswerten ist damit leider im Allgemeinen immer noch viel zu groß, als dass es möglich wäre, eine Optimallösung in akzeptabler Zeit dadurch zu bestimmen, dass man aus jeder der "Klassen" der Zielfunktionswerte nur eine Lösung berechnet. Die Tatsache, dass das Vertauschen von Objekten innerhalb eines Blocks nicht den Zielfunktionswert ändert, kann aber teilweise bei den in Kapitel 5 beschriebenen Heuristiken verwendet werden, um Rechenzeit einzusparen.

5 Heuristiken für das Quadratische Zuordnungsproblem

Für die Lösung von quadratischen Zuordnungsproblemen wurden viele heuristische Verfahren in der Literatur vorgestellt. Unter diesen sind vor allem die Folgenden zu nennen (in chronologischer Reihenfolge):

1. Konstruktionsverfahren
2. Begrenzte Enumeration
3. Verbesserungsverfahren
4. Tabu-Suche
5. Simulated Annealing
6. Genetische Algorithmen
7. Greedy Randomized Adaptive Search
8. Ameisenalgorithmen

Für eine Beschreibung dieser Verfahren siehe auch Kapitel 3 in [6]. Im Weiteren werden einige Heuristiken vorgestellt, welche hinsichtlich Lösungsgüte und Laufzeit untersucht und speziell im Hinblick auf die Optimierung der MPI-Prozess-Kommunikation ausgewählt wurden.

Durch eine verbesserte Zuordnung von Prozessen zu CPU-Cores erhofft man sich insbesondere eine geringere Laufzeit der zugrundeliegenden parallelen Anwendung. Damit diese Rechenzeitreduktion zum Tragen kommen kann, sollte natürlich auch die Zeit, die zum Berechnen einer Zuordnung benötigt wird, möglichst gering sein. Bei der Auswahl einer Heuristik für dieses Problem ist daher nicht nur die Lösungsgüte, sondern auch die Laufzeit des Verfahrens wichtig. Es ist anzumerken, dass viele der in der Literatur beschriebenen Heuristiken hauptsächlich hinsichtlich der Lösungsgüte konzipiert und nur für verhältnismäßig kleine Probleme getestet wurden. Insbesondere wurde die Suche nach einer geeigneten Heuristik dadurch erschwert, dass sich keine Ergebnisse für Probleme der Größe $n > 256$ finden ließen.

5.1 Konstruktionsverfahren

Konstruktionsverfahren dienen der Bestimmung einer (ersten) zulässigen Lösung eines Problems, welche zumeist durch sukzessives Festlegen von Lösungselementen konstruiert wird. Diese Algorithmen sind im Allgemeinen recht einfach konzipiert, was einerseits zu einfachen Implementierungen mit vergleichsweise kurzer Rechenzeit führt, andererseits aber auch oft zu qualitativ schlechten Lösungen.

Definition 5.1 (Teilpermutation)
Für eine Menge $N = \{1, 2, \ldots, n\}$ und Teilmengen $U, V \subsetneq N$ heißt ein bijektive Abbildung $\pi : U \to V$ eine *Teilpermutation* der Menge N.

Bei den Konstruktionsverfahren für das QAP wird mit Teillösungen bzw. Teilpermutationen gearbeitet. Üblicherweise wird hierbei mit der leeren Teilpermutation $\pi : \emptyset \to \emptyset$ begonnen, welche dann durch schrittweise Zuweisung von Objekten zu Orten zu einer zulässigen Lösung (Permutation) aufgebaut wird.

Die folgenden beiden Konstruktionsverfahren wurden von Müller-Merbach in dem Buch "Optimale Reihenfolgen" [35] vorgeschlagen. Sie gehören zu den Verfahren mit *abnehmendem Freiheitsgrad*. Das bedeutet, dass anfangs gute Zuordnungen gefunden werden, aber gegen Ende des Verfahrens nur noch recht schlechte Zuordnungen möglich sind.

Algorithmus 1 Einfache Zuordnung (EZ)

Eingabe: Flussmatrix $A \in \mathbb{R}^{n \times n}$, Distanzmatrix $B \in \mathbb{R}^{n \times n}$

Initialisiere: $Z_O = Z_L = \emptyset, \quad \pi : Z_O \to Z_L$

Schritt 1: Für $k = 1$ bis n berechne

$$f_k = \sum_{k=1}^{n} (a_{ik} + a_{ki})$$
$$d_k = \sum_{k=1}^{n} (b_{ik} + b_{ki})$$

Schritt 2: Bestimme $\phi \in \mathcal{S}_n$, so dass $f_{\phi(k)} \geq f_{\phi(k+1)} \ \forall\ k = 1, \ldots, n-1$

Bestimme $\psi \in \mathcal{S}_n$, so dass $d_{\psi(k)} \leq f_{\psi(k+1)} \ \forall\ k = 1, \ldots, n-1$

Schritt 3: Für $k = 1$ bis n setze

$$Z_O = Z_O \cup \{\phi(k)\}$$
$$Z_L = Z_L \cup \{\psi(k)\}$$
$$\pi(\psi(k)) = \phi(k)$$

Ausgabe: Permutation $\pi \in \mathcal{S}_n$

Bei der Beschreibung der Heuristiken wird mit N die Indexmenge der Objekte bzw. Orte bezeichnet. Z_O und Z_L bezeichnen die Indexmengen der bereits zugeordneten Objekte bzw. Orte. Bei beiden Verfahren sind diese Mengen zu Beginn leer, d. h. $Z_O = Z_L = \emptyset$. Durch schrittweises Hinzufügen von Elementen aus N zu Z_O und Z_L wird eine Zuordnung (Teilpermutation) $\pi : Z_O \to Z_L$ aufgebaut, solange bis alle Objekte jeweils einem Ort zugeordnet worden sind, d. h. $Z_O = Z_L = N$ und $\pi \in \mathcal{S}_n$ ist.

Algorithmus 2 Erweiterte einfache Zuordnung (EEZ)

Eingabe: Flussmatrix $A \in \mathbb{R}^{n \times n}$, Distanzmatrix $B \in \mathbb{R}^{n \times n}$

Initialisiere: $N = \{1, 2, \ldots, n\}, \quad Z_O = Z_L = \emptyset, \quad \pi : Z_O \to Z_L$

Schritt 1: Für $k = 1$ bis n berechne

$f_k = \sum_{k=1}^{n} (a_{ik} + a_{ki})$

$d_k = \sum_{k=1}^{n} (b_{ik} + b_{ki})$

Schritt 2: Bestimme $x \in N$, so dass $f_x \geq f_k \ \forall\, k \in N$

Bestimme $y \in N$, so dass $d_y \leq f_k \ \forall\, k \in N$

$Z_O = Z_O \cup \{x\}$

$Z_L = Z_L \cup \{y\}$

$\pi(y) = x$

Schritt 3: Für alle $k \in N \setminus Z_O$ berechne

$f_k = \sum_{i \in Z_O}^{n} (a_{ik} + a_{ki})$

Für alle $k \in N \setminus Z_L$ berechne

$d_k = \sum_{i \in Z_L}^{n} (b_{ik} + b_{ki})$

Schritt 4: Bestimme $x \in N \setminus Z_O$, so dass $f_x \geq f_k \ \forall\, k \in N \setminus Z_O$

Bestimme $y \in N \setminus Z_L$, so dass $d_y \leq f_k \ \forall\, k \in N \setminus Z_L$

$Z_O = Z_O \cup \{x\}$

$Z_L = Z_L \cup \{y\}$

$\pi(y) = x$

Schritt 5: Falls Z_O, $Z_L \neq N$

Gehe zurück zu Schritt 2

Ausgabe: Permutation $\pi \in \mathcal{S}_n$

Die Heuristik "Einfache Zuordnung" ist in Algorithmus 1 dargestellt. Das Verfahren besteht darin, dass man in Schritt 1 für jedes Objekt k den *Gesamtfluss* f_k berechnet, dass heißt die Summe der Flussbeziehungen zwischen k und allen anderen Objekten i. Ebenso wird für jeden Ort k die *Gesamtdistanz* d_k, dass heißt die Summe aller Entfernungen von k zu allen anderen Orten i, berechnet. In Schritt 2 werden die Gesamtflüsse aufsteigend und die Gesamtdistanzen absteigend sortiert, um in Schritt 3 sukzessive das Objekt $\phi(1)$ mit dem größten Gesamtfluss dem Ort $\psi(1)$ mit der kleinsten Gesamtentfernung, das Objekt $\phi(2)$ mit dem zweitgrößten Gesamtfluss dem Ort $\psi(2)$ mit der zweitkleinsten Gesamtentfernung usw. zuzuordnen.

Bei der in Algorithmus 2 dargestellten Heuristik "Erweiterte einfache Zuordnung" beginnt man in gleicher Weise, indem man zunächst das Objekt x mit dem größten Gesamtfluss f_x dem Ort y mit der kleinsten Gesamtdistanz d_y zuordnet (Schritte 1 und 2). Danach wird in Schritt 3 für jedes noch nicht zugeordnete Objekt $k \in N \backslash Z_O$ der Fluss f_k zu den bereits zugeordneten Objekten $i \in \mathbb{Z}_O$ bestimmt, sowie für jeden noch nicht zugeordneten Ort $k \in N \backslash Z_L$ die Distanz d_k zu den bereits zugeordneten Orten $i \in Z_L$. In Schritt 4 wird dann das Objekt x, für welches der Fluss f_x zu den bereits zugeordneten Objekten am größten ist, demjenigen Ort y zugeordnet, für welchen die Distanz d_y zu den bereits zugeordneten Orten am kleinsten ist. Die Schritte 3 und 4 werden so lange wiederholt, bis alle Objekte Orten zugeordnet worden sind, d. h. $Z_O = Z_L = N$ ist.

5.2 Verbesserungsverfahren

Verbesserungsverfahren gehören zu den Suchverfahren der *lokalen Suche*, bei der in der Nachbarschaft einer gegebenen zulässigen Anfangslösung iterativ nach einer besseren Lösung gesucht wird. Wird eine bessere Lösung gefunden, so wird die Anfangslösung durch diese ersetzt und nun die Nachbarschaft der neuen Lösung durchsucht. Diese Abfolge kann so lange wiederholt werden, bis sich keine bessere Lösung mehr findet.

Ein Nachteil dieser Verfahren ist, dass bessere Lösungen außerhalb der Nachbarschaft nicht gefunden werden können. Deshalb ist die Größe und Struktur der verwendeten Nachbarschaft ausschlaggebend dafür, welche Lösungen gefunden werden. Diese beiden Faktoren beeinflussen natürlich insbesondere auch die Komplexität des Verfahrens: Je größer eine Nachbarschaft ist und je aufwendiger es ist, sie zu durchsuchen, umso mehr Rechenzeit wird benötigt.

Zwei Möglichkeiten eine bessere Lösung auszuwählen sind üblich. Man kann zunächst die gesamte Nachbarschaft einer Lösung auswerten und dann die beste der gefundenen Lösungen wählen, oder man wählt immer die erste bessere Lösung, die gefunden wird ("gieriges" Verfahren). Die gierige Vorgehensweise erfordert meistens weniger Rechenzeit. Zu beachten ist, dass hierbei die Reihenfolge, in der die Nachbarschaft abgesucht wird, relevant sein kann.

Für das quadratische Zuordnungsproblem häufig verwendete Nachbarschaftsstrukturen sind die *pair-exchange Nachbarschaft* und die *cyclic triple-exchange Nachbarschaft*. Die pair-exchange Nachbarschaft einer Lösung (Permutation) besteht aus allen Permutationen, die man durch Vertauschen zweier Elemente der Permutation erhält. Die Größe dieser Nachbarschaft ist somit $\binom{n}{2} = \frac{n(n-1)}{2}$. Die cyclic triple-exchange Nachbarschaft einer Permutation besteht aus allen Permutationen, die man durch zyklisches Vertauschen von drei Elementen erhält. Die Größe dieser Nachbarschaft beträgt $2\binom{n}{3}$.

Die folgenden Verbesserungsverfahren für das QAP beruhen auf der von Heider in dem Paper "A Computationally Simplified Pair-Exchange Algorithm for the Quadratic Assignment Problem" [22] vorgeschlagenen Vorgehensweise. Die verwendete Nachbarschaftsstruktur ist hierbei die pair-exchange Nachbarschaft einer Lösung $\pi \in \mathcal{S}_n$. Diese wird jeweils kreisläufig nach einer besseren Lösung durchsucht. Die Auswahl einer neuen Lösung erfolgt nach dem gierigen Prinzip.

Definition 5.2 (Pair-exchange Nachbarschaft)
Die *pair-exchange Nachbarschaft* einer Permutation $\pi \in \mathcal{S}_n$ ist gegeben durch die Menge $N(\pi) = \{\pi^{p,q} \in \mathcal{S}_n \mid p, q = 1, 2, \ldots, n\}$ mit

$$\pi^{p,q}(i) := \begin{cases} \pi(i) & \text{für } i \notin \{p, q\} \\ \pi(p) & \text{für } i = q \\ \pi(q) & \text{für } i = p \end{cases} \tag{5.1}$$

Variante 1 (GPE)

In Algorithmus 3 ist das erste dieser Verfahren, das "Greedy Pair-Exchange" (GPE), dargestellt. Dem Algorithmus wird eine zulässige Lösung $\pi \in \mathcal{S}_n$ für QAP(A,B) übergeben. Dies kann die Identität *id*, eine zufällig gewählte oder eine durch eine Heuristik bestimmte Permutation sein. Zunächst wird die Variable *swap*, welche angibt, ob während eines Durchlaufs eine bessere Lösung gefunden wurde, mit `False` initialisiert. Die Reihenfolge, in der in Schritt 1 die Nachbarschaft $N(\pi)$ der aktuellen Lösung π abgesucht wird, ist durch die Variablen i und j vorgegeben. Beginnend mit $i = 1$ und $j = 2$ durchläuft dabei das Paar (i, j) alle möglichen Kombinationen[1] von Elementen aus $\{1, 2, \ldots, n\}$. Durch Tauschen der Elemente i und j von π, d. h. durch eine Zuordnung von Objekt $\pi(i)$ zu Ort j und von Objekt $\pi(j)$ zu Ort i, erhält man eine Nachbarpermutation $\pi^{i,j}$ der jeweils aktuellen Lösung π. Für diese wird nun getestet, ob sie einen besseren Zielfunktionswert als π besitzt, d. h. ob die Differenz $\Delta = Z(A, B, \pi) - Z(A, B, \pi^{i,j})$ positiv ist. Falls dies der Fall ist, wird π durch $\pi^{i,j}$ ersetzt und die Variable *swap* auf `True` gesetzt. Schritt 1 wird nun beim nächsten Paar (i, j) fortgesetzt und die Nachbarschaft der neuen Lösung durchsucht.

[1] Hierbei wird zwischen den Kombinationen (i, j) und (j, i) nicht unterschieden, da $\pi^{i,j} = \pi^{j,i}$ gilt.

Nachdem alle Paare (i, j) durchlaufen wurden wird festgestellt, ob eine bessere Lösung gefunden wurde (Schritt 2). In diesem Fall ist $swap = \texttt{True}$ und der Algorithmus beginnt mit der aktuellen Lösung wieder von vorn. Wurde die gesamte Nachbarschaft der aktuellen Lösung durchsucht ohne eine bessere Lösung zu finden, ist $swap = \texttt{False}$ und der Algorithmus wird beendet.

Algorithmus 3 Greedy Pair-Exchange (GPE)

Eingabe: Flussmatrix $A \in \mathbb{R}^{n \times n}$, Distanzmatrix $B \in \mathbb{R}^{n \times n}$, Permutation $\pi \in \mathcal{S}_n$

Initialisiere: $swap = \texttt{False}$

Schritt 1: Für $i = 1$ bis $n - 1$

Für $j = i + 1$ bis n

Berechne $\Delta = Z(A, B, \pi) - Z(A, B, \pi^{i,j})$

Falls $\Delta > 0$ ist setze

$\pi = \pi^{i,j}$

$swap = \texttt{True}$

Schritt 2: Falls $swap = \texttt{True}$ ist

Setze $swap = \texttt{False}$

Gehe zurück zu Schritt 1

Ausgabe: Permutation $\pi \in \mathcal{S}_n$

Die Berechnung der Differenz Δ in Schritt 1 trägt einen großen Teil zur Laufzeit des Algorithmus bei. Die Bestimmung des Zielfunktionswerts $Z(A, B, \pi)$ erfordert $\mathcal{O}(n^2)$ Rechenschritte: n^2 Terme der Form $a_{\pi(i)\pi(j)} \cdot b_{ij}$ müssen berechnet und aufsummiert werden. Ist dieser bekannt, so lässt sich jedoch der Zielfunktionswert einer Nachbarpermutation $\pi^{p,q}$ in Zeit $\mathcal{O}(n)$ bestimmen, indem nur die Terme neu berechnet werden, die bei einem Tausch der Elemente p und q verändert werden. Die Differenz Δ ergibt sich dann folgendermaßen (vgl. [35, S. 165]):

$$\begin{aligned}\Delta = &\sum_{\substack{i=1 \\ i \neq p,q}}^{n} \Big(a_{\pi(i)\pi(p)} - a_{\pi(i)\pi(q)}\Big)\Big(b_{ip} - b_{iq}\Big) + \sum_{\substack{j=1 \\ j \neq p,q}}^{n} \Big(a_{\pi(p)\pi(j)} - a_{\pi(q)\pi(j)}\Big)\Big(b_{pj} - b_{qj}\Big) \\ &+ \Big(a_{\pi(p)\pi(q)} - a_{\pi(q)\pi(p)}\Big)\Big(b_{pq} - b_{qp}\Big) + \Big(a_{\pi(p)\pi(p)} - a_{\pi(q)\pi(q)}\Big)\Big(b_{pp} - b_{qq}\Big)\end{aligned}$$

Sind beide Matrizen A und B symmetrisch, so vereinfacht sich dieser Ausdruck zu

$$\Delta = 2 \sum_{\substack{i=1 \\ i \neq p,q}}^{n} \Big(a_{\pi(i)\pi(p)} - a_{\pi(i)\pi(q)}\Big)\Big(b_{ip} - b_{iq}\Big) + \Big(a_{\pi(p)\pi(p)} - a_{\pi(q)\pi(q)}\Big)\Big(b_{pp} - b_{qq}\Big)$$

und die Anzahl der für die Berechnung von Δ benötigten Rechenoperationen lässt sich nochmals etwa halbieren. Diese Tatsache wird bei der zweiten Variante, dem "symmetrisierenden" Greedy Pair-Exchange ($\mathrm{GPE_S}$), verwendet.

Variante 2 ($\mathbf{GPE_S}$)

Im Initialisierungsschritt werden zunächst A auf $\frac{1}{2}(A+A^t)$ und B auf $\frac{1}{2}(B+B^t)$ gesetzt, um symmetrische Matrizen zu erhalten. Der Rest des Algorithmus ist mit GPE gleich.

Im Falle, dass eine der beiden Koeffizientenmatrizen von vorneherein symmetrisch war, ist QAP(A,B) äquivalent zu QAP($\frac{1}{2}(A+A^t), \frac{1}{2}(B+B^t)$) (vgl. Satz 3.14), und die Verfahren GPE und $\mathrm{GPE_S}$ unterscheiden sich nur in der Laufzeit, nicht im Ergebnis. Dies ist beim QAP zur Optimierung der Prozess-Kommunikation der Fall, da hier die nach den Definitionen 4.5 und 4.6 aufgestellten Distanzmatrizen immer symmetrisch sind.

Variante 3 ($\mathbf{GPE_{BS}}$)

Bei der dritten Variante des Greedy Pair-Exchange ($\mathrm{GPE_{BS}}$) wird weiterhin die Tatsache ausgenutzt, dass sich durch ein Vertauschen von zwei Permutationselementen, die im selben "Block" liegen, keine Änderung im Zielfunktionswert ergibt (vgl. Kapitel 4.2.2). Das Auswerten von Nachbarpermutationen π^{ij} kann weggelassen werden, wenn $\pi(i)$ und $\pi(j)$ eine Zuordnung zum selben Rechenknoten beschreiben, da durch den Algorithmus nur Lösungen mit echt kleinerem Zielfunktionswert ausgewählt werden. Dies wird erreicht, indem in Schritt 1 des Verfahrens nicht über alle Paare (i,j) iteriert wird, sondern nur über die Paare, die Zuordnungen zu verschiedenen Rechenknoten angeben. Sei c die Anzahl der CPU-Cores pro Rechenknoten. Dann sind das genau die Paare (i,j), bei denen i die Zahlen 1 bis $n-1$ und j die Zahlen $i-(i \bmod c)+c$ bis n durchläuft. Der Parameter c wird dem Algorithmus zu Beginn übergeben.

Variante 4 ($\mathbf{GPE_{PBS}}$)

Die vierte Variante des Verfahrens unterscheidet sich am deutlichsten von den vorhergehenden. Sie ist in Algorithmus 4 dargestellt. Hierbei werden ebenfalls die Matrizen A und B zu Beginn symmetrisiert und nur Nachbarpermutationen $\pi^{i,j}$ betrachtet, die zu einer Änderung im Zielfunktionswert führen können. Der Unterschied besteht darin, dass für die aktuelle Lösung π jeweils nur ein stark eingeschränkter Bereich der Nachbarschaft $N(\pi)$ betrachtet wird. Die Größe dieses Suchbereichs wird dem Algorithmus durch den Parameter s übergeben. Im Initialisierungsschritt wird die Variable *start*, welche den Beginn des Suchbereichs markiert, auf den Wert 1 und die Variable *stop*, welche das Ende des Suchbereichs markiert, auf den kleineren der Werte s und n gesetzt.

In jedem Durchlauf von Schritt 1 werden nur Paare (i, j) aus dem jeweiligen Suchbereich betrachtet, d. h. mit $i, j \in [start, stop]$. Wurde keine Verbesserung gefunden ($swap$ ist `False`) werden in Schritt 2 die Variablen $start$ und $stop$ jeweils um den Wert s erhöht, bzw. $stop$ auf den Wert n gesetzt, falls $start + s \geq n$ ist. Solange $start < n$ ist, wird in Schritt 3 der nächste Durchlauf des Algorithmus bei Schritt 1 gestartet.

Algorithmus 4 GPE_{PBS}

Eingabe: Flussmatrix $A \in \mathbb{R}^{n \times n}$, Distanzmatrix $B \in \mathbb{R}^{n \times n}$, Permutation $\pi \in \mathcal{S}_n$, Suchbereichsgröße s, Anzahl CPU-Cores pro Knoten c

Initialisiere: $A = \frac{1}{2}(A + A^t), \quad B = \frac{1}{2}(B + B^t)$

$swap = \texttt{False}$,

$start = 1$

$stop = \min\{start + s, n\}$

Schritt 1: Für $i = start$ bis $stop - 1$

Für $j = i - (i \bmod c) + c$ bis $stop$

Berechne $\Delta = Z(A, B, \pi) - Z(A, B, \pi^{i,j})$

Falls $\Delta > 0$ ist setze

$\pi = \pi^{i,j}$

$swap = \texttt{True}$

Schritt 2: Falls $swap = \texttt{False}$ ist setze

$start = start + s$

$stop = \min\{start + s, n\}$

Schritt 3: Falls $start < n$ ist

Setze $swap = \texttt{False}$

Gehe zurück zu Schritt 1

Ausgabe: Permutation $\pi \in \mathcal{S}_n$

Für $s = n$ ergibt sich wieder der Algorithmus GPE_{BS}. Für $s < n$ entspricht ein Durchlauf von GPE_{PBS} einer partiellen Ausführung von GPE_{BS}. Die Berechnung von Δ ist weiterhin abhängig von der Problemgröße n, durch die Einschränkung der möglichen Paarkombinationen (i, j) von $\mathcal{O}(n^2)$ auf $\mathcal{O}(s^2)$ lässt sich aber die Laufzeit des Verfahrens im Vergleich zu den anderen Varianten deutlich reduzieren. Da aber Tauschversuche von i und j für $j > i + stop$ ausgeschlossen werden, ist zu erwarten, dass dieses Verfahren weniger gute Ergebnisse als die anderen liefert.

5.3 Genetische Algorithmen

Bei den genetischen Algorithmen wird mit Hilfe von evolutionsbiologischen Strategien versucht, möglichst gute Lösungen zu erzeugen.

Das zugrunde liegende Konzept ist hierbei die Übertragung von einfachen Evolutionsmechanismen wie Gen-Rekombination durch Fortpflanzung, zufällige Mutation von Genen und Selektion ("survival of the fittest") auf Optimierungsprobleme. Ziel ist es, aus einer Menge (Population) von zulässigen Lösungen (Individuen), bessere Lösungen zu erzeugen, indem durch die Anwendung dieser Mechanismen die vorteilhaften Merkmale der Individuen an eine neue Generation weitergegeben werden.

Um diese Mechanismen zu implementieren, müssen die Lösungen in geeigneter Weise codiert sein. Üblicherweise werden sie als Folge bzw. Sequenz von Bits, Zahlen oder Zeichen gespeichert. Die einzelnen Elemente der Sequenz repräsentieren die *Gene* eines Individuums. Die ganze Gensequenz wird als *Chromosom* bezeichnet und als *Allel* der Wert, mit dem ein Gen belegt ist.

Der grundsätzliche Ablauf eines genetischen Algorithmus ist folgender:

1. Zunächst wird eine Anfangspopulation von Individuen generiert, d. h. eine Menge von passend codierten zulässigen Lösungen für das Problem. Die Auswahl der Lösungen kann entweder zufällig oder nach einem heuristischen Verfahren erfolgen. Durch die Größe und Zusammensetzung der Anfangspopulation wird im Wesentlichen die Größe des Suchraums für den Algorithmus bestimmt.

2. Anschließend werden aus der aktuellen Population diejenigen Individuen ausgewählt, die an der Fortpflanzung teilnehmen dürfen und in einen sogenannten *mating pool* kopiert. Die Auswahl erfolgt mit Hilfe eines *Selektionsschemas*, welches den Fitnesswert der einzelnen Individuen berücksichtigt.

   Die *Fitness* eines Individuums ist ein Maß für seine Fortpflanzungswahrscheinlichkeit. Je fitter ein Individuum ist, umso größer soll seine Chance sein, sich fortzupflanzen. Die Fitnessfunktion muss daher einen Wert liefern, der die Qualität einer Lösung ausdrückt in Bezug zu den anderen Lösungen derselben Generation.

   Je nach verwendetem Auswahlverfahren können Individuen auch mehrfach im mating pool vertreten sein.

3. Die Individuen im mating pool werden nun zu Paaren gruppiert. Aus jedem dieser Paare werden ein oder zwei Nachkommen erzeugt, indem die Gene der Eltern miteinander kombiniert werden. Bei dieser *Rekombination* werden üblicherweise Teile der Gensequenzen der Elternchromosomen ausgetauscht.

Da ein einfaches Austauschen von Chromosomstücken nicht in jedem Fall wieder zu einer zulässigen Lösung führt, müssen die Chromosomen der Nachfahren im Anschluss gegebenenfalls "repariert" werden. Dies ist insbesondere bei Permutationen der Fall.

4. Eine vorher festgelegte *Mutationsrate* bestimmt nun, mit welcher Wahrscheinlichkeit die Gene der Nachkommen mutieren. Die *Mutation* eines Individuums erfolgt üblicherweise dadurch, dass einzelne seiner Gene zufällig verändert werden. Auch hierbei muss wieder sichergestellt werden, dass sich zulässige Lösungen ergeben.

   Die Mutation ermöglicht das Entstehen von Individuen, welche man durch die Rekombination nicht erhalten könnte. Hierdurch wird die genetische Varianz der Individuen und damit die Größe des Suchraums erhöht. Ziel dabei ist es, eine allzu schnelle Konvergenz des Algorithmus in ein lokales Optimum zu verhindern.

5. Durch ein Ersetzen von Individuen der aktuellen Population durch Nachfahren wird eine neue Population erzeugt. Diese bildet die nächste Generation. Welche der neu entstandenen Individuen in die Population eingefügt und welche der "alten" Individuen durch sie ersetzt werden, wird durch ein *Ersetzungsschema* bestimmt. Häufig benutzte Schemata sind das *generational replacement*, bei dem alle Individuen durch Nachfahren ersetzt werden und das *Prinzip der Eliten*. Bei letzterem wird ein vorher festgelegter Prozentsatz der besten Individuen in die neue Population übernommen und der Rest durch die besten der Nachfahren ersetzt ("survival of the fittest").

6. Für die neu entstandene Population beginnt man nun wieder bei Schritt 2. Dies wird solange fortgesetzt, bis ein Abbruchkriterium für den Algorithmus erfüllt ist. In diesem Fall wird die beste im Laufe des Verfahrens gefundene Lösung ausgegeben.

   Abbruchbedingungen können beispielsweise sein, dass die aktuelle Population keine oder kaum unterschiedliche Individuen mehr enthält, dass eine vorher festgelegte Anzahl von Generationen erreicht ist oder dass die aktuell beste Lösung "gut genug" ist.

Algorithmus 5 zeigt einen genetischen Algorithmus für das QAP. In welcher Weise die einzelnen Schritte implementiert wurden, wird im Folgenden beschrieben.

Die Anfangspopulation P_1 wird aus p zufällig[2] gewählten Permutationen $\pi_i \in \mathcal{S}_n$ zusammengestellt (Schritt 1). Die Zahl $p \in \mathbb{N}$ entspricht hierbei dem Wert des Übergabeparameters *Populationsgröße* und die Menge $\mathcal{S}_n$ bildet die Menge aller zulässigen Lösungen

[2] "Zufällig" bedeutet hierbei: Mit Hilfe eines Zufallsgenerators ausgewählt.

Algorithmus 5 Genetischer Algorithmus für das QAP (GA)

Eingabe: Flussmatrix $A \in \mathbb{R}^{n \times n}$, Distanzmatrix $B \in \mathbb{R}^{n \times n}$

Parameter: Populationsgröße, Generationsanzahl, Mutationsrate, Eliteprozentsatz

Initialisiere: Generationszähler $k = 1$

Schritt 1: Generierung einer Anfangspopulation P_1 von Individuen $\pi_i \in \mathcal{S}_n$

Bestimme $\pi_{best} \in P_1$ mit $Z(A, B, \pi_{best}) \leq Z(A, B, \pi_i) \ \forall\, \pi_i \in P_1$

Setze $Z_{best} = Z(A, B, \pi_{best})$

Schritt 2: Bewertung der Individuen $\pi_i \in P_k$ mit Hilfe einer *Fitnessfunktion*

Schritt 3: Auswahl der Eltern $\pi_i \in P_k$ nach einem *Selektionsschema*

Schritt 4: Erzeugung der Nachkommen durch *Rekombination* der Elternchromosomen

Schritt 5: *Mutation* der Gene der Nachkommen

Schritt 6: Bildung einer neuen Population P_{k+1} mit Hilfe eines *Ersetzungsschemas*

Bestimme $\pi_x \in P_{k+1}$ mit $Z(A, B, \pi_x) \leq Z(A, B, \pi_i) \ \forall\, \pi_i \in P_{k+1}$

Falls $Z(A, B, \pi_x) < Z_{best}$ setze

$\pi_{best} = \pi_x$

$Z_{best} = Z(A, B, \pi_x)$

Setze $k = k + 1$

Schritt 7: Wenn Abbruchbedingung nicht erfüllt, gehe zurück zu Schritt 2

Ausgabe: Individuum $\pi_{best} \in \mathcal{S}_n$

für QAP(A,B). Die Chromosomen der einzelnen Individuen π_i werden repräsentiert durch Sequenzen bzw. n-Tupel der Form

$$\Big(\pi_i(1), \pi_i(2), \ldots, \pi_i(n)\Big)$$

Weiterhin besteht die Möglichkeit, bestimmte Individuen in die Anfangspopulation "einzuschleusen". Dies können beispielsweise Lösungen aus früheren Läufen des Algorithmus oder durch andere Heuristiken bestimmte Lösungen sein. Die bisher beste Lösung π_{best} und der bisher beste Zielfunktionswert Z_{best} werden bestimmt. Dafür muss für alle Individuen π_i der Zielfunktionswert $Z(A, B, \pi_i)$ berechnet werden.

In Schritt 2 werden alle Individuen gemäß ihrer Fitness bewertet. Die Fitness $f(\pi_i)$ eines Individuums π_i in Generation k wird berechnet durch

$$f(\pi_i) = Z^k_{\max} + Z^k_{\min} - Z(A, B, \pi_i)$$

wobei $Z^k_{\max}$ das Maximum und $Z^k_{\min}$ das Minimum der Zielfunktionswerte in Generation k bezeichnet. Hierdurch bekommen Lösungen mit vergleichsweise kleinem Zielfunktionswert einen hohen Fitnesswert zugewiesen und umgekehrt.

Die Selektion der Eltern in Schritt 3 erfolgt mit Hilfe des Selektionsschemas *gewichtete Rouletterad-Selektion* (vgl. [40]). Hierfür wird zunächst für jedes Individuum $\pi_i \in P_k$ seine *Selektionswahrscheinlichkeit* $s(\pi_i)$ berechnet. Es ist

$$s(\pi_i) = \frac{f(\pi_i)}{\sum_{k=1}^{p} f(\pi_k)}$$

Je größer also die Fitness des Individuums ist, umso größer ist auch seine Selektionswahrscheinlichkeit. Als nächstes wird für jedes Individuum π_i die jeweilige *kumulierte* Selektionswahrscheinlichkeit $q(\pi_i)$ bestimmt. Es ist

$$q(\pi_i) = \sum_{k=1}^{i} s(\pi_i)$$

Solange bis p Eltern ausgewählt wurden, wird nun wiederholt eine Zufallszahl $r \in (0, 1]$ generiert. Ist $r < q(\pi_1)$, so wird das Individuum π_1 ausgewählt. Andernfalls wird das Individuum π_i mit $i \geq 2$ gewählt, für das gilt $q(\pi_{i-1}) < r \leq q(\pi_i)$.

In Schritt 4 werden jeweils zwei Individuen aus der Menge der Eltern herausgenommen und durch Rekombination ihrer Gene zwei Nachkommen erzeugt. Hierfür wurden zwei verschiedene Rekombinationsverfahren, das *Partially Matched Crossover* und das *Cycle Crossover* (vgl. [40]), implementiert. Beide Verfahren werden hier kurz beschrieben und anhand eines Beispiels erläutert.

Beim Partially Matched Crossover werden zunächst beide Nachkommen als exakte Kopien der Eltern erstellt. Dann werden zwei *Crossover-Stellen* zufällig festgelegt. Die Belegung der Gensequenzen innerhalb dieser Stellen bestimmt, wie die Chromosomen verändert werden: Es werden jeweils die Allele einander zugeordnet, die sich an einander entsprechenden Stellen der Chromosomen befinden. Nacheinander werden nun bei beiden Kindern die Allele gemäß diesen Zuordnungen ausgetauscht.

Abbildung 5.1 zeigt ein Beispiel für den Ablauf dieses Verfahrens. Im ersten Schritt werden die Chromosomen $C1$ und $C2$ der Kinder als Kopien der Elternchromosomen $E1$ und $E2$ angelegt und ein Crossoverbereich festgelegt (hier gekennzeichnet durch ||). Innerhalb dieses Bereichs sind die Gene von $C1$ mit den Werten 4, 5 und 6 belegt und die Gene von $C2$ mit 7, 1 und 4. Daraus ergeben sich die Zuordnungen 4-7, 5-1 und 6-4. In jedem weiteren Schritt werden nun die Allele gemäß jeweils einer dieser Zuordnungen bei beiden Chromosomen ausgetauscht. Die vertauschten Allele sind hier jeweils durch Fettschrift gekennzeichnet.

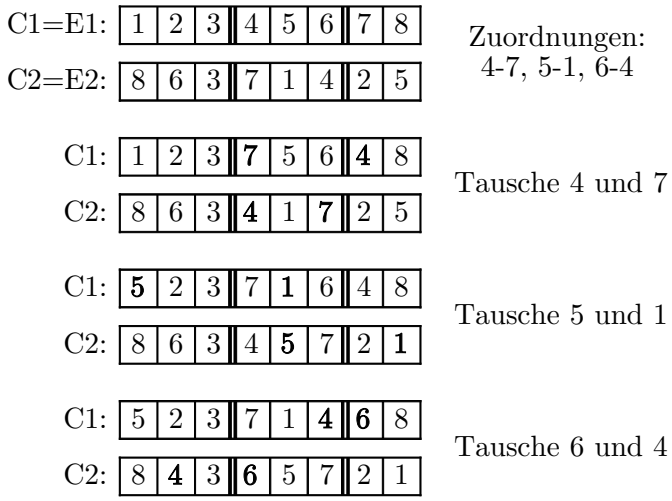


Abbildung 5.1: Partially Matched Crossover

Beim *Cycle Crossover* wird eine geschlossene Abfolge von Allelen innerhalb der beiden Elternchromosomen identifiziert. Dieser Zyklus gibt vor, welche Gene ausgetauscht werden um zwei Nachfahren zu erhalten. Zunächst wird dazu ein beliebiges Gen als Startposition des Verfahrens gewählt. Das Allel dieses Gens auf dem Chromosom des ersten Elternteils bildet das erste Element des Zyklus. Nun wird im zweiten Elternchromosom das Gen gesucht, welches dieses erste Zykluselement trägt. Das entsprechende Allel des ersten Elternchromosoms bildet das zweite Zykluselement, welches nun wieder im Chromosom des zweiten Elternteils gesucht wird. Dies wird so lange fortgeführt, bis sich ein Kreis gebildet hat. Die Nachfahren erhält man nun, indem jeweils die Gene, deren Allele im Zyklus vorkommen, direkt von einem Elternteil übernommen werden. Die restlichen Gene werden vom jeweils anderen Elternteil übernommen. Hierbei führen Zyklen der Länge 1 oder n dazu, dass mit den Eltern identische Kinder erzeugt werden.

In Abbildung 5.2 wird der Ablauf dieses Verfahrens anhand eines Beispiels demonstriert. Als Startposition wird das zweite Gen gewählt und das zugehörige Allel **2** auf Chromosom $E1$ bildet das erste Element des Zyklus. In Chromosom $E2$ findet sich dieses Allel auf dem siebten Gen. Das Allel **7** des entsprechenden Gens von $E1$ bildet das zweite Zykluselement. Auf diese Weise werden weiterhin die Allele **4** und **6** als drittes und viertes Zykluselement identifiziert. Die Suche nach Allel **6** in Chromosom $E2$ führt zurück zur Startposition, somit ergibt sich insgesamt der Zyklus 2-7-4-6. Die zugehörigen Gene 2, 4, 6, und 7 werden von $E1$ in den Nachfahren $C1$ übernommen und von $E2$ in $C2$. Die restlichen Gene 1, 3, 5 und 8 werden von $E1$ in $C2$ und von $E2$ in $C1$ übernommen.

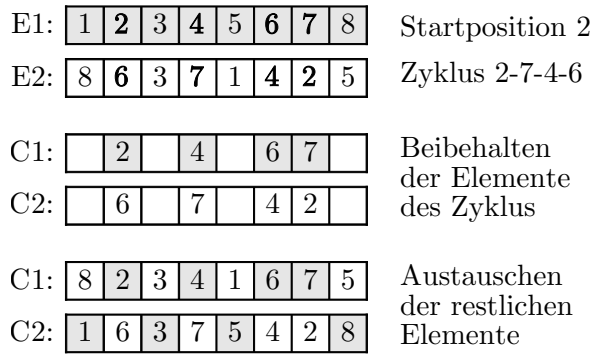


Abbildung 5.2: Cycle Crossover

In Schritt 5 des Algorithmus wird für jeden der insgesamt p Nachkommen eine Zufallszahl $r \in (0, 1]$ generiert und mit dem Wert des Parameters *Mutationsrate* $m \in (0, 1]$ verglichen. Ist $r < m$ wird eine *Mutationsfunktion* $M : \mathcal{S}_n \to \mathcal{S}_n$ auf den entsprechenden Nachkommen angewandt. Hierfür wurden zwei verschiedene Mutationsfunktionen implementiert. Bei beiden wird eine bestimmte Anzahl an zufällig gewählten Genen eines Individuums zyklisch vertauscht. Der Unterschied bei den Funktionen besteht darin, dass in einem Fall diese Anzahl im Voraus festgelegt ist und im anderen Fall bei jedem Aufruf der Mutationsfunktion zufällig bestimmt wird.

Die Bildung einer neuen Generation von Individuen in Schritt 6 wird nach dem oben beschriebenen Prinzip der Eliten durchgeführt. Für alle Individuen π_i der neuen Population P_{k+1} wird der Zielfunktionswert $Z(A, B, \pi_i)$ bestimmt und gegebenenfalls die Variablen π_{best} und Z_{best} aktualisiert.

Das Verfahren wird beendet (Schritt 7), wenn eine maximale Anzahl von Generationen erreicht ist. Diese wird durch den Übergabeparameter *Generationsanzahl* festgelegt.

Zu bemerken ist noch, dass sich der Algorithmus anhand der Parameter Populationsgröße, Generationsanzahl, Mutationsrate und Eliteprozentsatz in gewisser Weise steuern lässt. Je größer die Populationsgröße gewählt wird, umso höher ist auch die Wahrscheinlichkeit, dass sich schon in der Anfangspopulation "gute" Individuen befinden und dass ein breiter Bereich des gesamten Suchraums $\mathcal{S}_n$ abgedeckt wird. Die Evolution einer großen Population erfordert allerdings auch einen hohen Rechenaufwand und damit eine hohe Laufzeit des Verfahrens. Als "Daumenregel" für die Populationsgröße gilt, dass sie mindestens der Problemgröße (in diesem Fall n) entsprechen sollte. Direkten Einfluss auf die Laufzeit nimmt natürlich auch die maximale Generationsanzahl, bis zu der sich die Individuen entwickeln dürfen. Allerdings bedarf es auch immer einer gewissen Zeit, bis der gewünschte Effekt der Evolution spürbar zum Tragen kommt.

Mit der Mutationsrate und dem Eliteprozentsatz lässt sich das Konvergenzverhalten des Algorithmus beeinflussen. Bei einer Mutationsrate von 0% ist nur durch das Rekombinationsverfahren festgelegt, welche neuen Individuen entstehen können. Dies führt in den meisten Fällen dazu, dass es innerhalb recht kurzer Zeit nur noch sehr wenig oder gar keine genetische Varianz innerhalb der Populationen gibt. Wird die Mutationsrate zu groß gewählt, wird durch die Mutation möglicherweise jeder Effekt der Rekombination zunichte gemacht, indem die "guten Merkmale", die bei einer Reproduktion weitergegeben wurden, wieder zerstört werden. Üblicherweise wählt man deshalb kleine Mutationsraten von 1% bis maximal 10%. Bei einem Eliteprozentsatz von 0 werden alle Individuen der alten Population durch ihre Nachfahren ersetzt. Dies kann, abhängig von der Rekombinationsfunktion, dazu führen, dass die durchschnittliche Fitness der Population nur sehr langsam steigt oder sogar sinkt und gute Merkmale sehr schnell wieder verloren gehen. Auch mit einem sehr hohen Eliteprozentsatz wird die Entwicklung der Populationen eher gebremst. Ergebnis und Laufzeit des Verfahrens hängen somit von diesen Größen ab. Die "richtige" Wahl dieser Parameter ist sehr schwierig und erfordert insbesondere viel Zeit, um verschiedene Einstellungen auszuprobieren.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der in Kapitel 5 beschriebenen Heuristiken vorgestellt. Die praktische Anwendbarkeit der Heuristiken zur Optimierung der MPI-Prozess-Kommunikation wird diskutiert und Ergebnisse für die optimierte Zuordnung von MPI-Prozessen zu CPU-Cores beim DLR TAU-Code Strömungslöser gegeben. Als Testdatensätze sind dabei die folgenden (zum Teil frei verfügbaren) aerodynamischen Konfigurationen verwendet worden (siehe Abbildung 6.1):

a. NACA-0012: unstrukturiertes Netz mit ca. 5.000 Punkten

b. RAE-2822: hybrides Netz mit ca. 150.000 Punkten

c. DLR-F6: hybrides Netz mit ca. 2 Mio. Punkten

d. Hochauftriebskonfiguration: hybrides Netz mit ca. 13 Mio. Punkten

Die Kommunikationsdaten des TAU-Strömungslösers wurden direkt bei der Partitionierung des jeweiligen Rechennetzes extrahiert. Die Koeffizientenmatrizen für das QAP zur Optimierung der Prozess-Kommunikation wurden nach der in Kapitel 4 beschriebenen Modellierung aufgestellt. Da sich in den meisten Fällen keine oder nur unwesentliche Unterschiede zwischen der Verwendung der einfachen und der erweiterten Distanzmatrix ergeben haben, werden hier nur die Ergebnisse für die einfache Modellierung der Distanz nach Definition 4.5 gegeben. Der Wert für die Distanz zwischen CPU-Cores auf demselben Rechenknoten wurde dabei auf den Wert 1 und der Wert für die Distanz zwischen CPU-Cores auf verschiedenen Knoten auf den Wert 10 gesetzt. Weiterhin wurde jeweils von einer Anzahl von 8 CPU-Cores pro Rechenknoten ausgegangen.

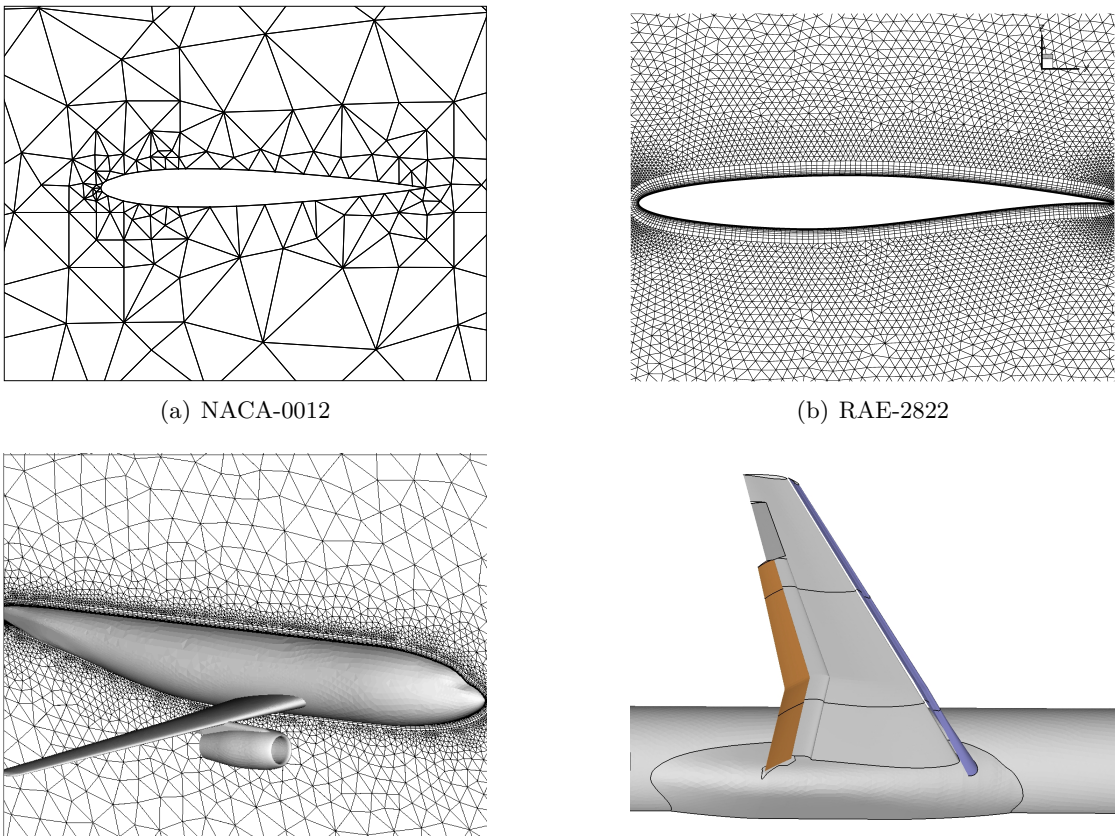
Bei der praktischen Anwendbarkeit der Heuristiken ist natürlich neben der erzielten Verbesserung im Zielfunktionswert auch die Laufzeit für das Verfahren relevant. Dies soll das folgende Beispiel einer typischen Simulationsrechnung noch einmal verdeutlichen:

Beispiel 6.1 (Parallele TAU Rechnung) Eine parallele Rechnung der DLR-F6 Konfiguration (Datensatz c) mit dem TAU-Strömungslöser[1] braucht auf einem Rechencluster mit 32 CPUs für 1300 Iterationen ca. 1950 Sekunden. Bei der Verwendung von 128 CPUs benötigt die gleiche Rechnung noch 832 Sekunden und bei 256 CPUs lediglich 663 Sekunden (vgl. [2] Seite 39).

[1] Einstellungen: Runge-Kutta Verfahren, 4w Mehrgitterzyklus, Spalart-Allmaras Turbulenzmodell

Bei der Anwendung einer der Heuristiken für die Optimierung der Prozess-Kommunikation beim TAU-Strömungslöser muss die Laufzeit der Heuristik unterhalb der durch die Optimierung erzielten Zeiteinsparung beim Strömungslöser liegen, um insgesamt zu einer Einsparung von Rechenzeit zu führen. Somit ist für die praktische Verwendung der Heuristiken deren Laufzeit ein kritisches Ausschlusskriterium.

Nach obigem Beispiel ist weiterhin zu erwarten, dass die absolute Zeiteinsparung beim TAU-Strömungslöser durch die Optimierung der Kommunikation mit steigender Anzahl der verwendeten CPUs sinkt. Dies verstärkt noch die Wichtigkeit der Zeitkomponente bei der Auswahl einer Heuristik für diese Aufgabe.



(a) NACA-0012

(b) RAE-2822

(c) DLR-F6

(d) Hochauftriebskonfiguration

Abbildung 6.1: Aerodynamische Konfigurationen

6.1 Ergebnisse der Heuristiken

Die Ergebnisse der einzelnen Heuristiken sind, je nach Verfahren zusammengefasst, in den folgenden Tabellen dargestellt. Der Zielfunktionswert einer durch eine Heuristik bestimmten Lösung wurde jeweils mit dem Zielfunktionswert der initialen Zuordnung verglichen. Neben der prozentualen Verbesserung in der Zielfunktion ist auch die Laufzeit der jeweiligen Heuristik in Sekunden mit aufgeführt.

Konstruktionsverfahren

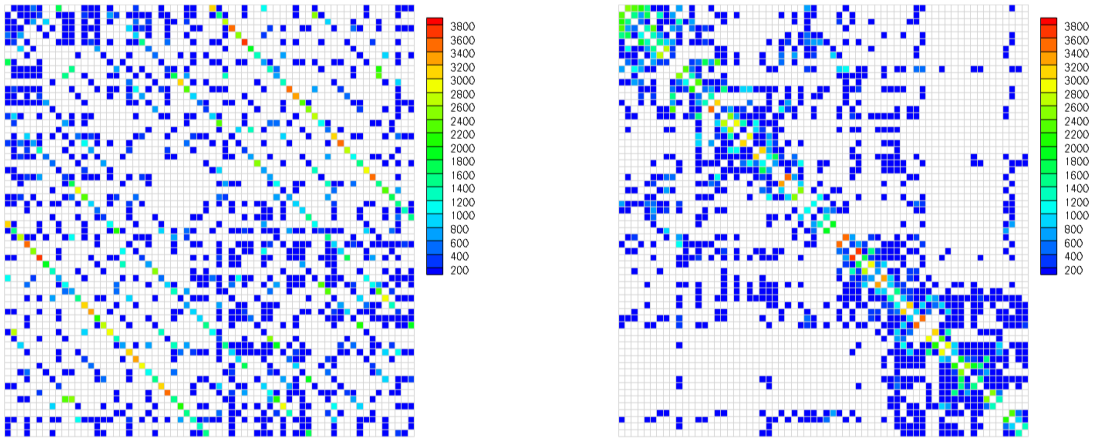
Mit dem Verfahren "Einfache Zuordnung" (EZ) konnte bei allen Testfällen nur eine sehr geringe Verbesserung im Zielfunktionswert von 0,7% bis maximal 9% erzielt werden. Daher werden hier nur die Ergebnisse für die Heuristik "Erweiterte Einfache Zuordnung" (EEZ) explizit angegeben. Die Werte sind für alle aerodynamischen Konfigurationen in Tabelle 6.1 zu finden.

n	64		128		256		1024	
	%	sec	%	sec	%	sec	%	sec
NACA-0012	44,15	0,01	35,54	0,04	32.40	0.17	23,34	2,72
RAE-2822	55,28	0,01	56,22	0,04	50,84	0,16	44,46	2,77
DLR-F6	62,00	0,01	54,41	0,04	55,67	0,17	47,13	2,97
HL-Konf.	65,96	0,01	64,91	0,04	58,29	0,16	48,91	2,86

Tabelle 6.1: Ergebnisse des Konstruktionsverfahrens EEZ für die aerodynamischen Konfigurationen a, b, c und d.

Auffallend ist, dass die Laufzeit der EEZ, bedingt durch ihre einfache Konstruktion, sehr kurz ist und selbst bei $n = 1024$ weniger als 3 Sekunden in Anspruch nimmt. In allen Fällen konnte hier der Zielfunktionswert deutlich verbessert werden.

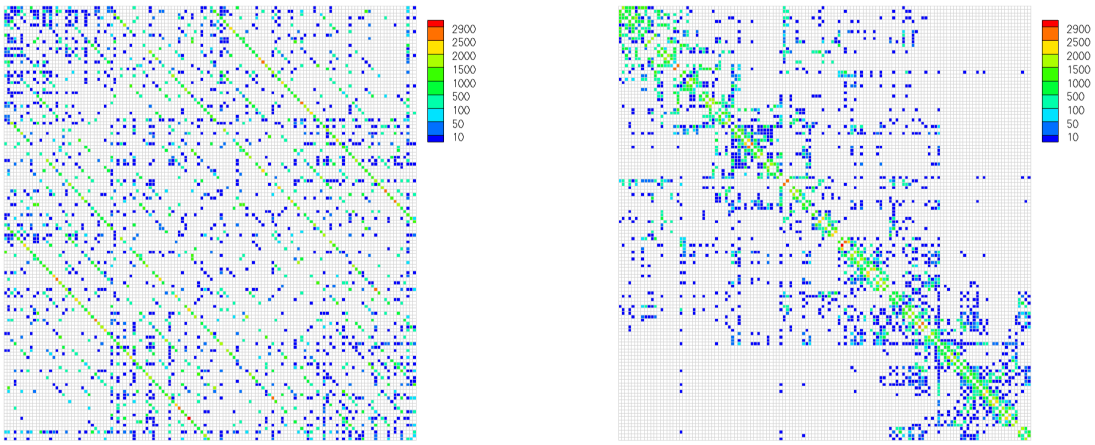
In den Abbildungen 6.2, 6.3 und 6.4 ist das Ergebnis der verbesserten Zuordnung für den Testfall DLR-F6 noch einmal grafisch dargestellt. Auf der linken Seite ist jeweils die nach der Beschreibung in Kapitel 4.1.1 aufgestellte Kommunikationsmatrix zu sehen. Auf der rechten Seite wurden die Einträge der Matrix entsprechend der durch die Heuristik EEZ berechneten Lösung permutiert. Alle nicht farblich gekennzeichneten Einträge entsprechen dabei dem Wert 0. Wie in Kapitel 4.2.2 gezeigt, befinden sich in der zugehörigen Distanzmatrix entlang der Hauptdiagonalen die 8×8-Blöcke, die den Wert für die Distanz zwischen CPU-Cores auf demselben Rechenknoten angeben. "Ideal" wäre natürlich, wenn sich nach der Umordnung alle positiven Einträge der Kommunikationsmatrix innerhalb entsprechender Diagonalblöcke befinden würden. In diesem Fall wäre



(a) Initiale Zuordnung (b) Optimierte Zuordnung (EEZ)

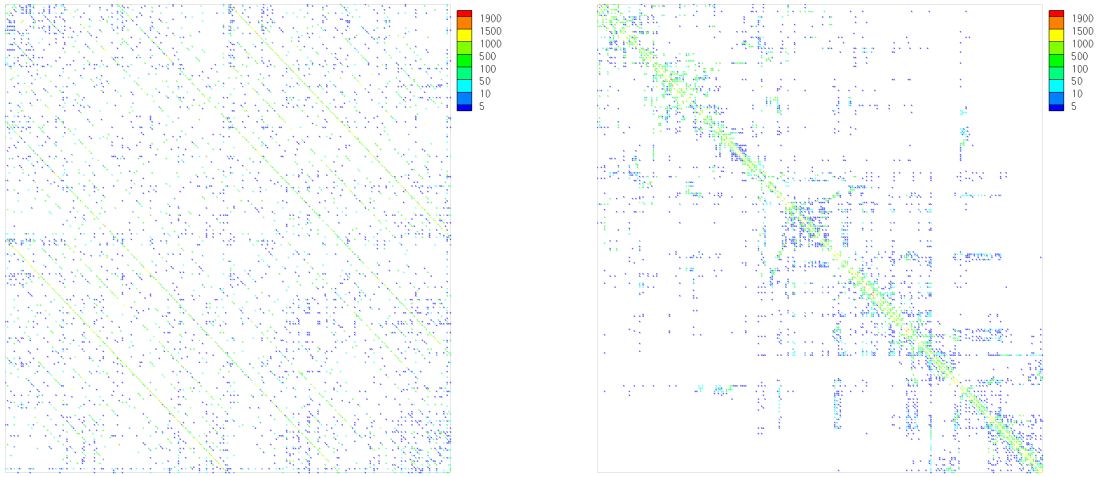
Abbildung 6.2: Kommunikationsmatrizen DLR-F6 für 64 Partitionen

keine knotenexterne Kommunikation über das Netzwerk mehr erforderlich. Durch die Abbildungen wird aber deutlich, dass dies hier gar nicht möglich ist. Im Allgemeinen hat jeder Prozess mehr als 8 Kommunikationspartner, so dass immer auch knotenexterne Kommunikation stattfinden muss. Man kann aber gut erkennen, dass sich durch die optimierte Zuordnung insbesondere die Einträge mit großem Wert um die Hauptdiagonale konzentrieren, so dass hauptsächlich kleine Nachrichten noch über das Netzwerk kommuniziert werden müssen.



(a) Initiale Zuordnung (b) Optimierte Zuordnung (EEZ)

Abbildung 6.3: Kommunikationsmatrizen DLR-F6 für 128 Partitionen



(a) Initiale Zuordnung (b) Optimierte Zuordnung (EEZ)

Abbildung 6.4: Kommunikationsmatrizen DLR-F6 für 256 Partitionen

Verbesserungsverfahren

Die verschiedenen Varianten des "Greedy Pair-Exchange" wurden für die aerodynamischen Konfigurationen NACA, RAE und F6 getestet. Die Laufzeit aller Verfahren wurde dabei auf maximal 30 Minuten (1800 Sekunden) begrenzt. In den folgenden Tabellen ist die Laufzeit jeweils mit einem Stern (*) gekennzeichnet, wenn der Lauf nach 30 Minuten abgebrochen wurde.

n	64		128		256		1024	
	%	sec	%	sec	%	sec	%	sec
NACA-0012	47,45	34,11	43,88	321,51	41,53	1800*	2,17	1800*
RAE-2822	60,41	33,87	55,23	325,28	55,33	1788	3,73	1800*
DLR-F6	62,29	26,77	58,63	272,13	53,97	1800*	2,59	1800*

Tabelle 6.2: Ergebnisse des Verbesserungsverfahrens GPE für die aerodynamischen Konfigurationen a, b und c.

In Tabelle 6.2 sind die Ergebnisse für die Heuristik GPE für alle drei Konfigurationen zusammengefasst. Als Startlösung wurde hierbei die initiale Zuordnung, d. h. die identische Permutation $id \in \mathcal{S}_n$ verwendet. Man sieht sofort, dass mit zunehmender Problemgröße die Laufzeit der Heuristik enorm steigt. Während die Verbesserungen in der Zielfunktion im Vergleich zur EEZ bei $n = 256$ noch gut sind, ist die Laufzeit mit teilweise über

1800 Sekunden schon nicht mehr akzeptabel. Bei einer Laufzeit von einer halben Stunde erzielt diese Heuristik für $n = 1024$ kaum noch eine Verbesserung. Grade bei größeren n (DLR-F6 ab $n = 256$) fällt auf, dass GPE in der Lösungsgüte zum Teil deutlich hinter die EEZ zurückfällt.

Um die Unterschiede besser beurteilen zu können, sind die Ergebnisse der unterschiedlichen Varianten des "Greedy Pair-Exchange" für jede der aerodynamischen Konfigurationen einzeln in den Tabellen 6.3, 6.4 und 6.5 aufgeführt. Dies ermöglicht auch einen direkten Vergleich im Hinblick auf die Laufzeit der unterschiedlichen Varianten. Hierbei wurde als Startlösung die durch Heuristik EEZ berechnete Permutation verwendet.

n	64		128		256		1024	
	%	sec	%	sec	%	sec	%	sec
GPE	48,87	21,03	46,60	222,35	43,74	1800*	23,95	1800*
GPE_S	48,87	3,40	46,60	36,31	43,94	461	27,39	1800*
GPE_{BS}	48,87	2,94	46,60	35,04	43,94	439	27,39	1800*
GPE_{PBS}	48,87	2,97	44,15	16,23	43,79	53,14	31,66	877

Tabelle 6.3: Ergebnisse der Verbesserungsverfahren für den NACA-0012 Fall

n	64		128		256		1024	
	%	sec	%	sec	%	sec	%	sec
GPE	61,74	26,07	60,93	156,63	55,72	1800*	44,46	1800*
GPE_S	61,74	4,24	60,93	25,26	55,72	330,14	46,61	1800*
GPE_{BS}	61,74	3,71	60,93	24,19	55,72	321,66	46,61	1800*
GPE_{PBS}	61,74	3,78	60,42	13,18	55,61	39,44	51,37	891

Tabelle 6.4: Ergebnisse der Verbesserungsverfahren für den RAE-2822 Fall

n	64		128		256		1024	
	%	sec	%	sec	%	sec	%	sec
GPE	62,40	14,08	60,06	168,56	59,63	1236	47,26	1800*
GPE_S	62,40	2,31	60,06	26,75	59,63	196,01	48,98	1800*
GPE_{BS}	62,40	2,05	60,06	26,07	59,63	197,31	48,98	1800*
GPE_{PBS}	62,40	2,02	60,06	8,12	58,56	33,50	51,64	814

Tabelle 6.5: Ergebnisse der Verbesserungsverfahren für den DLR-F6 Fall

Auf den ersten Blick lässt sich feststellen, dass das Verfahren GPE mit der Startzuordnung der EEZ bei deutlich geringerer Laufzeit bessere Ergebnisse liefert, als mit der initialen Zuordnung. Dies gilt ebenso für die anderen Varianten, weshalb deren Ergebnisse für die initiale Zuordnung hier nicht mehr explizit angegeben wurden.

Wie erwartet unterscheiden sich die Verfahren GPE, GPE_S und GPE_BS nur in der Laufzeit, sofern sie komplett durchlaufen und nicht nach 30 Minuten abgebrochen werden. Gegenüber GPE lässt sich die Laufzeit durch das "Symmetrisieren" der Matrizen bei GPE_S deutlich verringern. Die durch das Einsparen von Tauschversuchen innerhalb der zu einem Rechenknoten gehörenden "Blocks" einer Permutation erzielte weitere Laufzeitreduktion bei der Variante GPE_BS, fällt allerdings bei 8 CPUs pro Knoten noch nicht sonderlich groß aus.

Die Suchbereichsgröße s für das Verfahren GPE_PBS wurde bei den hier vorgestellten Tests auf 64 gesetzt, weshalb sich die Ergebnisse für $n = 64$ auch für diese Variante nicht von den anderen unterscheiden. Für $n = 128$ und $n = 256$ lieferte GPE_PBS nur geringfügig schlechtere Ergebnisse, bei allerdings deutlich geringerer Laufzeit. Für $n = 1024$ sind die Ergebnisse sogar besser als bei den anderen Varianten, was daran liegt, dass dieses Verfahren hierbei als einziges in deutlich weniger als 30 Minuten komplett durchgelaufen ist, also nicht wie die anderen zwischendurch abgebrochen wurde.

Die Laufzeit der Verbesserungsverfahren ist im direkten Vergleich zum Konstruktionsverfahren EEZ deutlich höher (z. B. DLR-F6, $n = 1024$, EEZ zu GPE_PBS, Faktor 274 bei lediglich 4.51% besserem Zielfunktionswert). GPE_PBS ist in dem Vergleich das schnellste Verbesserungsverfahren und eignet sich aufgrund der moderaten Laufzeit (bei $n = 256$ zwischen 31 und 39 Sekunden) für Probleme bis $n = 256$, sofern man das Ergebnis der EEZ als Startlösung verwendet.

Genetischer Algorithmus

Der genetische Algorithmus (GA) ist nur mit der DLR-F6 Konfiguration getestet worden. Die in Tabelle 6.6 angegebenen Parametereinstellungen haben sich dabei als sinnvoll erwiesen. Alle Tests wurden mit der Rekombination "Partially Matched Crossover" sowie den folgenden Mutationsfunktionen M_i durchgeführt:

- M_1**:** Jeweils 2 Gene werden vertauscht.
- M_2**:** Jeweils 10 Gene werden zyklisch vertauscht.
- M_3**:** Jeweils 10% der Gene werden zyklisch vertauscht.
- M_4**:** Eine jedesmal zufällig bestimmte Anzahl an Genen wird zyklisch vertauscht.

n	64	128	256	384	512	1024
Populationsgröße	100	100	200	384	512	1024
Mutationsrate	1%	1%	1%	1%	1%	1%
Eliteprozentsatz	10%	10%	10%	10%	10%	10%
Durchläufe	20-22	20	9-20	5	5	1

Tabelle 6.6: Parametereinstellungen für den genetischen Algorithmus

Eine Besonderheit des genetischen Algorithmus ist, dass dieser aufgrund der Zufallsfaktoren bei der Generierung der Anfangspopulation sowie bei der Mutations-, Reproduktions- und Selektionsfunktion nicht deterministisch ist. Das bedeutet, dass zwei Läufe des GA nicht unbedingt zum selben Ergebnis führen müssen. Aus diesem Grund wurden für jede Probleminstanz mehrere Durchläufe angestoßen und in den Ergebnistabellen nur die Mittelwerte eingetragen. Die Anzahl der Durchläufe pro Probleminstanz sind ebenfalls in Tabelle 6.6 angegeben.

Tabellen 6.7 und 6.8 zeigen die Ergebnisse des genetischen Algorithmus (GA) für die in Tabelle 6.6 gegebenen Einstellungen und die Mutationsfunktion M_1. Dabei bezeichnet GA+EEZ, dass in die Anfangspopulation die Lösung der Heuristik EEZ eingeschleust wurde. Ihr Ergebnis ist zum Vergleich jeweils noch einmal mit angegeben (ohne Laufzeit).

n	64		128		256		384		512	
	%	sec	%	sec	%	sec	%	sec	%	sec
EEZ	62,00	-/-	54,41	-/-	55,67	-/-	51,58	-/-	49,77	-/-
GA	57,60	122	50,52	380	43,09	4620	40,88	23496	36,79	100904
GA+EEZ	62,39	121	58,71	257	57,93	2139	53,64	9825	51,33	34929

Tabelle 6.7: Ergebnisse des Genetischen Algorithmus für den DLR-F6 Fall mit Generationenanzahl 10.000

Betrachtet man die Ergebnisse aus Tabelle 6.7 wird schnell klar, dass die Variante GA sehr viel Zeit benötigt, um eine Verbesserung zu erzielen, die an die Resultate der EEZ jedoch nicht ganz herankommt. Ab $n = 256$ ist die Laufzeit im Vergleich zur EEZ enorm und die Verbesserung unterdurchschnittlich. Bei der Verwendung des Ergebnisses der EEZ in der Anfangspopulation sieht man klare Verbesserungen im Zielfunktionswert, sowohl gegenüber dem Ergebnis von GA als auch von EEZ. Besonders für große n ist auch die Laufzeit von EEZ+GA deutlich geringer als die von GA, jedoch wird auch hierbei schon mehr als eine halbe Stunde Laufzeit für $n = 256$ benötigt.

Aus Platzgründen nicht in der Tabelle mit aufgeführt sind die Ergebnisse für $n = 1024$. Hier benötigte ein Durchlauf von GA 1738662 Sekunden (ca. 20 Tage) und erzielte dabei nur eine Verbesserung im Zielfunktionswert von $30,21\%$ und GA+EEZ benötigte 531994 Sekunden (ca. 6 Tage) für eine Verbesserung von $48,12\%$.

n	64		128		256	
	%	sec	%	sec	%	sec
EEZ	62,00	-/-	54,41	-/-	55,67	-/-
GA	57,70	1116	55,56	2677	53,22	23369
GA+EEZ	62,40	1114	60,08	2351	59,52	18972

Tabelle 6.8: Ergebnisse des Genetischen Algorithmus für den DLR-F6 Fall mit Generationenanzahl 100.000

Beim Vergleich der Ergebnisse der Tabellen 6.7 und 6.8 kann man erkennen, dass durch eine Erhöhung der Generationenanzahl zwar eine (bei GA deutliche) Verbesserung in der Zielfunktion herbeizuführen ist, dafür aber auch der Zeitaufwand exorbitant steigt (GA $n = 128$ von $50,52\%$ bei 380 s auf $55,56\%$ bei 2677 s bzw. $n = 256$ von $43,09\%$ bei 4620 s auf $53,22\%$ bei 23369 s). Bei GA+EEZ ergibt sich, trotz erheblich gestiegener Laufzeit, nur eine marginale Verbesserung im Vergleich zum Ergebnis aus Tabelle 6.7.

Eine Verkleinerung der Populationsgröße führt im Allgemeinen zu weniger guten Ergebnissen, aber leider nicht unbedingt zu einer akzeptablen Laufzeit. Ein Beispiel hierfür wird in Tabelle 6.9 gegeben. Obwohl hier bei GA+EEZ durch die Beschränkung der Populationsgröße von 200 auf 100 die Lösungsqualität nicht einmal wesentlich sinkt, ist die Laufzeit von 1019 Sekunden trotzdem noch viel zu lang.

Populationsgröße	200		100	
	%	sec	%	sec
GA	43,09	4620	37.73	2138
GA+EEZ	57,93	2139	57.39	1019

Tabelle 6.9: Ergebnisse des Genetischen Algorithmus für den DLR-F6 Fall mit $n = 256$ und Generationenanzahl 10.000 sowie unterschiedlicher Populationsgröße

Auf alle bisher beschriebenen Probleminstanzen wurde das Verfahren GA+EEZ auch mit den Mutationsfunktionen M_2, M_3 und M_4 angewandt. Dabei ergaben sich aber nur in drei Fällen spürbare Unterschiede zur Verwendung von M_1: Bei Generationenanzahl 10.000 war mit M_3 und M_4 bei $n = 384$ die Verbesserung im Zielfunktionswert um durchschnittlich 1% und bei $n = 512$ um durchschnittlich 3% besser. Bei Generationenanzahl 10.000 war mit M_4 bei $n = 256$ dagegen das Ergebnis durchschnittlich um 1% schlechter.

Aus Zeitgründen konnten bisher nicht alle Test mit der Rekombinationsfunktion "Cycle-Crossover" wiederholt werden. Erste Tests zeigten allerdings keine wesentlichen Unterschiede in Zielfunktion und Laufzeit.

Gilmore-Lawler-Schranke

Zum Schluss sollen hier auch noch die Ergebnisse der in Kapitel 3.4 vorgestellten unteren Schranken für das quadratische Zuordnungsproblem gegeben werden. Die Eigenwertschranke[2] erreichte bei allen Testfällen und -größen negative Werte und erwies sich damit leider für diese Fälle als völlig unbrauchbar. Die Ergebnisse werden hier deshalb nicht explizit angegeben.

n	64	128	256	1024
NACA-001	78,77	75,37	70,16	59,93
RAE-2822	88,83	87,50	86,95	85,54
DLR-F6	86,08	85,57	85,28	84,83
HL-Konf.	89,26	88,73	87,81	86,15

Tabelle 6.10: Ergebnisse der Gilmore-Lawler Schranke als Verbesserung in Prozent gegenüber dem initialen Zielfunktionswert.

Die Ergebnisse der Gilmore-Lawler-Schranke für alle aerodynamischen Konfigurationen sind in Tabelle 6.10 aufgelistet. Die maximal mögliche Verbesserung des initialen Zielfunktionswertes wird hierdurch für den NACA Fall durchschnittlich auf ca. 71% und für die Fälle RAE, F6 und die Hochauftriebskonfiguration auf ca. 86% begrenzt. Ob der optimale Zielfunktionswert näher an den Ergebnissen der Heuristiken oder an der Gilmore-Lawler-Schranke liegt, ist dabei unklar.

Zum Vergleich mögen hier noch die jeweils besten, nicht in den obigen Tabellen aufgelisteten Ergebnisse für den DLR-F6 dienen, welche durch den genetischen Algorithmus bei einzelnen Läufen gefunden wurden. Sie sind zusammen mit den Parametern Populationsgröße p, Generationenanzahl g und Mutationsfunktion M in Tabelle 6.11 angegeben.

n	Verbesserung	Verfahren	p	g	M
64	64,02%	GA	200	10.000	M_1
128	62,02%	GA+EEZ	200	100.000	M_3
256	60,42%	GA+EEZ	100	100.000	M_4

Tabelle 6.11: Die besten für den DLR-F6 erzielten Ergebnisse.

[2]Für die Bestimmung wurden die symmetrischen Kommunikationsmatrizen $\frac{1}{2}(A + A^t)$ verwendet.

6.2 Ergebnisse der Kommunikationsoptimierung beim DLR-TAU Code

Aufgrund der hohen Laufzeit der Verbesserungsverfahren und des genetischen Algorithmus kam für die Anwendungstests nur noch das Konstruktionsverfahren EEZ in Frage, obwohl dieses bei der Lösungsqualität etwas schlechter abschnitt. Die verbesserte Zuordnung von MPI-Prozessen zu CPU-Cores wurde dadurch erreicht, dass das MPI-Machinefile für den jeweiligen Lauf des TAU-Strömungslösers, entsprechend der durch die Heuristik EEZ bestimmten Permutation, umsortiert wurde. Durch das MPI-Machinefile wird festgelegt, welcher MPI-Prozess auf welchem CPU-Core gestartet wird. Für eine möglichst realistische Einschätzung des tatsächlichen Performancegewinns erfolgten die Tests auf zwei unterschiedlichen Rechenclustern des DLRs.

A. Rechencluster des DLR Instituts für Aerodynamik und Strömungstechnik in Göttingen, bestehend aus 60 Knoten mit jeweils 2 AMD Barcelona 1.9 GHz QuadCore Chips (480 Cores), 16 GByte RAM und einen Infiniband Single Data Rate (10 GBit/sec) Netzwerk [13].

B. Rechencluster des DLR Instituts für Antriebstechnik Köln in München, bestehend aus 200 Knoten mit jeweils 2 Intel Nehalem 2.53 GHz QuadCore Chips (1600 Cores), 24 GByte RAM und einen Infiniband Double Data Rate (20 GBit/sec) Netzwerk.

Für die Tests sind nur die beiden Flugzeugkonfigurationen DLR-F6 (Testdatensatz c) und die Hochauftriebskonfiguration (Testdatensatz d) eingesetzt worden. Die dabei erzielten Laufzeiten und Verbesserungen sind in den folgenden Diagrammen und Tabellen je nach Rechencluster und Testfall separat aufgeführt.

CPUs	16	32	40	48	64	96	128	160	192	256
Standard	702	389	318	277	219	167	134	119	116	94
Optimiert	693	381	310	265	204	152	124	109	98	85
Verbesserung in %	1,28	2,06	2,52	4,33	6,85	8,98	7,46	8,4	15,52	9,57

Tabelle 6.12: Laufzeiten in Sekunden für 500 Iterationen des DLR-F6 Falls ohne Mehrgitterbeschleunigung auf Rechencluster A.

In Tabelle 6.12 sind die Laufzeiten des TAU-Strömungslösers bei der optimierten und bei der nicht optimierten Standardzuordnung von MPI-Prozessen zu CPU-Cores für den DLR F6 Fall[3] auf dem Rechencluster A angegeben. Die prozentuale Verbesserung in der Laufzeit durch die optimierte Zuordnung der MPI-Prozesse beträgt zwischen $1,28\%$ ($n = 16$) und $15,52\%$ ($n = 192$).

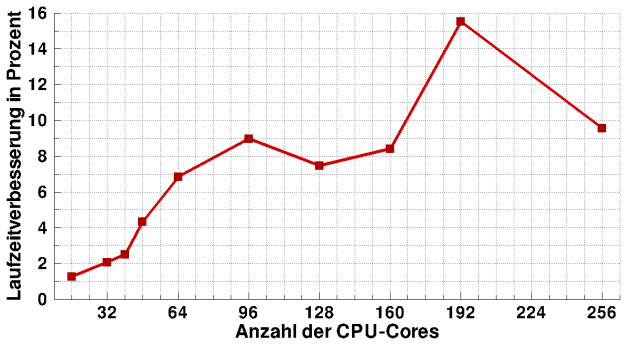
[3] Einstellungen: Runge-Kutta Verfahren, Spalart-Allmaras Turbulenzmodell

In Tabelle 6.13 sind die Laufzeiten für die Hochauftriebskonfiguration[3] angegeben. Hier liegt der Performancegewinn zwischen $0,64\%$ ($n = 32$) und $9,52\%$ ($n = 256$).

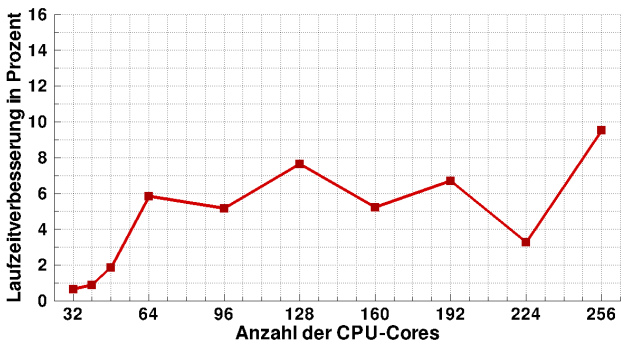
CPUs	32	40	48	64	96	128	160	192	224	256
Standard	2328	1906	1619	1282	873	695	554	493	427	399
Optimiert	2313	1889	1589	1207	828	642	525	460	413	361
Verbess. in %	0,64	0,89	1,85	5,85	5,15	7,63	5,23	6,69	3,28	9,52

Tabelle 6.13: Laufzeiten in Sekunden für 500 Iterationen der Hochauftriebskonfiguration ohne Mehrgitterbeschleunigung auf Rechencluster A.

In Abbildung 6.5 ist die durch die Optimierung der Kommunikation erzielte Laufzeitreduktion des TAU-Strömungslösers noch einmal grafisch dargestellt. Man kann gut erkennen, dass für beide Testfälle mit zunehmender CPU-Anzahl auch die Laufzeitverbesserung tendenziell ansteigt.

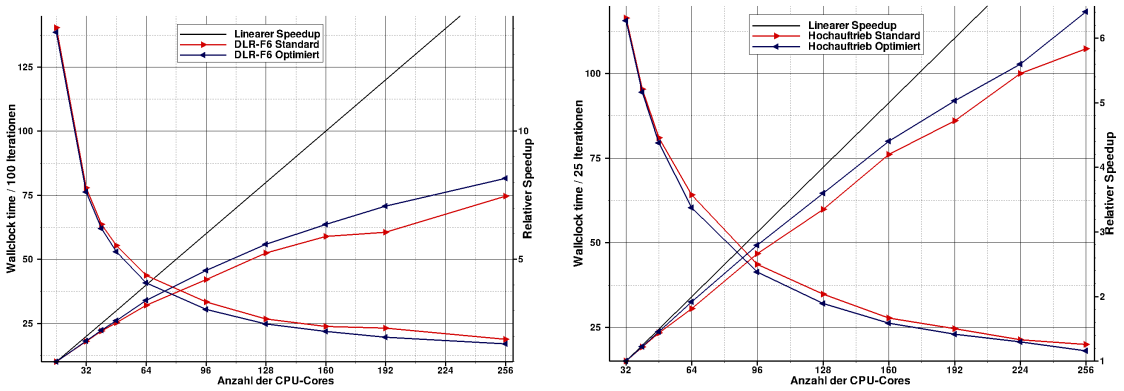


(a) DLR-F6



(b) Hochauftriebskonfiguration

Abbildung 6.5: Verbesserung der Laufzeit für den DLR-F6 (a) und die Hochauftriebskonfiguration (b) ohne Mehrgitterbeschleunigung auf Cluster A.



(a) DLR-F6 (b) Hochauftriebskonfiguration

Abbildung 6.6: Skalierung und Laufzeiten für den DLR-F6 und die Hochauftriebskonfiguration ohne Mehrgitterbeschleunigung auf Cluster A.

In Abbildung 6.6 ist für die beiden Flugzeugkonfigurationen neben der Laufzeit auch noch die Skalierung aufgeführt. Anhand der Skalierungskurve (Relativer Speedup) wird sehr schön deutlich, dass die optimierte MPI-Prozesszuordnung durch die EEZ neben der absoluten Laufzeit auch die parallele Effizienz des TAU-Strömungslösers verbessert.

Die Testläufe auf dem Rechencluster B zeigen ein ganz ähnliches Bild. Insbesondere die Hochauftriebskonfiguration kann sehr deutlich von der optimierten Zuordnung von MPI-Prozessen zu CPU-Cores profitieren (zwischen $1,9\%$ bei $n = 96$ und $28,5\%$ bei $n = 786$). Die Ergebnisse sind in Tabelle 6.14 und Abbildung 6.7 dargestellt. Für diese Konfiguration sind aufgrund technischer Probleme leider nicht alle Vergleichsläufe mit Mehrgitterbeschleunigung durchgelaufen, daher kann nur vermutet werden, dass sich auch hier bei größeren CPU-Anzahlen ($n >> 192$) eine ähnliche Laufzeitverbesserung ergibt wie ohne Mehrgitterbeschleunigung.

CPUs	MG	64	96	128	192	256	384	512	640	768	896
Standard	sg	239	163	127	89	71	54	45	38	36	32
Optimiert	sg	234	160	121	84	66	46	37	31	28	26
Standard	3v	-/-	284	229	164	-/-	-/-	-/-	-/-	-/-	-/-
Optimiert	3v	-/-	275	210	147	-/-	-/-	-/-	-/-	-/-	-/-

Tabelle 6.14: Laufzeiten für 200 Iterationen der Hochauftriebskonfiguration ohne (sg) und mit Mehrgitterbeschleunigung (3v) auf Cluster B.

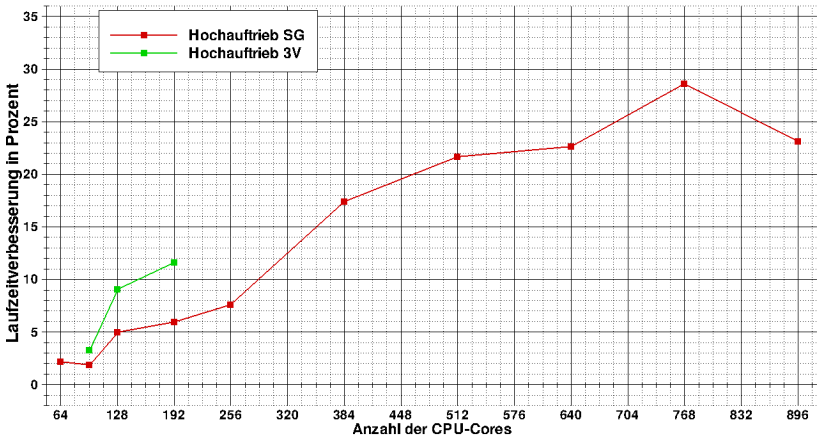


Abbildung 6.7: Laufzeitverbesserung für die Hochauftriebskonfiguration ohne (SG) und mit Mehrgitterbeschleunigung (3V) auf Cluster B.

Die Ergebnisse für den DLR-F6 auf dem Rechencluster B finden sich in Tabelle 6.15 und Abbildung 6.8. Hierfür sind insgesamt drei verschiedene Testrechnungen durchgeführt worden. Eine Testrechnung ist ohne Mehrgitterbeschleunigung (sg) gelaufen und zwei Tests sind mit einem 3v bzw. 4w Mehrgitterzyklus gerechnet worden. Die prozentuale Laufzeitverbesserung für den F6 liegt ab 128 CPU-Cores in den meisten Fällen bei über 10%. Dabei fällt auf, dass die Verbesserung in der Laufzeit beim 4w Zyklus erheblich stärker schwankt, was sich durch den erhöhten Kommunikationsaufwand der gröberen Netzlevel erklären lässt (siehe dazu auch [2] Seite 38 ff.).

CPUs	MG	32	48	64	96	128	192	256	320	384	512	768	1024
Standard	sg	76	53	41	31	24	19	15	13	12	12	9	10
Optimiert	sg	76	48	39	28	22	16	13	10	10	9	8	8
Standard	3v	127	92	72	56	45	40	36	31	32	30	30	32
Optimiert	3v	123	82	65	48	39	36	32	23	26	23	24	27
Standard	4w	144	105	87	77	67	67	69	66	69	65	72	83
Optimiert	4w	137	93	75	59	60	66	66	43	57	53	59	68

Tabelle 6.15: Ausgewählte Laufzeiten für 200 Iterationen DLR-F6 ohne (sg) und mit Mehrgitterbeschleunigung (3v, 4w) auf Cluster B.

Es ist zu beachten, dass der Testfall F6 mit seinen 2 Millionen Netzpunkten schon so klein ist, das es aus Effizienzgründen eigentlich nicht mehr sinnvoll ist, im 4w Zyklus mit mehr als 64 CPUs zu rechnen (vgl. [2] Seite 37). In Tabelle 6.15 sieht man, dass die Skalierung für die nicht optimierte Prozesszuordnung für den 4w Zyklus bei $n = 128$ endet.

Das heißt, es ist durch eine Hinzunahme von mehr CPU-Cores keinerlei nennenswerte Geschwindigkeitssteigerung mehr zu erzielen. Durch die optimierte Prozesszuordnung kann zwar die Laufzeit deutlich reduziert werden, aber die Skalierung verbessert sich dadurch nicht signifikant. Lediglich die Läufe mit $n = 320$ (43s), $n = 384$ (57s) und $n = 512$ (53s) sind noch schneller als der mit $n = 96$ (59s). Die Streuung der Ergebnisse ist hier so breit, dass man über eine Skalierungsverbesserung keine eindeutige Aussage treffen kann.

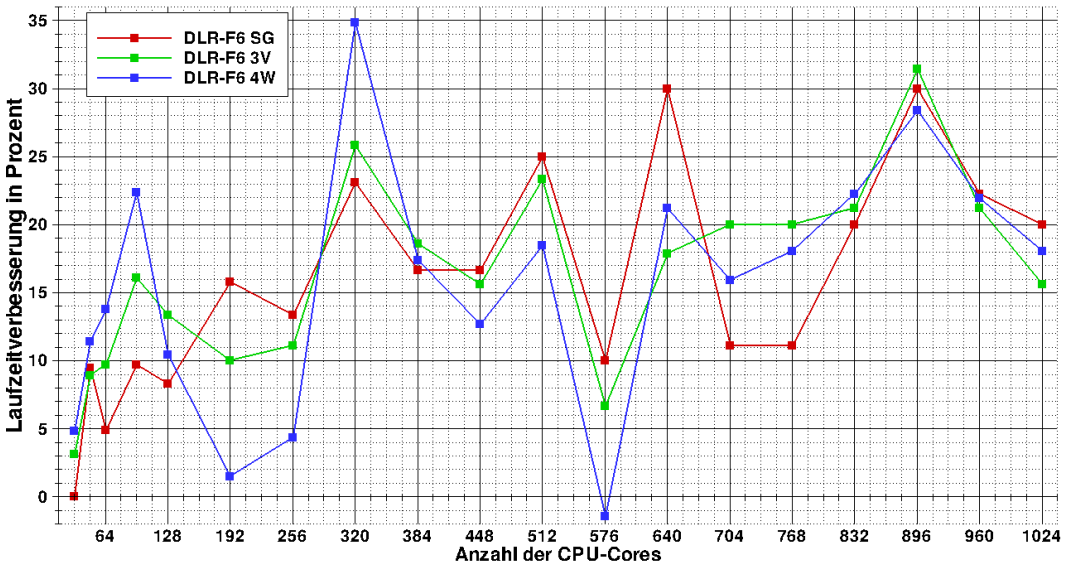


Abbildung 6.8: Laufzeitverbesserung für DLR-F6 ohne (SG) und mit Mehrgitterbeschleunigung (3V, 4W) auf Cluster B.

Für einen direkten Vergleich zwischen der Reduktion im Zielfunktionswert und der Reduktion der Laufzeit, ist in Abbildung 6.9 die durch Heuristik EEZ erzielte prozentuale Verbesserung in der Zielfunktion gegenüber der prozentualen Laufzeitverbesserung durch die Optimierung der Kommunikation beim TAU-Strömungslöser auf dem Cluster B für die Konfiguration DLR-F6 aufgetragen. Wie man sieht, sinkt die Verbesserung im Zielfunktionswert bei steigender Problemgröße. Diese Tendenz kann man bei vielen suboptimalen Lösungsverfahren für Optimierungsprobleme beobachten. Umso erfreulicher ist es, dass trotzdem mit steigender CPU-Anzahl (Problemgröße) auch die Laufzeitverbesserung tendenziell steigt. Dies lässt sich beispielsweise dadurch erklären, dass mit zunehmender Partitionsanzahl die Größe der zu kommunizierenden Nachrichten abnimmt (vgl. Abbildungen 6.2, 6.3 und 6.4) und insbesondere kleine Nachrichten von der optimierten Zuordnung profitieren (vgl. Tabelle 2.1 und Abbildung 2.1).

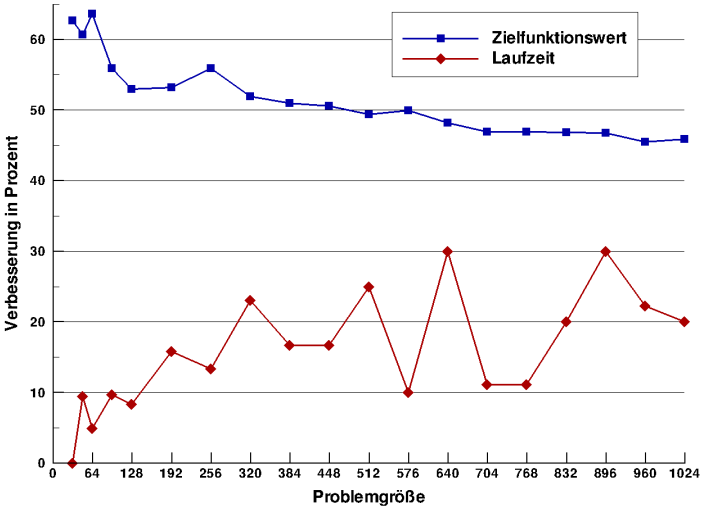


Abbildung 6.9: Vergleich der Verbesserung im Zielfunktionswert mit der Laufzeitverbesserung für den DLR-F6 Fall ohne Mehrgitterbeschleunigung auf B.

6.3 Zusammenfassung der Ergebnisse

Mit dem Konstruktionsverfahren EEZ konnte eine deutliche Verbesserung im Zielfunktionswert erreicht werden. Weiterhin zeichnet sich das Verfahren durch eine sehr kurze Laufzeit aus und eignet sich daher, im Gegensatz zu all den anderen getesteten Verfahren, am ehesten für die Aufgabe der Optimierung der MPI-Prozess-Kommunikation bei parallelen Anwendungen.

Den zum Teil deutlich besseren Ergebnissen im Zielfunktionswert bei den Verbesserungsverfahren steht insgesamt eine erheblich höhere Laufzeit gegenüber. Für die Alltagspraxis stellen diese Verfahren daher keine anwendbare Alternative zur EEZ dar.

Der genetische Algorithmus konnte bei einzelnen Läufen die besten Ergebnisse der hier vorgestellten Heuristiken liefern. Im Durchschnitt erzielte das Verfahren GA+EEZ bei Generationsanzahl 100.000 mit den Verbesserungsverfahren vergleichbare Werte, bei jedoch nicht mehr zu rechtfertigender Laufzeit. Daher ist auch der genetische Algorithmus keine Alternative zur EEZ, wenn man die praktische Anwendbarkeit betrachtet.

Als Endergebnis der Anwendungstests kann man festhalten, dass sich das Konstruktionsverfahren EEZ für die Aufgabe der Optimierung der MPI-Prozess-Kommunikation beim DLR-TAU Code bewährt hat. Es konnte mit zwei unterschiedlichen Flugzeugkonfigurationen und auf zwei unterschiedlichen Rechenclustern zum Teil eine deutliche Verbesserung im Laufzeitverhalten des TAU-Strömungslösers nachgewiesen werden.

6.4 Ausblick

Das generische Verfahren zur Optimierung der MPI-Prozess-Kommunikation, bei dem die verbesserte Prozesszuordnung durch eine Umordnung des Machinefiles, d. h. ohne Eingriff in den Sourcecode der Anwendung erfolgt, eignet sich auch sehr gut für andere parallele Applikationen aus dem Bereich der numerischen Strömungssimulation, sofern das Kommunikationsmuster der Anwendung extrahiert werden kann.

Bei der T-Systems Solutions for Research GmbH erfolgten mit der EEZ bereits erste Tests für die CFD Tool-Box OpenFOAM [32]. Weitere Tests sollen in Zusammenarbeit mit der Firma Cray auf dem 3D-Torus-Netzwerk der XT4 [43] erfolgen.

Die Heuristik EEZ wurde inzwischen auch direkt in den Partitionierer des DLR TAU-Codes implementiert, so dass die verbesserte Prozesszuordnung gleich im Anschluss an die Partitionierung des Rechennetzes berechnet und durchgeführt werden kann. Für die Aufstellung der Distanzmatrix muss hierfür vom Nutzer die Anzahl der CPU-Cores der verwendeten Rechenknoten im TAU-Parameterfile angegeben werden. Beim sequentiellen Einsatz des Partitionierers werden die einzelnen Netzpartitionen dann entsprechend der durch die EEZ berechneten Permutation so umsortiert, dass eine nachträgliche Umordnung des Machinefiles nicht mehr notwendig ist. Sind die einzelnen Netzpartitionen im Anschluss an eine parallel ausgeführte Partitionierung schon jeweils einem Prozess bzw. CPU-Core zugeordnet, wird die verbesserte Neuzuordnung dadurch erreicht, dass die Teilnetze über das Netzwerk zu dem jeweils für sie bestimmten Prozess bzw. CPU-Core verschickt werden.

Es ist weiterhin angedacht, auch die Heuristik $\mathrm{GPE}_{\mathrm{PBS}}$ in den TAU-Code zu integrieren, insbesondere für den Einsatz bei sehr lange dauernden Simulationsrechnungen (z. B. instationären DES-Simulationen [38]). Hierbei würde die Laufzeit der Heuristik selbst kaum noch ins Gewicht fallen, während sich jedes weitere Prozent, das sich in der Laufzeit des Strömungslösers einsparen ließe, durchaus deutlich bemerkbar machen könnte.

Literaturverzeichnis

[1] T. Alrutz. Investigation of the parallel performance of the unstructured DLR-TAU-Code on distributed computing systems. In A. Deane et al., editor, *Parallel Computational Fluid Dynamics*, pages 509–516, Proceedings of the 16th Parallel CFD Conference, College Park, MD, USA, May 24-27 2005. Elsevier.

[2] T. Alrutz. *Parallele dynamische Adaption hybrider Netze für effizientes verteiltes Rechnen*. Dissertation, Universität Göttingen, 2008.

[3] T. Alrutz, P. Aumann, A. Basermann, K. Feldhoff, T. Gerhold, J. Hunger, J. Jägersküpper, H.-P. Kersken, O. Knobloch, N. Kroll, O. Krzikalla, E. Kügeler, R. Müller-Pfefferkorn, M. Pütz, A. Schreiber, C. Simmendinger, C. Voigt, and C. Zscherp. HICFD - Hocheffiziente Implementierung von CFD-Codes für HPC-Many-Core-Architekturen, 2009. PARS-Mitteilungen, 22. PARS-Workshop, 04.-05. Juni 2009, Parsberg in der Oberpfalz, Deutschland. ISSN 0177-0454.

[4] K. M. Anstreicher and N. W. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89(3):341–357, 2001.

[5] R. E. Burkard, E. Çela, P. M. Pardalos, and L. S. Pitsoulis. Linear assignment problems and extensions. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume Supplement A, pages 75–149. Kluwer Academic Publishers, 1999.

[6] E. Çela. *The Quadratic Assignment Problem: Theory and Algorithms*, volume 1 of *Combinatorial Optimization*. Kluwer Academic Publishers, Dordrecht, 1998.

[7] CentaurSoft. CENTAUR Grid Generator. `http://www.centaursoft.com/cms/index.php?section=799`.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[9] C. S. Edwards. The derivation of a greedy approximator for the Koopmans-Beckmann quadratic assignment problem. In *Proceedings of CP77 Combinatorial Programming Conference*, pages 55–86, 1977.

[10] G. Finke, R. E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31:61–82, 1987.

[11] National Center for Computational Science. World's Most Powerful Computer For Science!, 2009. `http://www.nccs.gov/jaguar`.

[12] DLR Institut für Aerodynamik und Strömungstechnik. HPC-Cluster im Institut für Aerodynamik und Strömungstechnik, 2005. `http://www.dlr.de/as/desktopdefault.aspx/tabid-127/1082_read-5094/`.

[13] DLR Institut für Aerodynamik und Strömungstechnik. Rechner-Cluster Einweihung im DLR Göttingen; *72 Rechenknoten mit je 2.2 GHz Dualcore AMD Opteron*, 2006. `http://www.dlr.de/desktopdefault.aspx/tabid-2666/4016_read-5996/`.

[14] DLR Institut für Aerodynamik und Strömungstechnik. $C^2A^2S^2E$ - Hochleistungsrechner für die Luftfahrtforschung beim DLR in Betrieb genommen; *768 Rechenknoten mit je 1.9 GHz Quadcore AMD Opteron*, Mai, 2008. `http://www.dlr.de/as/desktopdefault.aspx/tabid-127/1082_read-12509/`.

[15] T. Gerhold, J. Evans, and M. Galle. Technical Documentation of the DLR TAU-Code. IB 223-97 A 43, DLR, 1997.

[16] T. Gerhold, V. Hannemann, and D. Schwamborn. On the Validation of the DLR-TAU Code. In *AG STAB Symposium, Berlin, 10.-12. November 1998*, volume 72, pages 426 – 433. Vieweg, 1999.

[17] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, 10:305–313, 1962.

[18] Pallas GmbH. Pallas MPI Benchmarks - PMB, 2000. `http://people.cs.uchicago.edu/~hai/vm1/vcluster/PMB/PMB-MPI1.pdf`.

[19] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[20] S. W. Hadley, F. Rendl, and H. Wolkowicz. A new lower bound via projection for the quadratic assignment problem. *Mathematics of Operations Research*, 17(3):727–739, 1992.

[21] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, Cambridge, 1952.

[22] C. H. Heider. A Computationally Simplified Pair-Exchange Algorithm for the Quadratic Assignment Problem. Professional Paper 101, Center for Naval Analysis, Arlington (Va), 1972.

[23] Voltaire Infiniband, 2006. `http://www.voltaire.com/Products/InfiniBand`.

[24] A. Jameson. Time-dependent calculations using multi-grid, with applications to unsteady flows past airfoils and wings. Paper 1596, AIAA, 1991.

[25] V. Kaibel. Ein Weltrekord zum Geburtstag. DMV-Mitteilungen IV/01, 16–18, 2001.

[26] Scientific Supercomputing Center (SSC) Karlsruhe. HP XC4000, 750 Rechenknoten mit je 2 AMD Opteron Dual Core 2.6GHz CPUs, 2006. `http://www.rz.uni-karlsruhe.de/ssck/hpxc4000.php`.

[27] J. Klenner, K. Becker, M. Cross, and N. Kroll. Future Simulation Concept, September 10.-13. 2007. CEAS-2007-105, 1. CEAS European Air and Space Conference, Berlin, Germany.

[28] T. Knopp. Validation of the Turbulence Models in the DLR TAU Code for Transonic Flows - A Best Practice Guide, 2006.

[29] T. C. Koopmans and M. J. Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25:53–76, 1957.

[30] N. Kroll and J. K. Fassbender, editors. *MEGAFLOW — Numerical Flow Simulation for Aircraft Design Results of the second phase of the German CFD initiative MEGAFLOW presented during its closing symposium at DLR, Braunschweig, Germany, December 10th and 11th 2002*, volume 89 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*. Springer Verlag, 2005.

[31] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[32] OpenCFD Ltd. The open source CFD toolbox OpenFOAM. `http://www.openfoam.com/`.

[33] D. J. Mavriplis. A three-dimensional mulitgrid reynolds-averaged navier-stokes solver for unstructured meshes. *AIAA*, 33(3):445–453, 1995.

[34] Message Passing Interface (MPI), 2007. `http://www.mcs.anl.gov/mpi`.

[35] H. Müller-Merbach. *Optimale Reihenfolgen*. Springer, Berlin, 1970.

[36] Heise Online. AMD Neue Serverprozessoren kommen früher, 22. April 2009. `http://www.heise.de/newsticker/meldung/AMD-Neue-Serverprozessoren-kommen-frueher-214860.html`.

[37] Heise Online. AMDs 12-Kerner legen los, 29. März 2010. `http://www.heise.de/newsticker/meldung/AMDs-12-Kerner-legen-los-966187.html`.

[38] S. Reuß and D. Schwamborn. Massively Parallel Computation of a Full Aircraft Configuration with Delayed Detached Eddy Simulation. HPCN-Workshop, T-Systems Solutions for Research GmbH, 4.–5. Mai, Braunschweig, 2010.

[39] S. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

[40] K. Sastry, D.Goldberg, and G.Kendall. Genetic algorithms. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 4, pages 97–152. Springer, 2005.

[41] D. Schwamborn, T. Gerhold, and R. Heinrich. The DLR TAU-Code: Recent Applications in Research and Industry. In *Proc. of Europ. Conf. on Computational Fluid Dynamics ECCOMAS CFD*, 2006.

[42] Texas Adavanced Computing Center (TACC). Ranger - 3936 Rechenknoten mit je 4 AMD Opteron Quad Core 2.3GHz CPUs, 2008. `http://www.tacc.utexas.edu/resources/hpcsystems/`.

[43] HECToR: UK National Supercomputing Service. `http://www.hector.ac.uk/`.

[44] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. In *Contributions to the Theory of Games, Vol. 2*, volume 28 of *Annals of Mathematics Studies*, pages 5–12. Princeton University Press, 1953.

[45] I. Wegener. *Theoretische Informatik: Eine algorithmenorientierte Einführung.* Vieweg+Teubner Verlag, 2005.

[46] S. Yoon and A. Jameson. An LU-SSOR scheme for the Euler and Navier-Stokes equations. *AIAA Journal*, (26):1025–1026, 1988.

Danksagung

An dieser Stelle möchte ich mich bei all denen bedanken, die mich während meines Studiums und bei der Entstehung dieser Arbeit fachlich und persönlich unterstützt haben.

Ich bedanke mich ganz herzlich bei Frau Prof. Dr. Anita Schöbel und Herrn Prof. Dr. Gert Lube für die motivierende Betreuung während der Anfertigung meiner Diplomarbeit.

Mein ganz besonderer Dank gilt Herrn Dr. Thomas Alrutz von der Firma T-Systems Solutions for Research GmbH, der mir immer mit Rat und Tat zur Seite gestanden hat und mit dem ich viele hilfreiche Diskussionen speziell zum Thema wissenschaftliches Rechnen führen konnte.

Desweiteren möchte ich mich auch ganz herzlich bei Herrn Dr. Thomas Gerhold vom DLR-Institut für Aerodynamik und Strömungstechnick in Göttingen bedanken, insbesondere für die Hilfe bei der Einarbeitung in den DLR-TAU Code.

Diese Arbeit ist im Rahmen eines Kooperationsprojektes der T-Systems Solutions for Research GmbH und dem Deutschen Zentrum für Luft- und Raumfahrt entstanden. Für die mir gewährte finanzielle Untestützung bin ich beiden Institutionen sehr dankbar.

Meiner Familie und meinen Freunden, die immer für mich da waren, möchte ich ebenfalls für ihre Unterstützung danken. Für unermüdliches Korrekturlesen und Hilfe beim Seitenlayout bin ich insbesondere meiner Mutter sehr dankbar.