RWTH Aachen

Lehrstuhl für Operations Research

— Prof. Dr. Marco E. Lübbecke —

Masterarbeit

Sommersemester 2012

Christian Kind

Permutieren einer Matrix in Blockdiagonalform mittels Graph-Partitionierung

Inhaltsverzeichnis

Eidesstattliche Erklärung

Eidesstattliche Erklärung
zur Masterarbeit

Name: _____ Vorname: _____

Ich versichere, dass ich die Masterarbeit selbstständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst habe.

Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Aachen, den _____
(Unterschrift)

Einleitung

In der Linearen Optimierung haben wir es immer wieder mit Gleichungen der Form $Ax = b$ zu tun. Dabei ist es nicht ungewöhnlich, dass die Matrix A mehrere tausend Zeilen und Spalten hat, und die Berechnung der Gleichungen dadurch sehr lange dauert. Aus diesem Grund ist ein großes Interesse daran entstanden, bestimmte Eigenschaften der Gleichungen auszunutzen, um eine schnellere Berechnung zu ermöglichen.

Grundsätzlich ist es von Vorteil, wenn viele Einträge der Matrix A den Wert 0 haben, da wir diese bei der Berechnung der Gleichungen ignorieren können. Wenn die Matrix A die Form

$$A = \begin{pmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_K \\ R_1 & R_2 & \cdots & R_K \end{pmatrix}$$

hat, können wir die Gleichungen $Ax = b$ umformulieren zu

$$\begin{aligned} B_k x_k &= b_k\,, \qquad k = 1, \ldots, K \\ \sum_{k=1}^{K} R_k x_k &= b_c\,, \end{aligned}$$

wobei x_k bzw. b_k die Einträge des Vektors x bzw. b für den $k-$ten Block bezeichnen. Da die Gleichungen $B_k x_k = b_k$ des $k-$ten Blocks unabhängig von den Gleichungen der anderen $K - 1$ Blöcken sind, können wir die Berechnung dieser Gleichungen parallel auf K Prozessoren durchführen. Wenn wir einen großen Teil der Gleichungen $Ax = b$ parallel berechnen können, d.h., wenn die Teilmatrix $(R_1, R_2, \ldots, R_K)$ wenige Zeilen hat und die Anzahl der Blöcke groß ist, können wir die Berechnung erheblich beschleunigen.

In dieser Arbeit befassen wir uns mit der Frage, wie wir die Matrix A in eine solche Form permutieren können, um dadurch die Berechnung von $Ax = b$ zu parallelisieren. Wir werden sehen, dass wir dieses Problem mit Hilfe von Graph-Partitionierung lösen können, und dass wir die Matrix dazu am besten als Hypergraph darstellen.

Darauf aufbauend werden wir untere Schranken für die Anzahl der Zeilen der Teilmatrix $(R_1, R_2, \ldots, R_K)$ herleiten. Diese Schranken hängen von verschiedenen Eigenschaften der Matrix sowie der Anzahl der Blöcke ab, und ergeben sich aus Abschätzungen für den Hypergraph der Matrix. Zur Herleitung der unteren Schranken werden wir verschiedene Ansätze verfolgen und anhand von Beispielen vergleichen. Zudem werden wir uns fragen, wie viele Blöcke maximal möglich sind bzw. inwiefern eine zu große Anzahl von Blöcken kontraproduktiv sein kann.

1 Dünnbesetzte Matrizen in Linearen Programmen

In der Linearen Programmierung haben wir es oft mit dünnbesetzten Matrizen zu tun, d.h. mit Matrizen, in denen nur wenige Einträge nicht Null sind. Wir wollen diese Einträge so anordnen, dass wir die Struktur der Matrix ausnutzen können. Das Umordnen der Zeilen und Spalten der Matrix erreichen wir durch Permutation.

1.1 Dünnbesetzte Matrizen

Beim Rechnen mit einer Matrix ist es oft von Interesse, ob und wie *dünnbesetzt* (engl.: *sparse*) die Matrix ist. Eine genaue Definition von Dünnbesetztheit gibt es aber nicht. Eine von mehreren bekannten Definitionen besagt, dass die Matrix $O(\min\{m, n\})$ Einträge haben muss, eine andere lautet, dass die Anzahl der Nicht-Null-Einträge mit Ordnung kleiner als n^2 wächst. Der Mathematiker J.H. Williamson gab eine ziemlich unförmliche, aber zugleich sehr passende Definition für die Dünnbesetztheit einer Matrix. In seinen Augen ist eine Matrix dünnbesetzt, wenn es sich lohnt, die Existenz von Nicht-Null-Einträgen auszunutzen [78]. Diese Aussage ist zum einen natürlich sehr unpräzise, da nicht klar ist, wann genau es sich lohnt, die Anzahl der Null-Einträge der Matrix auszunutzen. Auf der anderen Seite beschreibt diese Definition aber genau den Grund für das Interesse an dünnbesetzten Matrizen, da sich viele Probleme vereinfachen lassen, wenn wir die dünnbesetzte Struktur einer Matrix beachten. Das Gegenteil einer dünnbesetzten Matrix ist eine *vollbesetzte* Matrix, die fast nur Einträge ungleich Null hat.

Dünnbesetzte Matrizen sind vor allem mit Blick auf die Rechenzeit und den Speicheraufwand interessant. Da eine dünnbesetzte Matrix nur wenige Einträge ungleich Null enthält, ist es vorteilhaft, wenn wir statt der gesamten Matrix nur die Einträge ungleich Null abspeichern. Es gibt verschiedene Methoden, um dünnbesetzte Matrizen abzuspeichern, die hier aber nicht weiter diskutiert werden sollen.

Auch die Berechnung der Gleichungen $Ax = b$ lässt sich beschleunigen, wenn A dünnbesetzt ist. Sei

$$\sum_{j=1}^{n} a_{ij} x_j = b_i$$

die i-te Gleichung von $Ax = b$. Wenn A dünnbesetzt ist, gibt es wahrscheinlich viele Einträge $a_{ij} = 0$, sodass wir eigentlich nur

$$\sum_{j:a_{ij}\neq 0} a_{ij} x_j = b_i$$

berechnen müssen, wodurch viel weniger Operationen nötig sind. Im Bereich der Optimierung ist diese Beobachtung natürlich besonders interessant, da wir es hier ständig mit Gleichungen der Form $Ax = b$ zu tun haben.

1.2 Arrowhead-Form

Im Laufe dieser Arbeit sind wir nicht nur an der Tatsache interessiert, dass eine Matrix A dünnbesetzt ist, sondern wir wollen darüber hinaus wissen, ob sich die Matrix in eine besonders „schöne“ Form bringen lässt. Mit „besonders schön“ meinen wir im Folgenden die *Arrowhead-Form*, bei der die Matrix A die Gestalt

$$A = \begin{pmatrix} B_1 & & & & C_1 \\ & B_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & B_K & C_K \\ R_1 & R_2 & \cdots & R_K & D \end{pmatrix}$$

hat, mit $B_k \in \mathbb{R}^{m_k \times n_k}$, $C_k \in \mathbb{R}^{m_k \times n_c}$, $R_k \in \mathbb{R}^{m_r \times n_k}$, $D \in \mathbb{R}^{m_r \times n_c}$ und $\sum_{k=1}^{K} m_k + m_r = m$, $\sum_{k=1}^{K} n_k + n_c = n$. Die Struktur der Matrix ähnelt einem Pfeil, daher die Bezeichnung „Arrowhead“-Form, sie ist auch als *Doubly-Bordered Block Diagonal Form (dt.: Blockdiagonalform mit Doppelrand)* bekannt. Jede Teilmatrix B_k, $k = 1, \ldots, K$, bezeichnen wir als *Block*, jede Zeile der $m_r \times n$-Teilmatrix $(R_1, \ldots, R_K, D)$ nennen wir *Coupling Constraint* (dt.: *koppelnde Nebenbedingung*), und jede Spalte der $m \times n_c$-Teilmatrix $(C_1, \ldots, C_K, D)^T$ heißt *Linking Column* (dt.: *verbindende Spalte*).

Wir wollen zulassen, dass die Arrowhead-Form der Matrix A keine Linking Columns bzw. keine Coupling Constraints hat, d.h. $m_r = 0$ bzw. $n_c = 0$. In diesem Fall sprechen wir von einer *Singly-Bordered Block Diagonal Form* (dt.: *Blockdiagonalform mit Rand*) oder kurz *SB-Form*. Wenn die Linking Columns fehlen, werden wir eine solche Gestalt hier als *SB-Form mit Zeilenrand* bezeichnen, wenn die Coupling Constraints fehlen, werden wir von einer *SB-Form mit Spaltenrand* sprechen. Eine Struktur ganz ohne Rand, d.h. mit $m_r = n_c = 0$, bezeichnen wir als *strikte Blockdiagonalform*.

In manchen Fällen werden wir verallgemeinernd davon sprechen, dass eine Matrix in *Blockdiagonalform* ist, wenn sie in Arrowhead-, SB- oder strikter Blockdiagonalform ist. Unabhängig davon, ob die Arrowhead-Form einen Rand hat, werden wir sagen, dass wir die Matrix *partitionieren* wollen, wenn es darum geht, die Matrix in Arrowhead-Form zu bringen.

Wir können die Linking Columns leicht entfernen, wodurch sich allerdings die Anzahl der Coupling Constraints erhöht. Dazu formen wir die Spalten

$$(C_1, \ldots, C_K, D)^T$$

der Linking Columns durch *Column Splitting* (dt.: *Spaltenaufteilung*) um, sodass wir jede Spalte einem bestimmten Block zuordnen können, und so den Spaltenrand auflösen können.

Wir betrachten dazu die j-te Spalte des Spaltenrands. Diese Spalte entspricht den Koeffizienten einer Variablen x_j, d.h., jeder Eintrag ungleich Null in dieser Spalte impliziert, dass wir die Variable x_j für die entsprechende Nebenbedingung benötigen. Insbesondere benötigen wir x_j für mindestens zwei Nebenbedingungen, die zu verschiedenen Blöcken gehören, denn sonst hätten wir diese Spalte in einen der Blöcke permutieren können.

Für jeden Block, in dem die $j-$te Variable benötigt wird, legen wir eine Kopie der Variable x_j an. Zudem stellen wir die j-te Spalte als Summe von Spaltenvektoren

dar, sodass jeder dieser Spaltenvektoren nur die Einträge eines Blocks enthält. Da die Nicht-Null-Einträge dieser Spalten nur einen einzigen Block betreffen, können wir diese einzeln aus dem Rand heraus permutieren. Wenn wir dies für alle Spalten im Spaltenrand tun, können wir dadurch den Spaltenrand auflösen.

Da jede neue Variable x_j^k eigentlich einer ursprünglichen Variable x_j entspricht, müssen wir dafür sorgen, dass alle Kopien den selben Wert haben wie die originale Variable. Hierzu müssen wir zusätzliche Nebenbedingungen aufstellen, die dann etwa die Form $x_j^k - x_j^l = 0$ haben, wodurch wir sicher stellen, dass x_j^k und x_j^l den gleichen Wert haben. Wenn eine Linking Column k Blöcke verbindet, benötigen wir insgesamt $k - 1$ solcher Nebenbedingungen.

Weil sich die beiden Variablen einer neuen Nebenbedingung auf verschiedene Blöcke beziehen, verbinden wir durch die neue Nebenbedingung zwei Blöcke, weshalb diese Nebenbedingung in den Zeilenrand kommt. Insgesamt vergrößert sich der Zeilenrand um höchstens $(K - 1)n_c$ Nebenbedingungen, und wenn der Spaltenrand dünnbesetzt ist, sind es viel weniger. Diese Technik wird übrigens auch in der Stochastischen Programmierung verwendet, wenn man Antizipation unterbinden möchte [60].

Wir wollen diese Methode im Folgenden an einem Beispiel illustrieren. Die Matrix

$$\begin{pmatrix}
\times & \times & \times & & & & \times & \times \\
\times & \times & \times & & & & & \times \\
& & & \times & \times & & \times & \\
& & & & \times & \times & \times & \\
& & & \times & \times & \times & & \times \\
\times & & \times & & \times & \times & & \\
& \times & & \times & \times & & &
\end{pmatrix}$$

ist in Arrowhead-Form und soll in SB-Form gebracht werden, die $\times$ stehen dabei für Nicht-Null-Einträge. Der Spaltenrand besteht aus zwei Spalten, und die zugehörigen Variablen x_7 und x_8 werden sowohl für die Nebenbedingungen des ersten Blocks als auch für die Nebenbedingungen des zweiten Blocks benötigt. Anstelle der ersten Spalte im Spaltenrand legen wir zwei neue Spalten an, die in der Summe die erste Spalte ergeben. Das gleiche tun wir für die zweite Spalte, sodass wir die Linking Columns durch vier neue Spalten ersetzt haben. Hinzu kommen die Nebenbedingungen $x_7' - x_7'' = 0$ und $x_8' - x_8'' = 0$, die wir neu hinzufügen müssen. Damit hat die Matrix nun die Form

$$\begin{pmatrix}
\times & \times & \times & & & & \times & & \times & \\
\times & \times & \times & & & & & & \times & \\
& & & \times & \times & & & \times & & \\
& & & & \times & \times & & \times & & \\
& & & \times & \times & \times & & & & \times \\
\times & & \times & & \times & \times & & & & \\
& \times & & \times & \times & & & & & \\
& & & & & & 1 & -1 & & \\
& & & & & & & & 1 & -1
\end{pmatrix}.$$

Nun können wir die siebte und neunte Spalte in den ersten Block permutieren und

erhalten die Matrix

$$\begin{pmatrix}
\times & \times & \times & \times & \times & & & & & & \\
\times & \times & \times & & \times & & & & & & \\
& & & & & \times & \times & & \times & & \\
& & & & & & \times & \times & \times & \times & \\
& & & & & \times & \times & \times & & & \\
\times & & \times & & & & \times & \times & & & \\
& \times & & & & \times & \times & & & & \\
& & & 1 & & & & & -1 & & \\
& & & & 1 & & & & & -1 &
\end{pmatrix},$$

die nun in SB-Form ist, allerdings auch zwei Nebenbedingungen mehr als die ursprüngliche Matrix hat.

1.3 Parallele Berechnung

Eine Matrix in Arrowhead-Form bzw. in SB-Form ist natürlich dünnbesetzt, zumindest wenn wir davon ausgehen, dass die Linking Columns und Coupling Constraints nicht zu groß sind. Neben den generellen Vorteilen bei der Speicherung und Berechnung von dünnbesetzten Gleichungssystemen können wir die SB-Form auch noch auf eine andere Art und Weise ausnutzen.

Viele lineare Programme haben Lösungszeiten der Ordnung $O(n^2)$ oder $O(n^3)$, weshalb es effizienter ist, statt einem einzelnen großen Problem mehrere kleine Teilprobleme zu lösen. Darüber hinaus bietet die SB-Form die Möglichkeit der parallelen Berechnung. Das bedeutet, dass wir das lineare Programm nicht nur in mehrere kleinere Teilprobleme zerlegen können, sondern diese Teilprobleme auch noch auf mehreren Prozessoren gleichzeitig lösen können. Dies ist vor allem für sehr große Probleme interessant, bei denen die Berechnung mit einem einzigen Prozessor zu teuer oder sogar unmöglich ist.

Wenn die Matrix A in SB-Form ist, können wir die Gleichungen $Ax = b$ schreiben als

$$\begin{aligned}
B_k x_k &= b_k\,, \qquad k = 1, \ldots, K \\
\sum_{k=1}^{K} R_k x_k &= b_c\,,
\end{aligned}$$

wobei x_k bzw. b_k die Einträge des Vektors x bzw. b für den $k-$ten Block bezeichnen. Bei der Parallelisierung speichern wir die Nebenbedingungen und die Werte der Variablen des k-ten Blocks auf dem k-ten Prozessor und können diese unabhängig von den anderen Blöcken und Prozessoren berechnen. Die Coupling Constraints speichern wir ebenfalls auf einem der K Prozessoren statt auf einem weiteren Prozessor, da so immerhin schon ein Teil der benötigten Variablen auf dem Prozessor gespeichert ist. Leider ist eine vollständige Parallelisierung oft nicht möglich, denn während wir die Linking Columns wie erwähnt entfernen können, müssen wir oft eine gewisse Anzahl von Coupling Constraints in Kauf nehmen, sodass eine strikte Blockdiagonalform oft unerreichbar ist.

Für die Coupling Constraints benötigen wir die Werte von Variablen aus mehreren Blöcken, die sich dann auch auf mehreren Prozessoren befinden. Die Folge ist, dass

Kommunikation zwischen mehreren Prozessoren nötig ist, da der Prozessor mit den Coupling Constraints die fehlenden Werte der benötigten Variablen von den anderen Prozessoren erhalten muss. Diese Kommunikation bedeutet einen höheren Aufwand, sowohl zeitlich als auch physisch. Da es zudem allgemein am besten ist, so viele Nebenbedingungen wie möglich parallel zu berechnen, sollte unser Ziel also sein, die Kommunikation zwischen den einzelnen Prozessoren zu minimieren, d.h. die Kommunikation zwischen den einzelnen Blöcken und den Coupling Constraints minimal zu halten. Darüber hinaus sollten alle Prozessoren in etwa gleich ausgelastet sein, da die gesamte Rechenzeit natürlich in erster Linie durch den Prozessor bestimmt wird, der am längsten für die Berechnung braucht.

1.4 Permutation einer Matrix

Unser Ziel ist also, eine Matrix in eine SB-Form zu bringen, durch die wir dann eine parallele Berechnung erreichen können. In vielen linearen Programmen der Form

$$\begin{aligned} \min_x \quad & < c, x > \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

hat die Matrix A bereits annähernd SB-Form oder zumindest Arrowhead-Form, wie etwa die Matrix des Patient Distribution System-Problems [10] in Abbildung 1.1. Allerdings ist dies bei weitem nicht immer der Fall, und selbst wenn die Matrix schon so etwas ähnliches wie eine Arrowhead-Form hat, können wir daraus nicht unbedingt schnell auf eine tatsächliche Arrowhead-Form schließen.

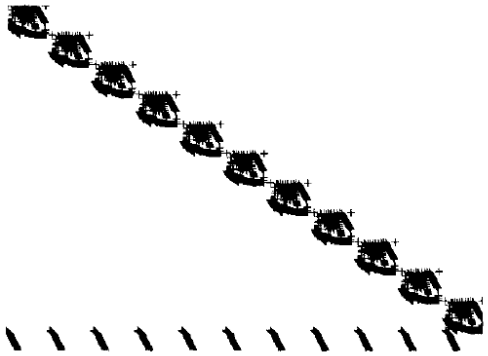

Abbildung 1.1: Matrix des Patient Distribution System-Problems

Wollen wir eine Matrix $A \in \mathbb{R}^{m \times n}$ permutieren, geschieht dies durch Multiplikation der Matrix A von links mit einer Matrix $P \in \{0,1\}^{m \times m}$ zur Vertauschung der Zeilen bzw. von rechts mit einer Matrix $Q \in \{0,1\}^{n \times n}$ zur Vertauschung der Spalten. In den Zeilen und Spalten der Matrizen P und Q ist jeweils ein Eintrag 1 und die anderen haben den Wert 0. Je nachdem wie die 1-Einträge in den Matrizen P und Q verteilt sind, erreichen wir durch die Multiplikationen PA bzw. AQ eine Vertauschung der Zeilen bzw. der Spalten in A. Eine 1 an der Stelle p_{ij} in der Matrix P hat zur Folge, dass die j-te Zeile der Matrix A in die i-te Zeile rutscht. Bei der Matrix Q bedeutet $q_{ij} = 1$, dass die i-te Spalte zur j-ten Spalte wird.

Durch die Permutation ändern wir nur die Reihenfolge der Einträge in den Zeilen und Spalten der Matrix A, es stehen allerdings weiterhin die selben Einträge in einer Zeile bzw. in einer Spalte wie zuvor. Es kann also nicht passieren, dass zwei Einträge aus der i-ten Zeile der Matrix A nach der Permutation in zwei verschiedenen Zeilen stehen, gleiches gilt für zwei Einträge aus der selben Spalte. Das ist sehr wichtig, damit die Nebenbedingungen auch nach der Permutation noch die gleiche Bedeutung haben. Spezialfälle für Permutationsmatrizen sind die Identitätsmatrix

$$Id = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix},$$

die keinerlei Veränderung bewirkt, und die Matrix

$$P = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \iddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{pmatrix},$$

die die Matrix durch Links-Multiplikation horizontal und durch Rechts-Multiplikation vertikal spiegelt.

Wir können die Matrix A also mit den Matrizen P und Q in die Matrix $A^{\Pi} = PAQ$ permutieren und das lineare Programm von oben umformulieren zu

$$\begin{aligned} \min_{\bar{x}} \quad & < Q^T c, \bar{x} > \\ s.t. \quad & A^{\pi}\bar{x} = Pb \\ & \bar{x} \geq 0\,. \end{aligned}$$

Da A^{π} eine Permutation der Matrix A ist, müssen wir auch die Vektoren x und b entsprechend umordnen, damit wir weiterhin die gleichen Nebenbedingungen haben. Die Lösung des ursprünglichen Problems ist dann $x^* = Q\bar{x}^*$.

1.5 Geschichtliche Hintergründe

Zum Abschluss des ersten Kapitels wollen wir einen kurzen geschichtlichen Überblick zur Entwicklung der Linearen Optimierung, dem Interesse an Matrizen in Blockdiagonalform und der Partitionierung von Matrizen geben.

Als „Vater der Linearen Optimierung“ gilt George B. Dantzig (1914-2005, im Bild). Auch wenn es bereits vorher Ansätze gab, die der Linearen Optimierung zugeschrieben werden können, war es Dantzig, der im Jahre 1947 mit der Entwicklung des Simplex-Verfahrens [16] den ersten Meilenstein in der Geschichte der Linearen Optimierung setzte. Schon bald beschäftigten sich Wissenschaftler mit diesem Verfahren, sodass keine zehn Jahre später schon zwei bedeutende Erweiterungen des Simplex-Verfahrens entstanden waren, das revidierte Simplex-Verfahren [16] und das duale Simplex-Verfahren [56]. Aber auch die Implementierung des Simplex-

Verfahrens wurde verbessert, als bedeutende Verbesserung gilt etwa die Verwendung der LU-Faktorisierung mit dem Markowitz-Kriterium [72].

Das Simplex-Verfahren hat zwar eine exponentielle Laufzeit, gilt aber dennoch in der Praxis oft als überlegen gegenüber anderen Verfahren. Ein Vorteil des Simplex-Verfahrens besteht darin, dass eine ermittelte Lösung schnell an kleine Veränderungen der Ausgangsdaten wie der Matrix A oder des Vektors b angepasst werden kann. In der Ganzzahligen Linearen Optimierung kann etwa eine Lösung, die nach Einfügen einer Schnittebene unzulässig geworden ist, durch einen einzigen dualen Simplex-Schritt in eine zulässige Lösung verwandelt werden.

Neben dem Simplex-Verfahren gibt es auch noch andere Algorithmen zur Lösung von Optimierungsproblemen. Eine davon ist die Ellipsoid-Methode, die 1979 von Leonid G. Khachiyan auf lineare Optimierungsprobleme angewandt wurde [31]. Ein weiteres Verfahren ist die Innere-Punkt-Methode, die erstmals im Jahre 1948 von einem weiteren großen Mathematiker, John von Neumann, auf die Lineare Optimierung angewandt wurde [16]. Im Gegensatz zu den anderen beiden Verfahren funktioniert die Innere-Punkt-Methode auch für nicht-lineare Optimierungsprobleme.

Etwa zur gleichen Zeit wie der Geburtsstunde der Linearen Optimierung wurden erste Resultate zu dünnbesetzten Matrizen erzielt. Es entstanden Algorithmen wie das Gauß-Seidel-Verfahren [15], das iterativ nach einer Lösung sucht und speziell die Dünnbesetztheit einer Matrix ausnutzt. Aber auch direkte Verfahren wurden entwickelt, bei der die Matrix oft als Produkt mehrerer Matrizen dargestellt wird, um durch Rückwärts-Substitution eine Lösung des Ausgangsproblems zu finden. Beispiele hierfür sind die LU-Zerlegung, die QR-Zerlegung und die Cholesky-Zerlegung [15].

Dass man die Blockdiagonalgestalt einer Matrix ausnutzen kann, haben wohl erstmals Dantzig und Wolfe festgestellt, die darauf die bekannte Dantzig-Wolfe-Dekomposition [17] aufbauten. Dabei wird für jeden Block die Menge der Lösungen, die die Nebenbedingungen des Blocks erfüllen, umformuliert, indem jede Lösung als Konvexkombination aller Extrempunkte und Extremstrahlen dieser Menge dargestellt wird. Diese Darstellung wird dann in das Master-Problem, das aus den restlichen Nebenbedingungen besteht, eingesetzt, sodass wir eine für das originale Problem zulässige Lösung erhalten.

Die Besonderheit an diesem Verfahren ist, dass wir nicht alle Extrempunkte und Extremstrahlen betrachten, sondern iterativ neue Extrempunkte und Extremstrahlen hinzufügen, wenn dies eine Verbesserung der Lösung verspricht. Ob wir eine Verbesserung erwarten können, und welche Variablen wir dazu zum Master-Problem hinzufügen müssen, entscheidet sich durch Lösen eines Subproblems. Dieses Verfahren, bei dem nach und nach Variablen zum Problem hinzugefügt werden, ist als *Spaltengenerierung* bekannt.

Weitere Verfahren, die die Blockdiagonalgestalt ausnutzen, sind die zur Dantzig-Wolfe-Dekomposition duale Benders-Dekomposition [5] oder der L-Shaped-Algorithmus [75], der die Ideen der Dekomposition auf stochastische Optimierungsprobleme anwendet.

Verfahren wie die Dantzig-Wolfe-Dekomposition setzen voraus, dass wir das Originalproblem wie oben erklärt umformulieren können. Theoretisch muss dazu natürlich keine SB-Form vorliegen, allerdings ist es wesentlich leichter, die Teilprobleme zu identifizieren, wenn die Matrix in SB-Form ist. Deshalb wurde begonnen nach Möglichkeiten zu suchen, wie eine Matrix in SB-Form gebracht werden kann.

Mit dem Problem, eine Matrix zwecks paralleler Berechnung in eine vorteilhafte Form wie die Arrowhead- oder die SB-Form zu bringen, beschäftigen sich Wissenschaftler noch nicht lange. In der Regel wird das Problem als Graph-Partitionierungsproblem dargestellt, allerdings stand lange Zeit lediglich das Standardmodell zur Verfügung, welches nur für symmetrische Matrizen angewandt werden kann. Hendrickson erweiterte die Anwendungsmöglichkeiten durch sein bipartites Modell [39] auf nichtsymmetrische und rechteckige Matrizen, stellte aber bald fest, dass dieses Modell einige Schwächen hat [38, 40]. Aykanat, Çatalyürek, Pınar et al. korrigierten die wichtigsten Schwächen durch ihr Hypergraph-Modell [2, 11], das heutzutage als gängiges Modell zur Matrix-Partitionierung gilt.

2 Graph- und Hypergrah-Partitionierung

Um eine Matrix zu partitionieren, werden wir uns verstärkt mit der Partitionierung eines Graphen oder eines Hypergraphen auseinandersetzen. In diesem Kapitel wollen wir einige Notationen und Aussagen für das Graph-Partitionierungsproblem festhalten, die wir im Laufe der Arbeit brauchen werden. Wir unterscheiden zwischen einem gewöhnlichen Graph und einem Hypergraph, wobei wir sehen werden, dass sich das Hypergraph-Partitionierungsproblem auch als Graph-Partitionierungsproblem darstellen lässt.

2.1 Graphen

Ein *Graph* $G = (V, E)$ besteht aus Knoten V und Kanten E. Eine Kante in E werden wir mit e_{ij} oder (i, j) bezeichnen, wobei v_i und v_j die Endpunkte der Kante e_{ij} sind. Die hier betrachteten Graphen haben keine Multikanten oder Schlingen, d.h., zwischen zwei Knoten verläuft höchstens eine Kante, und es gibt keine Kante, bei der beide Endpunkte identisch sind. Zwei Knoten v_i und v_j sind *adjazent*, wenn es in G eine Kante e_{ij} gibt. Die beiden Knoten heißen dann *inzident* zur Kante e_{ij}.

2.1.1 Graph-Partitionierung

Eine Partition eines Graphen lässt sich allgemein wie folgt definieren.

Definition 2.1.1 *Gegeben sei ein Graph $G = (V, E)$, eventuell mit Knoten- und Kanten-Gewichten. Eine Menge $\{V_k \subset V : 1 \leq k \leq K\}$ heißt* **K-Partition** *von V, wenn sie*

$$\bigcup_{k=1}^{K} V_k = V, \quad V_k \cap V_l = \emptyset, \quad k \neq l$$

erfüllt. Eine Partition mit

$$w(V_k) \approx \frac{W}{K}, \quad k = 1, \ldots, K\,,$$

wobei W das Gesamtgewicht aller Knoten ist, heißt **balanciert**. *Wenn darüber hinaus*

$$\sum_{\substack{(i,j)\in E:\\ i\in V_k, j\in V_l, k\neq l}} w_{ij}$$

minimal ist, dann ist diese Partition **optimal**.

Ein wichtiger Spezialfall einer Partition ist die *Bisektion*, bei der die Knoten in zwei Teilmengen aufgeteilt werden, die in der Regel gleich groß sein müssen oder sich in der Größe nur um einen Knoten unterscheiden. Die von uns betrachteten Graphen werden meistens Knoten ohne Gewichte haben, die Kanten können dagegen schon gewichtet sein. Ohne Knotengewichte reduziert sich die Bedingung der Balanciertheit zu

$$|V_k| \approx \frac{|V|}{K}, \quad k = 1, \ldots, K\,,$$

und ohne Kantengewichte bedeutet Optimalität, dass

$$|\{(i,j) \in E : i \in V_k, j \in V_l, k \neq l\}|$$

minimal ist.

Beim Graph-Partitionierungsproblem suchen wir also nach K Teilmengen $V_1, \ldots, V_K$ von V, die paarweise disjunkt sind und in etwa das gleiche Gewicht haben, sodass das Gesamtgewicht der Kanten zwischen einzelnen Teilmengen so gering wie möglich ist. Allerdings ist nicht von vornherein klar, wann die Teilmengen „ungefähr“ gleich groß sind, d.h. wie stark die Gewichte der Teilmengen voneinander abweichen dürfen. Eine alternative Definition, bei der dieser Aspekt klarer wird, ist

$$w(V_k) \leq \frac{W}{K}(1+\epsilon), \quad k = 1, \ldots, K\,,$$

mit einem entsprechend (klein) gewählten Wert für ϵ, bzw. mit einer zusätzlichen Beschränkung nach unten

$$\frac{W}{K}(1-\epsilon_1) \leq w(V_k) \leq \frac{W}{K}(1+\epsilon_2), \quad k = 1, \ldots, K\,.$$

Für Graphen ohne Knotengewichte wird häufig festgelegt, dass sich die Größen der Teilmengen um höchstens einen Knoten unterscheiden dürfen. Vor allem bei Bisektionen mit einer ungeraden Anzahl von Knoten ist dies üblich.

Definition 2.1.2 *Eine Partition $\{V_1, \ldots, U_K\}$ eines Graphen in K Teilmengen heißt* **gleichmäßig***, wenn*

$$||V_i| - |V_j|| \leq 1 \quad \forall\, 1 \leq i, j \leq K\,.$$

Edge Separator und Vertex Separator

Das Graph-Partitionierungsproblem können wir auf zwei verschiedene Arten lösen, als *Graph-Partitionierung via Edge Separator (GPES)* oder als *Graph-Partitionierung via Vertex Separator (GPVS)*.

Definition 2.1.3 *Gegeben sei ein Graph $G = (V, E)$. Eine Teilmenge $E_S \subset E$ heißt* **Edge Separator** *einer K-Partition Π_{GPES} von G, wenn G durch Entfernen von E_S in K Zusammenhangskomponenten zerfällt. Die Kanten in E_S werden als* **externe** *oder* **geschnittene Kanten** *bezeichnet, die anderen Kanten als* **interne** *oder* **nichtgeschnittene Kanten**. *Ein Knoten, der zu mindestens einer externen Kante inzident ist, heißt* **Randknoten**. *Die* **Cutsize** *$|E_S|$ einer Partition Π ist die Anzahl der geschnittenen Kanten, und das* **Cutweight** *$w(E_S)$ von Π ist das Gewicht der Kanten in E_S. Π_{GPES} ist* **optimal***, wenn $w(E_S)$ minimal ist, d.h., wenn es keinen Edge Separator einer Partition mit geringerem Cutweight gibt.*

Auf eine ähnliche Art und Weise lässt sich ein Graph durch eine Teilmenge der Knoten separieren.

Definition 2.1.4 *Gegeben sei ein Graph $G = (V, E)$. Eine Teilmenge $V_S \subset V$ heißt* **Vertex Separator** *einer K-Partition Π_{GPVS} von G, wenn G durch Entfernen von V_S in K disjunkte Teilmengen $V_1, \ldots, V_K$ mit $\bigcup_{k=1}^{K} V_k \cup V_S = V$ und $|V_k| \approx \frac{|V|}{K}$ zerfällt. Ein Knoten, der zu der Menge V_S adjazent ist, heißt* **Randknoten**. *Die* **Cutsize** $|V_S|$ *der Partition ist die Anzahl der Knoten in V_S, und das* **Cutweight** $w(V_S)$ *ist das Gewicht der Knoten in V_S. Ein Vertex Separator heißt* **fein**, *wenn keine echte Teilmenge von V_S einen Vertex Separator darstellt, ansonsten heißt der Vertex Separator* **grob**. Π_{GPVS} *ist* **optimal**, *wenn $w(V_S)$ minimal ist, d.h. wenn es keinen Vertex Separator einer Partition mit geringerem Cutweight gibt.*

Eine GPES und eine GPVS eines Graphen lassen sich leicht ineinander überführen, sodass es auf das zugrunde liegende Problem oder das persönliche Belieben ankommt, ob wir nach einem Edge Separator oder nach einem Vertex Separator suchen. Nehmen wir zum Beispiel einmal an, dass wir einen Edge Separator haben und daraus einen Vertex Separator erhalten möchten. Wir bezeichnen mit $\bar{G}$ den Graph, der durch die separierenden Kanten induziert wird. Wählen wir nun ein Vertex Cover von $\bar{G}$, decken wir dadurch alle Kanten aus dem Edge Separator ab. Wenn wir alle Knoten des Vertex Covers aus dem Graph entfernen, haben wir mindestens auch alle Kanten aus dem Edge Separator entfernt. Demnach ist jedes Vertex Cover von $\bar{G}$ ein Vertex Separator von G.

In diesem Sinne stellt die Cutsize einer optimalen GPES immer eine obere Schranke für die Cutsize einer optimalen GPVS dar, denn durch ein minimales Vertex Cover der Kanten in E_S erhalten wir einen zulässigen Vertex Separator V_S, der aber nicht optimal sein muss. Daher wird oft zunächst nach einer GPES gesucht wird, die dann zu einer GPVS verfeinert wird. Problematisch ist jedoch, dass eine gute GPES nicht automatisch zu einer guten GPVS führen muss.

Unbalanciertheit und variable Anzahl der Teilmengen

Algorithmen zur Graph-Partitionierung setzen oft voraus, dass alle Teilmengen gleich groß sind, was jedoch nicht immer möglich ist. Um dennoch eine Partition mit K gleich großen Teilmengen zu erhalten, können wir entsprechend viele Dummy-Knoten zum Graph hinzufügen, sodass insgesamt Km', $m' \in \mathbb{N}$, Knoten im Graph sind. Die Dummy-Knoten sind isoliert, haben also keine inzidenten Kanten. Nun bestimmen wir eine K-Partition, in der jede Teilmenge genau m' Knoten hat. Wenn wir die Dummy-Knoten wieder aus dem Graph löschen, erhalten wir eine Partition in der manche Teilmengen weniger als m' Knoten haben.

Wenn eine der Teilmengen nach dem Löschen der Dummy-Knoten leer ist, d.h. zuvor nur aus Dummy-Knoten bestand, haben wir im Endeffekt eine Partition in $K - 1$ Teilmengen bestimmt. Mit Hilfe dieser Erkenntnis können wir den Graph dahingehend manipulieren, dass wir eine Partition mit mindestens J und höchstens K Teilmengen erhalten, dazu fügen wir so viele Dummy-Knoten zum Graph hinzu, dass insgesamt $K\left\lceil \frac{n}{J} \right\rceil$ Knoten im modifizierten Graph sind. Wir erhalten dann eine Partition in K Teilmengen, die nach dem Löschen der Dummy-Knoten in eine Partition zerfällt, die zwischen J und K Teilmengen hat.

Beispiel 2.1.5 Wir betrachten einen Graph mit $n = 10$ Knoten. Wir wollen eine Partition erstellen, die zwischen drei und vier Teilmengen hat, also $J = 3$ und $K = 4$. Da der Graph nach den obigen Überlegungen aus $4\left\lceil \frac{10}{3} \right\rceil = 16$ Knoten bestehen muss, ist es nötig, dass sechs Dummy-Knoten hinzugefügt werden. Wenn wir nun eine Partition des modifizierten Graphen in vier Teilmengen erhalten, in der alle Teilmengen genau vier Elemente haben, ergibt sich daraus nach dem Löschen der sechs Dummy-Knoten eine Partition mit mindestens drei und höchstens vier Teilmengen, denn:

- Sind in keiner der vier Teilmengen mehr als drei Dummy-Knoten, enthalten alle Teilmengen nach dem Löschen mindestens noch ein Element, und wir erhalten auf dem originalen Graph eine Partition in vier Teilmengen.
- Besteht eine Teilmenge aus vier Dummy-Knoten, dann haben drei der vier Teilmengen nach dem Löschen der Dummy-Knoten noch mindestens ein Element, und wir erhalten auf dem originalen Graph eine Partition in drei Teilmengen.
- Eine Aufteilung der Dummy-Knoten, sodass nach dem Löschen zwei Teilmengen leer sind, ist nicht möglich, da es dazu mindestens acht Dummy-Knoten im Graph geben müsste. Deshalb ist keine Partition des originalen Graphen mit weniger als drei nicht-leeren Teilmengen möglich, insgesamt erhalten wir also eine Partition mit drei oder vier Teilmengen.

Anwendungsgebiete

Generell sind Graphen sehr beliebt, um Probleme zu beschreiben, da sich Beziehungen zwischen einzelnen Objekten durch die Kanten gut darstellen lassen und die Stärken der Beziehungen durch die Kantengewichte abgebildet werden können. Das Problem der Graph-Partitionierung wird daher für viele Fragestellungen benutzt, die nicht immer aus dem für Graphen typischen Gebiet der Netzwerk-Probleme stammen, wie etwa der Bildverarbeitung. Ein weiteres Beispiel ist das Lösen von Partiellen Differentialgleichungen (PDG), was unter Umständen sehr aufwendig werden kann und deshalb nicht exakt möglich ist. Ähnlich wie bei der Matrix-Partitionierung soll das Problem auf mehrere Prozessoren verteilt werden, wobei das PDG-Problem mit Hilfe von Grids dargestellt wird, um diese dann mit Hilfe von Graph-Partitionierung aufzuteilen.

Ein weiteres Anwendungsgebiet ist das VLSI Layout-Design, in dem Transistoren von integrierten Schaltkreisen auf einem Chip so platziert werden sollen, dass der Chip eine möglichst geringe Größe hat. Hier kommen oft sogenannte Hypergraphen zum Einsatz, um die Beziehungen zwischen mehreren Transistoren besser darstellen zu können. Mit der Hypergraph-Partitionierung werden wir uns gleich auch noch befassen.

2.1.2 Knoten- und Kanten-Konnektivität

Eine Vereinfachung des Partitionierungsproblems stellt die Suche nach der Knoten- oder Kanten-Konnektivität dar.

Definition 2.1.6 *Sei $G = (V, E)$ ein zusammenhängender Graph. Dann ist die* **Knoten-Konnektivität** *κ_V die minimale Anzahl zu entfernender Knoten, sodass G nicht mehr zusammenhängend ist, bzw. nur ein einzelner Knoten übrig bleibt. Analog dazu ist die* **Kanten-Konnektivität** *κ_E die minimale Anzahl zu entfernende Kanten, sodass G nicht mehr zusammenhängend ist.*

Die Knoten-Konnektivität ist kleiner als die Kanten-Konnektivität, außerdem können wir beide Werte durch den minimalen Knotengrad nach oben abschätzen.

Satz 2.1.7 *Sei $G = (V, E)$ ein Graph, κ_V die Knoten-Konnektivität, κ_E die Kanten-Konnektivität und d_1 der kleinste Grad in G. Dann gilt*

$$\kappa_V \leq \kappa_E \leq d_1 .$$

Beweis: Es sei E' eine Teilmenge von Kanten aus E, die G separiert, mit $|E'| = \kappa_E$. Jede dieser Kanten können wir entfernen, indem wir einen der beiden Endpunkte entfernen, insbesondere also, wenn wir ein Vertex Cover dieser Kanten entfernen. Ein minimales Vertex Cover dieser Kanten enthält höchstens $|E'|$ Knoten, wir können den Graph also durch Entfernen von höchstens $|E'|$ Knoten separieren. Also gilt $\kappa_V \leq \kappa_E$.
Nun sei v_i ein Knoten mit minimalem Grad d_1. Wenn wir alle inzidenten Kanten von v_i entfernen, ist v_i isoliert und G nicht mehr zusammenhängend. Also gilt $\kappa_E \leq d_1$. □

2.1.3 Die Laplace-Matrix

Sei $A(G)$ die *Adjazenzmatrix* eines Graphen G mit

$$a_{ij} = \begin{cases} 1 , & e_{ij} \in E \\ 0 , & \text{sonst} \end{cases}$$

und $D(G) = diag(d_i)$ die *Gradmatrix* von G, wobei d_i der Grad des Knoten i ist.

Definition 2.1.8 *Die $n \times n$-Matrix $L(G) = D(G) - A(G)$ heißt* **Laplace-Matrix** *von G.*

Die Laplace-Matrix wird oft nur für ungewichtete Graphen betrachtet, wir wollen sie aber auch für gewichtete Graphen definieren. Sei $A^w(G)$ die *gewichtete Adjazenzmatrix* von G mit

$$a^w_{ij} = \begin{cases} w_{ij} , & e_{ij} \in E \\ 0 , & \text{sonst} \end{cases}$$

und $D^w(G) = diag(d^w_i)$ die *gewichtete Gradmatrix* von G, wobei $d^w_i = \sum_{j:(i,j)\in E} w_{ij}$ der gewichtete Grad des Knoten i ist.

Definition 2.1.9 *Die $n \times n$-Matrix $Q(G) = D^w(G) - A^w(G)$ heißt* **gewichtete Laplace-Matrix** *von G.*

Natürlich entspricht die Matrix $Q(G)$ der Matrix $L(G)$, wenn alle Gewichte den Wert 1 haben. Im Folgenden werden wir sowohl $L(G)$ als auch $Q(G)$ abkürzend als „Laplace-Matrix“ bezeichnen. Ergebnisse werden wir oft nur für die Matrix $Q(G)$ beweisen, da sich diese sofort auf die Matrix $L(G)$ übertragen. Wenn Ergebnisse speziell für ungewichtete Graphen gelten, werden wir das anmerken, bzw. es wird aus dem Zusammenhang klar, dass es sich um einen ungewichteten Graph handelt.

Die Laplace-Matrix hat einige nützliche Eigenschaften, die wir in dem folgenden Lemma festhalten wollen.

Lemma 2.1.10 *Gegeben sei ein gewichteter Graph G. Dann gilt für die Laplace-Matrix Q(G):*

1. *Q(G) ist reellwertig und symmetrisch.*
2. *Die Eigenwerte $\lambda_1, \ldots, \lambda_n$ sind reellwertig, und die Eigenvektoren sind reellwertig und orthogonal zueinander.*
3. *Q(G) ist positiv semi-definit.*
4. *Alle Eigenwerte sind nicht-negativ, d.h. $\lambda_i \geq 0 \; \forall \; i = 1, \ldots, n$.*
5. *$\lambda_1 = 0$, und für den normierten Eigenvektor Λ_1 gilt $\Lambda_1 = (1/\sqrt{n})(1, \ldots, 1)$.*
6. *Die Anzahl der Eigenwerte mit Wert 0 entspricht der Anzahl der Zusammenhangskomponenten von G, insbesondere ist $\lambda_2 \neq 0$ äquivalent dazu, dass G zusammenhängend ist.*

Beweis:

1. Da $D^w(G)$ und $A^w(G)$ reellwertig und symmetrisch sind, ist auch $Q(G)$ als Differenz der beiden Matrizen reellwertig und symmetrisch.
2. Da $Q(G)$ eine symmetrische Matrix auf $\mathbb{R}^{n \times n}$ ist, hat sie nach Satz A.2.3 eine Orthonormalbasis aus Eigenvektoren und nach Satz A.2.4 nur reelle Eigenwerte.
3. Wir geben dem Graph G für den Moment eine Orientierung, d.h., bei jeder Kante ist einer der beiden Knoten der Startknoten und der andere der Endknoten. Dann ist die gewichtete Inzidenzmatrix $B^w(G)$ definiert durch

$$b_{ie} = \begin{cases} w_{ij}\,, & e_{ij} \in E \\ -w_{ij}\,, & e_{ji} \in E \\ 0\,, & \text{sonst} \end{cases}.$$

   Nach Satz A.1.1 können wir die Laplace-Matrix schreiben als $Q(G) = B^w(B^w)^T$. Nach Definition[1] ist $Q(G)$ genau dann positiv semi-definit, wenn $x^T Q x \geq 0$ für alle $x \neq 0$. Wir müssen demnach zeigen, dass

$$x^T Q x = x^T B^w (B^w)^T x = ((B^w)^T x)^T (B^w)^T x \geq 0$$

   ist. Wenn $e_{ij} \in E$ in der gewichteten Inzidenzmatrix die l-te Kante repräsentiert, ist die l-te Zeile von $(B^w)^T x$ genau

$$\ldots + w_{ij} \cdot x_i + \ldots + (-w_{ij}) \cdot x_j$$

   und somit ist

$$((B^w)^T x)^T (B^w)^T x = \sum_{(i,j) \in E} w_{ij}^2 (x_i - x_j)^2 \geq 0\,.$$

4. Da $Q(G)$ positiv semi-definit ist, sind alle Eigenwerte nach Satz A.2.2 nicht negativ.

[1] Definition A.2.1

5. Sei $e = (1, \ldots, 1)$ der 1-Vektor. Die Summe der Einträge in der i-ten Zeile von $A^w(G)e$ entspricht der Summe aller Gewichte der Kanten, mit denen der Knoten v_i inzident ist, also gerade d_i^w, was wiederum dem i-ten Eintrag von $D^w(G)e$ entspricht. Also ist $Q(G)e = D^w(G)e - A^w(G)e = 0$ und somit 0 ein Eigenwert und e ein Eigenvektor. Der normierte Eigenvektor ist dann $\Lambda_1 = (1/\sqrt{n})e$.

6. vgl. Elsner [22], S. 29. □

Die Laplace-Matrix ist auch als *Kirchhoff-Matrix* bekannt, was auf Ergebnisse des deutschen Physikers Gustav Robert Kirchhoff (1824-1887) zurückgeht. Kirchhoff arbeitete vor allem auf dem Gebiet der Elektrizität, im Zusammenhang mit elektronischen Netzwerken wird die Laplace-Matrix auch *Admittanz-Matrix* genannt.

Seit einigen Jahrzehnten spielt die Laplace-Matrix bei zahlreichen graphischen Problemen eine wichtige Rolle, dies gilt vor allem für den zweitkleinsten Eigenwert der Laplace-Matrix. Der tschechische Mathematiker Miroslav Fiedler (geb. 1926, im Bild) beschäftigte sich intensiv mit diesem Eigenwert und dem zugehörigen Eigenvektor [27, 28], dieser wird ihm zu Ehren als *Fiedler-Vektor* bezeichnet. Nach dem Matrix-Tree-Theorem von Kirchhoff [53] ist ein Graph genau dann zusammenhängend, wenn $\lambda_2 \neq 0$. Aus diesem Grund gab Fiedler dem zweitkleinsten Eigenwert den Namen *algebraische Konnektivität*. Generell können die Eigenwerte und Eigenvektoren als Maß für den Zusammenhang eines Graphen benutzt werden, so dient λ_2 zum Beispiel auch als untere Schranke für die Knoten- und Kanten-Konnektivität.

Satz 2.1.11 *Sei $G = (V, E)$ ein Graph, λ_2 der zweitkleinste Eigenwert der Laplace-Matrix $L(G)$, κ_V die Knoten- und κ_E die Kanten-Konnektivität. Dann gilt*

$$\lambda_2 \leq \kappa_V \leq \kappa_E \leq d_1 .$$

Beweis: vgl. Fiedler [27]

Wenn λ_2 groß ist, ist also auch die Konnektivität groß, d.h. wir müssen viele Knoten oder Kanten entfernen, um den Zusammenhang des Graphen zu zerstören.

Die Bedeutung von λ_2 für die Graphentheorie führte dazu, dass es mittlerweile viele Abschätzungen für diesen, aber auch andere Eigenwerte gibt [18, 59, 79].

2.2 Hypergraphen

Ein *Hypergraph* $H = (U, N)$ besteht aus *Knoten* U und *Hyperkanten* bzw. *Netzen* N. Die Netze $h_j \in N$ verbinden Knoten aus U, die Knoten eines Netzes werden *Pins* genannt und mit $Pins(h_j)$ notiert, zudem bezeichnet $p_j = |Pins(h_j)|$ die Anzahl der Pins des Netzes h_j. Ein Knoten u_i, der zu einem Netz h_j gehört, ist *inzident* zu diesem Netz, und $Nets(u_i)$ ist die Menge aller Netze, zu denen u_i inzident ist.

Bei der Definition fällt auf, dass jeder Graph auch ein Hypergraph ist, in dem jedes Netz aus genau zwei Knoten besteht. Die Begriffe „Netz“ und „Hyperkante“ haben dieselbe Bedeutung, jedoch sehen Hyperkanten in visualisierten Graphen oft eher aus

wie Netze, während im Bezug auf Matrizen der Begriff „Hyperkante“ wohl etwas anschaulicher ist, wie wir sehen werden. Wir bezeichnen die Netze mit h_j statt n_j, da die Parameter n_k bereits die Blockgrößen der Arrowhead-Form beschreiben.

2.2.1 Hypergraph-Partitionierung

Auch einen Hypergraph können wir partitionieren, die Definition ist ähnlich wie für Graphen.

Definition 2.2.1 *Eine* **(balancierte)** K**-Partition** Π *auf einem Hypergraph* H *ist gegeben durch eine Menge* $\{U_1 \ldots, U_K\}$ *mit*

- $U_i \cap U_j = \emptyset$, $i \neq j$, $\bigcup_{k=1}^{k} U_k = U$,
- $W_k \leq \frac{W}{K}(1 + \epsilon)$, $k = 1, \ldots, K$.

Ein Netz h_j **verbindet** *eine Teilmenge* U_i*, wenn mindestens ein Knoten in* U_i *inzident zu* h_j *ist. Wenn ein Netz* n_j *mehr als eine Teilmenge verbindet, heißt dieses Netz* **geschnitten** *oder* **extern**, *ansonsten* **nicht-geschnitten** *oder* **intern**. *Die Menge* N_S *ist die Menge aller geschnittenen Netze, und* N_S *ist* **optimal**, *wenn* $w(N_S)$ *minimal ist.*

Für die Cutsize gibt es mehrere Definitionen, die zumeist aus dem Bereich des VLSI Layout Designs kommen, zwei dieser Definitionen wollen wir hier vorstellen. Bei der ersten Definition wird das Gewicht aller geschnittenen Netze gezählt, die Cutsize ist daher gegeben durch

$$\xi(\Pi) = \sum_{h_j \in N_S} w_j \,,$$

wobei w_j das Gewicht von Netz h_j ist. Auch bei der zweiten Definition werden die Gewichte der geschnittenen Netze betrachtet, allerdings auch, wie viele Teilmengen ein Netz verbindet. Die Cutsize ist dann definiert als

$$\xi(\Pi) = \sum_{h_j \in N_S} w_j(c_j - 1) \,,$$

wobei c_j die Anzahl der Teilmengen ist, die durch h_j verbunden werden. Bei einem Graph ist $c_j = 1$ für jede externe Kante und $c_j = 0$ für jede interne Kante, weshalb beide Definitionen für Graphen äquivalent sind.

Wir werden im weiteren Verlauf dieser Arbeit die erste Cutsize-Definition verwenden. Natürlich können wir auch hier wieder zwischen „Cutsize“ und „Cutweight“ unterscheiden, die Cutsize ist dann die Anzahl der geschnittenen Netze, und das Cutweight entspricht einer der beiden obigen Definitionen.

2.2.2 Graph-Repräsentationen für einen Hypergraph

Da viele Partitionierungsalgorithmen nur auf normalen Graphen arbeiten können oder auf diesen wesentlich schneller arbeiten als auf Hypergraphen, ist es sinnvoll, eine Repräsentation des Hypergraphen als Graph zu finden. Dabei sollte das Gewicht eines Schnitts in der Graph-Repräsentation immer genauso groß sein, wie das Gewicht eines entsprechenden Schnitts im Hypergraph. Ihler, Wagner und Wagner [45] zeigten

jedoch, dass dies selbst bei Hinzufügen zusätzlicher Knoten nicht möglich ist. Wir stellen nun zwei der bekanntesten Graph-Modelle für Hypergraphen vor.

Beim *Clique Net Graph*-Modell (kurz: *CNG*-Modell) bleiben die Knoten des Hypergraphen bestehen, jedoch wird ein Netz nun als Clique der Pins dargestellt. Ist eine Kante in der Menge der geschnittenen Kanten einer GPES, dann sind alle Netze, die durch diese Kante repräsentiert werden, in der Menge der geschnittenen Netze des Hypergraphen und vice versa. Eine perfekte CNG-Repräsentation gibt es jedoch nicht, in Kapitel 5.2.1 werden wir aber versuchen, den Hypergraph durch eine geeignete Wahl von Gewichten im CNG so gut wie möglich zu approximieren.

Ein weiteres Modell ist das *Net Intersection Graph*-Modell (kurz: *NIG*-Modell) mit einem Graph $G = (V, E)$, in dem jeder Knoten $v_i \in G$ einem Netz h_i von H entspricht. Zwei Knoten v_i und v_j sind genau dann adjazent in G, wenn die beiden zugehörigen Netze $h_i, h_j \in N$ mindestens einen gemeinsamen Knoten haben, d.h. $e_{ij} \in E$ genau dann, wenn $Pins(h_i) \cap Pins(h_j) \neq \emptyset$. Der Vorteil des NIG-Modells ist, dass wir durch dieses Modell die Cutsize des Hypergraphen exakt bestimmen können.

Satz 2.2.2 *Sei H ein Hypergraph und G^{NIG} der entsprechende Net-Intersection-Graph. Dann induziert eine GPVS des NIG mit Cutsize $|V_S|$ eine Partition des Hypergraphen mit*

$$|N_S| = |V_S| .$$

Die Richtigkeit dieser Aussage werden wir gleich nachweisen.

Ein Problem des NIG-Ansatzes ist jedoch, dass es keine eindeutige Rückwärtsinduktion gibt [1]. Der NIG eines Hypergraphen ist zwar eindeutig, nicht aber der Hypergraph, den wir aus einem NIG ableiten können. Zudem können wir die Balanciertheit nicht direkt auf den NIG übertragen, da die Knoten des Hypergraphen beim NIG-Modell im Gegensatz zum CNG-Modell im Graph nicht mehr vorkommen. Allerdings gibt es Ansätze, um die Bedingung der Balanciertheit durch Gewichte abzubilden [51].

Hypergraph-Partitionierung mit dem NIG-Modell

Mit Hilfe des NIG-Modells können wir einen Hypergraph als Graph darstellen. Wenn wir also einen Hypergraph partitionieren wollen, können wir den NIG des Hypergraphen aufstellen und diesen dann mit Methoden der Graph-Partitionierung partitionieren. Aus der Partitionierung des NIG wollen wir dann auf eine Partitionierung des Hypergraphen schließen. Was auf den ersten Blick einfach klingt, bringt allerdings Probleme mit sich, denn die Partition des NIG stellt nicht unbedingt eine Partition des Hypergraphen dar.

Wenn wir eine GPES des NIG bestimmen, können wir versuchen, die Partition auf den Hypergraph zu übertragen, d.h., dass wir die Teilmengen V_k als Teilmengen N_k einer Partition des Hypergraphen betrachten. Wenn wir die Knoten des NIG wieder als Netze des Hypergraphen darstellen, gibt es wahrscheinlich einen Pin, der zu mindestens zwei verschiedenen Netzen inzidiert. Wenn die Knoten aller Netze, zu denen ein Pin inzident ist, in der gleichen Teilmenge V_k der Partition des NIG liegen, können wir auch den Pin einfach dieser Teilmenge zuordnen. Wenn es aber zwei Netze gibt, deren Knoten in der Partition des NIG in zwei verschiedenen Teilmengen sind, können wir den Pin, der zu beiden Netzen inzident ist, nicht eindeutig einer der beiden Teilmengen

zuordnen. Wir erhalten aus der Partition des NIG also keine wohldefinierte Partition für den Hypergraph, da sich die Teilmengen der Partition des NIG im Hypergraph überschneiden. Im Bereich des VLSI Layout Designs ist dieses Problem als *Module Contention Problem* bekannt.

Wir können dieses Problem umgehen, indem wir nach einer GPVS suchen. Wir starten mit einer Knoten-Partition $\Pi_{\text{GPVS}} = \{V_1, \ldots, V_K; V_S\}$ des NIG G, durch die wir dann eine Netz-Partition $\{N_1, \ldots, N_K; N_S\}$ des Hypergraphen H erhalten wollen.

Lemma 2.2.3 *Gegeben sei eine Partition $\Pi_{GPVS} = \{V_1, \ldots, V_K; V_S\}$ des NIG G. Dann wird dadurch eine $(K+1)$-Netz-Partition $\{N_1, \ldots, N_K; N_S\}$ des Hypergraphen H induziert.*
Beweis: Durch die Knoten-Teilmengen V_k des NIG erhalten wir Netz-Teilmengen N_k für $k = 1, \ldots, K$, $N_S = V_S$, da die Knoten im NIG die Netze in H repräsentieren. Weil Π_{GPVS} eine Partition ist, gibt es keinen Knoten einer Teilmenge V_k, der gleichzeitig in der Nachbarschaft einer Teilmenge V_l, $k \neq l$, liegt. Daraus folgt für die Netz-Teilmengen, dass $Pins(N_k) \cap Pins(N_l) = \emptyset$, für $k \neq l$, denn sonst gäbe es in G eine Kante e_{ij} zwischen $v_i \in V_k$ und $v_j \in V_l$ und somit gäbe es eine Kante zwischen V_k und V_l, die durch das Entfernen der Knoten V_S nicht weg fällt. Das wäre aber ein Widerspruch zur Annahme, dass die Partition eine GPVS ist. Also erhalten wir eine Netz-Partition $\{N_1, \ldots, N_K; N_S\}$. □

Kommen wir jetzt aber von der Netz-Partition auch auf eine Knoten-Partition des Hypergraphen? Dazu klammern wir zunächst einmal alle Netze in N_S aus und betrachten nur die internen Netze. Da diese Netze nur innerhalb einer Teilmenge verlaufen, liegen auch alle ihre Pins jeweils nur in einer Teilmenge. Daraus erhalten wir eine Partition $\Pi'_{\text{HP}} = \{U'_1, \ldots, U'_K\}$ mit $U'_k = Pins(N_k)$, $k = 1, \ldots, K$. Da wir nur die internen Netze betrachtet haben, sind die Teilmengen dieser Partition nicht verbunden, weshalb $N_S = V_S$ einen Net-Cut für $\Pi_{\text{HP}'}$ darstellt.

Allerdings haben wir dadurch noch nicht alle Knoten des Hypergraphen abgedeckt, es fehlen noch die Knoten, die nur zu externen Netzen inzident sind. Dies sind die Knoten

$$U'_F = U - \bigcup_{k=1}^{K} U'_k = U - \bigcup_{k=1}^{K} Pins(N_k) = \{u_i \in U : Nets(u_i) \subset N_S\},$$

es gilt also $Nets(U'_F) \subset N_S$. Dadurch können wir jeden Knoten der Menge U'_F zu einer beliebigen Teilmenge von Π'_{HP} hinzufügen, ohne dass der Net-Cut N_S zerstört wird, diese Knoten bezeichnen wir daher als *freie Knoten*.

Lemma 2.2.4 *Der Net-Cut N_S wird durch das Hinzufügen der Knoten U'_F zu den Teilmengen der Partition nicht verändert.*
Beweis: Wir betrachten eine beliebige Zuweisung der Knoten U'_F zu den Teilmengen von Π'_{HP} um die K-Partition $\Pi_{\text{HP}} = \{U_1, \ldots, U_K\}$ zu komplettieren. Da $Nets(U'_F) \subset N_S$, sind alle Netze von U'_F schon im Cut von Π'_{HP}, der Net-Cut wird also nicht größer. Da N_S zudem ein Net-Cut von Π'_{HP} ist, wird durch die Zuweisung der Knoten aus U'_F kein Netz aus dem Cut entfernt. □

Die freien Knoten können leicht identifiziert werden, sie entsprechen den Randknoten einer Teilmenge, die keine Verbindung zu internen Netzen haben, sondern nur zu geschnittenen Netzen inzident sind.

Wir haben also eine K-Partition Π_{HP} des Hypergraphen H gefunden mit $|N_S| = |V_S|$, es folgt also die Aussage von Satz 2.2.2. Diese Formulierung ist allerdings nur für die erste Cutsize-Definition gültig.

3 Algorithmen für das Graph-Partitionierungsproblem

Es gibt viele Verfahren zur Partitionierung eines Graphen, wir wollen an dieser Stelle einige bekannte Algorithmen vorstellen. Wir konzentrieren uns dabei auf Algorithmen, die wir sinnvoll für unser Matrix-Partitionierungsproblem anwenden können, weitere Verfahren werden in [13, 22, 29] vorgestellt.

Es gibt *globale Verfahren*, die auf dem gesamten Graph arbeiten und alle Informationen des Graphen nutzen, und es gibt *lokale Verfahren*, die immer nur kleine Teilmengen des Graphen betrachten und auch nur auf einigen wenigen Knoten zur gleichen Zeit arbeiten. Die meisten der Verfahren sind *deterministisch*, also ohne zufällige Entscheidungen, doch es gibt auch *randomisierte* Algorithmen. Manchmal ist es gut, den Graph zusammenzufassen, was bei *Multilevel-Algorithmen* ausgenutzt wird. Und zu guter Letzt kann man auch bei den Algorithmen über eine Parallelisierung nachdenken, um die Berechnung zu beschleunigen.

Das Graph-Partitionierungsproblem ist NP-vollständig [30], d.h., es existiert kein exakter Algorithmus mit polynomieller Laufzeit. Da jeder Graph gleichzeitig auch ein Hypergraph ist, ist auch das Hypergraph-Partitionierungsproblem NP-schwer. Zwar gibt es für das Graph-Partitionierungsproblem auch exakte Algorithmen, doch die funktionieren nur für Graphen mit vergleichsweise wenigen Knoten gut und werden schon für mittelgroße Graphen zu langsam. Deshalb sind eigentlich alle in der Praxis verwendeten Algorithmen Heuristiken.

Die Heuristiken unterscheiden sich meistens in Bezug auf Qualität und Laufzeit, wobei sich die Frage stellt, welche dieser beiden Eigenschaften wichtiger ist. Im VLSI Layout-Design nimmt man längere Laufzeiten wahrscheinlich eher in Kauf, da eine bessere Lösung für viele verkaufte Chips benutzt wird und somit bares Geld wert sein kann.

Bei der Matrix-Vektor-Multiplikation sieht das aber etwas anders aus, denn hier benutzen wir Graph-Partitionierung, um die Berechnung zu beschleunigen. Wenn eine Heuristik eine sehr lange Laufzeit hat, macht sie dadurch womöglich den Zeitgewinn zunichte, den wir durch die Matrix-Partitionierung erreichen. Deshalb ziehen wir bei zwei Lösungen eher die etwas schlechtere Lösung vor, wenn die Laufzeit bedeutend kürzer ist. Natürlich haben wir dabei trotzdem noch einen gewissen Anspruch an die Qualität der Lösung, sodass wir zu schlechte Lösungen mit sehr kurzer Laufzeit auch nicht akzeptieren werden.

Hinzu kommt, dass wir die Matrix wohl nicht allzu oft benutzen werden, weshalb wir ständig neue Graph-Partitionierungsprobleme lösen müssen. Sollten wir die Matrix aber über einen längeren Zeitraum verwenden, können wir überlegen, ob wir in das Graph-Partitionierungsproblem etwas mehr Zeit investieren, um eine bessere Lösung zu erzielen. Auch durch eine großzügigere Auslegung der Balanciertheit sind bessere Ergebnisse möglich.

3.1 Globale Verfahren

Globale Verfahren arbeiten auf dem gesamten Graph und nutzen die Informationen aller Knoten und Kanten. Für eine vorgegebene Zahl K generieren sie eine Partition des Graphen in K Teilmengen. Dies geschieht oft rekursiv, d.h., der Graph wird in zwei Teilmengen zerlegt, diese beiden Teilmengen werden wiederum zerlegt usw. bis wir schließlich eine K-Partition haben. Viele Algorithmen suchen speziell nach Bisektionen oder wurden ursprünglich für das Bisektions-Problem konzipiert. Globale Verfahren werden zudem oft in Verbindung mit lokalen Verbesserungsmethoden angewandt, um die Qualität der Partitionen zu verbessern.

3.1.1 Rekursive Graph-Bisektion

In der Praxis kommt es oft vor, dass wir eine K-Partition finden möchten, bei der K eine Zweierpotenz ist, beispielsweise haben Computer oft 2^t Prozessoren. Da das Prinzip der Bisektion zudem relativ einfach ist, sind Bisektions-Verfahren sehr beliebt.

Bei der Rekursiven Graph-Bisektion [69] bestehen die beiden Teilmengen zunächst aus zwei Knoten, die eine möglichst große Distanz zueinander haben, d.h. möglichst viele Kanten auf einem kürzesten Weg. Die anderen Knoten werden dann jeweils zu derjenigen Teilmenge des Ausgangsknoten hinzugefügt, zu dem sie die kleinere Distanz haben, wobei die Teilmengen am Ende gleich groß sein sollen. Die Rekursive Bisektion kann allerdings beliebig schlecht werden, für planare[1] Graphen schneidet sie aber ganz gut ab [70]. Ein besseres Verfahren ist die Rekursive Spektrale Bisektion.

3.1.2 Rekursive Spektrale Bisektion

Die Rekursive Spektrale Bisektion [69] liefert in der Regel sehr gute Partitionen von Graphen und benutzt dabei, wie der Name schon sagt, Eigenwerte und Eigenvektoren.

Der Einfachheit halber nehmen wir zunächst an, dass V eine gerade Anzahl Knoten hat, und G ungewichtet ist. Wir verwenden einen Vektor $x \in \{-1, 1\}^{|V|}$ mit

$$x_i = \begin{cases} 1\,, & i \in V_1 \\ -1\,, & i \in V_2 \end{cases}$$

und $\sum_{i \in V} x_i = 0$, um eine Bisektion (V_1, V_2) zu repräsentieren. Dadurch können wir die Funktion

$$f(x) = \frac{1}{4} \sum_{e_{ij} \in E} (x_i - x_j)^2$$

aufstellen, die dann wegen

$$(x_i - x_j)^2 = \begin{cases} 0\,, & \text{falls } i \in V_1, j \in V_1 \text{ oder } i \in V_2, j \in V_2 \\ 4\,, & \text{falls } i \in V_1, j \in V_2 \text{ oder } i \in V_2, j \in V_1 \end{cases}$$

die Anzahl der Kanten zwischen den beiden Teilmengen V_1 und V_2 zählt. Wenn wir also die Anzahl der Kanten zwischen den Teilmengen minimieren wollen, müssen wir die Funktion $f(x)$ minimieren.

[1] Ein Graph heißt *planar*, wenn er in einer 2-dimensionalen Ebene so visualisiert werden kann, dass sich keine Kanten schneiden.

Das Problem als Ganzzahliges Programm

Wir wollen das Graph-Partitionierungsproblem als Ganzzahliges Programm formulieren und formen dazu die Funktion f um. Mit der zweiten binomischen Formel wird der Ausdruck $(x_i - x_j)^2$ in der Funktion f zu

$$(x_i - x_j)^2 = x_i^2 - 2x_i x_j + x_j^2 ,$$

und daraus ergibt sich dann

$$\sum_{e_{ij} \in E} (x_i - x_j)^2 = \sum_{e_{ij} \in E} (x_i^2 + x_j^2) - 2 \sum_{e_{ij} \in E} x_i x_j .$$

Diesen Ausdruck können wir nun durch die Adjazenzmatrix und die Gradmatrix darstellen. Zum einen ist

$$\sum_{e_{ij} \in E} (x_i^2 + x_j^2) = \sum_{e_{ij} \in E} 2 = 2|E| = \sum_{i \in V} d_i = x_i^2 d_i = x^T D(G) x ,$$

wobei wir beim dritten Gleichheitszeichen Satz A.1.2 benutzt haben, und das vierte Gleichheitszeichen wegen $x_i = \pm 1$, $i \in V$ gilt. Zum anderen erhalten wir

$$2 \sum_{e_{ij} \in E} x_i x_j = \sum_{i \in V} \sum_{j \in V} x_i x_j a_{ij} = \sum_{i \in V} x_i \sum_{j \in V} a_{i,j} x_j = x^T A(G) x .$$

Insgesamt gilt also

$$\sum_{e_{ij} \in E} (x_i - x_j)^2 = x^T D(G) x - x^T A(G) x = x^T (D(G) - A(G)) x = x^T L x .$$

Hier bezeichnet L die in Kapitel 2.1.3 definierte Laplace-Matrix.

Das Graph-Partitionierungsproblem können wir nun als das folgende ganzzahlige Programm formulieren:

$$\begin{aligned} \min \ & \frac{1}{4} x^T L x \\ s.t. \ & \sum_{i=1}^{n} x_i = 0 \\ & x_i \in \{-1, 1\} . \end{aligned}$$

Wir relaxieren dieses Programm, indem wir die Bedingungen $x_i \in \{-1, 1\}$ verallgemeinern:

$$\begin{aligned} \min \ & \frac{1}{4} x^T L x \\ s.t. \ & \sum_{i=1}^{n} x_i = 0 \\ & x^T x = 0 . \end{aligned}$$

Eine Lösung der Relaxation können wir auf den zulässigen Bereich des diskreten Problems projizieren, um dadurch eine Schätzung für die optimale Lösung des originalen Programms zu erhalten. Da das zweite Programm eine Relaxation des ersten Programms ist, ist sein Zielfunktionswert kleiner, weshalb wir durch die optimale Lösung der Relaxation eine untere Schranke für den optimalen Zielfunktionswert des originalen Programms erhalten.

Lösung der Relaxation

Wie wir nun zeigen werden, können wir die optimale Lösung der Relaxation eindeutig angeben. Es sei $(\Lambda_1, \ldots, \Lambda_n)$ eine Orthonormalbasis der Eigenvektoren von $L(G)$ mit den zugehörigen Eigenwerten $\lambda_1 \leq \ldots \leq \lambda_n$. Da die Λ_i eine Basis bilden, können wir den Vektor x aus der Relaxation darstellen als

$$x = \sum_{i=1}^{n} \alpha_i \Lambda_i$$

für entsprechende Koeffizienten α_i. Für die Nebenbedingung $x^T x = n$ ergibt sich dann

$$n = x^T x = \left(\sum_{i=1}^{n} \alpha_i \Lambda_i \right)^T \sum_{i=1}^{n} \alpha_i \Lambda_i = \sum_{i=1}^{n} {\alpha_i}^2 {\Lambda_i}^T \Lambda_i = \sum_{i=1}^{n} {\alpha_i}^2 , \tag{3.1.1}$$

da die Λ_i normiert sind und somit ${\Lambda_i}^T \Lambda_i = 1$ gilt. Die Funktion $f(x)$ können wir jetzt schreiben als

$$f(x) = \frac{1}{4} x^T L x = \frac{1}{4} \left(\sum_{i=1}^{n} \alpha_i \Lambda_i \right)^T L \sum_{i=1}^{n} \alpha_i \Lambda_i = \frac{1}{4} \left(\sum_{i=1}^{n} \alpha_i \Lambda_i \right)^T \sum_{i=1}^{n} \alpha_i L \Lambda_i .$$

Da die Λ_i Eigenvektoren sind und daher $L\Lambda_i = \lambda_i \Lambda_i$ gilt, erhalten wir

$$\frac{1}{4} \left(\sum_{i=1}^{n} \alpha_i \Lambda_i \right)^T \sum_{i=1}^{n} \alpha_i L \Lambda_i = \frac{1}{4} \left(\sum_{i=1}^{n} \alpha_i \Lambda_i \right)^T \sum_{i=1}^{n} \alpha_i \lambda_i \Lambda_i = \frac{1}{4} \sum_{i=1}^{n} {\alpha_i}^2 \lambda_i {\Lambda_i}^T \Lambda_i = \frac{1}{4} \sum_{i=1}^{n} {\alpha_i}^2 \lambda_i$$

bzw. wegen $\lambda_1 = 0$

$$f(x) = \frac{1}{4} \sum_{i=2}^{n} {\alpha_i}^2 \lambda_i .$$

Mit Gleichung 3.1.1 und da $\lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_n$ können wir $f(x)$ abschätzen durch

$$f(x) = \frac{1}{4} \sum_{i=2}^{n} {\alpha_i}^2 \lambda_i \geq \frac{1}{4} \lambda_2 \sum_{i=2}^{n} {\alpha_i}^2 = \frac{1}{4} \lambda_2 n .$$

Wählen wir nun $x^* = \sqrt{n} \Lambda_2$, so gilt

$$f(x^*) = \frac{1}{4} (\sqrt{n} {\Lambda_2}^T L \sqrt{n} \Lambda_2) = \frac{1}{4} (\sqrt{n} \sqrt{n} {\Lambda_2}^T \lambda_2 \Lambda_2) = \frac{1}{4} \lambda_2 n .$$

Demnach wird die obere Schranke für x^* erreicht. Zudem erfüllt x^* die Bedingung $\sum_{i=1}^{n} {x_i}^* = 0$, denn mit $\Lambda_1 = (1/\sqrt{n}) e$ bzw. äquivalent dazu $\sqrt{n} \Lambda_1 = e$, und der Tatsache, dass die Eigenvektoren orthogonal zueinander sind, folgt

$$\sum_{i=1}^{n} {x_i}^* = e^T x^* = (\sqrt{n} \Lambda_1)^T (\sqrt{n} \Lambda_2) = n {\Lambda_1}^T \Lambda_2 = 0 .$$

Also ist x^* eine optimale Lösung der Relaxation, und der Zielfunktionswert $\frac{\lambda_2 n}{4}$ ist eine untere Schranke für die Cutsize des Graph-Partitionierungsproblems.

Vom Fiedler-Vektor zur Bisektion

Wir haben gesehen, dass wir die Relaxation des Graph-Partitionierungsproblems mit Hilfe des Fiedler-Vektors Λ_2 lösen können. Nun wollen wir vom Fiedler-Vektor auf eine Lösung für das ganzzahligen Problems schließen. Das folgende Theorem von Fiedler [28] liefert dabei die entscheidende Idee.

Theorem 3.1.1 *Sei $G = (V, E)$ ein zusammenhängender Graph mit $V = \{v_1, \ldots, v_n\}$, und u der Fiedler-Vektor des Graphen. Für $r \geq 0$ und $V_1 := \{v_i \in V : u_i \geq -r\}$ ist der durch V_1 induzierte Teilgraph zusammenhängend. Für $r \leq 0$ ist der durch $V_2 := \{v_i \in V : u_i \leq -r\}$ induzierte Teilgraph zusammenhängend.*

Interessant ist, dass dieses Ergebnis unabhängig von der Länge und den Vorzeichen der Einträge des Fiedler-Vektors gilt. Der Fiedler-Vektor separiert also quasi die Knoten des Graphen G in zwei Mengen. Diese Interpretation nutzen wir zur Aufteilung der Knoten in die zwei Teilmengen einer Bisektion.

Wir bestimmen zunächst den Median[2] μ der Einträge des Fiedler-Vektors. Dann definieren wir die beiden Teilmengen V_1 und V_2 der Bisektion durch $V_1 := \{v_i \in V : u_i > \mu\}$ und $V_2 := \{v_i \in V : u_i < \mu\}$. Im Fall $u_i = \mu$ wollen wir die Knoten so aufteilen, dass die Partition balanciert ist. Dadurch befinden sich die Knoten, die zu den $\frac{n}{2}$ größten Einträgen gehören, in V_1, und die $\frac{n}{2}$ anderen Knoten in V_2.

Wenn wir diese Methode benutzen, müssen wir natürlich den Fiedler-Vektor berechnen, zum Beispiel mit dem Algorithmus von Lanczos [55]. Die Berechnung des Eigenvektors muss nicht allzu genau sein, da wir nur an einer Sortierung der Einträge des Fiedler-Vektors nach der Größe interessiert sind.

Wenn wir eine Partition mit $K = 2^t$ Teilmengen finden möchten, müssen wir die Spektrale Bisektion rekursiv anwenden, d.h., wir müssen die Spektrale Bisektion noch einmal getrennt auf die beiden Teilmengen V_1 und V_2 anwenden, bis wir K Teilmengen haben. Wenn der Graph eine ungerade Anzahl Knoten hat, können wir Dummy-Knoten hinzufügen, die mit keinem der anderen Knoten verbunden sind und am Ende wieder aus dem Graph heraus genommen werden.

Quadrisektion und Oktasektion

Wenn wir eine K-Partition mit großem $K = 2^t$ haben möchten, ist es unter Umständen etwas zeitaufwendig, dies mittels Bisektion zu tun. Schneller geht es, wenn wir den Graph pro Iteration gleich in mehr als zwei Teilmengen zerlegen. Hendrickson und Leland versuchen dies mit ihren Algorithmen zur *Spektralen Quadrisektion* und *Spektralen Oktasektion* [41], bei denen der Graph pro Iteration in vier bzw. acht Teilmengen zerlegt wird. Diese beiden Algorithmen bauen auf dem Verfahren der Spektralen Bisektion auf. Allerdings benötigt die Spektrale Quadrisektion für die Lösung zusätzlich noch den drittkleinsten und die Spektrale Oktasektion sogar noch den viertkleinsten Eigenwert der Laplace-Matrix.

[2] Der Median μ ist der mittlere Wert einer Zahlenreihe $(X_1 \leq \ldots \leq X_n)$, d.h. $\mu = X_{\frac{n+1}{2}}$, falls n gerade, und $\mu = \frac{1}{2}(X_{\frac{n}{2}} + X_{\frac{n}{2}+1})$, falls n ungerade.

Gewichtete Spektrale Bisektion

Die Rekursive Spektrale Bisektion können wir auch für einem gewichteten Graph durchführen, die Einträge der gewichteten Laplace-Matrix Q sind bekanntlich durch

$$q_{ij} = \begin{cases} -w_{e_{ij}}, & e_{ij} \in E \text{ und } i \neq j \\ \sum_{k=1}^{n} w_{e_{ik}}, & i = j \\ 0, & \text{sonst} \end{cases}$$

gegeben. Zusätzlich führen wir noch die Diagonalmatrix V ein, auf deren Diagonalen die Gewichte der Knoten stehen. Das Verfahren der *Gewichteten Spektralen Bisektion* verläuft dann relativ analog zum obigen Verfahren. Wir formulieren das Problem als ganzzahliges Programm

$$\begin{aligned} \min \ & \frac{1}{4} x^T L x \\ s.t. \ & x^T V e = 0 \\ & x \in \{-1, 1\}^n, \end{aligned}$$

das wir dann relaxieren zu

$$\begin{aligned} \min \ & \frac{1}{4} x^T L x \\ s.t. \ & x^T V e = 0 \\ & x^T V x = e^T V e. \end{aligned}$$

Die Lösung der Relaxation können wir wieder über ein Eigenwertproblem bestimmen. Sie ist gegeben durch den Eigenvektor des zweitkleinsten Eigenwertes der Matrix $V^{-1/2} L V^{-1/2}$. Diese Matrix ist genau wie L und Q positiv semi-definit und hat auch die gleiche Struktur wie L und Q.

3.1.3 Greedy- und Graph-Growing-Algorithmen

Greedy-Algorithmen sind generell sehr beliebte Algorithmen, weil sie sehr schnell eine Lösung finden. Da sie dabei meistens nach der aktuell bestmöglichen Verbesserung suchen, kann sich die Lösung gegen Ende des Durchlaufs allerdings stark verschlechtern.

Ein typisches Beispiel für einen Greedy-Algorithmus ist der *Greedy Graph Growing Algorithm* von Karypis und Kumar [48], der jeweils den Knoten mit den wenigsten noch nicht in einer Teilmenge befindlichen Nachbarknoten zu einer Teilmenge hinzufügt. Sobald eine Teilmenge genug Knoten hat, wird eine neue Teilmenge betrachtet. Während die Cutsize zu Beginn moderat ansteigt, da die Knoten mit wenigen inzidenten Kanten gewählt werden, kann es passieren, dass am Ende die Knoten übrig bleiben, die sehr viele inzidente Kanten haben und die Cutsize somit stark ansteigen lassen. Diese Problematik ist typisch für Greedy-Algorithmen, weshalb diese oft auch nur zur Bestimmung einer Startlösung genutzt werden.

Graph-Growing-Algorithmen versuchen die Partition auf eine bestimmte Art und Weise anwachsen zu lassen. Ein Beispiel für einen Graph-Growing-Algorithmus ist *Farhats Algorithmus* [24]. Ähnlich wie Greedy-Partitioning füllt Farhats Algorithmus die

Teilmengen der Partition der Reihe nach auf, wobei versucht wird, die Anzahl der Kanten zwischen zwei Teilmengen klein zu halten.

Da Greedy- und Graph-Growing-Algorithmen zwar sehr schnell, allerdings oft auch sehr schlecht sind, ist die Spektrale Bisektion generell das wesentlich bessere Verfahren. Dafür setzen Greedy- und Graph-Growing-Algorithmen nicht voraus, dass die Anzahl der Teilmengen eine Zweierpotenz ist.

3.2 Lokale Verfahren

Lokale Verfahren arbeiten oft auf einer schon bestimmten Partition und versuchen diese zu verbessern, weshalb sie oft auch als *Verbesserungsverfahren* bezeichnet werden. Wenn wir nicht mit einer zufälligen Partition starten möchten, funktionieren sie nur in Verbindung mit einem globalen Verfahren. Die Kombination aus globalen und lokalen Verbesserungsverfahren liefert in der Praxis oft sehr gute Ergebnisse. Die meisten lokalen Verfahren basieren oft auf dem Prinzip der *lokalen Suche*.

3.2.1 Der Kernighan-Lin-Algorithmus

Eine der ersten lokalen Verbesserungsmethoden war der *Algorithmus von Kernighan und Lin* (kurz: *KL-Algorithmus*) [52], und viele andere lokale Verbesserungsmethoden sind Varianten von diesem Algorithmus aus dem Jahr 1970. Ursprünglich war für den Algorithmus eine zufällige Startpartition vorgesehen, allerdings hat sich gezeigt, dass der Algorithmus vor allem für große Graphen eine gute Startpartition benötigt, um gute Resultate zu liefern. Der Algorithmus arbeitet auf einer Bisektion und tauscht dort lokal Nachbarn in verschiedenen Teilmengen aus, um die Cutsize zu verringern. Wenn wir eine K-Partition mit $K > 2$ verbessern möchten, müssen wir den KL-Algorithmus rekursiv anwenden.

Als Ausgangspunkt dient eine Partition (V_1, V_2), die durch Vertauschen von Knoten zwischen V_1 und V_2 verbessert werden soll. Dazu definieren wir für einen Knoten $u \in V_1$ die *externen Kosten*

$$E_u = \sum_{j \in V_2} a_{uj}$$

als die Anzahl aller Kanten, die aus V_1 herausführen, wobei $A(G) = (a_{ij})$ die Adjazenzmatrix des Graphen G ist. Die *internen Kosten*

$$I_u = \sum_{j \in V_1} a_{uj}$$

sind die Anzahl der Kanten, die u mit einem anderen Knoten in V_1 verbinden. Für einen Knoten $w \in V_2$ definieren wir die externen und internen Kosten analog. Wir wollen natürlich versuchen, die gesamten externen Kosten zu senken, da dies gleichbedeutend damit ist, dass weniger Kanten zwischen den beiden Partitionen V_1 und V_2 verlaufen. Zu diesem Zweck definieren wir für jedes $v \in V$ die Differenz zwischen externen und internen Kosten durch

$$D_v := E_v - I_v \,.$$

Mit Hilfe von D_v können wir messen, ob sich das Vertauschen von zwei Knoten $u \in V_1$ und $w \in V_2$ lohnt. Wir bestimmten dazu zunächst den Gewinn einer Vertauschung.

Lemma 3.2.1 *Sei $D_u := E_u - I_u$ und $D_w := E_w - I_w$. Dann ist der Gesamtgewinn durch Vertauschen der Knoten $u \in V_1$ und $w \in V_2$ gegeben durch*

$$g_{u,w} = \begin{cases} D_u + D_w - 2\,, & \textit{falls } u \textit{ und } w \textit{ adjazent} \\ D_u + D_w\,, & \textit{sonst} \end{cases}.$$

Beweis:
1. Fall: u und w sind nicht adjazent.
Wenn wir u und w vertauschen, d.h., u ist nun in V_2 und w ist in V_1, werden alle externen Kanten von u zu internen Kanten, da u nun in derselben Teilmenge V_2 liegt wie alle Knoten, über die u zuvor noch über externe Kanten verbunden war. Umgekehrt werden alle internen Kanten von u zu externen Kanten, da diese nun zwischen u und V_1 verlaufen und daher zu den externen Kanten gezählt werden müssen. Insgesamt können wir also die E_u externen Kanten in interne Kanten umwandeln, allerdings werden dadurch auch die I_u internen Kanten zu externen Kanten. Dies hat einen Profit von $E_u - I_u$ zur Folge, also genau D_u. Für w erreichen wir analog einen Profit von D_w. In der Summe ergibt sich durch das Vertauschen von u und w also ein Profit von $D_u + D_w$.
2. Fall: u und w sind adjazent.
Für die meisten Kanten gilt das gleiche Prinzip wie oben, allerdings nicht für die Kante e_{uw}. Da beide Knoten gegeneinander ausgetauscht werden und daher weiterhin in zwei verschiedenen Teilmengen liegen, bleibt die Kante e_{uw} eine externe Kante. Deshalb werden zwar weiterhin alle internen Kanten von u zu externen Kanten, von den externen Kanten werden aber nur $E_u - 1$ Kanten zu internen Kanten. Der Profit berechnet sich dann zu $E_u - 1 - I_u$, genauso für w. Der gesamte Profit ist dann $(E_u - 1 - I_u) + (E_w - 1 - I_u) = D_u + D_w - 2$. □

Die Werte D_u und D_w können auch negativ sein, weshalb der Profit auch zum Verlust werden kann. Ist der Profit positiv, lohnt sich das Vertauschen, ansonsten nicht.

Beim KL-Algorithmus bestimmen wir zunächst D_v für alle $v \in V$ und wählen dann zwei Knoten $u_1 \in V_1$ und $w_1 \in V_2$, die durch Vertauschen den maximalen Gewinn ermöglichen, d.h.

$$g_1 := \max_{(u,w)\in V_1 \times V_2} D_u + D_w - 2a_{uw}\,.$$

Hierbei ist a_{uw} wieder der entsprechende Eintrag in der Adjazenzmatrix, der sicher stellt, dass der letzte Summand nur abgezogen wird, wenn u und w adjazent sind.

Im weiteren Verlauf unterstellen wir, dass u und w vertauscht wurden, des Weiteren werden wir diese beiden Knoten zunächst nicht mehr betrachten. Dementsprechend aktualisieren wir die Werte der D_v für alle $v \in V \setminus \{u_1, w_1\}$, wobei wir die folgende Formel benutzen können.

Lemma 3.2.2 *Es seien die Knoten $u \in V_1$ und $w \in V_2$ vertauscht worden. Dann sind die aktualisierten Werte für die Gewinne D_u und D_w gegeben durch*

$$D_u := D_u + 2a_{u,u_1} - 2a_{u,w_1} \text{ für } u \in V_1 \setminus \{u_1\}$$
$$D_w := D_w + 2a_{w,w_1} - 2a_{w,u_1} \text{ für } w \in V_2 \setminus \{w_1\}\,,$$

wenn die Knoten u und w nicht weiter beachtet werden.
Beweis: Falls ein $u \in V_1 \setminus \{u_1\}$ mit u_1 in der Ausgangspartition (V_1, V_2) verbunden

war, wird diese Kante nach dem Vertauschen von u_1 und w_1 zu einer externen Kante, wodurch sich D_u um 2 erhöht. Falls dieses u mit w_1 verbunden war, wird diese Kante nach dem Vertauschen von u_1 und w_1 zu einer internen Kante, weshalb D_u in diesem Fall um 2 sinkt. Das gleiche gilt für einen Knoten $w \in V_2 \setminus \{w_1\}$. □

Nun gehen wir analog zu oben vor, d.h., wir suchen zwei Knoten $u_2 \in V_1$ und $w_2 \in V_2$, die den maximalen Gewinn

$$g_2 := \max_{(u,w) \in V_1 \setminus \{u_1\} \times V_2 \setminus \{w_1\}} D_u + D_w - 2a_{uw}$$

generieren. Der Wert g_2 ist der zusätzliche Gewinn durch das Vertauschen von u_2 und w_2, wenn wir vorher bereits u_1 und w_1 vertauscht haben. Auch für die Knoten u_2 und w_2 werden wir annehmen, dass wir diese beiden Knoten vertauscht haben, und auch sie werden wir nicht mehr betrachten. Der nächste Schritt ist dann wieder das Aktualisieren der D_v für alle $v \in V \setminus \{u_1, u_2, w_1, w_2\}$, und wir suchen analog zu oben nach g_3.

Wir führen diese Prozedur solange durch, bis alle Knoten $v \in V$ abgearbeitet sind, insgesamt gibt es also $n/2$ Schritte. Aber nicht alle Vertauschungen sind am Ende auch gewinnbringend, wie das folgende Lemma zeigt.

Lemma 3.2.3 *Für die Gewinne g_i gilt*

$$\sum_{i=1}^{n/2} g_i = 0\,.$$

Beweis: Wenn wir alle Vertauschungen durchführen, verschieben wir alle Knoten aus V_1 nach V_2 und alle Knoten aus V_2 nach V_1. Die Kanten zwischen den beiden Teilmengen sind aber weiterhin dieselben, daher ist der Gesamtgewinn 0. □

Aus diesem Grund müssen wir den maximalen Profit

$$G := \max_{1 \le t \le n} \sum_{i=1}^{t} g_i$$

finden. Ist $G > 0$, vertauschen wir die Knoten $u_1, \ldots, u_t$ mit den Knoten $w_1, \ldots, w_t$. Danach können wir den Algorithmus für die modifizierte Partition erneut durchführen.

Sobald $G = 0$ gilt, befindet sich der Algorithmus in einem lokalen Optimum, was aber nicht heißen muss, dass die Partition dann optimal ist. In diesem Fall können wir zum Beispiel eine anderen Startpartition oder einen anderen Algorithmus wählen, um nach einer weiteren Verbesserung zu suchen.

Die Suche nach dem größtmöglichen Gewinn g_i in der i-ten Iteration lässt sich vereinfachen, wenn wir alle D_u und D_w absteigend nach der Größe sortieren.

Lemma 3.2.4 *Die Werte der D_u und D_w seien absteigend nach der Größe sortiert. Es sei ein Paar $D_{u'}, D_{w'}$ gefunden mit $D_{u'} + D_{w'} \le g^+$, wobei g^+ die in der aktuellen Iteration bisher größte gefundene Verbesserung ist. Dann gilt $D_{u''} + D_{w''} \le g^+$ für alle $D_{u''}, D_{w''}$ mit $D_{u''} \le D_{u'}$ und $D_{w''} \le D_{w'}$.*

Beweis: g^+ gehöre zu einem Knoten-Paar (u, w), dessen Vertauschung die bislang größte Verbesserung liefert. Da $D_{u'} + D_{w'} \leq g^+$, gilt auch $g_{u'w'} = D_{u'} + D_{w'} - 2a_{u'w'} \leq g^+$, u' und w' können also keinen größeren Gewinn erzielen. Mit $D_{u''} \leq D_{u'}$ und $D_{w''} \leq D_{w'}$ gilt dann auch $g_{u''w''} = D_{u''} + D_{w''} - 2a_{u''w''} \leq g^+$. □

Dieses Lemma können wir nun zu einer schnelleren Berechnung der größtmöglichen Verbesserung nutzen. Mit den beiden maximalen Werten für D_u und D_w beginnend probieren wir die anderen D_u und D_w der Reihe nach aus. Sobald ein Paar $D_{u'}, D_{w'}$ erreicht ist, deren Summe kleiner als die aktuell größte Verbesserung ist, können wir die Suche beenden.

Laufzeit des KL-Algorithmus und der Algorithmus von Fiduccia und Mattheyses

Die Laufzeit des KL-Algorithmus beträgt $O(|V|^3)$ bzw. mit Sortieren der D_u und D_w nur $O(|V|^2 log(|V|))$ [52]. Dutt [21] gelang mit einer Modifizierung des KL-Algorithmus eine Verbesserung der Laufzeit auf $O(|E| \max\{log(|V|), d_n\})$, wobei d_n der maximale Knotengrad im Graphen ist.

Noch schneller ist die Variante von Fiduccia und Mattheyses [26], deren Algorithmus eine Laufzeit von $O(|E|)$ hat und somit in linearer Zeit arbeitet. Die Verbesserung der Laufzeit gegenüber dem KL-Algorithmus ist bei ihrem Algorithmus darauf zurückzuführen, dass einzelne Knoten statt Paare verschoben werden, und mit Bucketsort [14] ein bestimmter Algorithmus zum Sortieren der Knoten benutzt wird.

Andere Varianten des KL-Algorithmus

Es gibt auch noch andere Varianten des KL-Algorithmus [42, 48, 58, 64, 76], bei denen etwa die Anzahl der Iterationen fixiert wird oder die Anzahl der zu tauschenden Knoten limitiert wird, da große Gewinne oft zu Beginn einer Iteration zu erwarten sind. Zudem müssen wir nur Knoten mit mindestens einer externen Kante betrachten, da nur diese Knoten zu einer Verringerung der Cutsize beitragen können.

Den KL-Algorithmus können wir auch auf eine K-Partition mit $K > 2$ anwenden. Da der KL-Algorithmus aber nur auf zwei Teilmengen gleichzeitig arbeitet, müssen wir den KL-Algorithmus solange für jeweils zwei Teilmengen durchführen, bis alle Teilmengen paarweise optimal sind. Insgesamt sind das $\binom{k}{2}$ Paare. Zudem sind eventuell mehrere Durchläufe nötig, da das Vertauschen von Knoten zwischen zwei Teilmengen die Optimalität gegenüber anderen Teilmengen beeinflussen kann.

3.2.2 Typische Verfahren der lokalen Suche

Der KL-Algorithmus wurde speziell für das Graph-Partitionierungsproblem konzipiert. Wir können aber auch allgemeine Verfahren der lokalen Suche anwenden, die in der Nachbarschaft nach einer besseren Lösung suchen. Die Nachbarschaft einer Lösung ist definiert durch alle Lösungen, die aus der vorgegebenen Lösung mittels definierter elementarer Operationen generiert werden können.

Generische und Evolutionäre Algorithmen

Generische und *Evolutionäre Algorithmen* richten sich nach dem Prinzip „Survival of the fittest“, das heutzutage vor allem mit Charles Darwin und seinem Buch „Über die Entstehung der Arten“ in Verbindung gebracht wird. Dahinter steht der Gedanke, dass eine Population von Individuen ihre Fitness dadurch erhöhen kann, dass sich Individuen mit überdurchschnittlicher Fitness häufiger vermehren als Individuen mit niedrigerer Fitness und ihre Merkmale an ihre Nachkommen weitergeben. Während generische Algorithmen vor allem die Weitergabe von Genen als Vorbild nutzen, basieren evolutionäre Algorithmen eher auf Mutation und Selektion innerhalb einer Population. Mittlerweile sind die Übergänge allerdings fließend, weshalb man beide Arten von Algorithmen oft zusammenfasst.

Wir wollen aus zwei Lösungen eine oder zwei neue Lösungen generieren, etwa durch Kreuzung. Aus welchen beiden Lösungen die neuen Lösungen generiert werden, entscheidet sich durch eine Fitnessfunktion, die jeder Lösung eine gewisse Fitness zuordnet. Je größer die Fitness einer Lösung, desto wahrscheinlicher wird sie ausgewählt. Als Wahrscheinlichkeit wird etwa der Anteil der eigenen Fitness an der gesamten Fitness aller betrachteten Lösungen gewählt. Nachdem eine neue Lösung geniert wurde, verändern wir diese durch einen Mutationsoperator, der die Lösung je nach vorgegebener Mutationswahrscheinlichkeit mehr oder weniger stark verändert. Wenn eine generierte Lösung für das Optimierungsproblem unzulässig ist, können wir diese entweder mit Strafkosten belegen und dann trotzdem verwenden, oder mit einem Reparaturalgorithmus in eine zulässige Lösung umwandeln.

Ein Vorteil dieser Algorithmen ist, dass sie auf viele Probleme anwendbar sind und keine Linearität oder Stetigkeit der Probleme voraussetzen.

Simulated Annealing

Beim *Simulated Annealing* (dt.: *Simuliertes Abkühlen*) wird die Nachbarschaft einer Lösung nach besseren Lösungen durchsucht, wobei teilweise aber auch temporäre Verschlechterungen im Zielfunktionswert in Kauf genommen werden. Während verbessernde Lösungen auf jeden Fall akzeptiert werden, werden schlechtere Lösungen nur mit einer bestimmten Wahrscheinlichkeit übernommen, die von der Verschlechterung des Zielfunktionswerts und einem Steuerungsparameter abhängt.

Nach einer festgelegten Anzahl von Schritten passen wir den Steuerungsparameter an, um die Wahrscheinlichkeit für die Annahme einer verschlechternden Lösung zu senken. Wenn innerhalb einer bestimmten Anzahl von Durchläufen keine verbessernde Lösung mehr gefunden wurde, terminiert das Verfahren.

Da wir auch verschlechternde Lösungen akzeptieren, bleibt der Algorithmus nicht so schnell in einem lokalen Optimum hängen. Das Verfahren baut auf einem physikalischen Modell auf, bei dem ein aufgewärmtes Metall durch gezieltes Abkühlen in eine bestimmte kristalline Form gebracht werden soll. Als Steuerungsparameter dient hierbei die Temperatur, die schrittweise gesenkt wird. Dieses Verfahren wurde unabhängig voneinander von Kirkpatrick, Gelatt und Vecchi [54] und Černý [12] entwickelt.

Tabu Search

Wie Simulated Annealing versucht auch *Tabu Search* das Festfahren in einem lokalen Optimum zu verhindern. Tabu Search verwendet dazu eine Art „Gedächtnis“, das den bisherigen Suchverlauf speichert. Um eine Lösung nicht mehrmals zu überprüfen, werden Tabu-Restriktionen genutzt, die bestimmte Eigenschaften einer Lösung verbieten.

Problematisch ist jedoch, dass verschiedene Lösungen womöglich ähnliche Eigenschaften haben, und wir dadurch eventuell verbessernde Lösungen verbieten, wenn wir Eigenschaften einer anderen Lösung verbieten wollen. Um das zu verhindern, können wir zum einen das Gedächtnis als Kurzzeit-Gedächtnis modellieren, sodass nur die letzten k Durchläufe gespeichert werden, zum anderen können wir durch Anspruchskriterien den Tabu-Status einer Lösung aufheben lassen. Solche Anspruchskriterien sind etwa „Anspruch durch eine Zielfunktion“, durch das ein Tabu aufgehoben wird, wenn dadurch eine verbessernde Lösung gewählt werden kann, oder „Anspruch durch Ermangelung“, wodurch die Lösung, die „am wenigsten“ tabu ist, gewählt wird, wenn alle Nachbarlösungen tabu sind. Tabu Search wurde von Glover [33, 34] entwickelt.

3.3 Multilevel-Ansätze

Im Vergleich zu anderen Verfahren ist die Kombination aus Bisektion und Lanczos' Algorithmus zur Berechnung des zweitgrößten Eigenwerts eher langsam, weshalb Barnard und Simon [4] versucht haben, die Struktur der Laplace-Matrix auszunutzen, um die Berechnung zu beschleunigen. Ihre Idee bei der Entwicklung der *Multilevel Spektralen Bisektion* war, dass ein kleinerer Graph zu einer kleineren Laplace-Matrix führt, was wiederum die Berechnung der Eigenwerte beschleunigt.

3.3.1 Multilevel Spektrale Bisektion

Die Multilevel-Variante der Spektralen Bisektion besteht aus drei Phasen: Zuerst wird der Graph G zu einem Graph G' zusammengefasst, dann wird der Fiedler-Vektor des zusammengefassten Graphen berechnet, der dann wiederum als Startlösung zur Approximation des Fiedler-Vektors des originalen Graphen benutzt wird.

Zusammenfassen des Graphen

Zunächst fassen wir alle Knoten, die viele gemeinsame Kanten haben, zu einem „Superknoten“ zusammen, diesen Prozess bezeichnen wir als *Coarsening* (dt.: *Vergröberung*). Dazu wählen wir eine maximale stabile Menge, d.h. eine stabile Menge, die nicht durch Hinzufügen von weiteren Knoten erweitert werden kann. Eine maximale stabile Menge können wir - im Gegensatz zu einer optimalen stabilen Menge - schnell durch ein Greedy-Verfahren bestimmen.

Lemma 3.3.1 *Sei $G = (V, E)$ ein Graph und $V' \subset V$. V' ist genau dann eine maximale stabile Menge, wenn alle Knoten aus $V \setminus V'$ adjazent zu V' sind.*
Beweis: Wir nehmen für den Beweis in die eine Richtung an, dass V' eine maximale stabile Menge ist, dann können wir keinen Knoten aus $V \setminus V'$ zu V' hinzufügen, ohne dass V' keine stabile Menge mehr ist. Gäbe es nun einen Knoten $v \in V \setminus V'$, der nicht zu V' adjazent ist, so sind auch alle Nachbarn von v nicht in V'. Dann könnten wir die

stabile Menge aber durch v erweitern, was im Widerspruch zur Annahme steht.
Zur Rückrichtung: Wenn alle Knoten in $V \setminus V'$ zu V' adjazent sind, haben all diese Knoten mindestens einen Nachbarn, der in der stabilen Menge liegt. Somit kann kein Knoten aus $V \setminus V'$ zur stabilen Menge hinzugefügt werden. □

Nachdem wir durch eine maximale stabile Menge die Knotenmenge verkleinert haben, bestimmen wir die Kanten von G'. Dazu errichten wir um jeden Superknoten $v'_i \in V' \cap V$ in G eine „Domain“, so etwas wie das „Einzugsgebiet des Knoten“. Wir vergrößern diese Domain, indem wir alle Nachbarn der Domain um den Knoten v'_i zur Domain hinzufügen. Wenn wir dann erneut versuchen, die Domain durch die Nachbarschaft zu erweitern, kann es passieren, dass sich die Domains von zwei verschiedenen Knoten $v'_i, v'_j \in V'$ schneiden. In diesem Fall fügen wir in dem Graph G' eine Kante zwischen v'_i und v'_j hinzu.

Nach der zweiten Erweiterung der Domains ist der komplette Graph G abgearbeitet, denn sonst hätte es schon zu Beginn einen Knoten gegeben, der noch in die stabile Menge gepasst hätte. Insbesondere sind zwei Knoten $v'_i, v'_j \in V'$ genau dann über eine Kante $e_{ij} \in E'$ verbunden, wenn es in dem Graph G einen Weg zwischen v'_i und v'_j gibt mit höchstens vier Knoten, denn bei einem fünften Knoten hätten wir den mittleren Knoten noch in die stabile Menge aufnehmen können. Außerdem ist G' genau dann zusammenhängend, wenn G zusammenhängend ist, denn wenn zwei Knoten in G über keinen Weg verbunden sind, sind sie es auch nicht in G' und vice versa.

Approximation des Fiedler-Vektors

Um den Fiedler-Vektor von G zu bestimmen, berechnen wir zunächst den Fiedler-Vektor f' von G', um aus diesem eine Approximation f des Fiedler-Vektors von G herzuleiten. Da G mehr Knoten als G' hat, hat der Fiedler-Vektor von G und somit auch f mehr Einträge als f'. Für die Knoten aus V' übertragen wir die Einträge von f' auf f, wobei zu beachtet ist, dass die Knoten eventuell unterschiedliche Indizes haben. Für einen Knoten aus $V \setminus V'$ wählen wir den Mittelwert der Einträge aller Knoten in G', zu denen dieser Knoten adjazent ist. Da V' eine maximale stabile Menge in V ist, hat jeder Knoten in $V \setminus V'$ mindestens einen Nachbarn, der in V' liegt, und somit haben alle Einträge von f einen Wert.

Anstelle des Algorithmus von Lanczos benutzen Barnard und Simon die *Rayleigh-Quotient-Iteration*, um den Fiedler-Vektor von G zu bestimmen. Der Vektor f dient dabei als Startvektor.

Bei großen Graphen bietet es sich an, das Coarsening des Graphen rekursiv durchzuführen, bis der Graph klein genug ist. Dann berechnen wir den Fiedler-Vektor des kleinsten Graphen, und passen diesen schrittweise an den jeweils nächstgrößeren Graph an, bis wir den Fiedler-Vektor des originalen Graphen bestimmt haben.

3.3.2 Multilevel-Partitionierung

Da die Multilevel Spektrale Bisektion zum einen gute Ergebnisse liefert und zum anderen schneller als die Rekursive Spektrale Bisektion ist, übertrugen Hendrickson und Leland [42] die Idee der Multilevel Spektralen Bisektion auf das allgemeine Graph-Partitionierungsproblem. Generell schneidet die Multilevel-Partitionierung in Kombi-

nation mit der Rekursiven Spektralen Bisektion und dem Kernighan-Lin-Algorithmus bei Tests sehr gut ab.

Coarsening des Graphen

Um den Graph zusammenzufassen, wählen wir diesmal ein maximales Matching. Zwei Knoten, die über eine Matching-Kante verbunden sind, fassen wir dann zu einem Knoten zusammen. Jeder neue Knoten wird mit allen anderen zusammengefassten Knoten, die mindestens einen Nachbarn von einem der beiden alten Knoten enthalten, über eine Kante verbunden. Im Gegensatz zur Version von Barnard und Simon bekommt diese Kante ein Gewicht, das sich aus der Summe der Gewichte der ursprünglichen Kanten ergibt.

Zur Bestimmung eines maximalen Matchings gibt es mehrere Methoden, eine recht einfache Methode ist die des *randomisierten Matchings*. Wir wählen zunächst zufällig einen Knoten aus V, dessen inzidente Kanten noch nicht im Matching sind. Wenn dieser Knoten einen Nachbarn hat, dessen indzidente Kanten auch noch nicht im Matching sind, fügen wir die Kante zwischen diesen beiden Knoten zum Matching hinzu. Dies wiederholen wir so lange bis das Matching nicht mehr erweitert werden kann.

Das randomisierte Matching ist ein recht einfaches und schnelles Verfahren, bei dem wir allerdings nicht die Kantengewichte beachten. Karypis und Kumar [48] entwickelten daher das Verfahren *Heavy Edge Matching* (*HEM*), welches unter den Kanten zu den noch freien Nachbarn die schwerste wählt. Möglicherweise werden dadurch aber trotzdem noch einige schwere Kanten ausgelassen.

Eine bessere Version des Heavy Edge Matchings ist das *Heaviest Edge Matching* von Gupta [36]. Hierbei sortieren wir zunächst alle Kanten absteigend nach dem Gewicht und fügen dann der Reihe nach die schwerstmögliche Kante zum Matching hinzu. Nur wenn zwei Kanten das gleiche Gewicht haben und beide noch zum Matching hinzugefügt werden können, entscheidet der Zufall darüber, welche Kante gewählt wird. Vor allem bei großen Unterschieden zwischen den Kantengewichten und für kleine Graphen, d.h., wenn das Coarsening weit fortgeschritten ist, ist dieses Verfahren recht gut.

Andere Verfahren zur Bestimmung eines maximalen Matchings sind beispielsweise die *modifizierte HEM*-Methode *HEM** von Karypis und Kumar [49] oder das *Heavy-Triangle Matching* (*HTM*) von Gupta [36].

Partitionierung des kleinsten Graphen

Wenn der Graph klein genug ist, oder wenn eine weitere Verkleinerung des Graphen keine große Veränderung mit sich bringt, partitionieren wir den Graph. Da der Graph recht klein ist und die meistens Verfahren daher sehr schnell arbeiten, können wir mehrere Verfahren ausprobieren und dann die beste Partition wählen. Auf der anderen Seite ist eine Partition in K balancierte Teilmengen nicht immer möglich, weshalb wir das Balance-Kriterium unter Umständen etwas auflockern sollten, um die Qualität der Partition zu verbessern.

Uncoarsening des Graphen

Nun müssen wir den Graph wieder „aufklappen“, diese Phase nennen wir *Uncoarsening* (dt.: *Entgröberung, Verfeinerung*). Jede Partition eines kleineren Graphen ist auch im größeren Graph eine Partition. Allerdings kann eine balancierte Partition durch das Aufklappen seine Balanciertheit verlieren, wenn in den Teilmengen unterschiedlich viele neue Knoten hinzukommen. Und auch die Cutsize kann sich deutlich verschlechtern, weshalb wir nach dem Hochprojizieren in die nächsthöhere Ebene eine lokale Verbesserungsmethode wie den KL-Algorithmus anwenden sollten.

3.4 Parallele Verfahren

Wir wollen Graph-Partitionierung benutzen, um die Matrix-Vektor-Berechnungen parallel ausführen zu können. Doch auch bei den dabei benutzten Verfahren können wir über eine Parallelisierung nachdenken, um die Laufzeit weiter zu senken. Bei den parallelen Verfahren handelt es sich meistens um parallele Versionen der oben vorgestellten Verfahren. Testergebnisse zeigen, dass parallele Verfahren ähnlich gut sein können wie konventionelle Verfahren, allerdings bei einer geringeren Laufzeit.

Während wir mehrere Versuche mit verschiedenen Verfahren oder unterschiedlichen Startpartitionen natürlich ohne Weiteres parallel durchführen können, lassen sich einzelne Verfahren nicht immer so leicht parallelisieren. Generell gelten P-vollständige Probleme, zu denen etwa Simulated Annealing und der KL-Algorithmus gehören [66], als schwer zu parallelisieren [35].

Auch Verfahren wie die Rekursive Spektrale Bisektion und die Multilevel-Verfahren können wir nicht ohne Weiteres parallelisieren, da Matchings oder stabile Mengen in stark zusammenhängenden Graphen bei Parallelisierung nicht so einfach bestimmt werden können. Dennoch gibt es einige Ansätze zu parallelen Verfahren.

3.4.1 Parallele Erweiterungen bekannter Verfahren

In [29] und den Referenzen darin werden einige parallele Versionen der Rekursiven Spektralen Bisektion bzw. der Multilevel Spektralen Bisektion präsentiert. Gilbert und Zmijevski [32] sowie Savage und Wloka [66] konstruierten jeweils parallele Algorithmen, die im Wesentlichen auf Bisektion und der Idee des KL-Algorithmus aufbauen. Gilbert und Zmijevski lassen zwei Leader-Prozessoren entscheiden, ob und wie viele Paare zwischen den zwei Teilmengen getauscht werden. Savage und Wloka entwickelten die *Mob-Heuristik*, bei der im Vorhinein eine maximal zu tauschende Anzahl an Knoten festgelegt wird. Auch Karypis und Kumar parallelisierten ihre zuvor entwickelten Multilevel-Verfahren teilweise mit Erfolg [47, 48, 49, 50].

3.4.2 Verfahren zur Balancierung

Es gibt einige parallele Verfahren, um eine Partition zu balancieren. Dabei sollen Teilmengen, die zu viele Knoten haben, Knoten an weniger große Teilmengen abgeben.

Özturan, de Cougny, Shephard und Flaherty [62] stellten einen Repartitionierungsalgorithmus vor, der auf einem Quotient-Graph arbeitet. Für eine Partition $\{V_i : 1 \leq i \leq k\}$ eines Graphen $G = (V, E)$ ist der Quotient-Graph definiert durch $G_q = (V_q, E_q)$, wobei

ein Knoten $v_i \in V_q$ eine Teilmenge V_i repräsentiert und eine Kante e_{ij} genau dann in E_q ist, wenn es eine Kante zwischen den Teilmengen V_i und V_j gibt. Jeder Knoten $v \in V_q$ entspricht einem Prozessor. Mit Hilfe einer minimalen Kantenfärbung des Graphen G_q wird dann bestimmt, wie die Auslastung zwischen den einzelnen Prozessoren ausgetauscht wird.

Das Verfahren von Ou und Ranka [61] arbeitet vor allem auf Partitionen, die durch Vergrößerung oder Verkleinerung des Graphen ihre Balanciertheit verloren haben. Die Partition soll so balanciert werden, dass möglichst wenige Knoten verschoben werden, dieses Problem wird als lineares Programm mit der Anzahl der Verschiebungen zwischen zwei Teilmengen als Variablen formuliert.

Walshaw, Cross und Everett [77] benutzen eine ähnliche Idee und verwenden dabei die Laplace-Matrix des Quotient-Graphen der Partition. Ihr Algorithmus wurde ursprünglich von Hu und Blake [44] entwickelt und gehört zu den Diffusionsalgorithmen. Auch Schloegel, Karypis und Kumar [67] beschäftigten sich mit dem Diffusions-Ansatz und verknüpften diese mit ihrer Multilevel-Methode.

3.5 Weiterführende Ideen und Ausblick

Selbst für sehr große Graphen liefern die Algorithmen in sehr kurzer Zeit gute Ergebnisse. Lohnt es sich also überhaupt noch nach Verbesserungen zu suchen? Der technische Fortschritt wird kaum stillstehen, weshalb eine Beschleunigung der Berechnungen allein schon durch bessere Hardware erfolgen kann. Verbesserungen lassen sich trotzdem wohl immer erreichen, wenn auch nur für spezielle Probleme oder bei einem bestimmten Vorwissen. Und auch bei einzelnen Schritten der Graph-Partitionierung, wie der Wahl des maximalen Matchings oder der Berechnung des Fiedler-Vektors, könnte nach Verbesserungen gesucht werden, da keines der Verfahren perfekt arbeitet.

Oft wissen wir nicht, wie weit eine Lösung von einer optimalen Partition entfernt ist, weshalb untere Schranken für die Cutsize ganz interessant wären. Ist die Cutsize der aktuell besten Lösung weit weg von einer guten unteren Schranke, lohnt sich der Aufwand zur Verbesserung der Lösung eher, als bei einer relativ guten Lösung. Genauso wie man die Lösungen immer weiter verbessern kann, ist es auch möglich und interessant, immer bessere untere Schranken zu finden. Diesem Problem wird im weiteren Verlauf der Arbeit speziell für Matrizen noch nachgegangen.

Haben wir bereits eine Partition in K Teilmengen gefunden, können wir das eventuell ausnutzen, wenn wir eine Partition in $K' > K$ Teilmengen finden wollen. Zudem stellt sich die Frage, ob sich eine Partition in K' Blöcke überhaupt lohnt bzw. wie viele Teilmengen sich maximal lohnen. Auch dieser Idee werden wir im Folgenden nachgehen, ebenfalls speziell für Matrizen.

4 Modelle zur Partitionierung von Matrizen

In diesem Kapitel werden wir sehen, wie wir eine Matrix als Graph darstellen können, um die Matrix dann durch den Graph partitionieren zu können.

Es gibt verschiedene Modelle, um eine Matrix als Graph darzustellen. Das Standardmodell ist das einfachste Modell, allerdings können wir es nur auf strukturell symmetrische Matrizen anwenden. Das bipartite Modell können wir auf jede Matrix anwenden, aber auch dieses Modell hat noch Schwächen im Hinblick auf die Zielfunktion, zumindest in Bezug auf die parallele Berechnung, die wir anstreben. Deshalb schauen wir uns auch noch das Hypergraph-Modell an, welches eine Matrix am besten repräsentiert. Unabhängig davon, welches Modell wir wählen, bezeichnen wir den Graph, der eine Matrix repräsentiert, als den *assoziierten Graph* der Matrix.

4.1 Das Standardmodell für strukturell symmetrische Matrizen

Das *Standardmodell* ist das einfachste Modell, um eine Matrix als Graph darzustellen. Es funktioniert allerdings ausschließlich für strukturell symmetrische Matrizen, weshalb es für Matrizen aus linearen Programmen, die meistens unsymmetrisch und rechteckig sind, oft nicht in Frage kommt. In einer *symmetrischen Matrix A* gilt $a_{ij} = a_{ji}$, sodass wir im Prinzip nur die Werte oberhalb der Diagonalen abspeichern müssen (inkl. der Diagonalen), weil wir dadurch auch die Einträge unterhalb der Diagonalen kennen. Für eine *strukturell symmetrische Matrix* gilt $a_{ij} \neq 0$ genau dann, wenn $a_{ji} \neq 0$. Eine strukturell symmetrische Matrix muss also nur hinsichtlich der Struktur der Nicht-Null-Einträge symmetrisch sein.

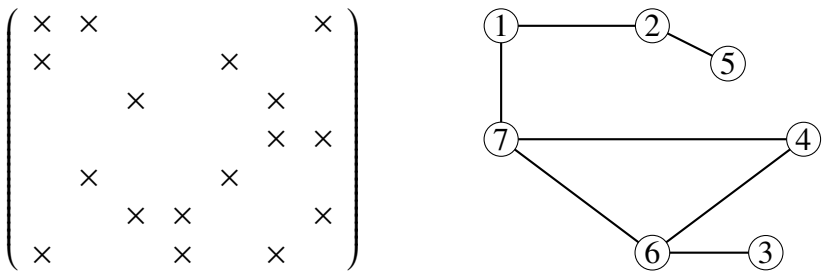

Abbildung 4.1: Strukturell symmetrische Matrix und der assoziierte Graph

Bei der Aufstellung des assoziierten Graphen betrachten wir nur die Einträge oberhalb der Diagonalen. Für jede Zeile fügen wir einen Knoten zum Graph hinzu, bei einer $n \times n$-Matrix besteht der Graph demnach aus n Knoten. Die Kanten des Graphen kommen nun zeilenweise hinzu. Zwei Knoten v_i und v_j sind genau dann über eine Kante e_{ij} verbunden, wenn $a_{ij} \neq 0$. Da die Matrix strukturell symmetrisch ist, gilt dann auch

$a_{ji} \neq 0$, weshalb die Kante nicht gerichtet sein muss. Abbildung 4.1 zeigt eine strukturell symmetrische Matrix und den assoziierten Graph der Matrix. Wir sehen, dass der Graph relativ klein ist, dies ist der größte Vorteil des Standardmodells.

Das Partitionieren der Matrix geschieht durch Partitionierung des Graphen. Für eine Blockdiagonalgestalt mit K Blöcken suchen wir im assoziierten Graph nach einer Partition mit K Teilmengen, sodass die Anzahl bzw. das Gewicht der Kanten zwischen den Teilmengen minimal ist. Aus dieser Partition leiten wir dann eine Partition der Matrix ab. Aus den Zeilen, deren Knoten mit Knoten aus anderen Teilmengen verbunden sind, entsteht der Rand, und die Teilmengen liefern die Blöcke. Den Rand bestimmen wir durch eine minimale Anzahl der Knoten, die mit Knoten aus anderen Teilmengen verbunden sind, sodass der restliche Graph nicht mehr zusammenhängend ist. Die entsprechenden Zeilen der Knoten kommen dann in den Zeilenrand. Da die übrigen Knoten nur mit Knoten innerhalb der eigenen Teilmenge verbunden sind, können wir aus diesen Zeilen durch Permutation die Blöcke formen.

Da wir die strukturelle Symmetrie der Matrix ausgenutzt haben, müssen wir simultan Zeilen und Spalten vertauschen, um die strukturelle Symmetrie beizubehalten. Daher hat die Arrowhead-Form der Matrix genau dann einen Zeilenrand, wenn sie einen Spaltenrand hat. Eine SB-Form ist daher für strukturell symmetrische Matrizen nicht möglich, lediglich eine Arrowhead-Form oder eine strikte Blockdiagonalform.

4.2 Das bipartite Modell

Das Standardmodell ist relativ simpel und war lange Zeit das einzige Modell zur Matrix-Partitionierung, allerdings kann es nur für strukturell symmetrische Matrizen benutzt werden. Wenn wir eine nicht strukturell symmetrische, aber quadratische Matrix haben, können wir hoffen, dass wir die Matrix in eine strukturell symmetrische Form permutieren können, leider ist das bei weitem nicht immer der Fall.

Statt einer Matrix A können wir die Matrix $A' = A + A^T$ betrachten, wegen

$$a'_{ij} = a_{ij} + a_{ji} = a_{ji} + a_{ij} = a'_{ji}$$

ist diese dann symmetrisch. Bei der Addition von A und A^T ist es allerdings möglich, dass wir eine Matrix erhalten, die weniger Null-Einträge als A hat und demzufolge nicht mehr so dünnbesetzt ist, wenn es viele Fälle mit $a_{ij} = 0$, aber $a_{ji} \neq 0$ gibt. Ist die Matrix A dagegen fast strukturell symmetrisch, ist die Anzahl der Nicht-Null-Einträge in der Matrix $A + A^T$ nicht viel größer als in der Matrix A. Zu diesem Zweck gibt es Verfahren, die eine Matrix A so weit wie möglich symmetrisieren, d.h., die die Anzahl der Paare (a_{ij}, a_{ji}) mit $a_{ij} \neq a_{ji}$ minimieren. Da es uns lediglich um strukturelle Symmetrie geht, können wir aus der Matrix A eine Matrix U erstellen, mit

$$u_{ij} = \begin{cases} 1, & a_{ij} \neq 0 \\ 0, & sonst \end{cases}$$

und versuchen, diese zu symmetrisieren. Das Symmetrisierungsproblem ist allerdings NP-vollständig, da bereits das Entscheidungsproblem, ob eine Matrix symmetrisierbar ist, NP-vollständig ist. Es gibt aber Heuristiken für dieses Problem [74].

Für rechteckige Matrizen haben wir dann aber immer noch kein graphisches Modell gefunden. In der Linearen Programmierung haben wir es jedoch häufig mit rechteckigen Matrizen zu tun, weshalb das Standardmodell auf keinen Fall ausreicht. Wir könnten zwar die Zeilen und Spalten einzeln betrachten, d.h. die Zeilen und Spalten getrennt voneinander permutieren und dann die Ergebnisse kombinieren, doch dies liefert oft schlechte Resultate.

Aus diesem Grund wurde von Hendrickson das *bipartite Modell* [39] als Erweiterung des Standardmodells entwickelt. Ein Graph $G = (V, E)$ heißt *bipartit*, wenn es eine Partition (V_1, V_2) gibt mit $V_1 \cap V_2 = \emptyset$ und $V_1 \cup V_2 = V$, bei der alle Kanten aus E zwischen den beiden Teilmengen V_1 und V_2 verlaufen. Die beiden Teilmengen müssen nicht gleich groß sein, wichtig ist nur, dass keine Kante zwei Knoten aus der gleichen Teilmenge verbindet.

Beim bipartiten Modell besteht der Graph einer Matrix $A \in \mathbb{R}^{m \times n}$ aus der Knotenmenge $V = R \cup C$ mit $R = \{r_1, \ldots, r_m\}$ und $C = \{c_1, \ldots, c_n\}$, wobei R die Zeilen und C die Spalten repräsentiert. Der Graph hat also $m + n$ Knoten und damit m mehr als bei einer strukturell symmetrischen $n \times n$-Matrix. Es existiert genau dann eine Kante zwischen zwei Knoten r_i und c_j, wenn der Eintrag a_{ij} nicht Null ist, d.h., es gilt $(r_i, c_j) \in E$ genau dann, wenn $r_i \in R, c_j \in C$ und $a_{ij} \neq 0$. Innerhalb der Teilmengen R und C gibt es keine Kanten, weshalb der Graph bipartit ist. In Abbildung 4.2 sehen wir eine rechteckige Matrix und die bipartite Graph-Repräsentation.

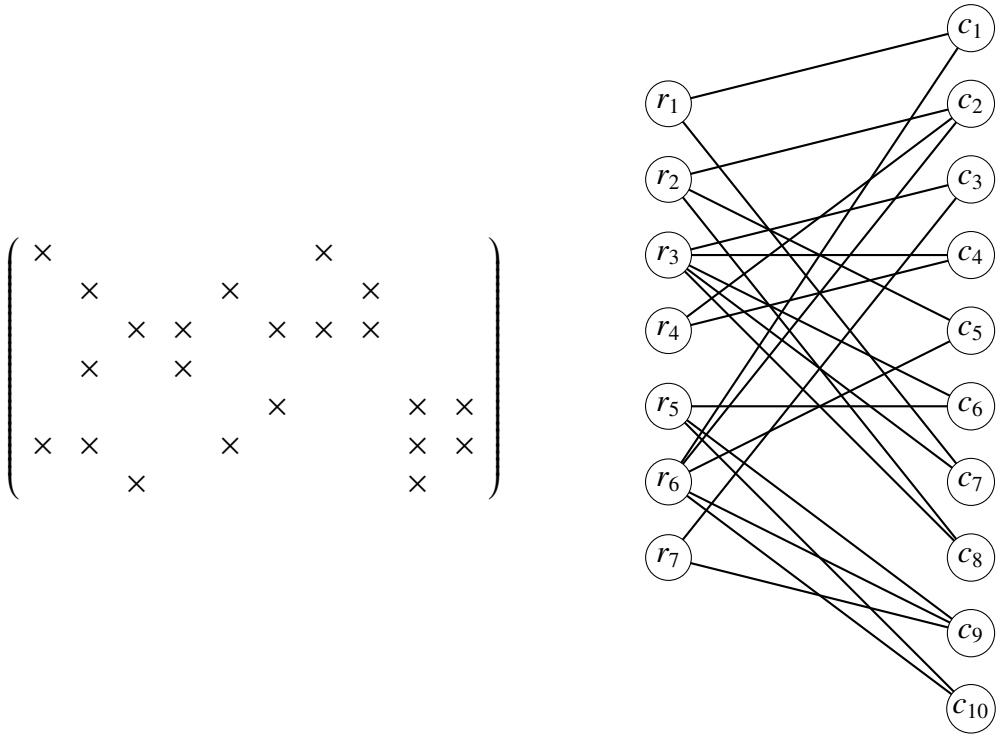

Abbildung 4.2: Rechteckige Matrix und der assoziierte Graph

Wie auch beim Standardmodell partitionieren wir zunächst den assoziierten Graph in eine GPES, durch die wir die Matrix dann in Arrowhead-Form bringen wollen. Aus den Knoten, die mit einer geschnittenen Kante inzidieren, d.h. die zu einem Knoten aus einer anderen Teilmenge adjazent sind, bestimmen wir den Rand der Arrowhead-Form, aus den restlichen Knoten können wir die Blöcke formen. Die Anzahl der Knoten, durch die wir den Rand erhalten, soll natürlich möglichst gering sein, wir suchen

daher nach einem minimalen Vertex Cover auf dem Graph, der durch die geschnittenen Kanten induziert wird. Zur Bestimmung eines Vertex Covers benutzen wir einen Greedy-Algorithmus [25].

Zunächst notieren wir für alle Knoten $v \in R \cup C$ die Anzahl der geschnittenen Kanten, zu denen der Knoten v inzidiert, und bezeichnen diesen Wert als *externe Kosten* E_v. Nun wählen wir den Knoten mit den größten externen Kosten und entfernen diesen aus dem Graph. Danach aktualisieren wir alle anderen externen Kosten, d.h., wir senken E_w um 1 für alle w, die zum entfernten Knoten v über eine externe Kante adjazent waren. Wenn v einem Zeilenknoten entspricht, d.h. $v \in R$, sind davon nur Knoten $w \in C$ betroffen, wenn wir einen Spaltenknoten entfernt haben, betrifft dies nur Zeilenknoten. Als nächstes entfernen wir wieder den Knoten mit den größten externen Kosten aus dem Graph und aktualisieren die externen Kosten der anderen Knoten. Diese Prozedur läuft so lange bis alle externen Kosten Null sind. Sollten mehrere Knoten die selben größten externen Kosten haben, ziehen wir Zeilenknoten gegenüber Spaltenknoten vor, weil sich der Zeilenrand durch das spätere Entfernen des Spaltenrands ohnehin vergrößern wird. Wenn mehr als ein Zeilenknoten in Frage kommt, wählen wir zufällig einen der Knoten.

Die Knoten, die aus dem Graph entfernt wurden, entsprechen den Zeilen und Spalten im Rand der permutierten Matrix. Haben wir einen Knoten r_i aus dem Graph entfernt, so permutieren wir die i-te Zeile der Matrix A in den Rand der Matrix A^π, haben wir einen Knoten c_j aus dem Graph entfernt, so permutieren wir die j-te Spalte von A in den Spaltenrand von A^π. Ohne diese Zeilen und Spalten können wir die verbleibende Matrix in eine strikte Blockdiagonalform permutieren, da der restliche Graph in K nicht verbundene Teilmengen zerfällt, von denen jede Teilmenge einen Block repräsentiert. Es ist durchaus möglich, dass einzelne Teilmengen leer sind, weshalb die Matrix auch weniger als K Blöcke haben kann. Insgesamt erhalten wir also eine Permutierung A^π einer Matrix A in Arrowhead-Form. Wenn wir eine gewisse Flexibilität bzgl. der Anzahl der Blöcke oder der Größe der Blöcke erreichen wollen, müssen wir Dummy-Knoten einfügen. Wie das geht, haben wir in Kapitel 2.1.1 beschrieben.

Ohne die Knoten, die den Rand der Matrix induzieren, zerfällt der Graph in K Zusammenhangskomponenten, die Matrix-Partitionierung entspricht also eigentlich einer GPVS. Stattdessen haben wir aber zunächst eine GPES erstellt und diese dann durch den Greedy-Algorithmus in eine GPVS umgewandelt, alternativ hätten wir auch direkt nach einer GPVS suchen können. Das Problem an dem indirekten Weg über die GPES ist, dass aus einer optimalen GPES nicht unbedingt eine optimale GPVS folgen muss. Das gilt erst recht für den von uns verwendeten Greedy-Algorithmus, denn wie bereits in Kapitel 3.1.3 angemerkt, finden Greedy-Algorithmen zwar schnell eine Lösung, können dagegen aber vor allem gegen Ende des Durchlaufs ziemlich schlecht werden. Auch hier kann es vorkommen, dass wir gegen Ende des Durchlaufs viele Knoten aus dem Graph entfernen, die mit vergleichsweise wenigen geschnittenen Kanten inzidieren.

Es ist schwer vorauszusagen, wie gut sich eine GPES eignet, um daraus eine GPVS zu bestimmen. Neben oberen Schranken für das Vertex Cover-Problem können wir versuchen, durch Gewichte bessere Resultate mit dem indirekten Weg über eine GPES zu erzielen. Pınar und Aykanat [63] beobachteten, dass nicht alle Kanten gleich wichtig sind, denn ein Knoten mit einem hohen Grad ist mit großer Wahrscheinlichkeit Teil des Vertex Separators, da es sehr wahrscheinlich ist, dass eine seiner inzidenten Kanten

zum Edge Separator gehört. Daher belegen sie jede Kante mit dem Gewicht

$$w_{ij} = \frac{1}{\max(d_i, d_j)},$$

wobei d_i und d_j die Grade der beiden zur Kante $e_{ij} = (r_i, c_j)$ inzidenten Knoten sind. Um Zeilenknoten gegenüber Spaltenknoten zu präferieren, betrachten sie nur noch den Zeilengrad und vereinfachen dieses Gewicht daher zu

$$w_{ij} = \frac{1}{d_i}.$$

4.3 Das Hypergraph-Modell

Das bipartite Modell gilt als relativ gutes Modell, um eine Matrix zu partitionieren. Wenn wir jedoch eine Matrix in Arrowhead-Form permutieren wollen, um sie dann durch Column Splitting in eine SB-Form zu bringen, unterstellen wir dadurch, dass eine gute Arrowhead-Form auch eine gute SB-Form induziert. Das muss aber nicht so sein. Vor allem, wenn der Spaltenrand vollbesetzt ist, kommen viele neue Nebenbedingungen in den Zeilenrand dazu. Zudem wollen wir die Matrix aus einem bestimmten Grund partitionieren, nämlich, um die Berechnung von $Ax = b$ auf mehrere Prozessoren aufzuteilen. Dass das bipartite Modell in diesem Fall nicht optimal ist, zeigt das folgende Beispiel.

Die Matrix

$$\begin{pmatrix} \times & \times & & & & \\ & \times & & & & \\ & & \times & \times & \times \\ & & \times & & \times \\ \times & & \times & & \times \\ \times & \times & & \times & \times \end{pmatrix}$$

ist in SB-Form, mit zwei Blöcken und zwei Zeilen im Zeilenrand. Dadurch können wir die Nebenbedingungen auf verschiedene Prozessoren verteilen, z. B., indem wir die ersten beiden Nebenbedingungen durch Prozessor 1 berechnen lassen, und die dritte und vierte Nebenbedingung sowie die Coupling Constraints durch Prozessor 2. Während die Nebenbedingungen des ersten und zweiten Blocks jeweils unabhängig voneinander berechnet werden können, benötigt Prozessor 2 für die Berechnung der Coupling Constraints die Werte der ersten beiden Variablen, die aber auf Prozessor 1 gespeichert sind. Nun muss also Prozessor 1 die Werte der ersten beiden Variablen an Prozessor 2 schicken.

Wie sieht das etwa für die erste Variable aus? Prozessor 1 schickt den Wert der Variable an Prozessor 2 und dann kann Prozessor 2 alle Nebenbedingungen mit der ersten Variable berechnen, vorausgesetzt, dass auch die Werte der anderen benötigten Variablen bekannt sind. In der Praxis müssen Prozessor 1 und Prozessor 2 wegen der ersten Variable also nur ein einziges Mal kommunizieren, unabhängig davon, in wie vielen Coupling Constraints die erste Variable vorkommt.

Wie aber wird das in der Zielfunktion des bipartiten Modells dargestellt? Hier wird jede Kante zwischen zwei Teilmengen gezählt, d.h., im vorliegenden Fall werden sowohl die Kante (r_5, c_1) als auch die Kante (r_6, c_1) zu den geschnittenen Kanten gezählt.

Es werden also alle Kanten zwischen c_1 und Randknoten aus R gezählt, obwohl es in der Praxis egal ist, wie viele Kanten zwischen c_1 und den Randknoten aus R verlaufen, da Prozessor 1 höchstens einmal mit Prozessor 2 kommunizieren muss. Wesentlich logischer wäre es daher, nur zu zählen, ob es eine Verbindung von c_1 in den Rand gibt und nicht durch wie viele Kanten diese Verbindung gestützt wird. Insgesamt sollte dann auch nicht die Anzahl der Kanten zwischen den Blöcken und dem Rand gezählt werden, sondern nur die Anzahl der Verbindungen. Dieser Umstand wurde u. a. von Hendrickson und Kolda [38, 40] selbst kritisiert und durch das Hypergraph-Modell von Çatalyürek et al. [2, 11] bereinigt.

4.3.1 Das Row-Net-Modell und das Column-Net-Modell

Statt durch einen Graph wollen wir die Matrix nun also durch einen Hypergraph darstellen.

Im *Row-Net-Modell* wird eine Matrix A durch einen Hypergraph $H_C = (U_C, N_R)$ repräsentiert, wobei die Spalten durch die Knoten U_C und die Zeilen durch die Netze N_R dargestellt werden. Ein Netz n_i enthält alle Spaltenknoten, für die der entsprechende Eintrag in der i-ten Zeile der Matrix A nicht Null ist. Ein Knoten u_j gehört also genau dann zu einem Netz n_i, wenn der Eintrag a_{ij} nicht Null ist.

Das *Column-Net-Modell* ist das duale Modell des Row-Net-Modells. Hier wird eine Matrix A durch einen Hypergraph $H_R = (U_R, N_C)$ dargestellt, wobei nun die Zeilen durch die Knoten U_R und die Spalten durch die Netze N_C repräsentiert werden. Ein Knoten u_i gehört genau dann zu einem Netz n_j, wenn der entsprechende Eintrag a_{ij} in der Matrix A nicht Null ist. Ein Netz n_j enthält demnach alle Zeilenknoten, für die der entsprechende Eintrag in der j-ten Spalte der Matrix A ungleich Null ist.

Sowohl beim Row-Net-Modell als auch beim Column-Net-Modell entspricht die Anzahl der Pins genau der Anzahl der Nicht-Null-Einträge in der Matrix A. In Abbildung 4.3 sehen wir die Row-Net- und die Column-Net-Hypergraph-Repräsentation der Matrix aus Abbildung 4.2.

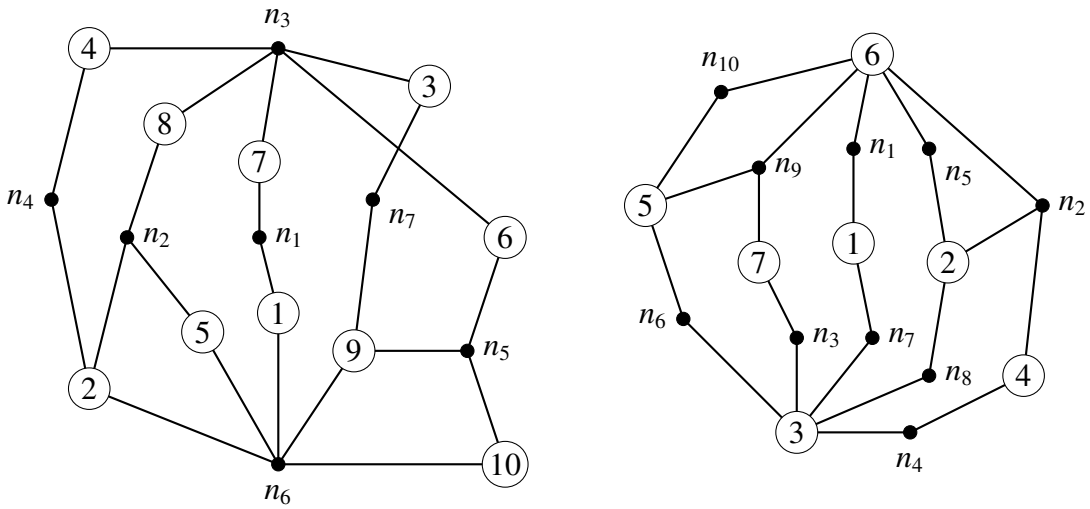

Abbildung 4.3: Row-Net- und Column-Net-Hypergraph-Repräsentation

Wir können das Row-Net-Modell und das Column-Net-Modell auch als Standardmodell einer symmetrischen Matrix formulieren, was wir nun für das Row-Net-Modell

demonstrieren werden. Sei $G^{NIG} = (V, E)$ der Net-Intersection-Graph des assoziierten Hypergraphen $H_R = (U, N)$ einer Matrix A. Die Knoten des NIG repräsentieren die Netze des Hypergraphen und somit die Zeilen der Matrix, und eine Kante e_{ij} existiert genau dann in G^{NIG}, wenn $Pins(n_i) \cap Pins(n_j) \neq \emptyset$. Da $Pins(n_i)$ genau die Spalten sind, die einen Nicht-Null-Eintrag in der i-ten Zeile haben, ist die Bedingung $Pins(n_i) \cap Pins(n_j) \neq \emptyset$ gleichbedeutend damit, dass die Zeilen i und j mindestens einen Nicht-Null-Eintrag in der gleichen Spalte haben.

Wir betrachten nun die Matrix $Z = AA^T$.

Lemma 4.3.1 *Sei $Z = AA^t$ und $G^{NIG} = (V, E)$ der NIG des Row-Net-Hypergraphen $H_R = (U, N)$ von A. Dann gilt $z_{ij} \neq 0$ genau dann, wenn $e_{ij} \in E$.*
Beweis: Im Hypergraph H_R werden die Spalten durch die Knoten U und die Zeilen durch die Netze N repräsentiert. In $G^{NIG} = (V, E)$ entsprechen die Knoten V den Netzen N des Hypergraphen, und zwei Knoten sind genau dann miteinander verbunden, wenn die zugehörigen Netze mindestens einen gemeinsamen Pin haben. Bezogen auf die Matrix bedeutet das, dass zwei Knoten v_i und v_j im NIG genau dann verbunden sind, wenn es eine Spalte gibt, die sowohl in der $i-$ten als auch in der j-ten Zeile keine Null stehen hat. Auf der anderen Seite ergibt sich der Eintrag z_{ij} durch Vektor-Multiplikation der i-ten und der j-ten Zeile von A, da $Z = AA^T$. Deshalb gilt $z_{ij} \neq 0$ ebenfalls genau dann, wenn es mindestens eine Spalte gibt, in der sowohl in der i-ten als auch in der j-ten Zeile ein Eintrag ungleich Null steht. Insgesamt folgt also, dass der Eintrag z_{ij} genau dann ungleich Null ist, wenn $e_{ij} \in E$. □

In der Adjazenzmatrix von G^{NIG} steht an der Stelle (i, j) genau dann eine 1, wenn es eine Kante $e_{ij} \in E$ gibt, ansonsten eine 0. Daher folgt aus dem Lemma, dass die Besetzungsstruktur der symmetrischen Matrix Z genau der Adjazenzmatrix von G^{NIG} entspricht, d.h. dass G^{NIG} äquivalent zum Graph des Standardmodells für Z ist.

Durch eine GPVS des NIG erhalten wir eine Partitionierung des Hypergraphen und dadurch eine Partitionierung der Matrix A. Da G^{NIG} äquivalent zum assoziierten Graph von AA^T ist, können wir das Matrix-Partitionierungsproblem daher als Partitionierungsproblem einer symmetrischen Matrix darstellen, wodurch wir einen zumeist wesentlich kleineren Graph erhalten, für den wir dann eine GPVS suchen können. Wie bereits in Kapitel 2.2.2 angemerkt, können wir die Balanciertheit der Matrix-Blöcke dann aber nicht mehr gewährleisten.

4.3.2 Von einer Hypergraph-Partition zur SB-Form der Matrix

Wir zeigen nun, dass wir durch die Hypergraph-Repräsentation tatsächlich auf eine SB-Form der Matrix schließen können. Wir betrachten nur das Row-Net-Modell, da die Column-Net-Hypergraph-Repräsentation einer Matrix A genau der Row-Net-Hypergraph-Repräsentation der Matrix A^T entspricht.

Angenommen, wir haben eine K-Partition $\{N_1, \ldots, N_K; N_S\}$ des assoziierten Hypergraphen bestimmt. Entfernen wir die Netze N_S, erhalten wir einen Hypergraph mit K disjunkten Knoten-Teilmengen $U_1, \ldots, U_K$. Wenn wir ein bestimmtes Balance-Kriterium berücksichtigt haben, sind diese Mengen hinsichtlich der Anzahl der Knoten balanciert.

Alle Netze aus einer Teilmenge N_k inzidieren nur mit Knoten aus U_k. Bezogen auf die Matrix bedeutet das, dass alle Zeilen, deren Netze zu einer Teilmenge N_k gehören, nur in den Spalten, die zu den Knoten aus der Teilmenge U_k gehören, einen Nicht-Null-Eintrag haben. Wenn wir nun die Spalten in der Reihenfolge der Teilmengen $U_1, \ldots, U_K$ und die Zeilen in der Reihenfolge der Teilmengen $N_1, \ldots, N_K$ anordnen, erhalten wir eine Matrix in strikter Blockdiagonalgestalt: Die Nicht-Null-Einträge der ersten $|U_1|$ Spalten sind alle in den ersten $|N_1|$ Zeilen, die Nicht-Null-Einträge der zweiten $|U_2|$ Spalten sind alle in den zweiten $|N_2|$ Zeilen usw.

Die Zeilen, die durch die Netze N_S repräsentiert werden, können wir als Zeilenrand der SB-Form auffassen. Da jedes Netz in N_S mindestens zwei Knoten aus verschiedenen Teilmengen verbindet, werden durch diese Zeilen mindestens zwei Spalten aus verschiedenen Blöcken verbunden, weshalb diese Zeilen auf jeden Fall in den Zeilenrand kommen müssen. Insgesamt permutieren wir also die j-te Spalte von A in den k-ten Block der permutierten Matrix A^π, wenn $u_j \in U_k$, und die i-te Zeile von A kommt in den k-ten Block von A^π, wenn $n_i \in N_k$, bzw. in den Rand, wenn $n_i \in N_S$.

Wir haben die Matrix dadurch in SB-Form gebracht. Die Zeilen-Dimension des k-ten Blocks entspricht der Anzahl der Netze in N_k, d.h. $m_k = |N_k|$, und die Spalten-Dimension des k-ten Blocks der Anzahl der Knoten in U_k, d.h. $n_k = |U_k|$. Wenn wir also bestimmte Blockgrößen erhalten wollen, können wir das über ein Balance-Kriterium für die Knoten und internen Netze sicher stellen. Außerdem entspricht die Anzahl der externen Netze genau der Anzahl der Coupling Constraints, was wir in dem folgenden Satz festhalten.

Satz 4.3.2 *Sei H der assoziierte Hypergraph einer Matrix A und $\{N_1, \ldots, N_K; N_S\}$ eine Partition des Hypergraphen. Dann hat die daraus hervorgehende SB-Form von A genau $|N_S|$ Coupling Constraints, d.h.*

$$|C| = |\mathcal{N}_S|\,,$$

wenn $|C|$ die Anzahl der Coupling Constraints bezeichnet.

Daraus folgt auch, dass wir die Cutsize-Definition

$$\xi(\Pi) = \sum_{n_j \in N_S} w_j$$

verwenden sollten, wenn wir das Gewicht der Coupling Constraints minimieren wollen. Die Cutsize-Definition

$$\xi(\Pi) = \sum_{n_j \in N_S} w_j(c_j - 1)$$

mit der Konnektivität c_j von Netz n_j beschreibt dagegen exakt das Kommunikationsaufkommen zwischen den Prozessoren, wie Çatalyürek und Aykanat [11] zeigten. Wir werden uns im Folgenden allerdings auf die Anzahl der Coupling Constraints beschränken und daher die erste Definition benutzen, insbesondere werden wir im folgenden Kapitel oft Satz 4.3.2 zitieren.

4.4 Zusammenfassung und Ausblick

Das einfachste Modell ist also das Standardmodell, welches jedoch nur auf strukturell symmetrische Matrizen angewendet werden kann. Eine Erweiterung für nicht strukturell symmetrische und rechteckige Matrizen ist das bipartite Modell, welches jedoch genau wie das Standardmodell im Bezug auf die Parallelisierung der Berechnung die falsche Zielfunktion hat. Deshalb wurde das Hypergraph-Modell entwickelt, durch das wir die Anzahl der Coupling Constraints bzw. das Kommunikationsaufkommen zwischen den Prozessoren genau abbilden können.

Den assoziierten Hypergraph einer Matrix A können wir wieder als Graph formulieren, etwa durch das CNG-Modell oder das NIG-Modell. Letzteres können wir sogar als Standardmodell der symmetrischen Matrix AA^T formulieren. Wichtiger wird für uns jedoch sein, dass wir durch eine Partition des Hypergraphen auf eine SB-Form der Matrix schließen können, wobei die Anzahl der Coupling Constraints genau der Anzahl der geschnittenen Netze entspricht. Dies werden wir im nächsten Kapitel ausnutzen, indem wir Abschätzungen für die Anzahl der geschnittenen Netze verwenden, um daraus untere Schranken für die Anzahl der Coupling Constraints herzuleiten.

5 Untere Schranken für die Anzahl der Coupling Constraints

Bei der Partitionierung einer Matrix wollen wir einerseits eine SB-Form mit möglichst vielen Blöcken erhalten, um die Berechnung auf möglichst viele Prozessoren aufteilen zu können, andererseits aber auch einen möglichst kleinen Rand, d.h. wenige Coupling Constraints, um so viele Gleichungen wie möglich parallel berechnen zu können. In diesem Kapitel wollen wir uns die Frage stellen, ob wir durch die Struktur einer Matrix abschätzen können, wie viele Coupling Constraints wir in Kauf nehmen müssen, wenn wir diese Matrix in eine SB-Form mit K Blöcken partitionieren wollen. Wir werden untere Schranken für die Anzahl der Coupling Constraints herleiten, die sowohl von Eigenschaften der Matrix, als auch von der Anzahl der Blöcke und der Balanciertheit abhängen.

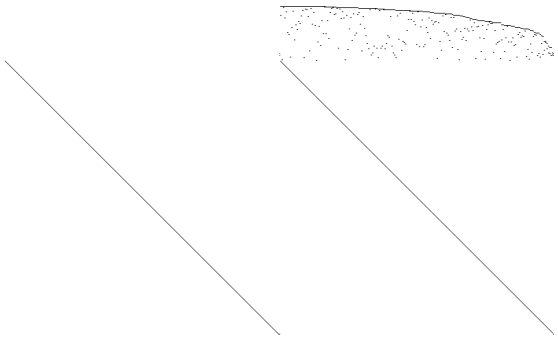

Abbildung 5.1: Eine sehr dünnbesetzte Matrix

Bei sehr dünnbesetzten Matrizen wie der Matrix in Abbildung 5.1 können wir schnell erkennen, ob wir diese gut partitionieren können. Zudem haben dünnbesetzte Matrizen von linearen Programmen nicht selten ein paar Zeilen mit sehr vielen Nicht-Null-Einträgen, die dann auf jeden Fall zu Coupling Constraints werden.

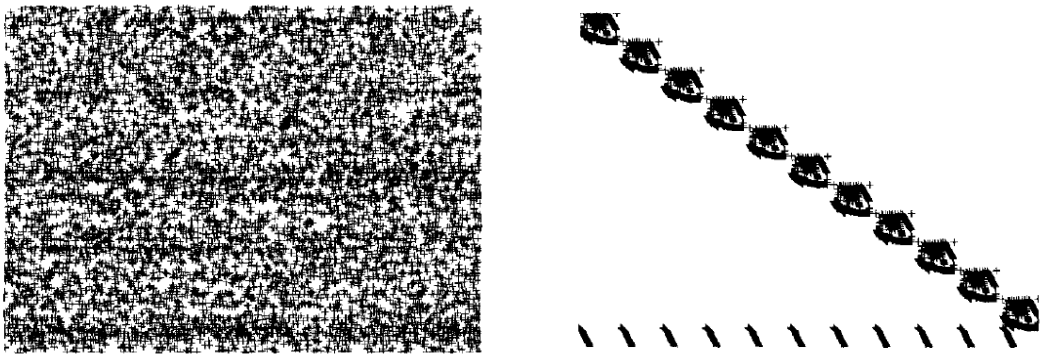

Abbildung 5.2: Randomisierte Permutation der PDS-Matrix, und die PDS-Matrix

Doch es gibt auch Matrizen, denen wir auf den ersten Blick nicht so einfach ansehen, wie gut wir sie partitionieren können. Die linke Matrix in Abbildung 5.2 wirkt zum Beispiel vollbesetzt, sodass eine Partitionierung der Matrix vermeintlich einen großen Rand haben wird. Allerdings ist diese Matrix nur eine randomisierte Permutation [25] der PDS-Matrix, die wir bereits in Kapitel 1.4 gesehen haben, und die von vornherein in einer SB-Form mit elf Blöcken und schmalem Rand ist. Dass wir die linke Matrix so gut partitionieren können, hätten wir der Matrix zuvor wohl kaum angesehen.

Um einen besseren Eindruck dafür zu bekommen, wie gut wir eine Matrix partitionieren können, wollen wir an Hand der Eigenschaften der Matrix untere Schranken für die Anzahl der Coupling Constraints herleiten. Wir konzentrieren uns dabei auf untere Schranken, da wir obere Schranken leicht durch das Lösen des Graph-Partitionierungsproblems mit einer Heuristik bekommen.

5.1 Untere Schranken für die Anzahl der Coupling Constraints

Im Folgenden werden wir zunächst einige untere Schranken für die Anzahl der Coupling Constraints angeben, deren Gültigkeit wir dann in Kapitel 5.2 beweisen werden. Die Schranken sollen sich aus den Eigenschaften der Matrix herleiten lassen, weshalb wir zunächst einige Notationen einführen. Wir interessieren uns nur für Partitionen mit $K \geq 2$ Blöcken, zudem gehen wir davon aus, dass jede Nebenbedingung mindestens zwei Variablen benutzt. Insbesondere gibt es also keine Variable, die unabhängig von allen anderen Variablen ist. Sonst könnten wir einer solchen Variable im Preprocessing[1] einen festen Wert zuordnen, sodass diese Variable im linearen Programm nicht weiter beachtet werden muss. Welchen Wert wir der Variable geben, hängt von der Zielfunktion ab und ob wir ein Minimierungs- oder ein Maximierungsproblem haben.

5.1.1 Notationen

Wir betrachten stets eine Matrix $A \in \mathbb{R}^{m \times n}$ eines linearen Programms mit m Nebenbedingungen und n Variablen. Diese Matrix wollen wir in eine SB-Form mit K Blöcken der Größen $n_k, 1 \leq k \leq K$, bringen, wobei wir die Menge der Coupling Constraints mit C bezeichnen und die Anzahl der Coupling Constraints mit $|C|$. Die Zeilen bzw. Nebenbedingungen von A bezeichnen wir mit $r_i, 1 \leq i \leq m$, und die Variablen mit $x_j, 1 \leq j \leq n$. Die Menge

$$rows(x_j) := \{r_i, 1 \leq i \leq m : a_{ij} \neq 0\}$$

beschreibt die Menge aller Nebenbedingungen, in denen die Variable x_j vorkommt. Mit

$$\Gamma(x_j) := \{x_i, 1 \leq i \leq n, i \neq j : rows(x_i) \cap rows(x_j) \neq \emptyset\}$$

bezeichnen wir die Menge aller Variablen, die durch mindestens eine Nebenbedingung mit der Variablen x_j verbunden sind, und mit $\gamma(x_j) := |\Gamma(x_j)|$ die Anzahl dieser

[1] Durch das Preprocessing wird ein Optimierungsproblem vor dem Lösen so stark wie möglich vereinfacht, ohne dabei den Zulässigkeitsbereich einzuschränken, z. B., indem Nebenbedingungen verschärft bzw. gestrichen werden, oder Variablen fixiert werden, wenn für sie nur ein Wert in Frage kommt. Oft kann hierbei die Ganzzahligkeit eines Problems ausgenutzt werden.

Variablen ohne doppelte Zählung. Wir unterscheiden also nicht, ob eine Variable nur in einer Nebenbedingung zusammen mit x_j auftaucht oder in mehreren Nebenbedingungen. Die Bestimmung der $\gamma(x_j)$ ist manchmal sehr einfach.

Satz 5.1.1 *Wenn es eine Nebenbedingung mit allen Variablen gibt, dann gilt*

$$\gamma(x_1) = \ldots \gamma(x_n) = n - 1\,.$$

Beweis: Durch diese Nebenbedingung ist jede Variable mit jeder anderen Nebenbedingung verbunden, somit folgt die Behauptung. □

Umgekehrt beschreibt die Menge

$$vars(r_i) := \{x_j, 1 \leq j \leq n : a_{ij} \neq 0\}$$

die Menge aller Variablen in der i-ten Nebenbedingung. Die Anzahl aller Nicht-Null-Einträge in der Zeile r_i bezeichnen wir mit

$$|r_i| := |\{a_{ij} \neq 0, 1 \leq j \leq n\}|\,.$$

Analog zu $\Gamma(x_j)$ definieren wir die Menge

$$\Delta(r_i) := \{r_j, 1 \leq j \leq m, j \neq i : vars(r_j) \cap vars(r_i) \neq \emptyset\}$$

als die Menge aller Nebenbedingungen, die mindestens eine Variable beinhalten, die auch in der i-ten Nebenbedingung vorkommt. Die Anzahl dieser Nebenbedingungen bezeichnen wir mit $\delta(r_i) := |\Delta(r_i)|$.

Wir werden jedem Paar zweier Variablen x_j und x_k ein Gewicht $\omega(x_j, x_k)$ zuteilen, das sich aus den gemeinsamen Nebenbedingungen ergibt. Dazu definieren wir zunächst für die $i-$te Nebenbedingung das Gewicht

$$\omega(r_i) := \frac{1}{\binom{|r_i|}{2} - \binom{\left\lfloor \frac{|r_i|}{K} \right\rfloor}{2}\left(K + K\left\lfloor \frac{|r_i|}{K} \right\rfloor - |r_i|\right) - \binom{\left\lceil \frac{|r_i|}{K} \right\rceil}{2}\left(|r_i| - K\left\lfloor \frac{|r_i|}{K} \right\rfloor\right)}\,, \tag{5.1.1}$$

welches sowohl von der Anzahl der Variablen in der i-ten Nebenbedingung als auch der Anzahl der vorgegebenen Blöcke K abhängt. Je mehr Variablen zu einer Nebenbedingung gehören, desto kleiner ist das Gewicht der Nebenbedingung, wie sich auch durch die Werte in Tabelle 5.1 erahnen lässt.

$\lvert r_i\rvert$	$K = 2$	$K = 3$	$K = 4$	$K = 5$
2	1	1	1	1
3	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
4	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{6}$
5	$\frac{1}{6}$	$\frac{1}{8}$	$\frac{1}{9}$	$\frac{1}{10}$
6	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{13}$	$\frac{1}{14}$
7	$\frac{1}{12}$	$\frac{1}{16}$	$\frac{1}{18}$	$\frac{1}{19}$
8	$\frac{1}{16}$	$\frac{1}{21}$	$\frac{1}{24}$	$\frac{1}{25}$
9	$\frac{1}{20}$	$\frac{1}{27}$	$\frac{1}{30}$	$\frac{1}{32}$

Tabelle 5.1: Gewichte $\omega(r_i)$ für verschiedene $|r_i|$ und K

Das Gewicht eines Variablen-Paars (x_j, x_k) ist dann die Summe aller Gewichte der Nebenbedingungen, in denen sowohl x_j als auch x_k auftauchen, d.h.

$$\omega(x_j, x_k) := \sum_{r_i \in rows(x_j) \cap rows(x_k)} \omega(r_i)\,. \tag{5.1.2}$$

Variablen-Paare, die in keiner gemeinsamen Nebenbedingung auftauchen, erhalten ein Gewicht von Null, im Folgenden werden wir solche Paare allerdings ignorieren. Woher diese Gewichte kommen, werden wir in Kapitel 5.2 sehen.

In einer Nebenbedingung mit $|r_i|$ Variablen gibt es $\binom{|r_i|}{2}$ Variablen-Paare. Da jedes dieser Paare durch die i-te Nebenbedingung das Gewicht $\omega(r_i)$ erhält, beträgt das Gesamtgewicht der Matrix

$$\mathcal{W} := \sum_{i=1}^{m} \omega(r_i)\binom{|r_i|}{2}.$$

Wir werden uns oft für die Anordnung der Gewichte interessieren, insbesondere für die kleinsten Gewichte. Aus diesem Grund werden wir mit $\omega_1 \le \omega_2 \le \ldots$ die Gewichte in aufsteigender Reihenfolge bezeichnen, analog dazu definieren wir $\gamma_1 \le \ldots \le \gamma_n$ und $\delta_1 \le \ldots \le \delta_m$. Mit $n_1 \ge \ldots \ge n_K$ bezeichnen wir die Anzahl der Variablen in den Blöcken in absteigender Reihenfolge.

Beispiel 5.1.2 Die Bedeutung der Symbole wollen wir an einem Beispiel veranschaulichen. Wir betrachten wieder die Matrix

$$\begin{pmatrix}
\times & & & & & \times & & & \\
 & \times & & & \times & & \times & & \\
 & & \times & \times & & \times & \times & \times & \\
 & \times & & \times & & & & & \\
 & & & & & \times & & \times & \times \\
\times & \times & & & \times & & & \times & \times \\
 & & \times & & & & & \times &
\end{pmatrix}$$

eines linearen Programms mit zehn Variablen und sieben Nebenbedingungen, die wir in eine SB-Form mit $K = 2$ Blöcken bringen wollen. Die Variable x_1 ist zusammen mit der Variable x_7 in der ersten Nebenbedingung, außerdem zusammen mit den Variablen x_2, x_5, x_9 und x_{10} in der sechsten Nebenbedingung, demzufolge ist

$$rows(x_1) = \{r_1, r_6\}$$

sowie

$$\Gamma(x_1) = \{x_2, x_5, x_7, x_9, x_{10}\} \text{ und } \gamma(x_1) = 5\,.$$

Die sechste Nebenbedingung benötigt die Variablen x_1, x_2, x_5, x_9 und x_{10}, also ist

$$vars(r_6) = \{x_1, x_2, x_5, x_9, x_{10}\} \text{ und } |r_6| = 5\,.$$

Die Variablen aus $vars(r_6)$ tauchen auch in der ersten, der zweiten, der vierten, der fünften und der siebten Nebenbedingungen auf, d.h.

$$\Delta(r_6) = \{r_1, r_2, r_4, r_5, r_7\} \text{ und } \delta(r_6) = 5\,.$$

Um die Gewichte der Variablen-Paare zu bestimmen, ermitteln wir zunächst die Gewichte der sieben Nebenbedingungen. Diese ergeben sich mit der Formel 5.1.1 zu

$$\omega(r_1) = 1,\ \omega(r_2) = \frac{1}{2},\ \omega(r_3) = \frac{1}{6},\ \omega(r_4) = 1,\ \omega(r_5) = \frac{1}{2},\ \omega(r_6) = \frac{1}{6},\ \omega(r_7) = 1\,.$$

Mit der Formel 5.1.2 können wir nun die Gewichte der Variablen-Paare bestimmen. Für die Variablen x_2 und x_5 erhalten wir etwa

$$\omega(x_2, x_5) = \omega(r_2) + \omega(r_6) = \frac{1}{2} + \frac{1}{6} = \frac{2}{3}$$

und für die Variablen x_2 und x_4

$$\omega(x_2, x_4) = \omega(r_4) = 1\,.$$

Obwohl die Variablen x_2 und x_5 in zwei Nebenbedingungen zusammen auftauchen, erhält dieses Paar ein geringeres Gewicht als das Paar (x_2, x_4), welches nur eine gemeinsame Nebenbedingung hat. Das liegt daran, dass in den Nebenbedingungen mit x_2 und x_5 auch noch andere Variablen vorkommen und die Gewichte der Nebenbedingungen daher kleiner sind.

5.1.2 Untere Schranken

Mit den soeben eingeführten Bezeichnungen wollen wir nun einige untere Schranken für die Anzahl der Coupling Constraints angeben, die wir dann in Kapitel 5.2 beweisen werden. Da wir davon ausgehen, dass jede Nebenbedingung mindestens zwei Variablen benutzt, gilt $\gamma(x_j) > 0$ für alle $j = 1, \ldots, n$. Zudem betrachten wir nur Matrizen, die sich nicht in strikte Blockdiagonalgestalt permutieren lassen, da wir viele Aussagen nur dann beweisen können.

Wir starten mit einer intuitiv relativ trivialen Aussage für die Partitionierung einer Matrix in SB-Form.

Satz 5.1.3 *Seien $A \in \mathbb{R}^{m \times n}$ eine Matrix, die in K Blöcke partitioniert werden soll, und $\omega_1 \leq \omega_2 \leq \ldots$ die Gewichte der Variablen-Paare in aufsteigender Reihenfolge. Wenn eine SB-Form der Matrix A mindestens einen Coupling Constraint hat, dann existiert ein $\tau \geq 1$ mit*

$$|C| \geq \left\lceil \sum_{i=1}^{\tau} \omega_i \right\rceil . \tag{5.1.3}$$

Wenn $\omega_1 \leq 1$ ist, bekommen wir mit $\tau = 1$ die Abschätzung, dass eine SB-Form der Matrix A mindestens einen Coupling Constraint hat. Doch auch, wenn die Gewichte $\omega(r_i)$ relativ klein sind und zudem höchstens 1 betragen, können die Gewichte ω_i größer als 1 sein, da sie als Summe mehrerer Gewichte definiert sind. Die Aussage von Satz 5.1.3 ist also nicht trivial.

Wenn wir Abschätzung 5.1.3 beweisen können, sollten wir natürlich versuchen, τ so maximal wie möglich zu wählen, um $|C|$ so gut wie möglich anzunähern. Auf der anderen Seite ist es aber auch gut, wenn das maximale τ so klein wie möglich ist, da dies auf einen kleinen Rand hinweist. Wir werden nun einige zulässige Werte für τ

angeben, insbesondere für bestimmte Fälle, in denen die Matrix eine gewisse Struktur hat. Wenn wir darüber hinaus einen Wert finden, für den Abschätzung 5.1.3 ohne Einschränkung gilt, haben wir dadurch auch Satz 5.1.3 bewiesen. Neben zulässigen Werten für τ werden wir aber auch einige direkte Abschätzungen für $|C|$ herleiten.

Für die erste Abschätzung müssen wir zuvor festlegen, wie viele Variablen ein Block höchstens enthalten darf. Außerdem muss jede Variable mit fast der Hälfte der anderen Variablen über eine Nebenbedingung verbunden sein.

Theorem 5.1.4 *Sei $\gamma_1 > \frac{n}{2} - 1$ und n_1 die maximale Anzahl der Variablen pro Block. Dann gilt Abschätzung 5.1.3 für*

$$\tau = \left\lceil \frac{(2\gamma_1 - n + 2)(n - n_1)}{2} \right\rceil .$$

Die Voraussetzung dieses Resultats ist eine notwendige Bedingung dafür, dass wir durch Abschätzung 5.1.3 keine triviale untere Schranke bekommen. Wenn γ_1 nicht groß genug ist, ist τ negativ, und wir erhalten die Abschätzung $|C| \geq 0$, da die Summe in Abschätzung 5.1.3 dann nach Definition leer ist. Wie wir später im Beweis sehen werden, ist diese Abschätzung zudem nur für $K = 2$ wirklich sinnvoll, da sie für größere K zu grob sein kann.

Das folgende Ergebnis kann ähnlich hergeleitet werden, wie die Aussage von Theorem 5.1.4, liefert allerdings für $K \geq 3$ in der Regel eine bessere Abschätzung.

Theorem 5.1.5 *Sei $K \geq 3$, $\gamma(x_j) > 0$ für alle $j = 1, \ldots, n$ und*

$$n_2(2\gamma_1 + 1) + \sum_{k=3}^{K} n_k(\gamma_k + k) > (n-1)(n - n_1) .$$

Dann gilt Abschätzung 5.1.3 für

$$\tau = \left\lceil \frac{n_2(2\gamma_1 + 1) - (n-1)(n-n_1)}{2} + \frac{1}{2} \sum_{k=3}^{K} n_k(\gamma_k + k) \right\rceil .$$

Die Voraussetzung für eine nicht-triviale Abschätzung ist etwas komplizierter, als in Theorem 5.1.4 und auch nicht so anschaulich. Wenn alle Blöcke gleich viele Variablen beinhalten sollen, können wir das Ergebnis allerdings ein bisschen anders formulieren.

Korollar 5.1.6 *Sei $K \geq 3$, $n_k = \frac{n}{K}$ für alle $k = 1, \ldots, K$, und es gelte $\gamma(x_j) > 0$ für alle $j = 1, \ldots, n$ sowie*

$$2\gamma_1 + \sum_{k=3}^{K} \gamma_k > n(K-1) - \frac{K(K+3)}{2} + 3 .$$

Dann gilt Abschätzung 5.1.3 für

$$\tau = \left\lceil \frac{n}{2K} \left(2\gamma_1 - n(K-1) + \frac{K(K+3)}{2} - 3 + \sum_{k=3}^{K} \gamma_k \right) \right\rceil .$$

Da $\gamma_2 \geq \gamma_1$, ist die Voraussetzung des Theorems insbesondere dann erfüllt, wenn

$$\sum_{k=1}^{K} \gamma_k > n(K-1) - \frac{K(K+3)}{2} + 3\,.$$

Wenn alle Blöcke gleich viele Variablen haben, lassen sich die Aussagen generell viel leichter zeigen bzw. kompakter darstellen. Bei dem folgenden Ergebnis setzen wir erneut voraus, dass alle Blöcke gleich viele Variablen haben.

Theorem 5.1.7 *Sei $n_k = \frac{n}{K}$, und es gelte*

$$\frac{\mathcal{W}^2}{\sum_{1\leq j<l\leq n} \omega(x_j, x_l)^2} > \frac{n-K}{n+K-2}\binom{n}{2}.$$

Dann gilt

$$|C| \geq \left\lceil \frac{2n\mathcal{W}(K-1) - n\sqrt{(K-1)(n-K)(n(n-1)\sum_{1\leq j<l\leq n} \omega(x_j, x_l)^2 - 2\mathcal{W}^2)}}{2K(n-1)} \right\rceil .$$

Zwar haben wir nun erstmals eine direkte Abschätzung für $|C|$ ohne Satz 5.1.3 gefunden, allerdings ist die Voraussetzung nicht sehr anschaulich. Wenn es viele Gewichte gibt, die größer als 1 sind, wird der Term auf der linken Seite der Voraussetzung relativ klein, das gleiche gilt aber auch, wenn es viele sehr kleine Gewichte gibt. Daher eignet sich Theorem 5.1.7 vor allem für Matrizen, in denen die Gewichte der Variablen-Paare nah bei 1 liegen.

Die folgende Aussage ist eine vereinfachte Version von Theorem 5.1.7 mit einer anschaulicheren Voraussetzung, aber auch einer indirekten Abschätzung über Satz 5.1.3.

Theorem 5.1.8 *Es sei $n_k = \frac{n}{K}$ für alle $k = 1, \ldots, K$, und es gelte*

$$\sum_{j=1}^{n} \gamma(x_j) > \frac{n(n-1)(n-K)}{n+K-2}\,.$$

Dann gilt Abschätzung 5.1.3 für

$$\tau = \left\lceil \frac{n(K-1)\sum_{j=1}^{n} \gamma(x_j) - n\sqrt{(K-1)(n-K)\frac{1}{2}\sum_{j=1}^{n} \gamma(x_j)(n^2 - n - \sum_{j=1}^{n} \gamma(x_j))}}{2K(n-1)} \right\rceil .$$

Da mit der Cauchy-Schwarz-Ungleichung[2]

$$\frac{1}{2}\sum_{j=1}^{n} \gamma(x_j) = \frac{1}{2}\sum_{j=1}^{n} \gamma(x_j)\frac{\mathcal{W}^2}{\mathcal{W}^2} \geq \frac{\frac{1}{2}\sum_{j=1}^{n} \gamma(x_j)\mathcal{W}^2}{\frac{1}{2}\sum_{j=1}^{n} \gamma(x_j)\sum_{1\leq j<l\leq n} \omega(x_j, x_l)^2} = \frac{\mathcal{W}^2}{\sum_{1\leq j<l\leq n} \omega(x_j, x_l)^2}$$

folgt, ist die Voraussetzung von Theorem 5.1.8 insbesondere dann erfüllt, wenn die Voraussetzung von Theorem 5.1.7 erfüllt ist. Die Bedingung von Theorem 5.1.8 gilt

[2] Satz A.2.11

zudem genau dann, wenn jede Variable im Schnitt mit mindestens $\frac{(n-1)(n-K)}{n+K-2}$ anderen Variablen verbunden ist. Nach Satz 5.1.1 kann daher schon eine einzige Nebenbedingung mit allen Variablen ausreichen, damit die Voraussetzung erfüllt ist.

Generell können wir für Matrizen, in denen alle Variablen untereinander verbunden sind, spezielle Ergebnisse herleiten.

Theorem 5.1.9 *Es gelte $\gamma(x_j) = n - 1$ für alle $j = 1, \ldots, n$. Dann gilt Abschätzung 5.1.3 für*

$$\tau = \binom{n}{2} - \sum_{k=1}^{K} \binom{n_k}{2}.$$

Hier brauchen wir keine Gauß-Klammern zu setzen, da τ immer ganzzahlig ist. Für eine Partition mit gleich vielen Variablen pro Block können wir den Term vereinfachen.

Korollar 5.1.10 *Es sei $n_k = \frac{n}{K}$ für alle $k = 1, \ldots, K$ und es gelte $\gamma(x_j) = n - 1$ für alle $j = 1, \ldots, n$. Dann gilt Abschätzung 5.1.3 für*

$$\tau = \frac{n^2}{2}\left(1 - \frac{1}{K}\right).$$

Und auch für eine Partition mit fast gleich vielen Variablen pro Block erhalten wir eine Darstellung ohne Summe.

Korollar 5.1.11 *Es sei $n_k \in \{\lfloor n/K \rfloor, \lceil n/K \rceil\}$ für alle $k = 1, \ldots, K$, und es gelte $\gamma(x_j) = n - 1$ für alle $j = 1, \ldots, n$. Dann gilt Abschätzung 5.1.3 für*

$$\tau = \binom{n}{2} - \binom{\left\lfloor \frac{n}{K} \right\rfloor}{2}\left(K + K\left\lfloor \frac{n}{K} \right\rfloor - n\right) - \binom{\left\lceil \frac{n}{K} \right\rceil}{2}\left(n - K\left\lfloor \frac{n}{K} \right\rfloor\right).$$

Dieses Ergebnis ähnelt sehr dem Nenner in den Gewichten $\omega(r_i)$ für die Nebenbedingungen. Wie wir sehen werden, ist das kein Zufall.

Die bisherigen Resultate hingen fast ausschließlich von der Anzahl der Variablen pro Block ab, nun folgen einige Ergebnisse, die unabhängig von den Blockgrößen sind.

Für das folgende Ergebnis setzen wir wieder voraus, dass alle Variablen untereinander verbunden sind.

Theorem 5.1.12 *Es gelte $\gamma(x_j) = n - 1$ für alle $j = 1, \ldots, n$ und $n \geq 3$. Dann gilt*

$$|C| \geq \lceil \omega_1 + \omega_2 \rceil .$$

Es ist fraglich, ob dieses Resultat gute Abschätzungen liefern kann, da wir nur die beiden kleinsten Gewichte zur Abschätzung benutzen dürfen. Zudem haben wir durch Theorem 5.1.9 schon eine Abschätzung gefunden, die wesentlich bessere Ergebnisse liefern dürfte.

Ein ähnliches Resultat können wir erzielen, wenn es zwar eine Variable gibt, die in allen Nebenbedingungen auftaucht, es aber auch eine Variable gibt, die nicht mit jeder anderen Variable verbunden ist. Dieses Ergebnis ist allerdings noch schlechter, als das von Theorem 5.1.12.

Theorem 5.1.13 *Es gelte $\gamma(x_j) = n - 1$ für mindestens eine Variable x_j, und $\gamma(x_l) \leq n - 2$ für mindestens eine weitere Variable x_l. Dann gilt*

$$|C| \geq \lceil \omega_1 \rceil \, .$$

Das folgende Ergebnis können wir schon eher gebrauchen, wir setzen dazu wieder voraus, dass alle Variablen miteinander verbunden sind.

Theorem 5.1.14 Es gelte $\gamma(x_j) = n - 1$ für alle $j = 1, \ldots, n$. Dann gilt Abschätzung 5.1.3 für

$$\tau = \left\lceil \frac{n-3}{2} \right\rceil .$$

Ein ähnliches Ergebnis bekommen wir direkt für $|C|$, wenn alle Nebenbedingungen über Variablen miteinander verbunden sind, d.h., wenn jedes Paar von Nebenbedingungen mindestens eine gemeinsame Variable teilt. Da es aber oft Nebenbedingungen mit nur wenigen Variablen gibt, ist diese Voraussetzung wohl eher selten erfüllt. Immerhin erhalten wir dann aber eine direkte Abschätzung für $|C|$.

Theorem 5.1.15 *Es gelte $\delta(r_i) = m - 1$ für alle $i = 1, \ldots, m$. Dann gilt*

$$|C| \geq \left\lceil \frac{m-3}{2} \right\rceil .$$

Einen sehr einfachen Wert für τ erhalten wir, wenn jedes Paar von nicht verbundenen Variablen zumindest in der Summe mit vielen anderen Variablen verbunden ist.

Theorem 5.1.16 *Es gelte $\gamma(x_j) + \gamma(x_l) \geq n - 1$ für alle Variablen x_j und x_l mit $\Gamma(x_j) \cap \Gamma(x_l) = \emptyset$. Dann gilt Abschätzung 5.1.3 für*

$$\tau = \gamma_1 \, .$$

Die Bedingung bedeutet nicht, dass die beiden Variablen x_j und x_l insgesamt mit allen anderen Variablen verbunden sein müssen, d.h., dass $\Gamma(x_j) \cup \Gamma(x_l) = \{x_1, \ldots, x_n\} \setminus \{x_j, x_l\}$. Vielmehr ist die Bedingung genau dann erfüllt, wenn bei allen Paaren nicht verbundener Variablen jede der beiden Variablen im Schnitt mit der Hälfte der anderen Variablen verbunden ist.

Da wir den Fall $\gamma_1 = 0$ ausschließen, ist das folgende Ergebnis ohne Einschränkung gültig, weshalb wir durch den Beweis der Aussage auch die Richtigkeit von Satz 5.1.3 folgern können.

Theorem 5.1.17 *Es sei $\gamma_1 > 0$, dann gilt Abschätzung 5.1.3 für*

$$\tau = \left\lceil \frac{\gamma_1}{2} \right\rceil .$$

Wie wir später im Beweis sehen werden, ist diese Aussage am besten, wenn die einzelnen γ_i ähnlich groß sind.

Zu guter Letzt präsentieren wir noch ein zweites Ergebnis, das nicht auf den Verbindungen zwischen den Variablen beruht, sondern auf den Verbindungen zwischen den Nebenbedingungen.

Theorem 5.1.18 *Es gilt*

$$|C| \geq \left\lceil \frac{2(\delta_1 + 1)}{3} \right\rceil .$$

Für eine Matrix-Partitionierung ohne Balance-Kriterium gilt

$$|C| = \delta_1 ,$$

wenn $\delta_1 \leq 4$.

Auch bei dieser Abschätzung werden wir sehen, dass diese für ähnlich große δ_i am besten ist.

5.2 Beweise der Abschätzungen

Nachdem wir einige untere Schranken vorgestellt haben, wollen wir nun die Gültigkeit dieser Aussagen beweisen. Wie wir sehen werden, basieren alle Abschätzungen auf Ergebnissen für den assoziierten Hypergraph, die wir wiederum durch eine Graph-Repräsentation des Hypergraphen erhalten. Bei manchen nutzen wir spektrale Ergebnisse, bei anderen die Vollständigkeit des Graphen, und auch durch die Konnektivität des Graphen können wir spezielle Abschätzungen angeben. Die Theoreme sind genau danach sortiert: Die Theoreme 5.1.4 bis 5.1.8 nutzen Eigenwerte, die Theoreme 5.1.9 bis 5.1.11 beruhen auf Ergebnissen für einen vollständigen Graph, und die Theoreme 5.1.12 bis 5.1.18 erhalten wir durch Abschätzungen für die Konnektivität. Generell setzen wir voraus, dass die Graphen zusammenhängend sind, deshalb können wir die Ergebnisse auch nur auf Matrizen beziehen, die keine strikte Blockdiagonalgestalt haben. Allerdings können wir die Blöcke dann separat voneinander partitionieren und die unteren Schranken dann gegebenenfalls auf die Teilmatrizen der Blöcke anwenden.

Wir werden eine Matrix stets durch das Row-Net-Modell als Hypergraph darstellen. Nach Satz 4.3.2 gilt dann

$$|C| = |N_S| ,$$

sodass wir Abschätzungen für die Anzahl der Coupling Constraints durch Abschätzungen für die Anzahl der geschnittenen Netze einer Partition des assoziierten Hypergraphs bekommen. Allerdings gibt es kaum Ergebnisse zu unteren Schranken für die Cutsize einer Hypergraph-Partition, weshalb wir versuchen müssen, durch die Graph-Repräsentation des Hypergraphen untere Schranken herzuleiten. Zwar gibt es für Graphen mehr Resultate zu unteren Schranken für die Cutsize einer Partition, aber auch deren Anzahl ist überschaubar, was wohl auch den guten Ergebnissen der besten Heuristiken für das Graph-Partitionierungsproblem geschuldet ist.

In Kapitel 2.2.2 haben wir zwei Modelle kennen gelernt, durch die wir einen Hypergraph als Graph darstellen können. Beim Net-Intersection-Graph-Modell entspricht jeder Knoten im NIG einem Netz im Hypergraph, und die Cutsize einer GPVS stimmt mit der Cutsize der entsprechenden Partition des Hypergraphen überein. In Abbildung 5.3 sehen wir den NIG des Row-Net-Hypergraphen aus Abbildung 4.3.

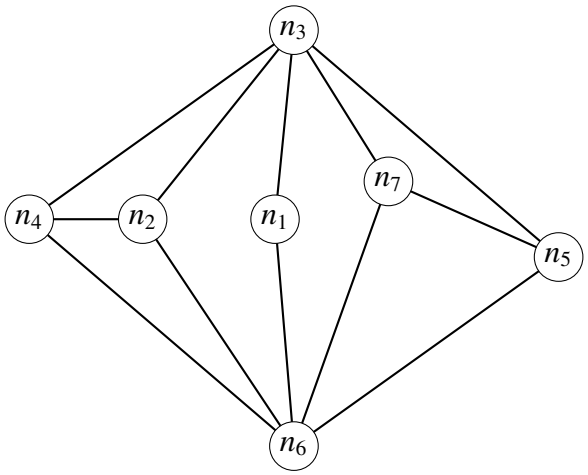

Abbildung 5.3: NIG des Row-Net-Hypergraphen aus Abbildung 4.3

Beim Clique-Net-Graph-Modell wird jedes Netz durch eine Clique ersetzt, aber im Gegensatz zum NIG-Modell können wir von der Cutsize einer GPES des CNG nicht direkt auf die Cutsize der entsprechenden Hypergraph-Partition schließen. Wir werden gleich jedoch ein Clique-Net-Graph-Modell vorstellen, das die Cutsize der Hypergraph-Partition durch eine geeignete Wahl von Kantengewichten von unten annähert.

Wir definieren einige Notationen, die wir im weiteren Verlauf dieses Kapitels benötigen werden. Mit $d(v_j)$ bezeichnen wir den Grad des Knoten v_j und mit $d_1 \leq \ldots \leq d_n$ die Knotengrade in aufsteigender Reihenfolge. Zudem definieren wir den gewichteten Knotengrad $d^w(v_j)$ als die Summe aller Gewichte der Kanten, zu denen v_j inzident ist. In einem ungewichteten Graph gilt natürlich $d^w(v_j) = d(v_j)$. Das Gesamtgewicht aller Kanten in G notieren wir mit W. An einigen Stellen werden wir zudem den *Diameter* $diam(G)$ eines Graphen G benutzen[3]. Wir betrachten nur Graphen mit mindestens zwei Knoten, da wir davon ausgehen, dass jede Matrix mindestens zwei Zeilen und zwei Spalten hat.

Um Abschätzungen für die Cutsize einer Partition des NIG oder des CNG auf das Matrix-Partitionierungsproblem übertragen zu können, müssen wir einige Beziehungen zwischen Kenngrößen einer Matrix und des assoziierten Graphen aufdecken. Die folgenden Beziehungen gelten für das Clique-Net-Graph-Modell, wenn wir die Matrix durch das Row-Net-Modell als Hypergraph darstellen.

- Ein Knoten v_j des CNG repräsentiert die $j-$te Spalte der Matrix und somit die Variable x_j, da v_j dem Knoten u_j des Hypergraphen entspricht.
- Da jede Nebenbedingung durch ein Netz repräsentiert wird, die im CNG als Clique dargestellt wird, sind zwei Variablen x_j und x_k genau dann durch eine Nebenbedingung miteinander verbunden, wenn es im CNG eine Kante e_{jk} gibt. Daraus folgt auch $d(v_j) = \gamma(x_j)$, da durch jede Verbindung der Variable x_j zu einer anderen Variable eine Kante im CNG induziert wird.
- Aus einer Partition des CNG erhalten wir eine Partition des Hypergraphen und daraus eine Partition der Matrix. Daher entspricht die Anzahl der Variablen in einem Block der Matrix-Partition genau der Anzahl der Knoten in der zugehörigen Teilmenge der CNG- bzw. Hypergraph-Partition, d.h. $n_k = |V_k|$.

[3]Die *Distanz* zwischen zwei Knoten ist die Anzahl der Kanten in einem kürzesten Weg zwischen diesen beiden Knoten. Der *Diameter* ist die längste Distanz zwischen zwei Knoten in G.

Für das Net-Intersection-Graph-Modell gelten die folgenden Äquivalenzen zwischen Kenngrößen einer Matrix und Kenngrößen des NIG.

- Ein Knoten v_i des NIG repräsentiert das Netz n_i des Hypergraphen und somit die Nebenbedingung r_i.
- Zwei Knoten sind genau dann im NIG über eine Kante verbunden, wenn die entsprechenden Netze mindestens einen gemeinsamen Knoten haben, d.h., wenn die entsprechenden Nebenbedingungen mindestens eine gemeinsame Variable haben.
- Die Anzahl der Nebenbedingungen, mit denen eine Nebenbedingung r_i über mindestens eine Variable verbunden ist, entspricht genau der Anzahl der Knoten, mit denen der Knoten v_i über eine Kante verbunden ist, d.h. $\delta(r_i) = d(v_i)$.

5.2.1 Ein Clique-Net-Graph-Modell

Wir wollen nun ein Clique-Net-Graph-Modell finden, durch das wir die Cutsize einer Partition des Hypergraphen so gut wie möglich approximieren können. Dazu suchen wir Gewichte für die Kanten des CNG, sodass die Anzahl der geschnittenen Netze durch das Gewicht der geschnittenen Kanten im CNG unterschätzt wird. Da jedes Netz durch eine Clique repräsentiert wird, ist dies insbesondere dann der Fall, wenn das Gewicht aller geschnittenen Kanten einer Clique immer höchstens 1 beträgt, unabhängig davon, welche und wie viele Kanten geschnitten werden. Hadley, Mark und Vannelli [37] haben ein solches Modell hergeleitet, das wir im Folgenden benutzen werden.

Den Clique-Net-Graph erhalten wir wie in Kapitel 2.2.2 beschrieben, indem wir jedes Netz durch eine Clique der Pins ersetzen. Bezeichnen wir mit K_n den vollständigen Graph auf n Knoten, dann ist eine Clique der vollständige Graph $K_{|Q|}$. Es gilt:

Lemma 5.2.1 *Die Anzahl der geschnittenen Kanten im K_n ist für gleichmäßige Partitionen am größten.*
Beweis: vgl. [37]

Nach Definition 2.1.2 unterscheiden sich die Größen der Teilmengen einer gleichmäßigen Partition um höchstens 1.

Die Größen der einzelnen Teilmengen einer gleichmäßigen Partition des K_n können wir leicht angeben.

Lemma 5.2.2 *In einer gleichmäßigen $K-$Partition des K_n haben $n - k\left\lfloor \frac{n}{K} \right\rfloor$ Teilmengen die Größe $\left\lceil \frac{n}{K} \right\rceil$ und $K - K\left\lfloor \frac{n}{K} \right\rfloor - n$ Teilmengen die Größe $\left\lfloor \frac{n}{K} \right\rfloor$. Ist K ein Teiler von n, sind alle Teilmengen gleich groß.*
Beweis: Die n Knoten müssen auf β_1 Teilmengen der Größe $\left\lceil \frac{n}{K} \right\rceil$ und β_2 Teilmengen der Größe $\left\lfloor \frac{n}{K} \right\rfloor$ verteilt werden, wobei β_1 und β_2 noch zu findende Zahlen sind, mit

$$\beta_1 + \beta_2 = K \tag{5.2.1}$$

und

$$\beta_1 \left\lceil \frac{n}{K} \right\rceil + \beta_2 \left\lfloor \frac{n}{K} \right\rfloor = n\,. \tag{5.2.2}$$

Formen wir Gleichung 5.2.2 um, erhalten wir mit Gleichung 5.2.1

$$\begin{aligned} n &= \beta_1 \left\lceil \frac{n}{K} \right\rceil + \beta_2 \left\lfloor \frac{n}{K} \right\rfloor \\ &= \beta_1 \left\lceil \frac{n}{K} \right\rceil + (K - \beta_1) \left\lfloor \frac{n}{K} \right\rfloor \\ &= \beta_1 \underbrace{\left(\left\lceil \frac{n}{K} \right\rceil - \left\lfloor \frac{n}{K} \right\rfloor \right)}_{=1} + K \left\lfloor \frac{n}{K} \right\rfloor \end{aligned}$$

und damit

$$\beta_1 = n - K \left\lfloor \frac{n}{K} \right\rfloor .$$

Setzen wir dies in Gleichung 5.2.1 ein, folgt

$$\beta_2 = K - \beta_1 = K - n + K \left\lfloor \frac{n}{K} \right\rfloor ,$$

sodass der erste Teil bewiesen ist. Wenn K ein Teiler von n ist, gilt $\left\lfloor \frac{n}{K} \right\rfloor = \left\lceil \frac{n}{K} \right\rceil$. □

Insbesondere ist die Anzahl der Teilmengen mit Größe $\left\lceil \frac{n}{K} \right\rceil$ bzw. $\left\lfloor \frac{n}{K} \right\rfloor$ also eindeutig.

Die Cutsize einer Partition des K_n in Teilmengen $V_1, \ldots, V_K$ können wir ebenfalls schnell bestimmen.

Satz 5.2.3 *Wenn der K_n in K Teilmengen $V_1, \ldots, V_K$ partitioniert wird, gilt*

$$|E_S| = \binom{n}{2} - \sum_{k=1}^{K} \binom{|V_k|}{2}.$$

Beweis: Der K_n hat $\binom{n}{2}$ Kanten. Nach der Partitionierung in die Teilmengen $V_1, \ldots, V_K$ sind alle Knoten innerhalb einer Teilmenge V_k miteinander verbunden, da sie bereits im K_n verbunden waren. Eine Teilmenge V_k hat daher $\binom{|V_k|}{2}$ Kanten. Insgesamt befinden sich also von den $\binom{n}{2}$ Kanten aus dem K_n genau $\sum_{k=1}^{K} \binom{|V_k|}{2}$ Kanten innerhalb der Teilmengen, weshalb die restlichen $\binom{n}{2} - \sum_{k=1}^{K} \binom{|V_k|}{2}$ zwischen zwei Teilmengen verlaufen müssen und daher geschnittene Kanten sind. □

Dadurch können wir auch die Anzahl der geschnittenen Kanten einer gleichmäßigen Partition des K_n genau angeben.

Satz 5.2.4 *Die Anzahl der geschnittenen Kanten einer Partition des K_n in K gleichmäßige Teilmengen beträgt*

$$|E_S| = \binom{n}{2} - \binom{\left\lceil \frac{n}{K} \right\rceil}{2} \left(n - K \left\lfloor \frac{n}{K} \right\rfloor \right) - \binom{\left\lfloor \frac{n}{K} \right\rfloor}{2} \left(K + K \left\lfloor \frac{n}{K} \right\rfloor - n \right) .$$

Beweis: Für den Moment setzen wir $t_1 := n - K \left\lfloor \frac{n}{K} \right\rfloor$ und $t_2 := K + K \left\lfloor \frac{n}{K} \right\rfloor - n$. Der K_n hat insgesamt $\binom{n}{2}$ Kanten. Nach Lemma 5.2.2 haben t_1 Teilmengen die Größe $\left\lceil \frac{n}{K} \right\rceil$ und t_2 Teilmengen die Größe $\left\lfloor \frac{n}{K} \right\rfloor$. Jede Teilmenge ist wieder ein vollständiger Graph

mit $\lceil \frac{n}{K} \rceil$ bzw. $\lfloor \frac{n}{K} \rfloor$ Knoten und hat daher $\binom{\lceil \frac{n}{K} \rceil}{2}$ bzw. $\binom{\lfloor \frac{n}{K} \rfloor}{2}$ Kanten. Insgesamt verlaufen also $\binom{\lceil \frac{n}{K} \rceil}{2}t_1 + \binom{\lfloor \frac{n}{K} \rfloor}{2}t_2$ Kanten innerhalb der Teilmengen, weshalb die restlichen $\binom{n}{2} - \left(\binom{\lceil \frac{n}{K} \rceil}{2}t_1 + \binom{\lfloor \frac{n}{K} \rfloor}{2}t_2\right)$ Kanten zwischen den Teilmengen verlaufen müssen und daher geschnittene Kanten sind. □

Nach Lemma 5.2.1 ist die Anzahl der geschnittenen Kanten einer Partition des K_n am größten, wenn die Partition gleichmäßig ist. Zudem haben wir durch Satz 5.2.4 die Anzahl der geschnittenen Kanten einer gleichmäßigen Partition des K_n exakt bestimmt. Da eine Clique Q der vollständige Graph $K_{|Q|}$ ist, erhalten wir daher das folgende Resultat.

Satz 5.2.5 *Sei Q eine Clique, die in K Teilmengen partitioniert werden soll. Wenn jede Kante e_{ij} der Clique das Gewicht*

$$w_{ij} = \frac{1}{\binom{|Q|}{2} - \binom{\lceil \frac{|Q|}{K} \rceil}{2}\left(|Q| - K\left\lfloor \frac{|Q|}{K} \right\rfloor\right) - \binom{\lfloor \frac{|Q|}{K} \rfloor}{2}\left(K + K\left\lfloor \frac{|Q|}{K} \right\rfloor - |Q|\right)}$$

hat, beträgt das Gesamtgewicht aller geschnittenen Kanten innerhalb dieser Clique höchstens 1.

Beweis: Angenommen, es gäbe eine Situation, in der das Gesamtgewicht der geschnittenen Kanten einer Clique echt größer als 1 ist. Da die Anzahl der geschnittenen Kanten nach Satz 5.2.4 genau

$$|E_S| = \binom{|Q|}{2} - \binom{\lceil \frac{|Q|}{K} \rceil}{2}\left(|Q| - K\left\lfloor \frac{|Q|}{K} \right\rfloor\right) - \binom{\lfloor \frac{|Q|}{K} \rfloor}{2}\left(K + K\left\lfloor \frac{|Q|}{K} \right\rfloor - |Q|\right)$$

ist, muss es dann eine Kante geben, deren Gewicht größer als $\frac{1}{|E_S|}$ ist, dies ist ein Widerspruch. □

Dieses Ergebnis können wir jetzt auf das CNG-Modell anwenden. Dazu erinnern wir uns, dass p_j die Anzahl der Pins des Netzes n_j definiert.

Satz 5.2.6 *Sei $H = (U, N)$ ein Hypergraph, der in K Teilmengen partitioniert werden soll, und $G^{CNG} = (V, E)$ der CNG des Hypergraphen. Jede Kante $e_{ij} \in E$ erhalte das Gewicht*

$$w_{ik} = \sum_{h_j \in Nets(u_i) \cap Nets(u_k)} \frac{1}{\binom{p_j}{2} - \binom{\lceil \frac{p_j}{K} \rceil}{2}\left(p_j - K\left\lfloor \frac{p_j}{K} \right\rfloor\right) - \binom{\lfloor \frac{p_j}{K} \rfloor}{2}\left(K + K\left\lfloor \frac{p_j}{K} \right\rfloor - p_j\right)}.$$

Sei $w(E_S)$ das Cutweight einer Partition des CNG. Dann gilt für die Cutsize der entsprechenden Partition des Hypergraphen

$$|N_S| \geq w(E_S).$$

Beweis: Ein Netz des Hypergraphen ist genau dann geschnitten, wenn mindestens eine Kante der zugehörigen Clique geschnitten ist, deshalb gibt es genau $|N_S|$ Cliquen mit geschnittenen Kanten. Wir betrachten jede dieser $|N_S|$ Cliquen einzeln, d.h. alle

Kanten einer Clique haben das gleiche Gewicht gemäß Satz 5.2.5. Das Cutweight der Partition des CNG ergibt sich aus der Summe der Gesamtgewichte der geschnittenen Kanten der Cliquen, wobei es möglich ist, dass eine Kante mehrfach unterschiedliche Gewichte zum Cutweight beiträgt, wenn diese zu mehreren Cliquen gehört. Nach Satz 5.2.5 ist das Gesamtgewicht der geschnittenen Kanten einer Clique höchstens 1. Wenn wir das Gewicht der geschnittenen Kanten der N_S Cliquen addieren, beträgt dieses daher höchstens $|N_S|$. □

Wir haben zwar keine perfekte Graph-Repräsentation des Hypergraphen gefunden, aber eine gute Approximation, da die bestimmten Gewichte sogar optimal sind [37]. Wir veranschaulichen die Aussage von Satz 5.2.6 an einem Beispiel.

Beispiel 5.2.7 In Abbildung 5.4 sehen wir den CNG des Row-Net-Hypergraphen aus Abbildung 4.3. Partitionieren wir die Knoten des Graphen in die beiden Teilmengen $V_1 := \{1, 2, 4, 5, 8\}$ und $V_2 := \{3, 6, 7, 9, 10\}$, erhalten wir mit den Gewichten aus Satz 5.2.6 ein Cutweight von

$$w(E_S) = \sum_{i \in V_1, j \in V_2} w_{ij}$$

$$= w_{1,7} + w_{1,9} + w_{1,10} + w_{2,9} + w_{2,10} + w_{3,4} + w_{3,8} + w_{4,6} + w_{4,7} + w_{5,9} + w_{5,10} + w_{6,8} + w_{7,8}$$

$$= 1 + 12 \cdot \frac{1}{6} = 3\,.$$

Tatsächlich werden durch diese Partition im Hypergraph die Netze $(1, 7)$, $(1, 2, 5, 9, 10)$ und $(3, 4, 6, 7, 8)$ geschnitten, d.h. $|N_S| = 3$. Also können wir die Anzahl der geschnittenen Netze durch $w(E_S)$ hier sogar perfekt abschätzen.

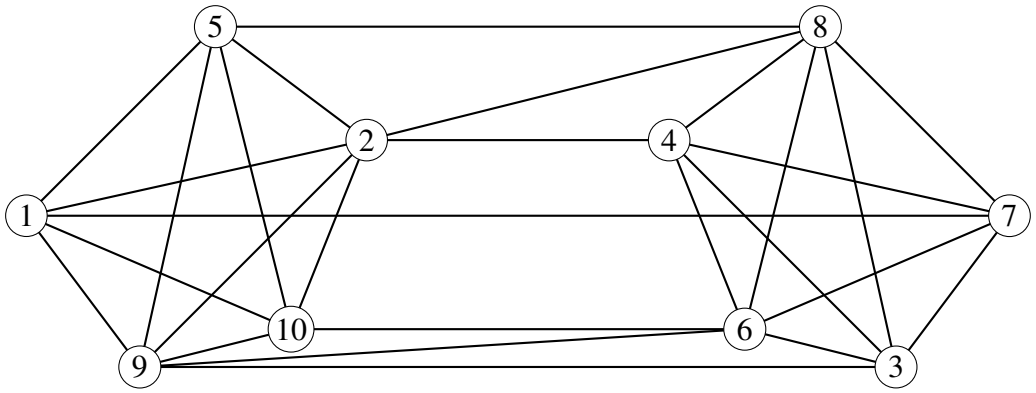

Abbildung 5.4: Clique-Net-Graph des Row-Net-Hypergraphen aus Abbildung 4.3

Die Aussage von Satz 5.2.6 können wir noch weiter nach unten abschätzen, indem wir die Cutsize des ungewichteten Graphen betrachten.

Satz 5.2.8 *Sei $H = (U, N)$ ein Hypergraph und $G^{CNG} = (V, E)$ der CNG des Hypergraphen. Weiter seien $|N_S|$ die Cutsize einer Partition des Hypergraphen, $|E_S|$ die Cutsize der entsprechenden Partition des CNG, $w(E_S)$ das Cutweight dieser Partition und $w_1 \leq \ldots \leq w_{|E_S|}$ die $|E_S|$ kleinsten Gewichte im Clique-Net-Graph. Dann gilt*

$$|N_S| \geq w(E_S) \geq \sum_{i=1}^{|E_S|} w_i\,.$$

Beweis: $w(E_S)$ ergibt sich aus den Kantengewichten der Kanten E_S. Die Kante mit dem kleinsten Gewicht in E_S hat mindestens ein Gewicht von w_1, die Kante mit dem zweitkleinsten Gewicht in E_S hat mindestens ein Gewicht von w_2 usw., sodass wir die Gewichte der $|E_S|$ Kanten gegen die $|E_S|$ kleinsten Gewichte im Graph abschätzen können. Also gilt

$$w(E_S) \geq w_1 + \ldots + w_{|E_S|}\,.$$

□

Anwendung auf Matrix-Partitionierung

Wegen $|C| = |N_S|$ erhalten wir durch Satz 5.2.8 auch untere Schranken für die Anzahl der Coupling Constraints. Dies gilt insbesondere, wenn wir untere Schranken für $|E_S|$ finden, d.h. untere Schranken für die Cutsize des ungewichteten CNG. Das Gewicht $\omega(r_i)$ der $i-$ten Nebenbedingungen ergibt sich natürlich direkt aus dem Gewicht der entsprechenden Clique des Netzes, und das Gewicht eines Variablen-Paares (x_j, x_k) ist genau das Gewicht der Kante e_{jk} aus Satz 5.2.6, d.h. $\omega(x_j, x_k) = w_{jk}$.

5.2.2 Abschätzungen durch Eigenwerte

Eine Möglichkeit, $w(W_S)$ bzw. $|E_S|$ abzuschätzen, führt über die Eigenwerte der gewichteten bzw. ungewichteten Laplace-Matrix des Graphen. Ein zentrales Resultat gelang Donath und Hoffman [20], das für die Partition eines ungewichteten Graphen gilt. Wir zeigen dieses Ergebnis für einen gewichteten Graph.

Satz 5.2.9 *Sei $G = (V, E)$ ein gewichteter Graph, $Q(G)$ die gewichtete Laplace-Matrix von G und $\lambda_1 \leq \ldots \lambda_n$ die Eigenwerte von Q. Weiter seien $|V_1| \geq \ldots \geq |V_K|$ die Größen der Teilmengen einer $K-$Partition von G und E_S die Menge der geschnittenen Kanten dieser Partition. Dann gilt*

$$w(E_S) \geq \frac{1}{2} \sum_{k=1}^{K} |V_k| \lambda_k(Q)\,.$$

Beweis: Wir definieren die Matrix U durch

$$U = \begin{pmatrix} U_1 & & \\ & \ddots & \\ & & U_K \end{pmatrix} \in \mathbb{R}^{n \times n}$$

mit

$$U_k = \begin{pmatrix} -1 & \cdots & -1 \\ \vdots & \ddots & \vdots \\ -1 & \cdots & -1 \end{pmatrix} \in \mathbb{R}^{|V_k| \times |V_k|}\,.$$

Sei u^k der Vektor, der an jeder Stelle, die zur Teilmatrix U_k gehört, eine 1 stehen hat und sonst eine 0. Dann gilt

$$Uu^k = -|V_k| u^k\,,$$

sodass $-|V_1| \leq \ldots \leq -|V_K|$ Eigenwerte von U sind. Definieren wir nun y^k als einen Vektor, der an einer Stelle der Teilmatrix U_k eine 1, an einer anderen Stelle der Teilmatrix U_k eine -1 und sonst eine 0 stehen hat, gilt

$$Uy^k = 0\,,$$

sodass die restlichen $n - K + 1$ Eigenwerte 0 sind. Zusammenfassend ist also

$$\begin{aligned} \lambda_k(U) &= -|V_k|\,, \quad k = 1, \ldots, K, \\ \lambda_k(U) &= 0\,, \qquad\quad k = K+1, \ldots, n\,. \end{aligned}$$

Wir fassen nun die Zeilen und Spalten von U_k als die Zeilen und Spalten der Knoten in V_k auf, und permutieren die Matrix so um, dass die Reihenfolge der Knoten mit der Reihenfolge in $Q(G)$ übereinstimmt. Durch die Permutation ändern sich die Eigenwerte der Matrix U nicht, daher gilt

$$\sum_{i=1}^{n} \lambda_i(U)\lambda_i(Q) = -\sum_{k=1}^{K} |V_k|\lambda_k(Q)\,.$$

Auf der anderen Seite ist

$$Spur(QU^T) = Spur(D^wU^T - A^wU^T) = Spur(D^wU^T) - Spur(A^wU^T)\,.$$

Da D^w eine Diagonalmatrix ist, und auf der Diagonalen von U^T immer eine -1 steht, gilt zum einen

$$Spur(D^wU^T) = -\sum_{i=1}^{n} d_{ii}^w = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\,,$$

und zum anderen folgt mit Satz A.2.5

$$Spur(A^wU^T) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^w u_{ij}\,.$$

Um diesen Ausdruck zu bestimmen, betrachten wir zunächst einmal die Matrix U', deren Einträge im Gegensatz zu U alle -1 sind. Dann gilt

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^w u'_{ij} = -\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^w = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\,.$$

In der Matrix U gilt jedoch nur dann $u_{ij} = -1$, wenn die $i-$te Zeile und die $j-$te Spalte zu zwei Knoten gehören, die in derselben Teilmenge V_k sind. Es fehlt also insbesondere dann eine -1, wenn die $i-$te Zeile und die $j-$te Spalte zu zwei Knoten v_i und v_j mit $v_i \in V_k$, $v_j \in V_l$, $k \neq l$, gehören. Daher erhalten wir nur noch die negative Summe aller Kantengewichte von Kanten, die innerhalb einer Teilmenge verlaufen, also ohne die geschnittenen Kanten. Deshalb ist

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^w u_{ij} = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} + \sum_{\substack{(i,j)\in E:\\ i\in V_k, j\in V_l, k\neq l}} 2w_{ij} = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} + 2w(EW_S)\,,$$

da jede Kante zweimal gezählt wird. Insgesamt ist also

$$Spur(D^wU^T) - Spur(A^wU^T) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} - \left(-\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} + 2w(E_S)\right) = -2w(E_S)\,.$$

Mit Satz A.2.6 folgt nun

$$-2w(E_S) = Spur(QU^T) \leq \sum_{i=1}^{n} \lambda_i(U)\lambda_i(Q) = -\sum_{k=1}^{n} |V_k|\lambda_k(Q)$$

und somit

$$w(E_S) \geq \frac{1}{2}\sum_{k=1}^{K} |V_k|\lambda_k(Q)\,.$$

□

Für einen ungewichteten Graph erhalten wir natürlich das folgende Korollar.

Korollar 5.2.10 *Seien $\lambda_1 \leq \ldots, \leq \lambda_K$ die K kleinsten Eigenwerte der Laplace-Matrix L(G), und $|V_1| \geq \ldots \geq |V_K|$. Dann gilt*

$$|E_S| \geq \frac{1}{2}\sum_{k=1}^{K} |V_k|\lambda_k(L)\,.$$

Für eine Bisektion mit zwei gleich großen Teilmengen bekommen wir wegen $\lambda_1 = 0$ die untere Schranke, die wir bereits beim Verfahren der Spektralen Bisektion hergeleitet hatten.

Korollar 5.2.11 *Für $|V_1| = |V_2| = n/2$ gilt*

$$|E_S| \geq \frac{n\lambda_2}{4}\,.$$

Mit Hilfe von Satz 5.2.9 und Korollar 5.2.10 werden wir nun die ersten Theoreme aus Kapitel 5.1.2 beweisen.

Beweis von Theorem 5.1.4

Um diese Aussage zu zeigen, benutzen wir das folgende Ergebnis von Fiedler [27].

Satz 5.2.12 *Sei G ein Graph, d_1 der kleinste Grad in G und λ_2 der zweitkleinste Eigenwert von L(G). Dann gilt*

$$\lambda_2 \geq 2d_1 - n + 2\,.$$

Nach Korollar 5.2.10 gilt

$$|E_S| \geq \frac{1}{2}\sum_{k=1}^{K} |V_k|\lambda_k(L)\,,$$

was wir nun mit $\lambda_1 = 0$ abschätzen können durch

$$\begin{aligned}
|E_S| \geq \frac{1}{2}\sum_{k=1}^{K} |V_k|\lambda_k(L) &\geq \frac{1}{2}\sum_{k=2}^{K} |V_k|\lambda_2 \\
&\geq \frac{1}{2}\sum_{k=2}^{K} |V_k|(2d_1 - n + 2) \\
&= \frac{2d_1 - n + 2}{2}\sum_{k=2}^{K} |V_k| \\
&= \frac{(2d_1 - n + 2)(n - |V_1|)}{2}\,.
\end{aligned}$$

Wegen $|C| = |N_S|$, $\gamma_1 = d_1$ und $n_1 = |V_1|$ folgt mit Satz 5.2.8 die Behauptung.

Hier sehen wir auch, dass diese Abschätzung für große K eher unbrauchbar ist, wenn die Eigenwerte nicht ähnlich groß sind. Die Abschätzung für λ_2 stellt dann nämlich eine schlechte Abschätzung für die größeren Eigenwerte dar, und somit wird auch die untere Schranke für $|C|$ schlecht.

Beweis von Theorem 5.1.5

Um Theorem 5.1.5 zu beweisen, nutzen wir eine Erweiterung von Satz 5.2.12, die eine Abschätzung für den k-ten Eigenwert angibt.

Satz 5.2.13 *Sei G ein Graph ohne isolierte Knoten, d_k der k-kleinste Grad in G und λ_k der k-kleinste Eigenwert von $L(G)$, $k = 2, \ldots, n$. Dann gilt*

$$\lambda_k \geq d_k - n + k + 1\,.$$

Dieses Ergebnis geht auf eine Conjecture von Guo zurück, die von Brouwer und Haemers für Graphen ohne isolierte Knoten bewiesen wurde [8]. Deshalb können wir diese Abschätzung auch nur benutzen, wenn der assoziierte Graph keine isolierten Knoten hat, d.h., wenn jede Variable mit mindestens einer anderen Variable über eine Nebenbedingung verbunden ist. Wenn wir für λ_2 weiterhin die Abschätzung aus Satz 5.2.12 benutzen, erhalten wir mit Korollar 5.2.10 und Satz 5.2.13

$$\begin{aligned}
|E_S| \geq \frac{1}{2}\sum_{k=1}^{K} |V_k|\lambda_k &\geq \frac{|V_2|(2d_1 - n + 2)}{2} + \frac{1}{2}\sum_{k=3}^{K} |V_k|(d_k - n + k + 1) \\
&= \frac{|V_2|(2d_1 - n + 2)}{2} + \frac{1}{2}\sum_{k=3}^{K} |V_k|(d_k + k) - \frac{n-1}{2}\sum_{k=3}^{K} |V_k| \\
&= \frac{|V_2|(2d_1 - n + 2)}{2} + \frac{1}{2}\sum_{k=3}^{K} |V_k|(d_k + k) - \frac{(n-1)(n - |V_1| - |V_2|)}{2} \\
&= \frac{|V_2|(2d_1 - n + 2) - (n-1)(n - |V_1| - |V_2|)}{2} + \frac{1}{2}\sum_{k=3}^{K} |V_k|(d_k + k) \\
&= \frac{|V_2|(2d_1 + 1) - (n-1)(n - |V_1|)}{2} + \frac{1}{2}\sum_{k=3}^{K} |V_k|(d_k + k)\,.
\end{aligned}$$

Wegen $|C| = |N_S|$, $\gamma_k = d_k$ und $n_k = |V_k|$ folgt mit Satz 5.2.8 die Behauptung.

Beweis von Korollar 5.1.6

Diese Aussage ist ein Spezialfall von Theorem 5.1.5. Mit $|V_k| = \frac{n}{K}$ folgt

$$\begin{aligned}
|E_S| &\geq \frac{\frac{n}{K}(2d_1+1) - (n-1)(n-\frac{n}{K})}{2} + \frac{1}{2}\sum_{k=3}^{K}\frac{n}{K}(d_k+k) \\
&= \frac{n(2d_1+1) - n(n-1)(K-1)}{2K} + \frac{n}{2K}\sum_{k=3}^{K} d_k + \frac{n}{2K}\sum_{k=3}^{K} k \\
&= \frac{n}{2K}\left(2d_1 + 1 - (n-1)(K-1) + \sum_{k=3}^{K} d_k + \sum_{k=3}^{K} k\right),
\end{aligned}$$

und dies können wir dann mit der Gaußschen Summenformel A.2.10 vereinfachen zu

$$\begin{aligned}
|E_S| &\geq \frac{n}{2K}\left(2d_1 - nK + n + K + \sum_{k=3}^{K} d_k + \sum_{k=3}^{K} k\right) \\
&= \frac{n}{2K}\left(2d_1 - n(K-1) + K + \sum_{k=3}^{K} d_k + \frac{K(K+1)}{2} - 3\right) \\
&= \frac{n}{2K}\left(2d_1 - n(K-1) + \frac{K(K+3)}{2} - 3 + \sum_{k=3}^{K} d_k\right).
\end{aligned}$$

Wegen $|C| = |N_S|$ und $\gamma_k = d_k$ folgt mit Satz 5.2.8 die Behauptung.

Beweis von Theorem 5.1.7

Der Beweis dieser unteren Schranke ist relativ aufwendig. Bevor wir starten, beweisen wir zunächst drei Hilfssätze.

Das folgende Lemma besagt, dass die Summe der Laplace'schen Eigenwerte genau der Summe der gewichteten Knotengrade entspricht.

Lemma 5.2.14 *Seien $\lambda_1 \leq \ldots \leq \lambda_n$ die Eigenwerte von $Q(G)$, $d^w(v_j)$ der gewichtete Knotengrad des Knoten v_j und $W = \sum_{(ij)\in E} w_{ij}$. Dann gilt*

$$\sum_{i=1}^{n} \lambda_i = \sum_{j=1}^{n} d^w(v_j) = 2W\,.$$

Beweis: Die Spur einer Matrix ist definiert als die Summe ihrer Diagonalelemente, weshalb für die Laplace-Matrix $Q(G)$

$$Spur(Q) = \sum_{i=1}^{n} q_{ii} = \sum_{i=1}^{n} d^w(v_i)$$

ist. Nach Satz A.2.7 gilt aber auch

$$Spur(Q) = \sum_{i=1}^{n} \lambda_i\,,$$

und somit folgt die erste Gleichung. Die zweite Gleichung gilt, da jedes Kantengewicht w_{ij} in der linken Summe sowohl durch $d^w(v_i)$ als auch durch $d^w(v_j)$ gezählt wird. □

Das zweite Lemma liefert eine ähnliche Aussage für die Summe der Quadrate der Eigenwerte.

Lemma 5.2.15 *Es gilt*

$$\sum_{i=1}^{n} \lambda_i^2 = \sum_{i=1}^{n} \left(d^w(v_i)^2 + \sum_{j=1}^{n} w_{ij} \right).$$

Beweis: Wieder argumentieren wir über die Spur, betrachten nun aber die Matrix Q^2. Diese hat die Form

$$Q^2 = \begin{pmatrix} d^w(v_1)^2 + \sum_{j=1}^{n} w_{1j}^2 & & * \\ & \ddots & \\ * & & d^w(v_n)^2 + \sum_{j=1}^{n} w_{nj}^2 \end{pmatrix},$$

sodass

$$Spur(Q^2) = \sum_{i=1}^{n} \left(d^w(v_i)^2 + \sum_{j=1}^{n} w_{ij}^2 \right).$$

Da Q symmetrisch ist, existiert nach Satz A.2.8 eine obere Dreiecksmatrix U und eine invertierbare Matrix P mit

$$Q = PUP^{-1},$$

wobei die Eigenwerte der Matrix Q auf der Diagonalen von U stehen. Dadurch ergibt sich

$$\begin{aligned} Spur(Q^2) =& Spur(PUP^{-1}PUP^{-1}) \\ =& Spur(PUUP^{-1}). \end{aligned}$$

Mit Satz A.2.9 folgt

$$Spur(PUUP^{-1}) = Spur(UP^{-1}PU) = Spur(UU) = \sum_{i=1}^{n} \lambda_i^2,$$

da U eine obere Dreiecksmatrix ist. Insgesamt folgt also

$$\sum_{i=1}^{n} \left(d^w(v_i)^2 + \sum_{j=1}^{n} w_{ij}^2 \right) = Spur(Q^2) = \sum_{i=1}^{n} \lambda_i^2.$$

□

Zudem werden wir die folgende Abschätzung für die gewichteten Knotengrade brauchen.

Lemma 5.2.16 *Es gilt*

$$\sum_{i=1}^{n} d^w(v_i)^2 \leq \frac{2|W|^2}{n-1} + (n-2) \sum_{(i,j)\in E} w_{ij}^2.$$

Beweis: In [19] wird gezeigt, dass

$$\left(\sum_{\{i,j\}} x_{ij}\right)^2 + \binom{n-1}{2}\sum_{\{i,j\}} x_{ij}^2 - \frac{n-1}{2}\sum_{i=1}^{n}\left(\sum_{j\neq i} x_{ij}\right)^2 \geq 0$$

für $x_{ij} \in \mathbb{R}$, wobei $\{i, j\}$ das ungeordnete Paar der Indizes i und j darstellt. Nun setzen wir

$$x_{ij} = a_{ij}^w = \begin{cases} w_{ij}, & i,j \in E \\ 0, & \text{sonst} \end{cases}$$

und erhalten dadurch

$$\begin{aligned} &\left(\sum_{\{i,j\}} a_{ij}\right)^2 + \binom{n-1}{2}\sum_{\{i,j\}} a_{ij}^2 - \frac{n-1}{2}\sum_{i=1}^{n}\left(\sum_{j\neq i} a_{ij}\right)^2 \geq 0 \\ \Leftrightarrow\ & W^2 + \frac{(n-1)(n-2)}{2}\sum_{(i,j)\in E} w_{ij}^2 \geq \frac{n-1}{2}\sum_{i=1}^{n} d^w(v_i)^2 \\ \Leftrightarrow\ & \frac{2W^2}{n-1} + (n-2)\sum_{(i,j)\in E} w_{ij}^2 \geq \sum_{i=1}^{n} d^w(v_i)^2 . \end{aligned}$$

□

Die Aussage von Theorem 5.1.7 beruht auf dem folgenden Satz.

Satz 5.2.17 *Sei $S_k := \sum_{i=1}^{K} \lambda_k$ die Summe der K kleinsten Eigenwerte von $Q(G)$ und*

$$\frac{W^2}{\sum_{(i,j)\in E} w_{ij}^2} > \frac{n-K}{n+K-2}\binom{n}{2}.$$

Dann gilt

$$S_k \geq \frac{2W(K-1) - \sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)}}{n-1}.$$

Beweis: Mit Lemma 5.2.14 und der Cauchy-Schwarz-Ungleichung erhalten wir zunächst die Abschätzung

$$\begin{aligned} (2W - S_K)^2 &= \left(\sum_{i=1}^{n}\lambda_i - \sum_{i=1}^{K}\lambda_i\right)^2 \\ &= \left(\sum_{i=K+1}^{n}\lambda_i\right)^2 \\ &\leq (n-K)\left(\sum_{i=K+1}^{n}\lambda_i^2\right) \\ &= (n-K)\left(\sum_{i=1}^{n}\lambda_i^2 - \sum_{i=1}^{K}\lambda_i^2\right). \end{aligned}$$

Wegen $\lambda_1 = 0$ und mit Lemma 5.2.15 folgt dann

$$(n-K)\left(\sum_{i=1}^{n}\lambda_i^2 - \sum_{i=1}^{K}\lambda_i^2\right) = (n-K)\left(\sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right] - \sum_{i=2}^{n}\lambda_i^2\right),$$

und erneut unter Anwendung der Cauchy-Schwarz-Ungleichung und wegen $\lambda_1 = 0$ erhalten wir

$$\begin{aligned}
&(n-K)\left(\sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right] - \sum_{i=2}^{n}\lambda_i^2\right)\\
\leq\ &(n-K)\left(\sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right] - \frac{1}{K-1}\left(\sum_{i=2}^{n}\lambda_i\right)^2\right)\\
=\ &(n-K)\left(\sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right] - \frac{1}{K-1}S_K^2\right),
\end{aligned}$$

also

$$(2W - S_K)^2 \leq (n-K)\left(\sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right] - \frac{1}{K-1}S_K^2\right).$$

Wir sammeln alle Terme mit S_K auf der linken Seite und formen diese um zu

$$\begin{aligned}
(2W - S_K)^2 + \frac{n-K}{K-1}S_K^2 &= 4W^2 - 4WS_K + S_K^2 + \frac{n-K}{K-1}S_K^2\\
&= 4W^2 - 4WS_K + \frac{K-1}{K-1}S_K^2 + \frac{n-K}{K-1}S_K^2\\
&= 4W^2 - 4WS_K + \frac{n-1}{K-1}S_K^2.
\end{aligned}$$

Mit $A := \sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n}w_{ij}^2\right]$ gilt dann

$$4W^2 - 4WS_K + \frac{n-1}{K-1}S_K^2 \leq (n-K)A,$$

und durch quadratische Ergänzung erhalten wir

$$\begin{aligned}
&4W^2 - 4WS_K + \frac{n-1}{K-1}S_K^2 \leq (n-K)A\\
\Leftrightarrow\ &S_K^2 - \frac{4W(K-1)}{n-1}S_K \leq \frac{(K-1)(n-K)}{n-1}A - \frac{K-1}{n-1}4W^2\\
\Leftrightarrow\ &S_K^2 - \frac{4W(K-1)}{n-1}S_K + \frac{4W^2(K-1)^2}{(n-1)^2} - \frac{4W^2(K-1)^2}{(n-1)^2} \leq \frac{K-1}{n-1}\left((n-K)A - 4W^2\right)\\
\Leftrightarrow\ &\left(S_K - \frac{2W(K-1)}{n-1}\right)^2 \leq \frac{(K-1)(n-1)}{(n-1)^2}\left((n-K)A - 4W^2\right) + \frac{4W^2(K-1)^2}{(n-1)^2}.
\end{aligned}$$

Da diese Abschätzung durch die Äquivalenz auf jeden Fall gelten muss, folgt daraus

$$S_K - \frac{2W(K-1)}{n-1} \leq \frac{\sqrt{(K-1)\left[(n-1)\left((n-K)A - 4W^2\right) + 4W^2(K-1)\right]}}{n-1}$$

und

$$S_K - \frac{2W(K-1)}{n-1} \geq -\frac{\sqrt{(K-1)\left[(n-1)\left((n-K)A - 4W^2\right) + 4W^2(K-1)\right]}}{n-1}.$$

Wir verfolgen natürlich den zweiten Fall

$$S_K \geq \frac{2W(K-1) - \sqrt{(K-1)\left[(n-1)\left((n-K)A - 4W^2\right) + 4W^2(K-1)\right]}}{n-1} \tag{5.2.3}$$

und formen den Term $B \coloneqq (n-1)\left((n-K)A - 4W^2\right) + 4W^2(K-1)$ unter der Wurzel um zu

$$\begin{aligned} B &= (n-1)\left((n-K)A - 4W^2\right) + 4W^2(K-1) \\ &= (n-1)(n-K)A - (n-1)4W^2 + 4W^2(K-1) \\ &= (n-1)(n-K)A - (n-K)4W^2 \\ &= (n-K)((n-1)A - 4W^2)\,. \end{aligned}$$

Schätzen wir den Term A mit Lemma 5.2.16 ab, ergibt sich

$$\begin{aligned} A = \sum_{i=1}^{n}\left[d^w(v_i)^2 + \sum_{j=1}^{n} w_{ij}^2\right] &\leq \frac{2W^2}{n-1} + (n-2)\sum_{\{i,j\}} w_{ij}^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}^2 \\ &= \frac{2W^2}{n-1} + (n-2)\sum_{\{i,j\}} w_{ij}^2 + 2\sum_{\{i,j\}} w_{ij}^2 \qquad (5.2.4) \\ &= \frac{2W^2}{n-1} + n\sum_{\{i,j\}} w_{ij}^2\,, \end{aligned}$$

wobei Zeile 5.2.4 wegen $w_{ii} = 0$ gilt. Somit erhalten wir

$$\begin{aligned} B = (n-K)((n-1)A - 4W^2) &\leq (n-K)\left((n-1)\frac{2W^2}{n-1} + n\sum_{\{i,j\}} w_{ij}^2 - 4W^2\right) \\ &= (n-K)\left(2W^2 + n(n-1)\sum_{\{i,j\}} w_{ij}^2 - 4W^2\right) \\ &= (n-K)\left(n(n-1)\sum_{\{i,j\}} w_{ij}^2 - 2W^2\right), \end{aligned}$$

und Abschätzung 5.2.3 wird zu

$$S_k \geq \frac{2W(K-1) - \sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)}}{n-1}.$$

Da $S_k > 0$, sollte auch

$$\frac{2W(K-1) - \sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)}}{n-1} > 0$$

gelten. Dies ist genau dann der Fall, wenn

$$
\begin{aligned}
&2W(K-1) > \sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)} \\
\Leftrightarrow\ & 4W^2(K-1)^2 > (K-1)(n-K)\left(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2\right) \\
\Leftrightarrow\ & 4W^2(K-1) > (n-K)\left(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2\right) \\
\Leftrightarrow\ & 4W^2(K-1) + 2(n-K)W^2 > (n-K)\left(n(n-1)\sum_{(i,j)\in E} w_{ij}^2\right) \\
\Leftrightarrow\ & 2W^2(n+K-2) > (n-K)\left(n(n-1)\sum_{(i,j)\in E} w_{ij}^2\right) \\
\Leftrightarrow\ & W^2 > \frac{n+K-2}{(n-K)}\left(\frac{n(n-1)}{2}\sum_{(i,j)\in E} w_{ij}^2\right) \\
\Leftrightarrow\ & \frac{W^2}{\sum_{(i,j)\in E} w_{ij}^2} > \frac{n+K-2}{(n-K)}\binom{n}{2}.
\end{aligned}
$$

□

Mit diesem Satz beweisen wir nun Theorem 5.1.7. Nach Satz 5.2.9 gilt

$$
w(E_S) \geq \frac{n}{2K} S_K \geq \frac{n2W(K-1) - n\sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)}}{2K(n-1)},
$$

sodass mit Satz 5.2.8 wegen $|C| = |N_S|$ und $\omega(x_i, x_j) = w_{ij}$ die Aussage von Theorem 5.1.7 folgt.

Beweis von Theorem 5.1.8

Für diese Abschätzung können wir ein Ergebnis von Zhou [81] benutzen oder Satz 5.2.17 auf einen ungewichteten Graph anwenden, d.h., es ist $w_{ij} = 1$ für alle $(i, j) \in E$. Das liefert uns

$$
\begin{aligned}
S_k \geq & \frac{2W(K-1) - \sqrt{(K-1)(n-K)(n(n-1)\sum_{(i,j)\in E} w_{ij}^2 - 2W^2)}}{n-1} \\
= & \frac{2|E|(K-1) - \sqrt{(K-1)(n-K)(n(n-1)|E| - 2|E|^2)}}{n-1} \\
= & \frac{2|E|(K-1) - \sqrt{(K-1)(n-K)|E|(n^2-n-2|E|)}}{n-1},
\end{aligned}
$$

und mit Korollar 5.2.10 folgt

$$
|E_S| \geq \frac{n}{2K} \frac{2|E|(K-1) - \sqrt{(K-1)(n-K)|E|(n^2-n-2|E|)}}{n-1}.
$$

Nach Satz A.1.2 gilt $\sum_{i=1}^{n} d(v_i) = 2|E|$, und daher folgt wegen $|C| = |N_S|$ und $\gamma_i = d_i$ mit Satz 5.2.8 die Behauptung.

5.2.3 Abschätzungen durch einen vollständigen Graph

Wenn der CNG des Hypergraphen vollständig ist, können wir die Cutsize einer Partition des CNG schnell angeben, daher finden wir dann auch schnell untere Schranken für $|C|$.

Beweis von Theorem 5.1.9

Wenn jede Variable mit jeder anderen Variable verbunden ist, ist der CNG des assoziierten Hypergraphen ein vollständiger Graph, weshalb die Cutsize dann nach Satz 5.2.3

$$|E_S| = \binom{n}{2} - \sum_{k=1}^{K} \binom{|V_k|}{2}$$

beträgt. Wegen $|C| = |N_S|$ und $n_k = |V_k|$ folgt mit Satz 5.2.8 die Aussage von Theorem 5.1.9.

Beweis von Korollar 5.1.10

Diese Abschätzung erhalten wir durch Einsetzen. Es gilt

$$\begin{aligned} |E_S| &= \frac{n(n-1)}{2} - K\binom{\frac{n}{K}}{2} \\ &= \frac{n(n-1)}{2} - K\frac{\frac{n}{K}\left(\frac{n}{K}-1\right)}{2} \\ &= \frac{n}{2}\left(n - 1 - \frac{n}{K} + 1\right) \\ &= \frac{n^2}{2}\left(1 - \frac{1}{K}\right). \end{aligned}$$

Wegen $|C| = |N_S|$ folgt mit Satz 5.2.8 die Behauptung.

Beweis von Korollar 5.1.11

Hier können wir direkt das Ergebnis von Satz 5.2.4 benutzen. Wegen $|C| = |N_S|$ folgt mit Satz 5.2.8 die Behauptung.

5.2.4 Abschätzungen durch die Konnektivität

Auch über die Konnektivität der Graph-Repräsentation des assoziierten Hypergraphen können wir untere Schranken herleiten, wobei wir zwischen Knoten- und Kanten-Konnektivität unterscheiden müssen.

Die Kanten-Konnektivität stellt eine untere Schranke für die Cutsize einer GPES dar.

Satz 5.2.18 *Sei G ein Graph, $\kappa_E(G)$ die Kanten-Konnektivität von G und $|E_S|$ die Cutsize einer GPES von G. Dann gilt*

$$|E_S| \geq \kappa_E(G).$$

Beweis: Angenommen, es gäbe eine GPES mit $|E_S| < \kappa_E(G)$. Dann könnten wir durch Entfernen der Kanten in E_S den Zusammenhang von G zerstören. Dies ist aber ein Widerspruch, da wir dazu mindestens $\kappa_E(G)$ Kanten benötigen. □

Eine ähnliche Aussage können wir für eine GPVS herleiten.

Satz 5.2.19 *Sei $\kappa_V(G)$ die Knoten-Konnektivität eines Graphen G und $|V_S|$ die Cutsize einer GPVS von G. Dann gilt*

$$|V_S| \geq \kappa_V(G)\,.$$

Beweis: Angenommen, es gäbe eine GPVS mit $|V_S| < \kappa_V(G)$. Dann wäre es möglich, den Zusammenhang von G durch Entfernen der Knoten in V_S zu zerstören. Dies ist ein Widerspruch, denn dazu sind mindestens $\kappa_V(G)$ Knoten nötig. □

Beweis von Theorem 5.1.12

Für diese Abschätzung nutzen wir das folgende Ergebnis von Esfahanian [23].

Satz 5.2.20 *Es gilt*

$$\kappa_E \geq \min\left\{\frac{n(d_n-2)}{(d_n-1)^{diam-1}+d_n(d_n-2)-1}, d_1\right\}\,.$$

Hier benutzen wir also zum ersten Mal den Diameter. Wenn jede Variable mit jeder anderen Variablen verbunden ist, dann ist im CNG jeder Knoten mit jedem anderen Knoten verbunden und daher $diam = 1$. Wegen $d_n = n-1$ folgt mit Satz 5.2.18 und Satz 5.2.20

$$\begin{aligned}|E_S| \geq \kappa_E &\geq \min\left\{\frac{n(n-3)}{1+(n-1)(n-3)-1}, n-1\right\}\\ &= \min\left\{\frac{n}{n-1}, n-1\right\}\\ &= \frac{n}{n-1}\,.\end{aligned}$$

Da κ_E ganzzahlig ist und $2 \geq \frac{n}{n-1} > 1$, folgt $|E_S| \geq \kappa_E \geq 2$ und wegen $|C| = |N_S|$ mit Satz 5.2.8 die Behauptung.

Beweis von Theorem 5.1.13

Um Theorem 5.1.13 zu beweisen, nutzen wir erneut Satz 5.2.20 und schätzen die Kanten-Konnektivität durch Satz 5.2.18 nach unten ab. Da es eine Variable x_j gibt, die mit allen anderen Variablen verbunden ist, gibt es im CNG einen Knoten v_j, der mit allen anderen Knoten verbunden ist. Diesen Knoten können wir als Verbindungsknoten zwischen zwei anderen Knoten nutzen, weshalb es zu jedem Paar (u, w) einen Weg (u, v_j, w) der Länge 2 gibt, und wir $diam \leq 2$ folgern können.

Mit $d_n = n - 1$ erhalten durch Satz 5.2.20

$$\begin{aligned} |E_S| \geq \kappa_E &\geq \min\left\{\frac{n(n-3)}{(n-2)^{2-1} + (n-3)(n-1) - 1}, d_1\right\} \\ &= \min\left\{\frac{n(n-3)}{n-3+(n-1)(n-3)}, d_1\right\} \\ &= \min\left\{\frac{n(n-3)}{n(n-3)}, d_1\right\} \\ &= 1 \end{aligned}$$

und wegen $|C| = |N_S|$ folgt mit Satz 5.2.8 die Aussage von Theorem 5.1.12.

Beweis von Theorem 5.1.14

Hier verwenden wir ein weiteres Ergebnis von Esfahanian [23].

Satz 5.2.21 *Es gilt*

$$\kappa_V \geq \frac{n(d_n - 2)}{(d_n - 1)^{diam} + d_n + 3}.$$

Diese Abschätzung können wir auch auf κ_E anwenden, da nach Satz 2.1.7 $\kappa_E \geq \kappa_V$ gilt. Weil alle Variablen miteinander verbunden sein sollen, ist der CNG vollständig, und es gilt $diam = 1$. Mit $d_n = n - 1$ liefert Satz 5.2.21

$$\begin{aligned} |E_S| \geq \kappa_E \geq \kappa_V &\geq \frac{n(n-3)}{n-2+n-1+3} \\ &= \frac{n(n-3)}{2n} \\ &= \frac{n-3}{2}, \end{aligned}$$

und wegen $|C| = |N_S|$ folgt mit Satz 5.2.8 die Behauptung.

Für $diam \geq 2$ wird die Abschätzung in Satz 5.2.21 kleiner als 1, sodass wir nur $|E_S| \geq 1$ erhalten.

Beweis von Theorem 5.1.15

Da Satz 5.2.21 eine Abschätzung für die Knoten-Konnektivität liefert, können wir diese wegen Satz 5.2.19 auch auf die Cutsize $|V_S|$ einer GPVS des NIG anwenden. Da jeder Knoten im NIG eine Nebenbedingung repräsentiert und alle Nebenbedingungen miteinander verbunden sind, ist der NIG ein vollständiger Graph, d.h. $diam = 1$. Da der NIG m Knoten hat, erhalten wir mit $d_m = m - 1$

$$\begin{aligned} |V_S| \geq \kappa_V &\geq \frac{m(m-3)}{m-2+m-1+3} \\ &= \frac{m(m-3)}{2m} \\ &= \frac{m-3}{2}. \end{aligned}$$

Mit Satz 2.2.2 und $|C| = |N_S|$ folgt die Aussage von Theorem 5.1.15.

Für $diam \geq 2$ erhalten wir die Abschätzung $|V_S| \geq 1$, sodass unbedingt alle Nebenbedingungen verbunden sein müssen, damit wir durch Satz 5.2.21 eine sinnvolle Abschätzung über den NIG erhalten.

Beweis von Theorem 5.1.16

Diese Abschätzung beruht auf einem Ergebnis von Lesniak [57].

Satz 5.2.22 *Sei $G = (V, E)$ ein Graph mit $d(u) + d(v) \geq n - 1$ für alle $(u, v) \notin E$. Dann gilt $\kappa_E = d_1$.*

Im CNG sind zwei Knoten genau dann nicht verbunden, wenn die beiden entsprechenden Variablen in keiner gemeinsamen Nebenbedingung auftauchen. Wegen $\gamma_1 = d_1$ und $|C| = |N_S|$ folgt mit Satz 5.2.18 und Satz 5.2.8 die Aussage von Theorem 5.1.16.

Beweis von Theorem 5.1.17

Ein *Digraph* ist ein Graph $D = (V, E)$, dessen Kanten gerichtet sind. Für einen Knoten $v \in V$ ist der Außengrad $d^+(v)$ die Anzahl aller Kanten, die v als Startpunkt haben, und der Innengrad $d^-(v)$ die Anzahl aller Kanten, die v als Endpunkt haben. Der Grad $d(v)$ ist dann das Minimum von Außengrad und Innengrad, d.h. $d(v) = \min\{d^+(v), d^-(v)\}$. Ein Digraph ist *symmetrisch*, wenn zu jeder Kante auch die entgegen gerichtete Kante existiert.

Um die Aussage von Theorem 5.1.17 zu zeigen, wandeln wir den CNG G in einen Digraph D um. Dazu entfernen wir genügend Kanten, sodass wir einen d_1-regulären Graph G' erhalten. Nun ersetzen wir jede verbliebene Kante in G' durch zwei gerichtete Kanten mit entgegengesetzter Orientierung. Auf diese Art und Weise erhalten wir einen symmetrischen, d_1-regulären Digraph D.

Nun benutzen wir das folgende Ergebnis, das in [3] bewiesen wird.

Satz 5.2.23 *Sei $D = (V, E)$ ein symmetrischer, d-regulärer Digraph. Dann gilt*

$$\kappa_E = d\,.$$

Da jede Kante in G' durch zwei gerichtete Kanten in D repräsentiert wird, ist das Entfernen einer Kante aus G' äquivalent zum Entfernen der entsprechenden gerichteten Kanten in D. Insbesondere müssen wir also doppelt so viele Kanten aus D entfernen wie aus G', um einen unzusammenhängenden Graph zu bekommen, d.h. $\kappa_E(D) = 2\kappa_E(G')$. Da G' ein Teilgraph von G ist, gilt natürlich $\kappa_E(G') \leq \kappa_E(G)$, sodass mit Satz 5.2.23

$$d_1 = \kappa_E(D) = 2\kappa_E(G') \leq 2\kappa_E(G) \;\Leftrightarrow\; \kappa(G) \geq \frac{d_1}{2}$$

folgt. Nach Satz 5.2.18 gilt dann auch

$$|E_S| \geq \kappa_E(G) \geq \frac{d_1}{2}\,,$$

und wegen $|C| = |N_S|$ folgt mit Satz 5.2.8 die Behauptung.

Es ist natürlich von Vorteil, wenn die Knotengrade in etwa gleich groß sind, da wir dann nicht viele Kanten entfernen müssen, um einen d_1-regulären Graph zu bekommen.

Beweis von Theorem 5.1.18

Auch dieses Ergebnis folgt aus einer Abschätzung für einen Digraph, das wir diesmal auf den NIG G anwenden. Wie schon im Beweis zu Theorem 5.1.17 verkleinern wir diesen zunächst so lange, bis wir einen d_1-regulären Graph G' haben. Danach wandeln wir den Graph G' wie oben in einen Digraph D um, indem wir jede Kante in G' durch zwei gerichtete Kanten mit entgegengesetzter Orientierung ersetzen.

Das folgende Resultat [3] liefert eine Abschätzung für die Knoten-Konnektivität eines Digraphen.

Satz 5.2.24 *Sei $D = (V, E)$ ein symmetrischer, d-regulärer Digraph. Dann gilt*

$$\kappa_V \geq \frac{2(d+1)}{3}$$

und $\kappa_V = d$, wenn $d \leq 4$.

Eine Knotenmenge ist genau dann ein Vertex Separator von D, wenn sie ein Vertex Separator von G' ist. Diese Aussage gilt, da die Orientierung der Kanten nicht von Belang ist, wenn wir Knoten entfernen. Wir können daher

$$\kappa_V(D) = \kappa_V(G') \leq \kappa_V(G)$$

annehmen.

Nach Satz 5.2.24 und Satz 5.2.19 gilt

$$\frac{2(d_1+1)}{3} \leq \kappa_V(D) \leq \kappa_V(G) \leq |V_S|\,,$$

und wegen $|C| = |N_S|$ folgt mit Satz 2.2.2 die Behauptung.

Auch bei diesem Ergebnis ist es natürlich gut, wenn die Knotengrade in etwa gleich groß sind, damit wir nicht zu viele Kanten entfernen müssen. In Satz 5.2.24 gilt zwar Gleichheit, wenn $d_1 \leq 4$, doch diese Gleichheit gilt dann nicht automatisch für $|C|$. Denn, wenn wir von einer Partition eine gewisse Balanciertheit verlangen, kann die Cutsize $|V_S|$ dieser Partition durchaus größer als κ_V sein. Wenn wir aber keine Balanciertheit einfordern, hat ein Vertex Separator mit minimaler Knotenzahl genau $\kappa_V(G)$ Knoten, und somit können wir die Gleichheit auch auf $|C|$ übertragen.

5.3 Beispiele

Natürlich wollen wir die unteren Schranken für die Anzahl der Coupling Constraints auch testen. Dazu betrachten wir in zwei Beispielen zwei Matrizen, wobei die zweite Matrix einen vollständigen CNG induziert.

5.3.1 Beispiel 1

Wir betrachten die Matrix

$$\begin{pmatrix}
\times & & & \times & & & & & \\
 & \times & & & \times & & & & \\
 & \times & & & \times & \times & \times & & \times \\
\times & & & \times & & \times & & & \\
 & & & \times & \times & \times & \times & \times & \\
\times & & \times & \times & & & & \times & \\
 & & & \times & & \times & & & \\
 & & & & \times & & & \times & \\
 & & \times & & & & \times & & \times \\
 & \times & & & & & & \times & \\
\times & \times & & & & & \times & & \times \\
 & & & & & & \times & & \times \\
 & & \times & & & & & & \times \\
\times & \times & & & & \times & & & \times
\end{pmatrix}$$

eines linearen Programms mit neun Variablen und 14 Nebenbedingungen. Wenn wir diese Matrix in SB-Form mit drei Blöcken und drei Variablen pro Block bringen, erhalten wir die Matrix

$$\begin{pmatrix}
\times & & \times & & & & & & \\
 & \times & \times & & & & & & \\
\times & \times & & & & & & & \\
 & & & \times & \times & \times & & & \\
 & & & & \times & \times & & & \\
 & & & \times & & \times & & & \\
 & & & & & & \times & & \times \\
 & & & & & & \times & \times & \times \\
 & & & & & & \times & \times & \\
\times & & \times & & \times & & \times & & \times \\
 & \times & & \times & & \times & & \times & \\
\times & & & \times & \times & & \times & & \\
 & \times & \times & & \times & \times & & & \times \\
\times & & & \times & & & \times & & \times
\end{pmatrix}$$

mit vier Coupling Constraints, d.h. $|C| = 4$. Nun wollen wir sehen, wie gut wir $|C|$ mit den Ergebnissen aus Kapitel 5.1 abschätzen können. Dazu bestimmen wir zunächst die Mengen $\Gamma(x_j)$ und somit auch $\gamma(x_j)$ für alle Variablen x_j, $j = 1, \ldots, 9$. Danach errechnen wir die Gewichte der Nebenbedingungen, aus denen sich dann die Gewichte der Variablen-Paare ergeben. Zudem werden wir noch die Mengen $\Delta(r_i)$ und dadurch auch $\delta(r_i)$ für alle Nebenbedingungen r_i, $i = 1, \ldots, 14$ bestimmen.

Um die Menge $\Gamma(x_j)$ einer Variable x_j zu bestimmen, erfassen wir die Variablen, die

in den Nebenbedingungen von x_j auftauchen. Wir erhalten

$$\begin{aligned}
\Gamma(x_1) &= \{x_2, x_3, x_4, x_6, x_7, x_8, x_9\}, & \gamma(x_1) &= 7,\\
\Gamma(x_2) &= \{x_1, x_5, x_6, x_7, x_8, x_9\}, & \gamma(x_2) &= 6,\\
\Gamma(x_3) &= \{x_1, x_4, x_7, x_8, x_9\}, & \gamma(x_3) &= 5,\\
\Gamma(x_4) &= \{x_1, x_3, x_5, x_6, x_7, x_8\}, & \gamma(x_4) &= 6,\\
\Gamma(x_5) &= \{x_2, x_4, x_6, x_7, x_8, x_9\}, & \gamma(x_5) &= 6,\\
\Gamma(x_6) &= \{x_1, x_2, x_4, x_5, x_7, x_8, x_9\}, & \gamma(x_6) &= 7,\\
\Gamma(x_7) &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_8, x_9\}, & \gamma(x_7) &= 8,\\
\Gamma(x_8) &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}, & \gamma(x_8) &= 7,\\
\Gamma(x_9) &= \{x_1, x_2, x_3, x_5, x_6, x_7\}, & \gamma(x_9) &= 6
\end{aligned}$$

und

$$\gamma_1 = 5\,,\ \gamma_2 = \ldots = \gamma_5 = 6,\ \gamma_6 = \ldots = \gamma_8 = 7,\ \gamma_9 = 8\,.$$

Das Gewicht der $i-$ten Nebenbedingungen ist definiert durch

$$\omega(r_i) := \frac{1}{\binom{|r_i|}{2} - \binom{\left\lfloor \frac{|r_i|}{K} \right\rfloor}{2}\left(K + K\left\lfloor \frac{|r_i|}{K} \right\rfloor - |r_i|\right) - \binom{\left\lceil \frac{|r_i|}{K} \right\rceil}{2}\left(|r_i| - K\left\lfloor \frac{|r_i|}{K} \right\rfloor\right)}\,.$$

Hier ist $K = 3$ und

$$\begin{aligned}
&|r_1| = 2\,,\ \ |r_2| = 2\,,\ \ |r_3| = 5\,,\ \ |r_4| = 3\,,\ \ |r_5| = 5\,,\ \ |r_6| = 4\,,\ \ |r_7| = 2\,,\\
&|r_8| = 2\,,\ \ |r_9| = 3\,,\ \ |r_{10}| = 2\,,\ \ |r_{11}| = 4\,,\ \ |r_{12}| = 2\,,\ \ |r_{13}| = 2\,,\ \ |r_{14}| = 4\,,
\end{aligned}$$

weshalb wir die Gewichte

$$\begin{aligned}
&\omega(r_1) = 1\,,\ \ \omega(r_2) = 1\,,\ \ \omega(r_3) = \frac{1}{8}\,,\ \ \omega(r_4) = \frac{1}{3}\,,\ \ \omega(r_5) = \frac{1}{8}\,,\ \ \omega(r_6) = \frac{1}{5}\,,\ \ \omega(r_7) = 1\,,\\
&\omega(r_8) = 1\,,\ \ \omega(r_9) = \frac{1}{3}\,,\ \ \omega(r_{10}) = 1\,,\ \ \omega(r_{11}) = \frac{1}{5}\,,\ \ \omega(r_{12}) = 1\,,\ \ \omega(r_{13}) = 1\,,\ \ \omega(r_{14}) = \frac{1}{5}
\end{aligned}$$

erhalten. Die Gewichte sind also nicht so klein, da die einzelnen Nebenbedingungen eher wenige Variablen haben. Aus den Gewichten der Nebenbedingungen können wir jetzt die Gewichte der Variablen-Paare bestimmen, so ist etwa

$$\omega(x_1, x_4) = \omega(r_1) + \omega(r_4) + \omega(r_6) = 1 + \frac{1}{3} + \frac{1}{5} = \frac{23}{15}\,.$$

Insgesamt gibt es 29 Variablen-Paare, die in gemeinsamen Nebenbedingungen auftauchen. Die Gewichte dieser Variablen-Paare errechnen sich zu

$$\begin{aligned}
&\omega(x_1, x_2) = \frac{2}{5}\,,\ \omega(x_1, x_3) = \frac{1}{5}\,,\ \omega(x_1, x_4) = \frac{23}{15}\,,\ \omega(x_1, x_6) = \frac{8}{15}\,,\ \omega(x_1, x_7) = \frac{1}{5}\,,\\
&\omega(x_1, x_8) = \frac{1}{5}\,,\ \omega(x_1, x_9) = \frac{2}{5}\,,\ \omega(x_2, x_5) = \frac{9}{8}\,,\ \omega(x_2, x_6) = \frac{13}{40}\,,\ \omega(x_2, x_7) = \frac{13}{40}\,,\\
&\omega(x_2, x_8) = 1\,,\ \omega(x_2, x_9) = \frac{21}{40}\,,\ \omega(x_3, x_4) = \frac{1}{5}\,,\ \omega(x_3, x_7) = \frac{1}{3}\,,\ \omega(x_3, x_8) = \frac{1}{5}\,,\\
&\omega(x_3, x_9) = \frac{4}{3}\,,\ \omega(x_4, x_5) = \frac{1}{8}\,,\ \omega(x_4, x_6) = \frac{35}{24}\,,\ \omega(x_4, x_7) = \frac{1}{8}\,,\ \omega(x_4, x_8) = \frac{13}{40}\,,\\
&\omega(x_5, x_6) = \frac{1}{4}\,,\ \omega(x_5, x_7) = \frac{1}{4}\,,\ \omega(x_5, x_8) = \frac{9}{8}\,,\ \omega(x_5, x_9) = \frac{1}{8}\,,\ \omega(x_6, x_7) = \frac{1}{8}\,,\\
&\omega(x_6, x_8) = \frac{1}{8}\,,\ \omega(x_6, x_9) = \frac{1}{5}\,,\ \omega(x_7, x_8) = \frac{1}{8}\,,\ \omega(x_7, x_9) = \frac{23}{15}
\end{aligned}$$

bzw. in aufsteigender Reihenfolge

$$\omega_1 = \ldots = \omega_6 = \frac{1}{8}\,,\ \ \omega_7 = \ldots = \omega_{12} = \frac{1}{5}\,,\ \ \omega_{13} = \omega_{14} = \frac{1}{4}\,,$$
$$\omega_{15} = \ldots = \omega_{17} = \frac{13}{40}\,,\ \ \omega_{18} = \frac{1}{3}\,,\ \ \omega_{19} = \omega_{20} = \frac{2}{5}\,,$$
$$\omega_{21} = \frac{21}{40}\,,\ \ \omega_{22} = \frac{8}{15}\,,\ \ \omega_{23} = 1\,,\ \ \omega_{24} = \omega_{25} = \frac{9}{8}\,,$$
$$\omega_{26} = \frac{4}{3}\,,\ \ \omega_{27} = \frac{35}{24}\,,\ \ \omega_{28} = \omega_{29} = \frac{23}{15}\,.$$

Das Gesamtgewicht aller Variablen-Paare ergibt sich zu $\mathcal{W} = \frac{589}{40}$.

Nun bestimmen wir zusätzlich noch die Mengen $\Delta(r_i)$ für die Nebenbedingungen r_i, $i = 1, \ldots, 14$, und erhalten

$$\begin{aligned}
\Delta(r_1) &= \{r_4, r_5, r_6, r_7, r_{11}, r_{14}\}\,, & \delta(r_1) &= 6\,,\\
\Delta(r_2) &= \{r_3, r_5, r_8, r_{10}, r_{11}, r_{14}\}\,, & \delta(r_2) &= 6\,,\\
\Delta(r_3) &= \{r_2, r_4, r_5, r_7, r_8, r_9, r_{11}, r_{12}, r_{13}, r_{14}\}\,, & \delta(r_3) &= 10\,,\\
\Delta(r_4) &= \{r_1, r_3, r_5, r_6, r_7, r_{11}, r_{14}\}\,, & \delta(r_4) &= 7\,,\\
\Delta(r_5) &= \{r_1, r_2, r_3, r_4, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{14}\}\,, & \delta(r_5) &= 12\,,\\
\Delta(r_6) &= \{r_1, r_4, r_4, r_7, r_8, r_9, r_{10}, r_{11}, r_{13}, r_{14}\}\,, & \delta(r_6) &= 10\,,\\
\Delta(r_7) &= \{r_1, r_3, r_4, r_5, r_6, r_{14}\}\,, & \delta(r_7) &= 6\,,\\
\Delta(r_8) &= \{r_2, r_3, r_5, r_6, r_{10}\}\,, & \delta(r_8) &= 5\,,\\
\Delta(r_9) &= \{r_3, r_5, r_6, r_{11}, r_{12}, r_{13}, r_{14}\}\,, & \delta(r_9) &= 7\,,\\
\Delta(r_{10}) &= \{r_2, r_5, r_6, r_8\}\,, & \delta(r_{10}) &= 4\\
\Delta(r_{11}) &= \{r_1, r_2, r_3, r_4, r_5, r_6, r_9, r_{12}, r_{13}, r_{14}\}\,, & \delta(r_{11}) &= 10\\
\Delta(r_{12}) &= \{r_3, r_5, r_9, r_{11}, r_{13}, r_{14}\}\,, & \delta(r_{12}) &= 6\\
\Delta(r_{13}) &= \{r_3, r_6, r_9, r_{11}, r_{12}, r_{14}\}\,, & \delta(r_{13}) &= 6\\
\Delta(r_{14}) &= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_9, r_{11}, r_{12}, r_{13}\}\,, & \delta(r_{14}) &= 11\,,
\end{aligned}$$

d.h. $\delta_1 = 4$.
Damit haben wir alle notwendigen Werte bestimmt und können sie auf die Abschätzungen anwenden.

Abschätzung durch Theorem 5.1.4

Es ist

$$\gamma_1 = 5 > \frac{7}{2} = \frac{n}{2} - 1\,,$$

weshalb wir dieses Theorem auf die Matrix anwenden können. Da $n_1 = 3$, erhalten wir

$$\tau = \left\lceil \frac{(2\gamma_1 - n + 2)(n - n_1)}{2} \right\rceil = \left\lceil \frac{(10 - 9 + 2)(9 - 3)}{2} \right\rceil = 9$$

und somit nach Satz 5.1.3

$$|C| \geq \left\lceil \sum_{i=1}^{9} \omega_i \right\rceil = \left\lceil 6 \cdot \frac{1}{8} + 3 \cdot \frac{1}{5} \right\rceil = 2\,.$$

Theorem 5.1.4 liefert also eine Abschätzung, die halb so groß ist, wie die tatsächliche Anzahl der Coupling Constraints.

Abschätzung durch Theorem 5.1.5

Da alle Blöcke gleich viele Variablen haben sollen, wenden wir direkt Korollar 5.1.6 an. Wegen

$$2\gamma_1 + \gamma_3 = 16 > 12 = n(K-1) - \frac{K(K+3)}{2} + 3$$

können wir das Theorem anwenden und erhalten

$$\begin{aligned}\tau &= \left\lceil \frac{n}{2K}\left(2\gamma_1 - n(K-1) + \frac{K(K+3)}{2} - 3 + \sum_{k=3}^{K}\gamma_k\right)\right\rceil \\ &= \left\lceil \frac{9}{6}(16-12)\right\rceil = 6\,.\end{aligned}$$

Nach Satz 5.1.3 ist daher

$$|C| \geq \left\lceil \sum_{i=1}^{6}\omega_i \right\rceil = \left\lceil 6 \cdot \frac{1}{8}\right\rceil = 1\,.$$

Theorem 5.1.5 liefert also eine schlechtere Abschätzung als Theorem 5.1.4, obwohl wir im Beweis zu Theorem 5.1.5 jeden Eigenwert einzeln abgeschätzt haben statt durch λ_2. Allerdings ist die Abschätzung für λ_2 aus Satz 5.2.12 teilweise besser als die Abschätzungen für die λ_k aus Satz 5.2.13, vor allem für kleine Graphen mit ähnlich großen Graden d_k. Für Partitionen von größeren Matrizen mit mehr Blöcken dürfte Theorem 5.1.5 daher bessere Abschätzungen liefern als Theorem 5.1.4.

Abschätzung durch Theorem 5.1.7

Wir überprüfen die Voraussetzung des Theorems und erhalten

$$\frac{\mathcal{W}^2}{\sum_{1\leq j<l\leq n}\omega(x_j,x_l)^2} = 15,5936 < 21,6 = \frac{n-K}{n+K-2}\binom{n}{2}.$$

Also können wir das Theorem nicht anwenden.

Abschätzung durch Theorem 5.1.8

Wir versuchen es mit der vereinfachten Version von Theorem 5.1.7. Da

$$\sum_{j=1}^{n}\gamma(x_j) = 58 > 43,2 = \frac{n(n-1)(n-K)}{n+K-2}\,,$$

ist die Voraussetzung des Theorem erfüllt, und wir erhalten

$$\begin{aligned}\tau &= \left\lceil \frac{n(K-1)\sum_{j=1}^{n}\gamma(x_j) - n\sqrt{(K-1)(n-K)\frac{1}{2}\sum_{j=1}^{n}\gamma(x_j)(n^2-n-\sum_{j=1}^{n}\gamma(x_j))}}{2K(n-1)}\right\rceil \\ &= \left\lceil \frac{415,8026}{48}\right\rceil = 9\,.\end{aligned}$$

Theorem 5.1.8 liefert also wie schon Theorem 5.1.4 mit Satz 5.1.3 die untere Schranke

$$|C| \geq 2\,.$$

Abschätzung durch Theorem 5.1.13

Da die Variable x_9 mit allen anderen Variablen verbunden ist, können wir Theorem 5.1.13 anwenden und erhalten

$$|C| \geq \lceil \omega_1 \rceil = \left\lceil \frac{1}{8} \right\rceil = 1 .$$

Wie erwartet ist die Abschätzung durch dieses Theorem nicht besonders gut.

Abschätzung durch Theorem 5.1.16

Es gilt

$$\gamma(x_j) + \gamma(x_k) \geq 5 + 6 = 11 > 8 = n - 1 ,$$

und daher können wir das Ergebnis von Theorem 5.1.16 benutzen. Wir erhalten also mit Satz 5.1.3

$$|C| \geq \left\lceil \sum_{i=1}^{\gamma_1} \omega_i \right\rceil = \left\lceil 5 \cdot \frac{1}{8} \right\rceil = 1 .$$

Abschätzung durch Theorem 5.1.17

Für dieses Theorem brauchen wir keine Voraussetzung zu überprüfen. Mit Satz 5.1.3 erhalten wir direkt

$$|C| \geq \left\lceil \sum_{i=1}^{3} \omega_i \right\rceil = \left\lceil 3 \cdot \frac{1}{8} \right\rceil = 1 .$$

Abschätzung durch Theorem 5.1.18

Für diese Abschätzung benötigen wir δ_1 statt den γ_i.

$$|C| \geq \left\lceil \frac{2(\delta_1 + 1)}{3} \right\rceil = \left\lceil \frac{10}{3} \right\rceil = 4 .$$

Nachdem $|C| \geq 2$ bislang die beste gefundene Abschätzung war, können wir die Anzahl der Coupling Constraints durch Theorem 5.1.18 exakt abschätzen. Wegen $\delta_1 = 4$ hätten wir für eine unbalancierte Partition der Matrix sogar direkt $|C| = 4$ folgern können.

Allerdings wäre dies auch ohne Theorem 5.1.18 möglich gewesen, da wir drei Variablen pro Block haben wollen und es vier Nebenbedingungen mit mehr als drei Variablen gibt, die dann auf jeden Fall zu Coupling Constraints werden müssen. Bei einer Partition mit zwei Blöcken wäre dies nicht so trivial gewesen, da dann Blöcke mit vier Variablen möglich gewesen wären und wir nicht gewusst hätten, ob die Nebenbedingungen mit den vier Variablen in die Blöcke kommen können oder in den Rand müssen. Theorem 5.1.18 liefert dann also keine triviale Abschätzung.

5.3.2 Beispiel 2

Wir konnten bislang noch nicht alle Theoreme anwenden, u.a., weil nicht alle Variablen untereinander verbunden waren. Deshalb betrachten wir in einem zweiten Beispiel die Matrix

$$\begin{pmatrix}
 & \times & \times & & & & & & \\
\times & & & & & & & & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & & & \times & \\
 & & & & & \times & & & \times \\
 & \times & \times & & & \times & & \times & \\
 & \times & \times & & & & \times & & \\
\times & & & \times & \times & & \times & & \times \\
 & & & \times & & & & \times &
\end{pmatrix}$$

eines linearen Programms mit neun Variablen und neun Nebenbedingungen. Wenn wir diese Matrix in drei Blöcke mit jeweils drei Variablen partitionieren wollen, erhalten wir die SB-Form

$$\begin{pmatrix}
\times & & \times & & & & & & \\
 & \times & \times & & & & & & \\
 & & & \times & & \times & & & \\
 & & & \times & \times & \times & & & \\
 & & & & & & \times & \times & \\
 & & & & & & & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & & \times & & \times & & \times & & \times \\
 & \times & & \times & & \times & & \times &
\end{pmatrix}$$

mit drei Coupling Constraints. Da in der siebten Nebenbedingung alle Variablen vorkommen, sind alle Variablen untereinander verbunden und es gilt $\gamma(x_j) = 8$ für alle $j = 1, \ldots, 9$.

Wir bestimmen zunächst wieder die Gewichte der Nebenbedingungen und erhalten

$$\omega(r_1) = 1\,,\ \ \omega(r_2) = 1\,,\ \ \omega(r_3) = \frac{1}{27}\,,\ \ \omega(r_4) = 1\,,\ \ \omega(r_5) = 1\,,$$
$$\omega(r_6) = \frac{1}{5}\,,\ \ \omega(r_7) = \frac{1}{3}\,,\ \ \omega(r_8) = \frac{1}{8}\,,\ \ \omega(r_9) = 1\,.$$

Bis auf das Gewicht der dritten Nebenbedingung sind die Gewichte also ähnlich groß wie bei der Matrix in Beispiel 1. Da die dritte Nebenbedingung alle Variablen benutzt, ist das Gewicht entsprechend klein. Dies könnte sich negativ auf die Abschätzungen durch Satz 5.1.3 auswirken, da wir dazu die kleinsten Gewichte verwenden.

Da jede Variable über die dritte Nebenbedingung mit jeder anderen Variablen verbunden ist, beträgt jedes Gewicht $\omega(x_j, x_k)$ zweier Variablen x_j und x_k mindestens $\frac{1}{27}$. Wir berechnen daher nur die Gewichte der 21 Variablen-Paare, die auch in anderen Neben-

bedingungen auftauchen, diese sind:

$$\omega(x_1, x_4) = \frac{35}{216}, \quad \omega(x_1, x_5) = \frac{35}{216}, \quad \omega(x_1, x_7) = \frac{35}{216}, \quad \omega(x_1, x_9) = \frac{251}{216}, \quad \omega(x_2, x_3) = \frac{212}{135},$$
$$\omega(x_2, x_6) = \frac{32}{135}, \quad \omega(x_2, x_7) = \frac{10}{27}, \quad \omega(x_2, x_8) = \frac{32}{135}, \quad \omega(x_3, x_6) = \frac{32}{135}, \quad \omega(x_3, x_7) = \frac{10}{27},$$
$$\omega(x_3, x_8) = \frac{32}{135}, \quad \omega(x_4, x_5) = \frac{35}{216}, \quad \omega(x_4, x_7) = \frac{35}{216}, \quad \omega(x_4, x_8) = \frac{28}{27}, \quad \omega(x_4, x_9) = \frac{35}{216},$$
$$\omega(x_5, x_7) = \frac{35}{216}, \quad \omega(x_5, x_8) = \frac{28}{27}, \quad \omega(x_5, x_9) = \frac{35}{216}, \quad \omega(x_6, x_7) = \frac{35}{216}, \omega(x_6, x_8) = \frac{32}{135},$$
$$\omega(x_6, x_9) = \frac{28}{27}.$$

Die Gewichte der restlichen 15 Variablen-Paare betragen $\frac{1}{27}$. Wir erhalten also

$$\omega_1 = \ldots = \omega_{15} = \frac{1}{27}, \quad \omega_{16} = \ldots = \omega_{24} = \frac{35}{216}, \quad \omega_{25} = \ldots = \omega_{29} = \frac{32}{135},$$
$$\omega_{30} = \omega_{31} = \frac{10}{27}, \quad \omega_{32} = \ldots = \omega_{34} = \frac{28}{27}, \quad \omega_{35} = \frac{251}{216}, \quad \omega_{36} = \frac{212}{135}$$

und $\mathcal{W} = \frac{587}{60}$.

Nun können wir die Theoreme auf die Matrix anwenden.

Abschätzung durch Theorem 5.1.7

Wir versuchen erneut, Theorem 5.1.7 anzuwenden. Doch wegen

$$\frac{\mathcal{W}^2}{\sum_{1 \le j < l \le n} \omega(x_j, x_l)^2} = 12,1852 < 28,8 = \frac{n-K}{n+K-2}\binom{n}{2}$$

ist dies auch für diese Matrix nicht möglich, obwohl alle Variablen untereinander verbunden sind. Allerdings werden fast die Hälfte aller Verbindungen zwischen zwei Variablen durch die dritte Nebenbedingung gestützt, in der alle Variablen vorkommen. Entsprechend ist das Gewicht vieler Variablen-Paare sehr klein, immerhin 15 Variablen-Paare haben nur ein Gewicht von $\frac{1}{27}$. Daher wäre es besser, wenn es keine Nebenbedingung mit allen Variablen gäbe, dafür aber viele Nebenbedingungen mit wenigen Variablen, sodass trotzdem alle Variablen untereinander verbunden sind.

Die Bedingung von Theorem 5.1.8 ist allerdings wegen

$$\sum_{j=1}^{n} \gamma(x_j) = 72 > 43,2 = \frac{n(n-1)(n-K)}{n+K-2}$$

erneut erfüllt. Während es in Bezug auf Theorem 5.1.8 also gut ist, wenn alle Variablen untereinander verbunden sind, ist es in Bezug auf Theorem 5.1.7 zusätzlich besser, wenn dies nicht durch eine einzige Nebenbedingung bewirkt wird.

Abschätzung durch Theorem 5.1.10

Da alle Variablen untereinander verbunden sind, können wir Theorem 5.1.10 anwenden und erhalten

$$\tau = \frac{n^2}{2}\left(1 - \frac{1}{K}\right) = 27\,.$$

Nach Satz 5.1.3 ist dann

$$|C| \geq \left\lceil \sum_{i=1}^{2} 7\omega_i \right\rceil = \left\lceil 15 \cdot \frac{1}{27} + 9 \cdot \frac{35}{216} + 3 \cdot \frac{32}{135} \right\rceil = 3\,.$$

Theorem 5.1.10 schätzt die Anzahl der Coupling Constraints also perfekt ab.

Abschätzung durch Theorem 5.1.12

Da alle Variablen untereinander verbunden sind, liefert Theorem 5.1.10 die Abschätzung

$$|C| \geq \lceil \omega_1 + \omega_2 \rceil = \left\lceil \frac{2}{27} \right\rceil = 1\,,$$

die erwartungsgemäß eher unbrauchbar ist.

Abschätzung durch Theorem 5.1.14

Eine bessere Abschätzung erhoffen wir uns durch Theorem 5.1.14. Da alle Variablen untereinander verbunden sind, gilt hier

$$\tau = \left\lceil \frac{n-3}{2} \right\rceil = 3\,,$$

weshalb wir ebenfalls die triviale Abschätzung

$$|C| \geq 1$$

erhalten.

5.4 Diskussion

Wir haben einige Abschätzungen für die Anzahl der Coupling Constraints hergeleitet und diese an zwei Matrizen getestet. Während die meisten Theoreme für die Matrix aus Beispiel 1 keine guten Abschätzungen liefern, schätzt Theorem 5.1.18 in diesem Fall die Anzahl der Coupling Constraints perfekt ab. Der Grund dürfte sein, dass wir bei den meisten anderen Theoremen zunächst die Cutsize der entsprechenden Hypergraph-Partition $|N_S|$ durch das Cutweight $w(E_S)$ der CNG-Partition abschätzen, die wir wiederum durch die $|E_S|$ kleinsten Gewichte abschätzen, und zu guter Letzt auch noch für $|E_S|$ eine Abschätzung benutzen. Durch dieses dreimalige Abschätzen kann die eigentliche Abschätzung für $|C|$ sehr schlecht werden.

Dass die Abschätzungen durch die Kanten-Konnektivität sehr schlecht sind, verwundert nicht. Wir müssen zwar mindestens κ_E Kanten entfernen, um überhaupt einen

nicht-zusammenhängenden Graph zu bekommen, beachten dabei aber keinerlei Balanciertheit. Daher ist die Abschätzung von $|E_S|$ durch κ_E ziemlich grob.

Im zweiten Beispiel, in dem wir eine Matrix betrachten, in der alle Variablen untereinander verbunden sind, liefert mit Theorem 5.1.9 dann doch ein Theorem, das auf dem CNG beruht, eine sehr gute Abschätzung. Hier können wir ausnutzen, dass wir die Cutsize einer Partition genau bestimmen können, wenn der CNG vollständig ist, sodass die untere Schranke für $|C|$ nur noch durch zwei Abschätzungen entsteht.

In Theorem 5.1.18 nutzen wir dagegen, dass für eine Partition des NIG $|V_S| = |N_S|$ gilt, und schätzen $|V_S|$ durch die Knoten-Konnektivität ab. Allerdings dürfte Theorem 5.1.18 für größere Matrizen schlechtere Abschätzungen liefern, da die Cutsize $|V_S|$ dann wegen der Balanciertheit wesentlich größer als die Knoten-Konnektivität sein kann. Dennoch ist das NIG-Modell im Vergleich zum CNG-Modell wohl die bessere Variante, um eine Matrix über einen Graph zu partitionieren.

Beim NIG haben wir jedoch das Problem, dass wir eine Balanciertheit der Anzahl der Variablen in den Blöcken nicht direkt erreichen können, da die Knoten der Variablen im NIG nicht mehr auftauchen. Auch bei den unteren Schranken durch den NIG beachten wir die Balanciertheit nicht, bei den Abschätzungen über den CNG ist das dagegen sehr wohl möglich. In [51] wurde eine Methode vorgeschlagen, um die Balanciertheit durch Gewichte abzubilden, doch gibt es auch eine Möglichkeit, die Balanciertheit ohne Gewichte zu erzwingen?

Balanciertheit bezieht sich stets auf die Knoten eines Graphen. Diese entsprechen beim Row-Net-Modell den Nebenbedingungen, beim Column-Net-Modell allerdings den Variablen. Wenn wir also das Column-Net-Model zur Partitionierung benutzen, können wir die Anzahl der Variablen in den Blöcken über die Teilmengen $V_1, \ldots, V_K$ einer Partition des NIG balancieren. Da die Knoten der Menge V_S einer solchen Partition nun aber Variablen entsprechen, erhalten wir aus der Partition des NIG bzw. des Hypergraphen eine SB-Form mit Spaltenrand. Dann können wir entweder die Matrix-Partition zur Parallelisierung benutzen, indem der $k-$te Prozessor jeweils den Teil einer Nebenbedingung mit den Variablen aus dem $k-$ten Block berechnet, und die Ergebnisse der einzelnen Blöcke dann addiert werden, oder wir wandeln den Spaltenrand wie in Kapitel 1.2 beschrieben durch Column Splitting in einen Zeilenrand um. Wenn der Spaltenrand n_c Spalten hat, erhalten wir durch das Column Splitting einen Zeilenrand mit höchstens $(K-1)n_c$ Zeilen, allerdings kann die erzeugte SB-Form mit Zeilenrand viele Coupling Constraints haben, obwohl der Spaltenrand klein war.

Eine Balancierung der Nebenbedingungen statt der Anzahl der Variablen pro Block, die wir durch den NIG ohne Weiteres realisieren können, ist nur sinnvoll, wenn alle Nebenbedingungen ähnlich viele Variablen haben, da die Berechnungszeiten der einzelnen Nebenbedingungen sonst stark variieren können. Zudem ist die Anzahl der Nebenbedingungen pro Prozessor nur schwer voraus zu sehen, da sie von der Anzahl der Coupling Constraints abhängt.

Die unteren Schranken gelten teilweise für festgelegte Blockgrößen. Doch ist es wirklich sinnvoll, die Anzahl der Variablen von vornherein festzulegen oder sollten wir nicht einen gewissen Spielraum bei den Blockgrößen lassen, sodass die Blockgrößen von der zu partitionierenden Matrix abhängen? In der Praxis werden wir wohl eher letzteres tun, weshalb sich die Frage stellt, wie aussagekräftig die Schranken dann wirklich sind. Wir könnten zwar verschiedene Konstellationen ausprobieren und dann

die kleinste Schranke als untere Schranke wählen, doch gerade bei großen Matrizen und vielen Blöcken wird das eventuell zu aufwendig.

Generell können wird überprüfen, ob wir das Matrix-Partitionierungsproblem zu Beginn vereinfachen können. In diesem Kapitel haben wir etwa nur Matrizen betrachtet, die sich nicht in strikte Blockdiagonalgestalt permutieren lassen, da der assoziierte Graph sonst nicht zusammenhängend wäre, und einige der Ergebnisse nicht anwendbar wären. Wir können allerdings auch ausnutzen, dass der assoziierte Graph nicht zusammenhängend ist, denn wenn die Matrix eine strikte Blockdiagonalgestalt hat, können wir die Teilmatrizen separat voneinander partitionieren. Zudem gibt es oft Nebenbedingungen, die auf jeden Fall zu Coupling Constraints werden, etwa wenn diese mehr Variablen benutzen als wir pro Block zulassen. Diese Nebenbedingungen können wir beim Matrix-Partitionierungsproblem ignorieren, um nur den Teil der Matrix zu partitionieren, der nicht trivial partitionierbar ist.

Die Qualität der unteren Schranken hängt nicht nur von der Matrix ab, sondern auch von der Anzahl der Blöcke K. Wenn wir durch ein $K' \neq K$ eine größere obere Schranke erhalten, können wir daraus nicht schließen, dass wir eine bessere Abschätzung für $|C|$ gefunden haben, oder dass eine SB-Form mit K' Blöcken einen größeren Rand hat. Mit der Frage, ob wir durch eine Partition mit K Blöcken etwas über eine Partition mit $K + 1$ Blöcken sagen können, werden wir uns im nächsten Kapitel befassen.

Zu guter Letzt wäre es natürlich interessant, die unteren Schranken für die Anzahl der Coupling Constraints auch an größeren Matrizen zu testen, insbesondere an solchen mit weit über 1000 Zeilen und Spalten, wie sie in praktischen Problemen der Linearen Optimierung oft vorkommen.

6 Obere Schranken für die Anzahl der Blöcke

In Kapitel 5 haben wir uns mit der Anzahl der Coupling Constraints beschäftigt und für diese einige untere Schranken gefunden. Nun wollen wir uns mit der Anzahl der Blöcke auseinander setzen und dabei gegebenenfalls die Abschätzungen aus Kapitel 5 anwenden.

6.1 Veranschaulichung

Die Anzahl der Blöcke ist natürlich genau wie die Anzahl der Coupling Constraints nicht uninteressant. Zwar haben wir zumeist eine bestimmte Anzahl an Prozessoren zur Verfügung, sodass die Anzahl der Blöcke in der SB-Form eigentlich vorgegeben ist. Doch es muss nicht immer sinnvoll sein, so viele Prozessoren wie möglich zu verwenden, denn mit der Anzahl der Blöcke steigt eventuell auch die Anzahl der Coupling Constraints, sodass eine Partition der Matrix in K Blöcke nicht sinnvoll oder überhaupt nicht möglich sein kann, wenn K zu groß ist. Allgemein gibt es zur Anzahl der Blöcke einer Matrix-Partition nur wenige Ergebnisse, die für unser Problem auch nicht unmittelbar brauchbar sind [9, 65].

Generell ist die Größe des Randes nicht monoton steigend in der Anzahl der Blöcke. Mit einer gewissen Toleranz bei der Balanciertheit können wir allerdings schon eine Abschätzung geben.

Satz 6.1.1 *Sei $|C^{K+1}|$ die minimale Anzahl der Coupling Constraints bei Partitionierung einer Matrix A in eine SB-Form mit $K+1$ Blöcken und Blockgrößen n_k, $k = 1, \ldots, K+1$. Dann gibt es eine optimale Partition der Matrix A in SB-Form mit K Blöcken und Blockgrößen n'_k mit $n'_k \leq n_k + \left\lceil \frac{1}{k} \left\lfloor \frac{n}{K+1} \right\rfloor \right\rceil$, $k = 1, \ldots, K$, und*

$$|C^K| \leq |C^{K+1}| .$$

Beweis: Wir definieren die Matrix $A^{\pi,K+1}$ als die optimal partitionierte Matrix in $K+1$ Blöcke und konstruieren daraus eine Matrix-Partition $A^{\pi,K}$ mit K Blöcken und der gleichen Anzahl von Coupling Constraints. Wir wählen einen der Blöcke mit kleinstem n_k und verteilen nun die Zeilen und Spalten dieses Blocks auf die verbleibenden K Blöcke. Da sich die $K+1$ Blöcke nicht überschneiden, können wir jede Spalte des $K+1$-ten Blocks an einen der anderen K Blöcke dran permutieren und die strikte Blockdiagonalform der Blöcke durch Zeilen-Permutation wiederherstellen. Der $K+1$-te Block hatte eine minimale Anzahl an Variablen und somit höchstens $\left\lfloor \frac{n}{K+1} \right\rfloor$ Spalten, daher können wir jedem der K anderen Blöcke gleichmäßig $\left\lfloor \frac{1}{k} \left\lfloor \frac{n}{K+1} \right\rfloor \right\rfloor$ Spalten zuordnen. Wenn dann noch Spalten übrig bleiben, da K kein Teiler von $\left\lfloor \frac{n}{K+1} \right\rfloor$ ist, müssen einige der K Blöcke noch zusätzliche Spalten bekommen, sodass diese $\left\lceil \frac{1}{k} \left\lfloor \frac{n}{K+1} \right\rfloor \right\rceil$ neue Spalten haben. Insgesamt erhöht sich die Anzahl der Spalten pro Block also um höchstens

$\left\lceil \frac{1}{k} \left\lfloor \frac{n}{K+1} \right\rfloor \right\rceil$.
Die gefundene Lösung ist zulässig, und die Anzahl der Coupling Constraints hat sich nicht verändert, weshalb es möglich ist, dass es eine Lösung mit geringerer Anzahl Coupling Constraints gibt. □

Die Beziehung zwischen den Blockgrößen n_k und n'_k können wir auch etwas einfacher interpretieren. Es seien n_k und $n_l, 1 \leq k \neq l \leq K$ zwei Blockgrößen der K größten Blöcke und n'_k und $n'_l, 1 \leq k \neq l \leq K$ die Blockgrößen der gleichen Blöcke nach der Transformation der $K+1$-Partition in eine K-Partition. Wenn der k-te Block um s Spalten größer ist als der l-te Block, d.h.

$$||n_k| - |n_l|| = s\,,$$

dann ist der k-te Block nach der Transformation mindestens um $s-1$ und höchstens um $s+1$ Spalten größer als der l-te Block, d.h.

$$||n'_k| - |n'_l|| \geq s-1 \text{ und } ||n'_k| - |n'_l|| \leq s+1\,.$$

Wenn die $K+1$ Blöcke der Matrix $A^{\pi,K+1}$ also auf eine gewissen Art und Weise balanciert waren, wird sich die Balanciertheit durch die Transformation höchstens geringfügig verschlechtern. Generell können wir eine solche Abschätzung wie in Satz 6.1.1 nur machen, wenn wir die Balanciertheit beachten, denn es ist klar, dass die Balanciertheit das Partitionierungsproblem einschränkt und somit schlechtere Partitionen bewirken kann als ohne Balanciertheit.

Durch kleinere Anpassungen bei der Balanciertheit ist es möglich, dass eine optimale Matrix-Partitionierung mit $K+1$ Blöcken einen kleineren Rand hat als eine optimale Partitionierung der gleichen Matrix in K Blöcke. Aus diesem Grund lässt sich aus dem obigen Satz keine allgemeine Aussage herleiten. Trotzdem verdeutlicht dieser Satz, dass eine steigende Anzahl von Prozessoren nicht automatisch einen Gewinn darstellen muss. Daher scheint es sinnvoll zu sein, dass wir uns mit oberen Schranken für die mögliche bzw. sinnvolle Anzahl von Blöcken auseinandersetzen.

6.2 Einige Ansätze

Zunächst eine relativ einfache obere Schranke für die Anzahl der Blöcke.

Satz 6.2.1 *Sei $A \in \mathbb{R}^{m\times n}$ eine Matrix und $|c_j|$ die Anzahl der Nicht-Null-Einträge in der j-ten Spalte von A. Dann gilt für eine SB-Form von A mit K Blöcken*

$$K \leq m - \max\{|c_j|, 1 \leq j \leq n\}\,.$$

Beweis: Eine SB-Form von A kann höchstens m Blöcke haben, da pro Block mindestens eine Zeile nötig ist. Sei c_j eine Spalte mit der maximalen Anzahl von Nicht-Null-Einträgen, dann müssen die $|c_j|$ Nicht-Null-Einträge entweder in einen Block oder in die Coupling Constraints. So oder so bleiben nur noch $m-|c_j|$ Zeilen für andere Blöcke, da sich die Blöcke zeilenweise nicht überschneiden dürfen. Also folgt

$$K \leq m - \max\{|c_j|, 1 \leq j \leq n\}\,.$$

□

Diese Abschätzung ist natürlich nur sehr grob und beruht auch nur auf der Zeile mit den meisten Nicht-Null-Einträgen. Wir können aber auch eine Abschätzung über alle Spalten finden.

Satz 6.2.2 *Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix, die in SB-Form mit K Blöcken gebracht werden soll. Sortiere die Spalten von A wie folgt: c'_1 ist die Spalte mit den meisten Einträgen ungleich Null. c'_2 ist die Spalte mit den meisten Einträgen ungleich Null in den Zeilen, in denen c'_1 eine Null stehen hat. c'_3 ist die Spalte mit den meisten Einträgen ungleich Null in den Zeilen, in denen c'_1 und c'_2 eine Null stehen haben usw. Sei nun $\eta_1 = |c'_1|$ und allgemein η_i die Anzahl der Nicht-Null-Einträge in Spalte c'_i in den Zeilen, in denen die Spalten $c'_j, 1 \leq j \leq i-1$ eine Null stehen haben. Dann gilt*

$$K \leq m - \sum_{i=1}^{m} \eta_i \,,$$

wobei $\eta_i = 0$ ab einem gewissen i möglich ist.
Beweis: Nach Satz 6.2.1 gilt $K \leq m - \eta_1$. Die η_2 Zeilen mit den Nicht-Null-Einträgen der Spalte c'_2 kommen entweder in den Block dieser Spalte oder in die Coupling Constraints. Da diese Zeilen keinen Nicht-Null-Eintrag in der Spalte c'_1 haben, reduziert sich die Anzahl der möglichen Blöcke weiter um η_2, d.h., dass

$$K \leq m - \eta_1 - \eta_2 \,.$$

Analog reduziert sich die mögliche Anzahl von Blöcken durch die Spalte c'_i jeweils um η_i, sodass sich insgesamt die zu beweisende Abschätzung ergibt. □

Wie wir aus einer SB-Form mit $K + 1$ Blöcken schnell eine SB-Form mit K Blöcken bekommen, haben wir im Beweis zu Satz 6.1.1 gesehen. Doch in der Praxis interessiert uns wohl eher der umgekehrte Fall: Wenn wir bereits eine Partitionierung mit K Blöcken gefunden haben, können wir dadurch dann auf eine Partitionierung mit $K + 1$ Blöcken schließen? Oder zumindest abschätzen, inwiefern sich der Rand der SB-Form vergrößern wird? Wie bereits erwähnt, ist diese Frage schon allein auf Grund der Balanciertheit nicht ganz einfach zu beantworten. Trotzdem wollen wir uns ein wenig mit diesem Problem befassen.

Wie können wir zum Beispiel aus einer Partition mit K Blöcken eine Partition mit $K + 1$ Blöcken machen? Eine Möglichkeit besteht darin, dass wir einen der Blöcke auswählen und versuchen, diesen in zwei Blöcke aufzuteilen. Wenn es keine zwei Teilblöcke des Blocks gibt, die sich nicht überschneiden, fügen wir die benötigten verbindenden Zeilen zu den Coupling Constraints hinzu. Wir können das etwa für alle Blöcke ausprobieren und dann den Block partitionieren, bei dem die wenigsten Zeilen zu den Coupling Constraints hinzu kommen. Daraus können wir ein paar Erkenntnisse ziehen, wenn es zum Beispiel in jedem Block eine Zeile mit allen Variablen des Blocks gibt, wird sich durch diese Methode der Rand auf jeden Fall um eine Zeile vergrößern.

Die daraus resultierende Partition wird wohl kaum noch der gewünschten Balanciertheit genügen, doch wenn es uns nur um eine untere Schranke für die Vergrößerung des Randes geht, ist dies nicht schlimm. Wenn wir dann wieder auf die Balanciertheit achten, kann diese untere Schranke allerdings relativ schlecht im Vergleich zum

tatsächlichen Ergebnis sein. Außerdem gilt sie nur für die Überführung einer einzigen Partition mit K Blöcken in eine Partition mit $K+1$ Blöcken, es kann aber durchaus vorkommen, dass es mehrere Partitionen mit der gleichen Anzahl Coupling Constraints gibt, die wir dann auch alle betrachten müssen.

Die Partition von K Blöcke auf $K+1$ Blöcke durch eine einfache Anpassung zu erweitern, scheint aufgrund der Balanciertheit also nicht wirklich hilfreich zu sein. Eine bessere Methode könnte sich durch einen der Algorithmen zur Graph-Partitionierung ergeben, nämlich durch die Rekursive Bisektion. Dabei teilen wir jeden Block in zwei Blöcke auf, wobei wir eventuell wieder zusätzliche Zeilen in den Coupling Constraints hinnehmen müssen. Doch wie auch beim Graph-Partitionierungsproblem muss aus einer optimalen Partition nicht folgen, dass auch die erneute Partition optimal ist. Analog zum Graph-Partitionierungsproblem könnten wir versuchen, die Bisektion durch Methoden wie das Kernighan-Lin-Verfahren zu verbessern, und generell durch untere Schranken für die Bisektion und die Gewinne beim Kernighan-Lin-Verfahren untere Schranken für die Vergrößerung des Randes zu bestimmten.

Natürlich können wir auch versuchen, spektrale untere Schranken herzuleiten, wie schon in Kapitel 5. Zum Beispiel könnten wir den Abstand zwischen λ_{K+1} und λ_K abschätzen, um dadurch auf untere Schranken für die Anzahl der Coupling Constraints schließen zu können. Da aber bereits die Schranken in Kapitel 5 nicht immer die besten waren, können wir nicht unbedingt erwarten, dass wir die Vergrößerung des Randes dadurch gut abschätzen können.

6.3 Ein lineares Programm

Nachdem wir uns einige Gedanken zur Anzahl der Blöcke gemacht haben, wollen wir nun versuchen, die größtmögliche Anzahl der Blöcke durch ein Optimierungsproblem zu bestimmen. Dabei fassen wir die Blöcke als eine Art Bins auf, in die wir die Nicht-Null-Einträge der Matrix verpacken wollen. Zusätzlich stellen wir noch einen weiteren Bin zur Verfügung, der die Coupling Constraints repräsentiert. Wir müssen beachten, dass alle Nicht-Null-Einträge einer Zeile, die nicht in die Coupling Constraints kommen, dem gleichen Block zugeordnet werden, d.h. dem gleichen Bin. Insgesamt wollen wir natürlich die Anzahl der Bins und somit der Blöcke maximieren.

Wir definieren die Variablen

$$x_{ij}^k = \begin{cases} 1, & \text{Eintrag } (i,j) \text{ kommt in Block } k \\ 0, & \text{sonst} \end{cases}$$

für die Zuordnung der Einträge in die Blöcke, bzw.

$$x_{ij}^c = \begin{cases} 1, & \text{Eintrag } (i,j) \text{ kommt in die Coupling Constraints} \\ 0, & \text{sonst} \end{cases}$$

für die Zuordnung der Einträge in den Rand, und

$$y_k = \begin{cases} 1, & \text{Block } k \text{ wird benutzt} \\ 0, & \text{sonst} \end{cases}$$

für die Benutzung der Blöcke, bzw.

$$y_c = \begin{cases} 1\,, & \text{Coupling Constraints werden benutzt} \\ 0\,, & \text{sonst} \end{cases}$$

für die Benutzung der Coupling Constraints. Wir gehen davon aus, dass wir eine gewisse Anzahl von Blöcken K haben, etwa $K = \min\{m, n\}$, und die Anzahl der nicht-leeren Blöcke maximieren wollen.

Der Einfachheit halber definieren wir $A^{\neq 0}$ als die Menge aller Nicht-Null-Einträge der Matrix A. Das lineare Programm könnte dann in etwa so aussehen:

$$\begin{aligned}
&\max \sum_{k=1}^{K} y_k \\
&s.d. \sum_{k=1}^{K} x_{ij}^k + x_{ij}^c = 1 && \forall\,(i,j) \in A^{\neq 0} && (1)\\
&\quad y_k \leq \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^k && \forall k = 1,\ldots,K && (2)\\
&\quad y_c \leq \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^c && && (3)\\
&\quad x_{ij}^k = x_{il}^k && \forall j,l : (i,j),(i,l) \in A^{\neq 0}, \forall k \in \{1,\ldots,K,c\} && (4)\\
&\quad x_{ij}^k + x_{lj}^{k'} \leq 1 && \forall i,l : (i,j),(l,j) \in A^{\neq 0}, \forall 1 \leq k \neq k' \leq K && (5)\\
&\quad x_{ij}^k \in \{0,1\}\,,\ \ y_k \in \{0,1\} && \forall k = 1,\ldots,K && \\
&\quad x_{ij}^c \in \{0,1\}\,,\ \ y_c \in \{0,1\}\,. &&
\end{aligned}$$

Die Zielfunktion maximiert also die Anzahl der benutzten Blöcke. Nebenbedingung (1) stellt sicher, dass jeder Eintrag genau einem Block oder den Coupling Constraints zugeordnet wird. Die Nebenbedingungen (2) und (3) sorgen dafür, dass ein Block bzw. die Coupling Constraints nur dann benutzt werden, wenn wir ihnen auch einen Eintrag zuordnen. Durch die Nebenbedingungen (4) und (5) wird verhindert, dass Einträge aus der gleichen Zeile oder Spalte in unterschiedliche Blöcke kommen.

Dieses lineare Programm ähnelt natürlich einem Bin-Packing-Problem, nur dass wir hier die Anzahl der Bins nicht minimieren, sondern maximieren wollen. Durch die vielen Nebenbedingungen ist es wohl schwer zu lösen, weshalb es in der Praxis wohl eher unbrauchbar ist. Wir könnten aber versuchen, obere Schranken für dieses Problem zu finden, etwa durch Schranken für das Bin-Packing-Problem oder das duale Problem.

Das Programm lässt sich auch noch erweitern, zum Beispiel um die Anzahl der Coupling Constraints zu beschränken. Dazu definieren wir durch $|r_i|$ wie in Kapitel 5 die Anzahl aller Nicht-Null-Einträge in der $i-$ten Zeile, und fügen die Nebenbedingung

$$\sum_{i=1}^{m} \frac{1}{|r_i|} \sum_{j=1}^{n} x_{ij}^c \leq C^{max}$$

zum Programm hinzu. Da alle Variablen x_{ij} einer Zeile r_i den gleichen Wert haben müssen, ist $\sum_{j=1}^{n} x_{ij}^c$ entweder 0 oder $|r_i|$. Daher zählen wir durch den Term auf der linken Seite der Nebenbedingung die Anzahl der Zeilen, deren Einträge in den Coupling

Constraints sind, und diese Anzahl beschränken wir dann durch C^{max}. Da $|r_i|$ keine Variable ist, ist diese Nebenbedingung für ein lineares Programm zulässig. Diese Nebenbedingung ist nützlich, wenn wir an der maximalen sinnvollen Anzahl der Blöcke interessiert sind, da eine SB-Form mit vielen Blocken, aber auch sehr großem Rand nicht immer empfehlenswert sein muss.

6.4 Alternierende Wege in der Matrix

Wir blicken noch einmal auf Kapitel 5 zurück, in dem wir Ergebnisse für die Matrix durch den CNG bzw. den NIG des assoziierten Hypergraphen hergeleitet haben. Insbesondere haben wir den Diameter *diam* eines Graphen benutzt, und beispielsweise festgestellt, dass $diam = 1$ gilt, wenn der Graph zusammenhängend ist. Um den Diameter zu bestimmen, müssen wir den kürzesten Weg zwischen allen Paaren zweier Knoten des Graphen finden. Da wir gesehen haben, wie wir eine Matrix als CNG oder NIG darstellen können, wollen wir nun versuchen, einen Weg in dem Graph auf die Matrix zu übertragen.

Ein Knoten im CNG entspricht einer Spalte der Matrix, und es gibt genau dann eine Kante zwischen zwei Knoten, wenn die beiden entsprechenden Variablen über eine Nebenbedingung miteinander verbunden sind. Wir betrachten einen Weg (v_1, v_2, v_3) im CNG zwischen zwei Knoten v_1 und v_3, der über den Knoten v_2 läuft. Jeder Knoten, der auf diesem Weg liegt, entspricht einer Spalte, und jede Kante auf diesem Weg entspricht einer Verbindung zweier Variablen durch eine Nebenbedingung. Deshalb können wir den Weg (v_1, v_2, v_3) in der Matrix durch einen Weg entlang der Zeilen und Spalten nachvollziehen.

Wir starten mit der Variable x_1, die zum Knoten v_1 gehört. Dann wählen wir eine der Nebenbedingungen, in denen sowohl x_1 als auch x_2 vorkommen, und gehen diese Zeile entlang bis zum Eintrag von x_2. Nun wählen wir eine der Zeilen, in denen sowohl x_2 als auch x_3 vorkommen. Wenn dies die gleiche Zeile ist, in der wir uns schon befinden, laufen wir weiter zum Eintrag der Variable x_3. Ansonsten gehen wir in der Spalte der Variable x_2 nach oben oder unten, bis wir zu einer Nebenbedingung kommen, die sowohl x_2 als auch x_3 benutzt. Zu guter Letzt laufen wir diese Zeile entlang bis zum Eintrag von x_3. Durch den Weg (x_1, x_2, x_3) haben wir einen Weg in der Matrix gefunden, der äquivalent zum entsprechenden Weg im CNG ist. Da dieser Weg sowohl Zeilen als auch Spalten entlang läuft, werden wir einen solchen Weg als *alternierenden Weg* in der Matrix bezeichnen.

Dadurch, dass wir einen Weg im CNG auf die Matrix übertragen können, ist es nun auch möglich, Aussagen durch Wege im Graph für die Matrix zu interpretieren. Wenn es zum Beispiel zwei Knoten gibt, zwischen denen es keinen Weg gibt, ist der Graph nicht zusammenhängend. Daraus folgt, dass die Matrix eine strikte Blockdiagonalform hat, wenn es zwei Variablen gibt, die wir durch keinen alternierenden Weg verbinden können. Wenn es einen alternierenden Weg gibt, der alle Variablen bzw. alle Nebenbedingungen enthält, können wir diese Spalten bzw. Zeilen nicht partitionieren.

Für den NIG können wir den Begriff eines alternierenden Wegs analog definieren, wobei wir dann Zeilen verbinden wollen, da die Knoten im NIG den Zeilen entsprechen.

Wenn ein alternierender Weg jede Zeile oder jede Spalte enthält, und jede Zeile und jede Spalte höchstens einmal besucht, können wir dadurch für eine Matrix $A \in \mathbb{R}^{m \times n}$

folgende Aussagen machen:

- Sei $m > n$: Dann haben wir alle Variablen benutzt, nicht aber alle Nebenbedingungen. Da wir die Teilmatrix, die durch die Zeilen des alternierenden Wegs induziert wird, nicht partitionieren können, besteht der Zeilenrand mindestens aus all diesen Zeilen, d.h. $|C| \geq n$.

- Sei $m < n$: Dann haben wir zwar alle Nebenbedingungen benutzt, nicht aber alle Variablen. Die Matrix hat keine SB-Form mit Zeilenrand, und n ist eine untere Schranke für die Anzahl der Linking Columns in einer SB-Form der Matrix mit Spaltenrand.

- Sei $m = n$: Hier kommen alle Zeilen und alle Spalten in den Rand, sodass die Matrix nicht partitioniert werden kann. Wir erhalten also höchstens eine Treppenform, d.h. eine Blockdiagonalform, in der sich zwei aufeinander folgende Blöcke durch mindestens eine Zeile oder eine Spalte überschneiden.

Insbesondere erhalten wir durch die alternierenden Wege Aussagen über die Auswirkungen einer Permutation der Matrix. Ein alternierender Weg in der Matrix beinhaltet alle Zeilen und Spalten, die in den gleichen Block einer SB-Form der Matrix kommen müssen, es sei denn, wir permutieren mindestens eine Zeile in den Zeilenrand bzw. eine Spalte in den Spaltenrand. In der Linearen Algebra gibt es viele Ergebnisse zu Permutationsgruppen, die sich hier eventuell anwenden lassen.

Das Problem, in einem Graph einen Kreis zu finden, der jeden Knoten nur einmal besucht, ist als das *Hamilton-Problem* bekannt, und ein Graph, der einen solchen Kreis enthält, heißt *hamiltonsch*. Wir können das Problem, in der Matrix einen alternierenden Weg zu finden, der jede Variable genau einmal benutzt, daher auch als das folgende Entscheidungsproblem auffassen: Ist der CNG hamiltonsch, wenn wir höchstens eine Kante hinzufügen? Analog dazu ist das Problem, in der Matrix einen alternierenden Weg zu finden, der jede Nebenbedingung genau einmal benutzt, äquivalent zum Entscheidungsproblem: Ist der NIG hamiltonsch, wenn wir höchstens eine Kante hinzufügen? Das Hinzufügen einer Kante sorgt dafür, dass wir einen Kreis finden, der alle Knoten genau einmal enthält und der Graph somit hamiltonsch ist. Ohne diese Kante erhalten wir einen Weg mit allen Knoten und somit einen alternierenden Weg in der Matrix.

Wenn wir eine Matrix haben, die einen alternierenden Weg mit allen Zeilen oder allen Spalten hat, und der NIG bzw. der CNG daher hamiltonsch ist, können wir versuchen, Ergebnisse für hamiltonsche Graphen auf den NIG bzw. den CNG und somit auch auf die Matrix anzuwenden. Wir könnten also versuchen, alternierende Wege zu benutzen, um Aussagen über die Partitionierbarkeit einer Matrix machen zu können. Das Prinzip der alternierenden Wege in der Matrix ist vergleichbar mit dem Vorgehen bei der Ungarischen Methode, um ein Matching in einem bipartiten Graph durch einen erweiternden Weg zu vergrößern [46]. Das bipartite Modell eignet sich daher auch gut zum Finden von alternierenden Wegen.

7 Fazit und Ausblick

7.1 Rückblick

Auf den vorangegangenen Seiten haben wir uns mit der Permutierung einer Matrix in SB-Form durch Graph-Partitionierung beschäftigt. Zunächst haben wir gesehen, warum eine dünnbesetzte Struktur einer Matrix vorteilhaft ist, und wie wir dies ausnutzen können. Wir haben gemerkt, dass es sinnvoll sein kann, die Matrix-Vektor-Multiplikation zu parallelisieren und die Matrix dazu in eine SB-Form zu partitionieren. Nachdem wir wichtige Begriffe und Zusammenhänge der Graph- und Hypergraph-Partitionierung eingeführt und einige wichtige Algorithmen zur Lösung kennen gelernt haben, sind wir dazu übergegangen, das Matrix-Partitionierungsproblem als Graph-Partitionierungsproblem darzustellen. Wir haben gesehen, dass wir eine Matrix am besten als Hypergraph darstellen, um sie zu partitionieren, dazu stehen das Row-Net- und das Column-Net-Modell zur Verfügung.

Nachdem wir genügend Vorwissen angesammelt hatten, haben wir uns näher mit der Anzahl der Coupling Constraints befasst, für die wir untere Schranken herleiten wollten. Mit Hilfe des assoziierten Hypergraphen der Matrix haben wir einige untere Schranken hergeleitet, die auf verschiedenen Eigenschaften der Matrix basieren. Auch wenn nicht alle Abschätzungen am Ende zufriedenstellende Ergebnisse lieferten, haben wir dennoch verschiedene Ansätze kennen gelernt, um untere Schranken herzuleiten. Wir haben zudem erkannt, dass wir durch das NIG-Modell gegenüber dem CNG-Modell wohl die bessere Hypergraph-Repräsentation erhalten, wenn wir untere Schranken für die Anzahl der Coupling Constraints bestimmen wollen.

In einem weiteren Kapitel haben wir uns mit der Anzahl der Blöcke beschäftigt. Zum einen haben wir uns gefragt, ob wir aus einer SB-Form mit K Blöcken Erkenntnisse für eine SB-Form mit $K' > K$ ableiten können. Zum anderen haben wir uns gefragt, wie viele Blöcke eine SB-Form einer Matrix maximal haben kann bzw. wie viele Blöcke sinnvoll sind. Dazu haben wir ein lineares Programm aufgestellt, das die maximale Anzahl der Blöcke bestimmt. Für dieses Programm könnten wir nun durch Relaxationen obere Schranken herleiten, um Abschätzungen für die maximale Anzahl der Blöcke zu erhalten. Wenn wir gleichzeitig wollen, dass die Anzahl der Coupling Constraints einen gewissen Wert nicht überschreitet, können wir dazu eine zusätzliche Nebenbedingung aufstellen. Zudem haben wir die Äquivalenz zwischen Wegen im Graph und alternierenden Wegen in der Matrix aufgezeigt, aus der sich eventuell Erkenntnisse zur Partitionierung der Matrix ziehen lassen.

7.2 Ausblick

Zum Schluss wollen wir uns noch mit einigen Ideen auseinander setzen, die wir eventuell noch hätten verfolgen können. Wir haben stets eine feste Matrix A betrachtet (*statischer Fall*), es kann aber durchaus möglich sein, dass sich diese Matrix im Laufe der

Zeit verändert (*dynamischen Fall*). Dann könnten wir versuchen, die Partition der alten Matrix an die neue Matrix anzupassen, bei der Graph-Partitionierung geschieht dies durch *Repartitionierungsalgorithmen*. Diese könnten wir auf den assoziierten Graph anwenden oder versuchen, Repartitionierungsalgorithmen für Matrizen herzuleiten.

Bei den unteren Schranken gibt es unter Umständen noch weitere Ansätze, die wir verfolgen könnten. So gibt es etwa auch für Hypergraphen Ansätze zur Definition der Laplace-Matrix und der algebraischen Konnektivität, die Tensoren benutzen [43]. Allerdings ist dies etwas komplizierter als für Graphen.

Ein weiterer Versuch könnte darin unternommen werden, die Multilevel-Idee aus Kapitel 3.3 zur Bestimmung von unteren Schranken für die Anzahl der Coupling Constraints zu nutzen. So könnten wir z. B. Nicht-Null-Einträge der Matrix, die sehr wahrscheinlich in den gleichen Block kommen werden, zu einem einzigen Nicht-Null-Eintrag zusammenfassen. Finden wir dann eine Partition der zusammengefassten Matrix, können wir dadurch unter Umständen auf untere Schranken für die Anzahl der Coupling Constraints der originalen Matrix schließen. Zusätzlich könnten wir untersuchen, wie stark sich der Rand der Matrix vergrößern wird, wenn wir die zusammengefasste Matrix wieder „aufklappen“.

Im Laufe der Arbeit haben wir uns mit der Anzahl der Coupling Constraints beschäftigt, also den Nebenbedingungen, für die mehrere Prozessoren miteinander kommunizieren müssen. Die Kommunikation haben wir dabei weitestgehend außer Acht gelassen, in Kapitel 4 haben wir aber schon gesehen, dass die Zielfunktion zur Minimierung der Kommunikation zwischen den Prozessoren eine andere ist, als zur Minimierung der Coupling Constraints.

Das Problem der Kommunikation zwischen den Prozessoren wird auf eine ähnliche Art und Weise behandelt wie das der Minimierung der Coupling Constraints, wie in Teilen der hier verwendeten Literatur nachgelesen werden kann. Allerdings gibt es auch andere Ansätze, bei denen das Problem z. B. als Knoten- oder Kantenfärbungsproblem modelliert wird [73].

Zudem haben wir uns voll und ganz auf die Matrix A eines Gleichungssystems $Ax = b$ konzentriert und die Vektoren x bzw. Ax dabei völlig außer Acht gelassen. Wenn wir die einzelnen Zeilen verschiedenen Prozessoren zuordnen, ist es jedoch nicht unbedeutend, welche Zeile welchem Prozessor zugeordnet wird, da dies insbesondere die Gleichmäßigkeit der Kommunikation beeinflusst. Daher gibt es neben der Matrix-Partitionierung auch Ansätze zur Vektor-Partitionierung [7], wenn auch nicht so viele.

Natürlich können wir die Modelle immer weiter verfeinern, Hendrickson und Kolda [40] schlagen etwa Gewichte zur Abbildung des Aufwands und der Menge an Daten vor. Zudem hängt die Übertragungszeit der Daten auch von der Latenz, der Anzahl der Prozessoren und Switches zwischen zwei kommunizierenden Prozessoren und der Leistungsfähigkeit der einzelnen Prozessoren ab. Dadurch würde allerdings eine ziemlich komplexe Zielfunktion entstehen, bei der wir zudem noch entscheiden müssten, wie stark die einzelnen Eigenschaften gewichtet werden.

Zu guter Letzt müssen wir erkennen, dass wir irgendwann an Grenzen stoßen werden. In Kapitel 3.4 wurde bereits die Klasse der P-vollständigen Probleme genannt, die so etwas wie eine obere Schranke für Parallelisierung darstellt. Aus diesem Grund beschäftigen sich Wissenschaftler bereits immer häufiger mit dieser Klasse der Probleme [35].

Anhang

Wir benutzen immer wieder Definitionen, die nicht von zentraler Bedeutung sind, oder Sätze, die uns in Beweisen weiterhelfen, aber mit dem eigentlichen Thema nur wenig zu tun haben. Diese können hier im Anhang nachgelesen werden. Neben Begriffen und Aussagen aus verschiedenen Gebieten der Mathematik sind am Ende auch einige Programme zur Partitionierung aufgelistet.

A.1 Graphentheorie

Dass die Laplace-Matrix positiv semi-definit ist, zeigen wir mit Hilfe einer Inzidenzmatrix.

Satz A.1.1 *Sei $G = (V, E)$ ein Graph, $Q(G)$ die Laplace-Matrix und $B(G)$ die Inzidenzmatrix, wenn alle Kanten eine Orientierung erhalten. Dann gilt für jede beliebige Orientierung der Kanten $Q(G) = BB^T$.*
Beweis: vgl. [6], S. 27 □

Das Handschlaglemma gehört zu den bekanntesten Sätzen der Graphentheorie.

Satz A.1.2 (**Handschlaglemma**) *Sei $G = (V, E)$ ein Graph und d_i der Grad des Knoten v_i. Dann gilt*

$$\sum_{v_i \in V} d_i = 2|E| \,.$$

Beweis: Für einen Graph mit einem oder zwei Knoten ist die Gleichung trivial, wir beweisen die Gleichung für $|V| \geq 3$. Sei also G ein Graph, zunächst ohne Kanten. Fügen wir nun eine Kante zum Graph hinzu, gibt es zwei Knoten v_i und v_j, die nun adjazent zueinander sind. Deren Grad erhöht sich um 1, insgesamt steigt also die Summe aller Knotengrade um 2. Da wir E um eine Kante erweitert haben, erhöht sich $|E|$ um 1, pro Kante steigt also die Summe aller Knotengrade um 2. Damit haben wir die Gültigkeit der obigen Gleichung gezeigt. □

A.2 Algebra und Analysis

Bei der Laplace-Matrix nutzen wir aus, dass diese positiv semi-definit ist.

Definition A.2.1 *Eine Matrix $A \in \mathbb{R}^{n \times n}$ heißt* **positiv semi-definit**, *wenn für alle Vektoren $x \neq 0$ gilt, dass $x^T Ax \geq 0$.*

Um positive Semi-Definitheit zu zeigen, wird oft das folgende Ergebnis benutzt.

Satz A.2.2 *Eine Matrix A ist genau dann positiv semi-definit, wenn* $\lambda_i \geq 0$ *für alle Eigenwerte* $\lambda_1, \ldots, \lambda_n$.
Beweis: Da die normierten Eigenvektoren der Matrix A eine Orthonormalbasis bilden, können wir jeden Vektor $x \in \mathbb{R}^n$ darstellen als $x = \sum_{i=1}^{n} \alpha_i \Lambda_i$. Somit gilt

$$x^T A x = (\sum_{i=1}^{n} \alpha_i \Lambda_i)^T A \sum_{i=1}^{n} \alpha_i \Lambda_i = (\sum_{i=1}^{n} \alpha_i \Lambda_i)^T \sum_{i=1}^{n} \alpha_i \lambda_i \Lambda_i = \sum_{i=1}^{n} \alpha_i^2 \lambda_i \,.$$

Somit gilt $x^T A x \geq 0$ für alle $x \in \mathbb{R}^n$ genau dann, wenn $\lambda_i \geq 0$ für alle $i = 1, \ldots, n$. □

Der Spektralsatz gehört zu den wichtigsten Sätzen der Linearen Algebra.

Satz A.2.3 (**Spektralsatz**, Matrix-Version) *Jede symmetrische Matrix besitzt eine Orthonormalbasis aus Eigenvektoren.*
Beweis: vgl. [71], S. 334 ff.

Mit dem Spektralsatz eng verbunden ist die folgende Aussage.

Satz A.2.4 *Alle Eigenwerte einer symmetrische Matrix sind reell.*
Beweis: vgl. [71], S. 336.

Der folgende Satz hilft uns, die Spur besser berechnen zu können.

Satz A.2.5 Seien $A, B \in \mathbb{R}^{m \times n}$. Dann gilt

$$Spur(AB^T) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij} \,.$$

Beweis: Es ist

$$Spur(AB^T) = \sum_{i=1}^{n} A_{i,-} B^T_{-,i} = \sum_{i=1}^{n} A_{i,-} \left(B_{i,-}\right)^T = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij} \,.$$

□

Wir können die Spur durch die Eigenwerte abschätzen.

Satz A.2.6 Seien $A, B \in \mathbb{R}^{n \times n}$ symmetrisch. Dann gilt

$$Spur(AB^T) \leq \sum_{i=1}^{n} \lambda_i(A) \lambda_i(B) \,.$$

Beweis: vgl. [20]

Wenn in Satz A.2.6 die Matrix B die Identitätsmatrix ist, gilt Gleichheit.

Satz A.2.7 Sei $A \in \mathbb{R}^{n\times n}$ und $\lambda_1, \ldots, \lambda_n$ die Eigenwerte von A. Dann gilt

$$Spur(A) = \sum_{i=1}^{n} \lambda_i(A) .$$

Beweis: vgl. [80], S. 148

Eine quadratische Matrix können wir durch eine obere Dreiecksmatrix darstellen.

Satz A.2.8 *Sei $A \in \mathbb{C}^{n\times n}$ eine quadratische Matrix. Dann gibt es eine obere Dreiecksmatrix $U \in \mathbb{C}^{n\times n}$ und eine invertierbare Matrix $P \in \mathbb{C}^{n\times n}$ mit*

$$U = P^{-1}AP$$

bzw.

$$A = PUP^{-1} .$$

Die Eigenwerte von A stehen auf der Diagonalen von U.
Beweis: vgl. [80], S. 199 f. □

Die Spur ist kommutativ.

Satz A.2.9 *Sei $A \in \mathbb{R}^{m\times n}$ und $B \in \mathbb{R}^{n\times m}$. Dann gilt*

$$Spur(AB) = Spur(BA) .$$

Beweis: vgl. [80], S. 22. □

Die folgende Formel hat Gauß angeblich schon während der Schulzeit zeigen können.

Satz A.2.10 (**Gaußsche Summenformel**) *Es gilt*

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} .$$

Beweis: vgl. [68], S. 39 □

Eine bekannte Ungleichung der Analysis ist die Cauchy-Schwarz-Ungleichung.

Satz A.2.11 (**Cauchy-Schwarz-Ungleichung**) *Für reelle Zahlen $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ gilt*

$$\left(\sum_{k=1}^{n} |a_k b_k|\right)^2 \leq \sum_{k=1}^{n} |a_k|^2 \cdot \sum_{k=1}^{n} |b_k|^2 .$$

Beweis: vgl. [68], S. 221

A.3 Software-Pakete zur Graph-Partitionierung

An dieser Stelle wollen wir einige Programme zur Graph- (GP), Hypergraph- (HP) und Matrix-Partitionierung (MP) auflisten.

- Chaco (GP), `http://www.cs.sandia.gov/~bahendr/chaco.html`
- METIS (GP), `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`
- ParMETIS (GP), `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`
- hMETIS (HP), `http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview`
- PaToH (HP), `http://bmi.osu.edu/~umit/software.html`
- Mondriaan (MP), `http://www.staff.science.uu.nl/~bisse101/Mondriaan/`

Dies sind nur einige Programme, für weitere siehe die Referenzen in der Literatur der Literaturliste.

Literaturverzeichnis

[1] Alpert, C. J., Kahng, A. B. (1995). *Recent directions in netlist partitioning: a survey*, Integration, the VLSI Journal, Vol. 19, No. 1-2, S. 1-81.

[2] Aykanat, C., Pınar, A., Çatalyürek, Ü. V. (2002). *Permuting Sparse Rectangular Matrices into Singly-Bordered Block-Diagonal Form for Parallel Solution of LP Problems*, Technical Report BU-CE-0203, Computer Engineering Department, Bilkent University, 2002.

[3] Babai, L. (1994). *Automorphism groups, isomorphism, reconstruction*, Handbook of Combinatorics, MIT Press, ISBN-10 0262571722, Kapitel 27, S. 1447-1539.

[4] Barnard, S.T., Simon, H.D. (1994). *A fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems*, Concurrency: Practice and Experience, Vol. 6, No. 2, S. 101-117.

[5] Benders, J. F. (1962). *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik 4, S. 238-252.

[6] Biggs, N. (1974). *Algebraic Graph Theory*, Cambridge Mathematical Library, ISBN 0-521-45897-8.

[7] Bisseling, R. H., Meesen, W. (2005). *Communication Balancing in Parallel Sparse Matrix-Vector Multiplication*, Electronic Transactions on Numerical Analysis, Vol. 21, S. 47-65.

[8] Brouwer, A. E., Haemers, W. H. (2008). *A lower bound for the Laplacian eigenvalues of a graph - proof of a conjecture by Guo*, Linear Algebra and its Applications, Vol. 429, S. 2131-2135.

[9] Brualdi, R. A., Hahn, G., Horak, P., Kramer, E. S., Mellendorf, S., Mesner, D. M. (1994). *On A Matrix Partition Conjecture*, Journal of Combinatorial Theory, Series A, Vol. 69, No. 2, S. 333-346.

[10] Carolan, W. J., Hill, J. E., Kennington, J. L., Niemi, S., Wichmann, S. J. (1990). *An empirical evaluation of the KORBX algorithms for military airlift applications*, Operations Research, Vol. 38, S. 240-248.

[11] Çatalyürek, Ü. V., Aykanat, C. (1999). *Hypergraph-Partitioning Based Decomposition for Parallel Sparse-Matrix Vector Multiplication*, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 7, S. 673-693.

[12] Černý, V. (1985). *Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm*, Journal of Optimization Theory and Applications, Vol. 45, No. l, S. 41-51.

[13] Chamberlain, B. L. (1998). *Graph Partitioning for Distributing Workloads of Parallel Computations*, Technical Report UW-CSE-98-10-03, University of Washington.

[14] T. H. Cormen, Leiserson, C. E., Rivest, R., Stein, C. (2009). *Algorithmen - Eine Einführung*, Oldenbourg Wissenschaftsverlag, ISBN 978-3-486-59002-9.

[15] Dahmen, W., Reusken, A. (2008). *Numerik für Ingenieure und Naturwissenschaftler*, Springer Verlag, ISBN 978-3-540-76492-2.

[16] Dantzig, G. B., Thapa, M. N. (2003). *Linear Programming 2: Theory and Extensions*, Springer Verlag, ISBN-10 0387986138.

[17] Dantzig, G. B., Wolfe, P. (1959). *Decomposition Principle for Linear Programs*, Operations Research, Vol. 8, No. 1, S. 101-111.

[18] de Abreu, N. M. M. (2007). *Old and new results on algebraic connectivity of graphs*, Linear Algebra and its Applications, Vol. 423, S. 53-73.

[19] de Caen, D. (1997). *An upper bound on the sum of squares of degrees in a graph*, Discrete Mathematics, Vol. 185, S. 245-248.

[20] Donath, W. E., Hoffman, A. J. (1973). *Lower Bounds for the Partitioning of Graphs*, IBM Journal of Research and Development, Vol. 17, No. 5, S. 420-425.

[21] Dutt, S. (1993). *New Faster Kernighan-Lin-Type Graph Partitioning Algorithms*, ICCAD '93 Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design, S. 370-377.

[22] Elsner, U. (1997). *Graph Partitioning - A survey*, Preprint SFB393/97-27, Technische Universität Chemnitz.

[23] Esfahanian, A. H. (1985). *Lower-Bounds on the Connectivities of a Graph*, Journal of Graph Theory, Vol. 9, S. 503-511.

[24] Farhat, C. (1988). *A simple and efficient automatic fem domain decomposer*, Computers and Structures, Vol. 28, No. 5, S. 579-602.

[25] Ferris, M. C., Horn, J. D. (1998). *Partitioning mathematical programs for parallel solution*, Mathematical Programming 80, S. 36-61.

[26] Fiduccia, C. M., Mattheyses, R.M. (1982). *A Linear-Time Heuristic for Improving Network Partitions*, Technical Report 82CRD130, General Electric Co, Corporate Research and Development Center, Schenectady, New York.

[27] Fiedler, M. (1973). *Algebraic connectivity of graphs*, Czechoslovak Mathematical Journal, Vol. 23, No. 98, S. 298-305.

[28] Fiedler, M. (1975). *A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory*, Czechoslovak Mathematical Journal, Vol. 25, No. 4, S. 619-633.

[29] Fjällström, P.-O. (1998). *Algorithms for Graph Partitioning: A Survey*, Linköping Electronic Articles in Computer and Information Science, Vol 3, No. 10, Linköping University Electronic Press, Linköping, Schweden.

[30] Garey, M. R., Johnson, D. R., Stockmeyer, L. (1987). *Some Simplified NP-Complete Problems*, Theoretical Computer Science, Vol. 1, No. 3, S. 237-267.

[31] Gács, P., Lovász, L. (1981). *Khachiyan's algorithm for linear programming*, Mathematical Programming Studies, Vol. 14, S. 61-68.

[32] Gilbert, J. R., Zmijewski, E. (1987). *A parallel graph partitioning algorithm for a message-passing multiprocessor*, International Journal of Parallel Programming, Vol. 16, No. 6, S. 427-449.

[33] Glover, F., (1989). *Tabu Search-Part I*, ORSA Journal on Computing, Vol. 1, No. 3, S. 190-206.

[34] Glover, F., (1990). *Tabu Search-Part II*, ORSA Journal on Computing, Vol. 2, No. 1, S. 4-32.

[35] Greenlaw, R., Hoover, H. J., Ruzzo, W. L., (1995). *Limits to Parallel Computation: P-Completeness Theory*, http://www.cs.washington.edu/ ruzzo/papers/limits.pdf, Oxford University Press, ISBN-10 0195085914.

[36] Gupta, A, (1997). *Fast and effective algorithms for graph partitioning and sparse-matrix ordering*, IBM Journal of Research and Development - Special issue: optical lithography I, Vol. 41, No. 1, S. 171-183.

[37] Hadley, S. W., Mark, B. L., Vannelli, A. (1992). *An Efficient Eigenvector Approach for Finding Netlist Partitions*, IEEE Transactions on Computer-Applied Design, Vol. 11, No. 7, S. 885-892.

[38] Hendrickson, B. (1998). *Graph Partitioning and Parallel Solvers : Has the Emperor No Clothes?*, Solving Irregularly Structured Problems in Parallel, Irregular '98, Springer Verlag, S. 218-225.

[39] Hendrickson, B., Kolda, T. (1998). *Partitioning Sparse Rectangular Matrices for Parallel Computations of Ax and A^Tv*, Applied Parallel Computing in Large Scientific and Industrial Problems, PARA '98, Springer Verlag, S. 239-247.

[40] Hendrickson, B., Kolda, T. (1999). *Graph partitioning models for parallel computing*, Parallel Computing 26, S. 1519-1534.

[41] Hendrickson, B., Leland, R. (1995). *An Improved Spectral Graph Partitioning Algorithm*, SIAM Journal on Scientific Computing, Vol. 16, No. 2, S. 452-469.

[42] Hendrickson, B., Leland, R. (1995). *A Multilevel Algorithm for Partitioning Graphs*, Supercomputing '95 Proceedings of the 1995 ACM/IEEE conference on Supercomputing, Art.-Nr. 28.

[43] Hu, S., Qi, L. (2011). *Algebraic connectivity of an even uniform hypergraph*, Journal of Combinatorial Optimization, DOI: 10.1007/s10878-011-9407-1, S. 1-16.

[44] Hu, Y. F., Blake, R. J. (1995). *An optimal dynamic load balancing algorithm*, Technical Report DL-P-95-011, Daresbury Laboratory, Warrington, UK.

[45] Ihler, E., Wagner, D., Wagner, F. (1993). *Modeling hypergraphs by graphs with the same mincut properties*, Information Processing Letters, Vol. 45, No. 4, S. 171-175.

[46] Jongen, H. T., Meer, K., Triesch, E. (2004). *Optimization Theory*, Kluwer Academic Publishers, Print ISBN 1-4020-8098-0.

[47] Karypis, G., Kumar, V. (1995). *Analysis of multilevel graph partitioning*, Supercomputing '95 Proceedings of the 1995 ACM/IEEE conference on Supercomputing, Art. No. 29.

[48] Karypis, G., Kumar, V. (1998). *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, SIAM Journal on Scientific Computing, Vol. 20, No. 1, S. 359-392.

[49] Karypis, G., Kumar, V. (1998). *Multilevel k-way Partitioning Scheme for Irregular Graphs*, Journal of Parallel and Distributed Computing, Vol. 48, S. 96-129.

[50] Karypis, G., Kumar, V. (1997). *A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm*, 8th SIAM Conference on Parallel Processing for Scientific Computing.

[51] Kayaaslan, E., Pınar, A., Aykanat, C., Çatalyürek, Ü. V. (2012). *Partitioning Hypergraphs in Scientific Computing Applications through Vertex Separators on Graphs*, SIAM Journal on Scientific Computing, Vol. 34, No. 2, S. 970-992.

[52] Kernighan, B. W., Lin, S. (1970). *An Efficient Heuristic Procedure for Partitioning Graphs*, The Bell System Technical Journal, Vol. 49, No. 2, S. 291-307.

[53] Kirchhoff, G. (1847). *Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird*, Annalen der Physik und Chemie, Vol. 72, S. 497-508.

[54] Kirkpatrick, S., Gelatt, C. D., Vecchi, M.P. (1983). *Optimization by Simulated Annealing*, Science, New Series, Vol. 220, No. 4598, S. 671-680.

[55] Lanczos, C. (1950). *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators*, Journal of Research of the National Bureau of Standards, Vol. 45, No. 4, S. 255-282.

[56] Lemke, C. E. (1954). *The dual method of solving the linear programming problem*, Naval Research Logistics, Vol. 1, No. 1, S. 36-47.

[57] Lesniak, L. (1974). *Results on the edge-connectivity of graphs*, Discrete Mathematics, Vol. 8, S. 351-354.

[58] Liu, L.-T., Kuo, M.-T., Huang, S.-T., Cheng, C.-K. (1995). *A gradient method on the initial partition of Fiduccia-Mattheyses algorithm*, ICCAD '95 Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design, S. 229-234.

[59] Mohar, B. (1991). *The laplacian spectrum of graphs*, Graph Theory, Combinatorics, and Applications, Vol. 2, S. 871-898.

[60] Mulvey, J. M., Ruszczyński, A. (1992). *A diagonal quadratic approximation method for large scale linear programs*, Operations Research Letters, Vol. 12, No. 4, S. 205-215.

[61] Ou, C.W., Ranka, S. (1997). *Parallel Incremental Graph Partitioning*, IEEE Transactions on Parallel and Distributed Systems, 8(8):123-137.

[62] Özturan, C., de Cougny, L., Shephard, M. S., Flaherty, J. E. (1994). *Parallel Adaptive Mesh Refinement and Redistribution on Distributed Memory Computers*, Computer Methods in Applied Mechanics and Engineering, Vol. 12, S. 123-137.

[63] Pınar, A., Aykanat, C. (1996). *An Effective Model to Decompose Linear Programs for Parallel Solution*, Applied Parallel Computing Industrial Computation and Optimization Lecture Notes in Computer Science, Vol. 1184, S. 592-601.

[64] Preis, R., Dieckmann, R. (1996). *The PARTY Partitioning-Library, User Guide - Version 1.1*, Technical Report tr-rsfb-96-024, University of Paderborn.

[65] Rho, Y. (2005). *On Kramer-Mesner Matrix Partitioning Conjecture*, Journal of the Korean Mathematical Society, Vol. 42, No. 4, S. 871-881.

[66] Savage, J. E., Wloka, M. G. (1991). *Parallelism in graph-partitioning*, Journal of Parallel and Distributed Computing, Vol. 13, No. 3, S. 257-272.

[67] Schloegel, K., Karypis, G., Kumar, V. (1997). *Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes*, Journal of Parallel and Distributed Computing, Vol. 47, S. 109 -124.

[68] Schulz, F. (2011). *Analysis I*, Oldenbourg Verlag, ISBN 978-3-486-70677-2.

[69] Simon, H. D. (1991). *Partitioning of Unstructured Problems for Parallel Processing*, Computing Systems in Engineering, Vol. 2, S. 135-148.

[70] Simon, H. D., Teng, S.-H. (1993). *How Good is Recursive Bisection?*, SIAM Journal on Scientific Computing, Vol. 18, S. 1436-1445.

[71] Strang, G. (2003). *Lineare Algebra*, Springer Verlag, ISBN 3-540-43949-8.

[72] Suhl, U. H., Suhl, L. M. (1990). *Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases*, INFORMS Journal on Computing, Vol. 2, No. 4, S. 325-335.

[73] Trifunovic, A., Knottenbelt, W. (1990). *A General Graph Model For Representing Exact Communication Volume in Parallel Sparse Matrix-Vector Multiplication*, Lecture Notes in Computer Science, Vol. 4263, S. 813-824.

[74] Uçar, B. (2008). *Heuristics for Matrix Symmetrization Problem*, PPAM 2007, LNCS 4967, Springer Verlag, S. 718-727.

[75] van Slyke, R. M., Wets, R. (1969). *L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming*, SIAM Journal on Applied Mathematics, Vol. 17, No. 4, S. 638-663.

[76] Walshaw, C., Cross, M., Everett, M.G. (1995). *A Parallelisable Algorithm for Optimising Unstructured Mesh Partitions*, Technical Report 95/IM/03, Computing and Mathematical Science, University of Greenwich, London, UK.

[77] Walshaw, C., Cross, M., Everett, M.G. (1997). *Parallel dynamic graph-partitioning for unstructured meshes*, Mathematical Research Report 97/IM/20, Centre for Numerical Modeling and Process Analysis, Univ. of Greenwich.

[78] Williamson, J. H., Reinsch, C. (1971). *Linear Algebra*, Springer Verlag, ISBN 3-540-05414-6.

[79] Zhang, X.-D. (2007). *The Laplacian eigenvalues of graphs: a survey*, Linear Algebra Research Advances, Nova Science Pub Inc, ISBN 1-60021-818-0, S. 203-230.

[80] Zurmühl, R., Falk, S. (1984). *Matrizen und ihre Anwendungen, Teil 1: Grundlagen*, Springer Verlag, ISBN 3-540-12848-4.

[81] Zhou, B. (2004). *On Laplacian Eigenvalues of a Graph*, Zeitschrift für Naturforschung, Vol. 59a, S. 181-184.

Abbildungsverzeichnis

Die Quellen aller Abbildungen, Graphen und Matrizen, die nicht aus eigenem Besitz stammen oder selbst erstellt wurden, sind hier aufgeführt.

- Seite 11, und Seite 53, Abbildung 5.2 rechts: Matrix von PDS [10], entnommen aus [25]
- Seite 12, Bild „Dantzig“:

  `http://www.freeinfosociety.com/media/images/2994.jpg`

- Seite 21, Bild „Fiedler“:

  `http://www-history.mcs.st-and.ac.uk/BigPictures/Fiedler_2.jpeg`

- Seite 53, Abbildung 5.1: Matrix von p80x400b, MIPLIB 2010,

  `http://miplib.zib.de/miplib2010/p80x400b.php`

- Seite 53, Abbildung 5.2 links: Randomisierte Permutation der Matrix von PDS [10], entnommen aus [25]