Mengenpartitionen
und ihre Anwendung auf
ein Maschinenbelegungsproblem

Diplomarbeit

von

Ralf Borndörfer

aus

Neusäß

eingereicht

beim Institut für Mathematik

der Naturwissenschaftlichen Fakultät

der Universität Augsburg

im Dezember 1991

Erstgutachter : Prof. Dr. Martin Grötschel
Zweitgutachter: Prof. Dr. Karl Heinz Borgwardt

Ich erkläre hiermit, daß ich die vorliegende Arbeit selbstständig unter Anleitung von Prof. Grötschel angefertigt und keine anderen als die erwähnten Hilfsmittel benutzt habe.

Inhalt

0 Vorwort

In dieser Arbeit geht es um zwei Fragestellungen, die beide gleichberechtigt den Anlaß zu ihrer Anfertigung gaben.
Zunächst interessierte ein aus der Praxis kommendes Problem der Steuerung von Maschinen, wie es in ähnlicher Form bei einem großen deutschen Elektronikkonzern auftritt. Leiterplatten sind hierbei in einer vollautomatischen Produktionsanlage mit Hilfe von Robotern schnellstmöglich mit elektronischen Bauteilen zu bestücken.
Andererseits besteht ein Zusammenhang zu einem eher theoretischen Problem der Partition von Mengensystemen. Hier ist die Aufgabe, die Grundmenge eines Systems von Teilmengen der Grundmenge so in zwei Teile aufzuteilen, daß auch jedes Element des Mengensystems möglichst "halbiert" wird.
In den folgenden Ausführungen werden die beiden skizzierten Probleme teils unabhängig, teils aufeinander bezogen untersucht. Der Autor will jedoch nicht leugnen, daß sein Ziel stets vorrangig eine Lösung des erstgenannten praktischen Problems war, zu dessen Erreichung das theoretische Mengenpartitionsproblem ein Hilfsmittel darstellt. Aus diesem Grunde mag die Verteilung der Gewichte zwischen den beiden Schwerpunkten nicht ganz symmetrisch ausgefallen sein.
Im einleitenden ersten Kapitel wird das Maschinenbelegungsproblem vorgestellt. Annahmen für eine mathematische Modellierung werden diskutiert und getroffen. Insbesondere ist hier die Einschränkung auf nur zwei verwendete Produktionsroboter zu nennen, die einer leichteren Behandelbarkeit des Problems Vorschub leisten sollen. Die Arbeit ist in diesem Sinne als ein erster Einstieg in die betrachtete Problematik zu sehen. Diese wird mathematisch formuliert, wobei sich zwei das Produktionsmodell unter den getroffenen Annahmen bestimmende Einflußgrößen herauskristallisieren. Die erste bestimmt, in welcher Reihenfolge bestimmte Aufgaben von den Automaten zu erledigen sind, die zweite teilt Aufgaben auf die beiden Produktionsroboter auf. Die erste Formulierung stellt sich als unbefriedigend heraus; eine trickreichere Modellierung wird erarbeitet und als praktisch einsetzbar nachgewiesen.
Im zweiten Kapitel wird der Reihenfolgeparameter der Modellierung unabhängig von seinem Aufteilungspendant untersucht. Ein neuer polynomialer Algorithmus zu seiner optimalen Bestimmung wird vorgestellt. Es handelt sich hierbei um eine Verallgemeinerung eines von Gilmore und Gomory [Gilmore & Gomory, 1964] entwickelten Verfahrens zur Lösung eines ähnlichen Problems. Um den Algorithmus darstellen und seine Korrektheit beweisen zu können, müssen verschiedene bekannte Resultate über "pyramidale" Lösungen von Travelling Salesman Problemen hergeleitet werden. Der Autor hat versucht, die Theorie straff und durchsichtig darzustellen, indem er etwa die zur Konstruktion von optimalen "Pyramidentouren" nötigen Operationen in einem Hilfssatz zusammengefaßt hat, was durchsichtige Beweise der wichtigsten Resultate ermöglicht. Zum Abschluß des Kapitels werden noch die Probleme untersucht, die bei der Verwendung von mehr als zwei Automaten auftreten. Es wird gezeigt, daß die Situation in diesem Fall im Sinne der Komplexitätstheorie wesentlich schwieriger ist.
Im dritten Kapitel wird der Aufteilungsparameter des Produktionsmodells untersucht. Allgemein wird das Problem der Partition von Mengensystemen betrachtet und formuliert. Es wird gezeigt, daß die gefundenen Formulierungen im Sinne der Komplexitätstheorie schwierige Probleme darstellen. Anschließend werden Spezialfälle der interessierenden Partitionsprobleme ebenfalls als schwierig nachgewiesen und eine scharfe Grenze zwischen polynomial lösbaren und $\mathcal{NP}$-vollständigen Unterproblemen festgestellt. Um hier zu guten Resultaten zu gelangen, war es nötig, einige bekannte Resultate der Komplexitätstheorie herzuleiten. Die Philosophie des Autors bestand darin, die Schwierigkeit "exotischer" Probleme nachzuweisen, während er einige andere, nämlich `3SAT`, `PARTITION`, `THREE DIMENSIONAL MATCHING` und `TSP` als allgemein bekannt vorausgesetzt hat. Obwohl die bewiesenen Resultate nicht vom Autor stammen, so tun es doch die hier vorgestellten Beweise. Im weiteren Verlauf des Kapitels wird dann auf die Existenz approximativer Algorithmen eingegangen. Zuletzt wird ein neuer Algorithmus zur approximativen Lösung eines Partitionsproblems vorgestellt, der zwar nicht polynomial, aber nach Meinung des Autors einem Enumerationsverfahren deutlich überlegen ist.
Der Appendix enthält Standardmaterial zu Verschiedenem, der Graphen- und der Komplexitätstheorie. Der Leser braucht ihn nicht notwendig zu studieren, jedoch empfiehlt der Autor ein kurzes Überfliegen, um Unklarheiten in der Notation von vornherein auszuschließen. Die im Appendix gemachten Aussagen sind meistenteils unmittelbaren Folgen der Definitionen, so daß der Autor auf Beweise ganz verzichtet hat.

Interessant ist vielleicht noch eine neue Definition eines Codierugsschemas im Teil über Komplexitätstheorie, die der Autor als den ihm bekannten Vorgehensweisen überlegen ansieht.
Die Arbeit wurde mit TEX 3.1 auf einer Sun Sparc Station ELC geschrieben.
Zuletzt möchte sich der Autor herzlich bei Carlos Ferreira für das aufmerksame Korrekturlesen einer ersten (von Fehlern wimmelnden) Fassung und viele anregende Diskussionen im Zusammenhang mit dieser Arbeit bedanken. Er hofft, daß der Felerteufel dieselbe nicht allzusehr heimgesucht hat und wünscht allen Lesern viel Spaß bei der Lektüre.

Ralf Borndörfer, im Dezember 1991

1 Einleitung

1.1 Leiterplattenproduktion

Dieser Abschnitt beschreibt den Prozeß der industriellen Produktion von Leiterplatten. Er führt damit hin zu einem realen Problem der Steuerung von Maschinen.
Bei der Produktion elektronischer Geräte werden elektronische Bauteile zu Baugruppen zusammengefaßt und auf **Platinen** gemeinsam montiert, die als Einheiten in das Endprodukt eingebaut werden. Eine Platine besteht gewöhnlich aus mehreren dünnen Plastikschichten, auf die jeweils auf einer Seite durch Wegätzen bestimmter Teile einer dünnen Metallbeschichtung ein Muster metallischer **Leiterbahnen** aufgebracht ist. Die Leiterbahnen dienen der elektrischen Verbindung von Punkten auf den einzelnen Flächen der Schichten der Platine. Die derart vorbehandelten Plastikschichten werden zu einer harten Platte, eben der Platine oder **Leiterplatte**, verarbeitet, in die an vorbestimmten Stellen Löcher gebohrt werden. Jedes Loch durchbohrt insbesondere auch jede einzelne Schicht der Platine und damit möglicherweise auch eine Leiterbahn in dieser Schicht. Bei sinnvoller Anordnung der Leiterbahnen verbinden diese die Bohrungen, in die die Anschlüsse der elektronischen Bauteile gesteckt werden. Das Einsetzen der Bauteile in diese Bohrungen wird als **Bestückung der Platine** bezeichnet. Jeder Anschluß wird anschließend mit einem Lötpunkt versehen, der für den elektrischen Kontakt zu denjenigen Leiterbahnen aller Schichten der Platine, durch die die jeweilige Bohrung verläuft, und für eine mechanische Fixierung der im allgemeinen kleinen Bauteile auf der Platine sorgt. Die Leiterbahnen verbinden auf diese Weise die elektronischen Bauteile auf der Platine miteinander. Sie ersetzen eine aufwendige Verdrahtung und erzwingen eine Modularisierung der elektronischen Funktionseinheiten. Mehrere Schichten von Leiterbahnen werden benötigt, um die Kreuzungsfreiheit der Leiterbahnen sicherstellen zu können.

1.1 Probleme bei der Bestückung von Leiterplatten. Bei der Errichtung einer Produktionseinrichtung zur Bestückung von Leiterplatten muß folgenden Problemen Rechnung getragen werden.

1. Durch technischen Fortschritt und Modeerscheinungen verändern sich die Endprodukte schnell.
2. Ebenfalls durch technische Weiterentwicklungen verändern sich die verwendeten elektronischen Bauteile häufig.
3. Es ist wünschenswert, wenn in derselben Produktionseinrichtung mehrere verschiedene Typen von Platinen bestückt werden können; es ist dann zum Beispiel möglich, am Tag X Aufträge der Form “200 PCs, 300 Workstations” und am Tag Y eine andere Auftragszusammenstellung zu bearbeiten.
4. Die elektronischen Bauteile sind klein und im allgemeinen empfindlich. □

Traditionell werden Leiterplatten manuell bestückt. Arbeiter stecken dabei in mehreren aufeinanderfolgenden Bearbeitungsstufen die Bauteile in die Bohrungen der Platinen.
Diese Handarbeitsmethode ist zeitintensiv, fehleranfällig und zumindest in Ländern mit hohem Lohniveau teuer. Sie ist aber gut geeignet, die gerade dargestellten Probleme zu bewältigen, denn Menschen können sich schnell auf eine veränderte Produktpalette und auf sich verändernde Bauteile einstellen, sie sind in der Lage, verschiedene Typen von Platinen zu erkennen und dann richtig zu bestücken und sie behandeln die Bauteile mit der nötigen Sorgfalt.
Diese Kombination von Fähigkeiten, von denen sich die erste, zweite und dritte unter dem Schlagwort einer **flexiblen Produktionseinrichtung** zusammenfassen lassen während die vierte zum Problemkreis Sensorik und Steuerung gehört, war bis vor kurzem bei automatischen Produktionsanlagen nicht anzutreffen. Der technische Fortschritt (der einige der betrachteten Probleme direkt oder indirekt verursacht) führte jedoch zur Entwicklung einer neuen Generation von Industrierobotern, die erstmals in der Lage sind, bei der Bestückung von Platinen in Konkurrenz zur Handarbeitsmethode zu treten oder diese sogar zu übertreffen (und damit ihre eigene Produktion zu beeinflußen).
Ein solcher **Bestückungsautomat** besteht aus einem **Barcodeleser**, der einen auf eine Platine gedruckten Code lesen und so einen Platinentyp identifizieren kann, Magazinen für auf den Leiterplatten zu montierende Bauteile, einem “Greifarm”, der die Bauteile aus den Magazinen entnimmt oder zugeführt bekommt und diese auf der Platine in entsprechende Bohrungen einsetzt, und der zugehörigen Meß- und Steuerungstechnik.

In der Praxis wird jetzt ein Fließband aufgebaut, auf das am Anfang un- oder teilbestückte Platinen aufgebracht werden, die dann von im allgemeinen mehreren hintereinander am Fließband aufgestellten Automaten in dementsprechend mehreren Bearbeitungstufen mit Bauteilen bestückt werden und von dem am Ende die fertig bestückten Platinen abgenommen werden können. Die Automaten sind in der Lage, verschiedene Platinentypen zu bearbeiten, die sie vorher mit ihrem Barcodeleser identifizieren. Im Unterschied zur Handarbeitsmethode, wo meist die Arbeiter aufeinanderfolgender Bestückungsstufen ihre Leiterplatten nicht direkt von einem Vorgänger erhalten, sondern etwa einen Korb voll Platinen von der vorherigen Bearbeitungsstufe geliefert bekommen und diesen dann vollständig bearbeitet an die nächste Stufe weitergeben, arbeiten die Automaten am Fließband im Takt. Jeder Automat gibt die gerade bearbeitete Platine direkt an seinen Nachfolger weiter und erhält entsprechend gleichzeitig eine Platine von seinem Vorgänger.
Im folgenden sei stets eine derartige Produktionseinrichtung vorausgesetzt.

1.2 Problemstellung

Nunmehr wird auf die Problematik der Produktion von Leiterplatten eingegangen; Annahmen für eine Modellierung des Produktionsprozesses werden diskutiert und getroffen.
In dem im letzten Abschnitt vorgestellten Produktionsmodell stehen einzelne Bestückungsautomaten genau dann still, wenn nicht alle Teilbestückungen die gleiche Zeit in Anspruch nehmen; das Fließband setzt sich erst dann wieder in Bewegung, wenn der langsamste Automat fertig geworden ist. Je geringer diese Leerzeiten und damit die gesamte benötigte Bearbeitungszeit sind beziehungsweise ist, um so grösser ist die Anzahl der in der Fabrik bestückten Leiterplatten pro Zeiteinheit und auf desto mehr Platinen verteilen sich die Fixkosten der Produktion, was einen Kostenvorteil nach sich zieht. Ziel ist es deshalb, die Gesamtbearbeitungszeit, also die Zielgröße Zeit, möglichst zu minimieren.
Selbstverständlich ist es schon aufgrund unbekannter Daten unmöglich, diese Minimierung für die gesamte Produktion, also das gesamte Leben einer solchen Bestückungsanlage auch nur zu formulieren, geschweige denn zu durchzuführen. Realistischer und naheliegend ist dagegen eine Unterteilung der Produktion in Jobs. Ein **Job** bestehe hierbei aus dem Auftrag, gewisse Anzahlen von bestimmten Typen von Leiterplatten zu produzieren. Denkbar wäre etwa eine Unterteilung in Tagesjobs, für die die den Job charakterisierenden Daten nicht nur (am Vorabend) bekannt, sondern für die sich die Menge dieser Daten auch in "bewältigbarem" Umfang bewegt. Offensichtlich kann ein solcher Umfang stets durch Betrachtung von entsprechend kleinen Jobs erreicht werden. Die Betrachtung des Problems der Minimierung der Gesamtbearbeitungszeit eines Jobs im vorgestellten Modell zur Produktion von Leiterplatten mit Hilfe eines Fließbandes und von Bestückungsautomaten bildet einen Schwerpunkt dieser Arbeit.
Zur genaueren Formulierung der Fragestellung selbst erscheinen einige Annahmen nötig.

1.2 Annahme. Folgende Annahmen seien getroffen.

1. Die Bestückungsautomaten sind baugleich. Jeder Bestückungsautomat ist insbesondere technisch in der Lage, jeden Typ von Platinen mit jedem Typ von Bauteilen zu bestücken.
2. Die Bestückung mit einem Bauteil eines bestimmten Typs nimmt für jeden Typ von Platinen in jedem Stadium der Bestückung die gleiche Zeit in Anspruch.
3. Es gibt mehr Bauteile, als auf einem Automaten untergebracht werden können. □

Annahme 1.22 besagt insbesondere, daß die zur Bestückung benötigte Zeit unabhängig von der Bestükkungsposition eines Bauteils auf der Platine ist und daß bereits plazierte Bauteile den Automaten nicht behindern. Alle drei in Annahme 1.2 getroffenen Annahmen stellen keine grobe Verzerrung der in der Praxis auftretenden Situation dar.
Auf welche Weise kann auf die Produktionseinrichtung Einfluß genommen werden, wenn diese selbst physisch nicht verändert werden soll? Offenbar geht es um die Aufstellung eines **Produktionsplanes**, aus dem hervorgeht, welche Bauteile auf jeder einzelnen Leiterplatte jedes Typs von welchem Automaten bestückt werden. Die Betrachtung von einen solchen Plan beeinflussenden Größen, das Treffen einschränkender Annahmen in Bezug auf den Produktionsprozeß und die Auswahl einer Klasse von Produktionsplänen unter diesen Annahmen ist Gegenstand der sich anschliessenden Diskussion.
Folgende Faktoren spielen bei der Bestimmung eins optimalen Planes eine Rolle.

1. Die Verteilung der Bauteile auf die Automaten.
2. Die Reihenfolge des Aufbringens von Platinen verschiedener Typen auf das Fließband.

Die beiden genannten Parameter bestimmen allein zunächst noch nicht einen Produktionsplan, da zum Beispiel für die Bestückung einer Platine mit einem Bauteil eines bestimmten Typs mehrere Automaten in Frage kommen können oder etwa Leiterplatten gleicher Typen in verschiedenem Umfang von verschiedenen Automaten bestückt werden können, möglicherweise in Abhängigkeit von der Reihenfolge des Aufbringens der Platinen auf das Fließband. Kurzum, ein Produktionsplan kann im Extremfall eine individuelle Bestückungsplanung für jede einzelne Platine und nicht etwa nur für Typen von Leiterplatten vorsehen. Auf diese Weise kann versucht werden, Nutzen aus der Reihenfolge des Aufbringens von Platinen verschiedener Typen auf das Fließband oder aus dem mehrfachen Vorkommen von Bauteilen gleichen Typs auf verschiedenen Automaten zu ziehen. Ein derartiges Vorgehen scheint dem Autor jedoch aus verschiedenen Gründen unzweckmäßig zu sein.

Aus praktischer Sicht ist festzustellen, daß es nicht möglich ist, unabhängig jeden Automaten mit einem Programm zu versehen, das vorschreibt, welche Bauteile von diesem Automaten auf jeder Platine eines bestimmten Typs zu montieren sind. Vielmehr muß eine Zentralsteuerung installiert werden, die jeweils die einzelnen Automaten anweist, im jeweiligen Zustand der Produktionsanlage eine Bestückung mit gewissen Bauteilen vorzunehmen. Dies erscheint aufwendig. Weiterhin ist es beim Auftreten von Fehlern nicht ohne weiteres möglich, diese zu lokalisieren.

Aus theoretischer Sicht ist —bereits ohne näher auf etwaige Formulierungen des Problems einzugehen— klar, daß eine individuelle Bestückungsplanung für jede einzelne Platine nicht in zur Codierungslänge eines Jobs polynomialer Zeit aufgeschrieben werden kann. Zur Charakterisierung eines Jobs reicht es nämlich aus anzugeben, wie die Bestückung jeder einzelnen Platine aussehen soll und wieviele Platinen von jedem einzelnen Typ zu produzieren sind, während im Produktionsplan für jede einzelne Platine Daten vorhanden sind. Bei k Platinen eines Typs sind hierzu im ersten Fall $O(\log_2 k)$ Bits zuzüglich der platinentypbezogenen Informationen über die auf dem entsprechenden Typ zu plazierenden Bauteile nötig, im zweiten dagegen bereits mindestens für jede Platine ein Bit, also mindestens $O(k)$ Bits.

Der Autor ist der Meinung, daß man sich aus diesen Gründen auf die Betrachtung von auf Platinentypen bezogenen Bestückungsplänen beschränken sollte.

Eine praktische Umsetzung einer optimalen Reihenfolge des Aufbringens von Platinen auf das Fließband ist dagegen nicht von gleicher Schwierigkeit. Es ist lediglich erforderlich, am Eingang des Fließbandes eine Zuführungseinrichtung zu installieren, die in der Lage ist, etwa aus Containeren oder ähnlichen Tranporteinheiten jeweils eine Platine eines bestimmten Typs zu entnehmen. Eine solche Zuführung ist sicherlich von technisch geringerer Raffinesse als ein Bestückungsautomat. Auch vom mathematischen Standpunkt aus bestehen gegen ein solches Vorgehen keine Bedenken, wie sich herausstellen wird.

Weiterhin sei noch auf die Möglichkeit der mehrfachen Zuordnung von Bauteiltypen zu Automaten eingegangen. Ein Produktionsplan kann mit Hilfe von derartigen Mehrfachzuordnungen Bearbeitungszeit von Automaten auf andere übertragen. Einfachste Beispiele mit einem Platinen- und einem Bauteiltyp, wo das Bauteil zweimal in jede Leiterplatte einzustecken ist, überzeugen bereits vom Wert dieser Möglichkeit für eine Minimerung der Produktionszeit. Tatsächlich weiß der Autor aber nicht, wie man diese Option in geschickter Weise modellieren soll; er betrachtet daher nur den Fall, daß die Menge der Bauteile auf die Menge der Automaten abgebildet wird, so daß sich jedes Bauteil auf genau einem Automaten findet. Jedenfalls sei bemerkt, daß man durch Aufteilung eines Bauteiltyps in mehrere Untertypen der Art "x.tes Bauteil des Typs y auf Platine vom Typ z", die dann verschiedenen Automaten zugeordnet werden können, eine Mehrfachzuordnung von Bauteiltypen in begrenztem Umfang simulieren kann. Kritisch ist hierbei die Vielzahl von auf Bestückungsvielfachheiten von Bauteilen bezogenen Typen, die die genannte Transformation als sicherlich nicht polynomial charakterisieren. Eine Einschränkung des Problems auf beschränkte Vielfachheiten scheint wenig sinnvoll, da verschiedene kleine Bauteile durchaus in großer Anzahl auf Leiterplatten vorkommen können.

Zuletzt wird die Situation noch durch die Komplexität der Wechselwirkungen zwischen vielen hintereinander aufgestellten Automaten verkompliziert, die immer alle jeweils auf den langsamsten warten. Der Autor will sich, um die Situation zu vereinfachen, auf den Fall von nur zwei Automaten beschränken. Selbstverständlich stellt auch diese Annahme die praktische Anwendbarkeit von aus diesem Modell gezogenen Schlüssen gelinde gesagt in Frage, da in der Praxis selbstverständlich in der Regel mehr als zwei Automaten eingesetzt werden.

Zusammenfassend seien folgende zusätzliche, wie gesagt zum Teil einschneidende Annahmen im weiteren vorausgesetzt.

1.2 Annahme (Fortsetzung).

4. Am Fließband werden genau zwei Automaten hintereinander aufgestellt.
5. Es werden nur solche Produktionspläne betrachtet, die jeden Typ von Bauteilen genau einem Automaten zuordnen. Insbesondere werden Leiterplatten gleicher Typen stets auf gleiche Weise bestückt. □

Die beiden vorhin genannten Parameter Verteilung der Bauteile auf die Automaten und Reihenfolge des Aufbringens von Platinen auf das Fließband konstituieren unter diesen zusätzlichen Voraussetzungen einen vollständigen Produktionsplan, denn aus ihnen geht in eindeutiger Weise hervor, welche Bauteile von welchem Automaten in welche Leiterplatten einzustecken sind.
Das Problem der Minimierung der Bearbeitungszeit kann damit folgendermaßen formuliert werden.

1.3 Frage. Welche Zuordnung von Bauteiltypen zu den zwei Automaten und welche Reihenfolge des Aufbringens von Platinen auf das Fließband führt unter den Voraussetzungen Annahme 1.2 bei gegebenem Job zu einer Minimierung der Gesamtbearbeitungszeit des Jobs? □

Unter den Voraussetzungen Annahme 1.2 werde im weiteren stets Frage 1.3 betrachtet.

1.3 Problemformulierung

In diesem Abschnitt wollen wir das in Frage 1.3 umgangsprachlich dargestellte Problem mathematisch formulieren. Zunächst stellen wir die entsprechenden Begriffe bereit.

1.4 Definition. Hat ein Hypergraph genau zwei Kanten, so heißt er **Partition** und seine Kanten heißen auch **Hälften**. □

Eine Aufteilung von n nummerierten Typen von Bauteilen auf zwei Bestückungsautomaten kann durch eine Partition $H = \big(\{\,1,\ldots,n\,\},\{\,e_1,e_2\,\}\big)$ mit der Interpretation

$$e_i = \{\,\text{Menge der von Automat } i \text{ zu bestückenden Typen von Bauteilen}\,\},\quad i = 1,2$$

dargestellt werden.

1.5 Definition. Sei m eine natürliche Zahl, s aus $\mathbb{N}^m$ und $S = \sum_{i=1}^{m} s_i$. Dann heißt eine Funktion $\sigma : \{\,1,\ldots,S\,\} \to \{\,1,\ldots,m\,\}$ mit der Eigenschaft

$$\big|\sigma^{-1}\big(\{\,i\,\}\big)\big| = s_i,\quad i = 1,\ldots,m$$

verallgemeinerte Permutation von $\{\,1,\ldots,m\,\}$, die Zahlen s_i heißen **Vielfachheiten** und S **Länge** der verallgemeinerten Permutation σ. □

1.6 Beispiel. $(1,2,2,3,3,3)^t$ und $(3,2,1,3,2,3)^t$ sind verallgemeinerte Permutationen der Länge 6 von $\{\,1,2,3\,\}$; der Vektor s der Vielfachheiten ist in beiden Fällen $(1,2,3)^t$, denn die 1 kommt jeweils einmal, die 2 zweimal und die 3 dreimal vor. Ausdrücklich sei darauf hingewiesen, daß die Notation einer verallgemeinerten Permutation als Vektor (vergleiche den Appendix Verschiedenes) bereits die Reihenfolge bezeichnet, in der die Elemente schließlich auftreten sollen. Anders als bei normalen Permutationen kommen Bilder mehrfach vor; eine Darstellung etwa in Analogie zur Darstellung einer Permutation als Produkt von Faktoren scheidet daher aus. □

Eine verallgemeinerte Permutation σ mit Vielfachheiten s_i und Länge S und eine Reihenfolge des Aufbringens von insgesamt S Leiterplatten m verschiedener Typen auf das Fließband unserer Produktionsanlage, wobei von Typ i jeweils s_i Platinen auf das Fließband aufzulegen sind, entsprechen einander über die Beziehung

$$\sigma(k) = i \iff \text{Die } k\text{-te auf das Fließband aufgebrachte Leiterplatte ist vom Typ } i\ ,\quad k = 1,\ldots,S.$$

Unser Optimierungsproblem kann damit wie folgt formuliert werden.

1.7 Problem. MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS.
Instanz. Gegeben sei ein Job, der die Herstellung von jeweils s_i Einheiten einer Leiterplatte vom Typ i für jeden von insgesamt m Typen verlangt. Auf einer Platine vom Typ i sind dabei $m_{ij} \in \mathbb{N}_0$ Einheiten eines Typs j von insgesamt n Typen von Bauteilen zu plazieren. Die Bestückung mit einer Einheit eines Bauteils vom Typ j nehme hierbei jeweils $c_j \in \mathbb{N}$ Zeiteinheiten in Anspruch. Es bezeichne $c_{ij} := m_{ij}c_j$ für alle Platinentypen i und alle Bauteiltypen j die zur Bestückung mit allen Bauteilen des Typs j auf einer Platine des Typs i insgesamt benötigte Zeit und S die Summe der Zahlen s_i.
Frage. Welche Partition $P = \big(\{\,1,\ldots,n\,\},\{\,f_1,f_2\,\}\big)$ auf der Menge der Bauteiltypen $\{\,1,\ldots,n\,\}$ und welche verallgemeinerte Permutation σ von $\{\,1,\ldots,m\,\}$ der Länge S mit Vielfachheiten s_i minimieren die für die Ausführung des Jobs insgesamt benötigte Zeit

$$c_{\sigma(1)\cdot\chi(f_1)} + \sum_{i=1}^{S-1} \max\big\{\, c_{\sigma(i)\cdot\chi(f_2)}, c_{\sigma(i+1)\cdot\chi(f_1)} \,\big\} + c_{\sigma(S)\cdot\big(\chi(f_2)\big)}?$$

(χ bezeichnet den Inzidenzvektor einer Menge, vergleiche den Symbolindex.) □

Die Zielfunktion von Problem 1.7 kann wie folgt interpretiert weren. Der Term

$$c_{\sigma(1)\cdot\chi(f_1)} = \sum_{j\in f_1} m_{\sigma(1)j}c_j$$

beschreibt die zur Bestückung der ersten auf das Fließband aufgebrachten Platine vom Typ $\sigma(1)$ mit den dem ersten Automaten zugeordneten Bauteilen, die in der Menge f_1 zusammengefaßt sind. Da es sich um die erste Platine handelt, ist noch keine Leiterplatte bis zum zweiten Automaten gelangt, der infolgedessen während dieser Zeit untätig ist. Bei der nächsten Bewegung des Fließbandes wird die erste Platine in den zweiten Automaten befördert, während eine Leiterplatte vom Typ $\sigma(2)$ in den ersten Automaten gelangt. Der zweite Automat bestückt die erste Platine in einer Zeit von

$$c_{\sigma(1)\cdot\chi(f_2)} = \sum_{j\in f_2} m_{\sigma(1)j}c_j = \sum_{j\notin f_1} m_{\sigma(1)j}c_j$$

Zeiteinheiten zu Ende, der erste Automat nimmt wieder eine erste Teilbestückung der zweiten Platine vor, für die er diesmal

$$c_{\sigma(2)\cdot\chi(f_1)}$$

Zeiteinheiten benötigt. Da das Fließband sich erst wieder in Bewegung setzen kann, wenn der langsamere der beiden Automaten mit der Bestückung fertig ist, vergehen insgsamt

$$\max\big\{\, c_{\sigma(1)\cdot\chi(f_2)}, c_{\sigma(1+1)\cdot\chi(f_1)} \,\big\}$$

Zeiteinheiten, bis das Fließband weiterläuft. Analog geht es weiter bis zur S-ten und letzten Platine vom Typ $\sigma(S)$, die keinen Nachfolger mehr hat. Bei ihrer abschliessenden Bestückung auf Automat 2 steht der erste Automat deshalb still und bis zur Beendigung des Jobs verstreichen nochmals

$$c_{\sigma(S)\cdot\big(\chi(f_2)\big)}$$

Zeiteinheiten.
Die Zeit, die zwischen den einzelnen Bestückungsvorgängen bei der Bewegung des Fließbandes verstreicht, ist für alle Bearbeitungsreihenfolgen gleich und geht in die Zielfunktion als Konstante ein. Bei der Formulierung von Problem 1.7 wurde deshalb auf eine Berücksichtigung entsprechender Terme verzichtet.

1.8 Beobachtung. Eine Codierung einer Instanz von Problem 1.7 ist durch eine Codierung der Matrix C und des Vektors der Vielfachheiten s gegeben, ihre Codierungslänge beträgt

$$O\big(\langle C\rangle + \langle s\rangle\big) \le O\left(mn \max_{\substack{i=1,\ldots,m\\ j=1,\ldots,n}} \langle c_{ij}\rangle + m \max_{i=1}^{m} \langle s_i\rangle\right),$$

die Codierungslänge einer Lösung $\big(\sigma, P\big)$ ist

$$\langle P\rangle + \langle \sigma\rangle = O\,(S+n) \le O\left(n + m \max_{i=1}^{m} s_i\right).$$

□

Die Codierungslänge einer Lösung für eine Instanz von Problem 1.7 ist nicht poynomial in der Codierungslänge der Instanz.

1.4 Modellierung

In diesem Abschnitt wollen wir einen Ausweg aus dem von Beobachtung 1.8 gezeigten Dilemma finden und eine bessere Formulierung des Problems angeben. Diese Formulierung wird eine verallgemeinerte Permutation nur noch indirekt enthalten; es wird daher insbesondere gezeigt, wie man aus der gegebenen Darstellung wieder eine verallgemeinerte Permutation rekonstruieren kann. Zunächst stellen wir einige Begriffe bereit.

1.9 Definition. Sei σ eine verallgemeinerte Permutation von $\{1,\ldots,m\}$ der Länge S mit Vielfachheiten s_i. Dann heißt die Matrix $\tau(\sigma) \in {\mathbb{N}_0}^{m\times m}$ definiert als

$$\big(\tau(\sigma)_{ij}\big) := \left(\left|\left\{\, k \in \{1,\ldots,S\} : \big(\sigma(k)\sigma(k \bmod S+1)\big) = (i,j) \,\right\}\right|\right)_{\substack{i=1,\ldots,m\\ j=1,\ldots,m}}$$

Nachfolgermatrix von σ. □

Eine Nachfolgermatrix einer verallgemeinerten Permutation σ zählt im Eintrag ij, wie oft in σ auf eine Komponente i eine Komponente j folgt; σ wird dabei als zyklisch verkettet betrachtet.

1.10 Beispiel. Die $(3,2,1)^t$-Permutation $(1,2,3,1,2,1)^t$ besitzt die Nachfolgermatrix

$$\tau\big((1,2,3,1,2,1)^t\big) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 1 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}.$$

$\tau\big((1,2,3,1,2,1)^t\big)_{12} = 2$ besagt, daß auf eine 1 zweimal eine 2 folgt, wohingegen $\tau\big((1,2,3,1,2,1)^t\big)_{11} = 1$ auf $\big(\sigma(6)\sigma(1)\big) = (1,1)$ zurückzuführen ist. □

1.11 Folgerung. Für eine gegebene verallgemeinerte Permutation σ der Länge S haben für jedes k aus $\{0,\ldots,S-1\}$ die verallgemeinerten Permutationen

$$\Big(\sigma\big((k+i-1) \bmod S+1\big)\Big)_{i=1,\ldots,S}$$

die gleiche Nachfolgermatrix. □

1.12 Definition. Sei $P = \big(V, \{f_1, f_2\}\big)$ eine Partition auf $\{1,\ldots,n\}$ und C eine Matrix aus ${\mathbb{N}_0}^{m\times n}$. Dann heißt die Matrix $\pi(C,P) \in {\mathbb{N}_0}^{m\times m}$ definiert als

$$\pi(C,P) := \big(\max\{\, c_{i\cdot}\chi(f_2), c_{j\cdot}\chi(f_1)\,\}\big)_{\substack{i=1,\ldots,m\\ j=1,\ldots,m}}$$

Kostenmatrix der Partition P bezüglich C. □

1.13 Beispiel. Wir betrachten die Partition $P = \Big(\{1,2,3,4,5\}, \big\{\{1,2\},\{3,4,5\}\big\}\Big)$ von $\{1,2,3,4,5\}$. Eine Matrix C aus ${\mathbb{N}_0}^{m\times 5}$ und die zugehörige Kostenmatrix $\pi(C,P)$ bezüglich P sind dann

$$C = \begin{pmatrix} 2 & 0 & 8 & 9 & 1 \\ 0 & 7 & 0 & 1 & 1 \\ 0 & 0 & 3 & 2 & 0 \end{pmatrix} \quad \text{und} \quad \pi(C,P) = \begin{pmatrix} 17 & 17 & 17 \\ 2 & 7 & 2 \\ 5 & 7 & 5 \end{pmatrix}.$$

□

Durch die Einführung eines "dummy"-Typs von Leiterplatten, der Anfang und Ende der Beschickung des Fließbandes mit Leiterplatten repräsentiert, läßt sich die lineare Bearbeitungsreihenfolge in einen Zyklus transformieren, was Fallunterscheidungen zur Betrachtung des Anfangs und des Endes der Sequenz überflüssig macht.

1.14 Folgerung. Sei eine Instanz von Problem 1.7 und eine Partition P gegeben; C^* entstehe aus C durch Hinzufügen einer Nullzeile als $m+1$-ter Zeile und für jede verallgemeinerte Permutation σ von $\{1,\ldots,m\}$ der Länge S zu Vielfachheiten s_i sei σ^* diejenige verallgemeinerte Permutation von $\{1,\ldots,m+1\}$ der Länge $S+1$ zu Vielfachheiten $(s_1,\ldots,s_m,1)$, die durch "Verlängern" von σ mit $m+1$ entsteht, das heißt

$$\sigma^* := \big(\sigma^t, m+1\big)^t.$$

Dann trifft folgende Beziehung zu.

$$c_{\sigma(1)\cdot\chi(f_1)} + \sum_{i=1}^{S-1} \max\big\{\, c_{\sigma(i)\cdot\chi(f_2)}, c_{\sigma(i+1)\cdot\chi(f_1)} \,\big\} + c_{\sigma(S)\cdot\chi(f_2)} = \big(\tau(\sigma^*), \pi(C^*,P)\big),$$

wobei $(\cdot,\cdot)$ das gewöhnliche Skalarprodukt bezeichne .

Beweis.

$$\begin{aligned}
&c_{\sigma(1)\cdot\chi(f_1)} + \sum_{k=1}^{S-1} \max\big\{\, c_{\sigma(k)\cdot\chi(f_2)}, c_{\sigma(k+1)\cdot\chi(f_1)} \,\big\} + c_{\sigma(S)\cdot\chi(f_2)} \\
=&\sum_{k=1}^{S+1} \max\big\{\, {c^*}_{\sigma^*(k)\cdot\chi(f_2)}, {c^*}_{\sigma^*\left(k \bmod (S+1)+1\right)\cdot\chi(f_1)} \,\big\} \\
=&\sum_{k=1}^{S+1} \pi(C^*,P)_{\sigma^*(k)\sigma^*\left(k \bmod (S+1)+1\right)} \\
=&\big(\tau(\sigma^*), \pi(C^*,P)\big).
\end{aligned}$$

□

Ordnet man also dem "dummy"-Typ eine Vielfachheit von 1 und Kosten von Null zu, so kann man eine optimale Bearbeitungsreihenfolge durch die Bestimmung einer "verallgemeinerten Hamiltontour" (beziehungsweise ihrer Nachfolgermatrix) zu durch die Kostenmatrix $\pi(C^*,P)$ bestimmten Kosten finden. Nach Folgerung 1.11 kann man umgekehrt den dummy $m+1$ kostenneutral an die letzte Stelle "shiften" und ihn dann streichen, das heißt eine Tour wird an dieser Stelle wieder ohne Kosten aufgeschnitten, so daß sich eine optimale Bearbeitungsreihenfolge ergibt. Problem 1.7 läßt sich jetzt prägnanter formulieren.

1.15 Problem. `MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS`.
Instanz. Gegeben sei eine Matrix C aus $\mathbb{N}_0{}^{m\times n}$ und m positive Zahlen s_i mit $S = \sum_{i=1}^m s_i$.
Frage. Welche Partition P von $\{1,\ldots,n\}$ und welche verallgemeinerte Permutation σ von $\{1,\ldots,m\}$ der Länge S mit Vielfachheiten s_i minimieren

$$\pi(C,P)\cdot\tau(\sigma)?$$

□

Will man unser Maschinenbelegungsproblem auf Problem 1.15 transformieren, so muß man natürlich in der Matrix C und den Vielfachheiten s_i einen dummy-Typ entsprechend Folgerung 1.14 vorsehen, indem man in C eine Nullzeile und für die entsprechende Vielfachheit eine 1 zur Verfügung stellt.
Welche Matrizen τ aus $\mathbb{N}_0{}^{m\times m}$ sind nun umgekehrt für gegebene Vielfachheiten s_i mögliche Nachfolgermatrizen? Aus der Definition entnimmt man unmittelbar die Notwendigkeit der Bedingungen

$$\sum_{j=1}^m \tau_{kj} = \sum_{i=1}^m \tau_{ik} = s_k, \quad k = 1,\ldots,m. \tag{1}$$

Diese allein sind aber noch nicht hinreichend, denn wie der Leser sieht, charakteriseren die Flusserhaltungsbedingungen (1) τ als eine Zirkulation im vollständigen Digraphen D_m mit m Knoten.

1.16 Satz. $\tau \in {\mathbb{N}_0}^{m\times m}$ erfülle die Bedingungen (1). Dann sind die folgenden Aussagen äquivalent.

1. τ ist Nachfolgermatrix einer verallgemeinerten Permutation σ auf $\{1,\ldots,m\}$ mit Vielfachheiten s_i, $i = 1,\ldots,m$.
2. τ, interpretiert als Zirkulation im vollständigen Digraphen D_m mit m Knoten, enthält eine aufspannende Arboreszenz mit Wurzel 1.
3. τ lässt sich nicht durch synchrones Vertauschen von Zeilen und Spalten auf Blockdiagonalform bringen, das heißt es existiert keine Permutation π, so daß $(\tau_{\pi(i)\pi(j)})$ eine Blockdiagonalmatrix ist.
4. Für jede Zerlegung von τ in eine Anzahl gerichteter Kreise C_k für gewisse Indizes k ist der Hypergraph $H = \Big(\{1,\ldots,m\}, \{\{C_k\}\}_k\Big)$ zusammenhängend.

Beweis.

1⇒2. Angenommen, τ enthält keine aufspannende Arboreszenz von D_m mit 1 als Wurzel und ohne Beschränkung der Allgemeinheit sei $\sigma(1) = 1$ (ansonsten "shiftet" man σ gemäß Folgerung 1.11). Dann existiert ein von $\sigma(1)$ verschiedener Knoten i, so daß τ keinen $\big(\sigma(1) = 1, i\big)$-Pfad enthält. Mit

$$k := \min\big\{\, k \in \{1,\ldots,S\} : \sigma(k) = i \,\big\}$$

enthält aber

$$\Big(\big(\sigma(1),\sigma(2)\big), \big(\sigma(2),\sigma(3)\big), \ldots, \big(\sigma(k-1),\sigma(k) = i\big)\Big)$$

einen $\big(\sigma(1), i\big)$-Pfad und damit auch τ, ein Widerspruch.

2⇒3. Angenommen, es existiert eine Permutation π, so daß $(\tau_{\pi(i)\pi(j)})$ Blockdiagonalgestalt besitzt. Sei I eine Mengen von Zeilen- beziehungsweise Spaltenindizes von τ, die zu einer quadratischen Untermatrix von $(\tau_{\pi(i)\pi(j)})$ gehören, sei also $(\tau_{\pi(i)\pi(j)})_{i,j\in I}$ ein Block von $(\tau_{\pi(i)\pi(j)})$. Wir dürfen annehmen, daß 1 zu I gehört. Betrachten wir den von I induzierten Schnitt in D_m. Da τ Blockdiagonalgestalt besitzt, gilt

$$\tau_a = 0 \quad \forall a \in \delta^+(I). \tag{2}$$

Ein Pfad von 1 zu einem Knoten im Komplement von I, wie er in einer aufspannenden Arboreszenz enthalten ist, muß aber den Schnitt $\delta^+(I)$ in einem Bogen ij mit $i \in I$ und $j \notin I$ einmal überwinden, was im Widerspruch zu (2) $\tau_{ij} > 0$ impliziert.

3⇒4. Angenommen, für eine Zerlegung der Zirkulation τ in eine Anzahl gerichteter Kreise C_k für gewisse Indizes k ist der Hypergraph $H = \Big(\{1,\ldots,m\}, \{\{C_k\}\}_k\Big)$ nicht zusammenhängend. Wir betrachten die Komponennte von H, deren Knotenmenge I den Knoten 1 enthält. Nach Voraussetzung gilt dann für jede von einem beliebigen Kreis C_k induzierte Zirkulation τ^k die Beziehung $i \in I, j \notin I \Longrightarrow \tau^k_{ij} = 0$ und damit auch $i \in I, j \notin I \Longrightarrow \tau_{ij} = \sum_k \tau^k_{ij} = 0$. Ist J das Komplement von I in $\{1,\ldots,m\}$ und schreiben wir $I = \{i_1,\ldots,i_{|I|}\}$ und $J = \{j_1,\ldots,j_{|J|}\}$, so bedeutet dies aber gerade, daß die Permutation π mit $\big(\pi(1),\ldots,\pi(m)\big) = (i_1,\ldots,i_{|I|},j_1,\ldots,j_{|J|})$ τ auf Blockdiagonalgestalt transformiert.

4⇒1. Sei die Zirkulation τ auf beliebige Weise in eine Anzahl k von gerichteten Kreisen $C_1,\ldots,C_k$ zerlegt. Bezeichnet $|\{C_i\}|$ für alle i die Anzahl der im Kreis C_i enthaltenen Knoten, so kann jeder Kreis C_i in eindeutiger Weise als Folge der in ihm in dieser Reihenfolge enthaltenen Knoten in der Form $C_i = \Big(c_{i_1},\ldots,c_{i_{|\{C_i\}|}}\Big)$ dargestellt werden. Hieraus konstruieren wir iterativ eine geeignete verallgemeinerte Permutation σ wie folgt. Wir starten mit $\sigma = \Big(c_{1_1},\ldots,c_{1_{|\{C_1\}|}}\Big)$. Hat σ einen Knoten j mit einem von C_1 verschiedenen Kreis C_i gemeinsam, wobei wir $\sigma(j) = c_{i_1}$ annehmen dürfen, so kann σ mit Hilfe von C_i auf

$$\Big(c_{1_1},\ldots,j = c_{i_1},\ldots,c_{i_{|\{C_i\}|}},c_{i_1} = j,\ldots,c_{1_{|\{C_1\}|}}\Big)$$

erweitert werden. Entweder kann diese Konstruktion iterativ für jeden Kreis C_i genau einmal durchgeführt werden, oder nach Erweiterung von σ durch mehrere (möglicherweise aber auch keinen) Kreis C_i zerfällt die Menge der Knoten von D_m in die beiden Mengen $U := \{i \in \{1,\ldots,m\} : \exists j : \sigma(j) = i\}$ und ihr Komplement $W := \{i \in \{1,\ldots,m\} : \nexists j : \sigma(j) = i\}$. Im zweiten Fall sind die Knoten von W genau die in den noch nicht zur Erweiterung von σ verwendeten Kreisen enthaltenen. Überschritte die Zirkulation τ aber einmal den Schnitt $\delta^+(U)$, so enthielte τ einen Bogen ij mit i aus U, j aus W

und positivem Fluß. Dieser Bogen muß dann aber in einem der Kreise C_i, sagen wir C_l, enthalten sein. C_l kann noch nicht zur Erweiterung von σ benutzt worden sein, denn sonst wäre j ein Element von U. Dann hätte aber σ mit Hilfe von C_l noch vergrößert werden können, was wir als unmöglich vorausgesetzt hatten: ein Widerspruch. Wenn die Zirkulation τ dagegen nie den Schnitt $\delta^+(U)$ überschreitet, dann wegen der Flußerhaltungsbedingungen (1) auch nie $\delta^-(U)$, das heißt es gilt

$$i \in U, j \in W \Longrightarrow \tau_{ij} = \tau_{ji} = 0,$$

für jeden Kreis C_k also für die von ihm induzierte Zirkulation τ^k ebenfalls $i \in U, j \in W \Longrightarrow \tau^k_{ij} = \tau^k_{ji} = 0$. Keine Kante von H enthält demnach sowohl Knoten aus U alsauch aus W, so daß im Falle, daß beide Mengen nicht leer sind, H nicht zusammenhinge, was unserer Voraussetzung widerspricht. Also produziert obige Vorgehensweise des Erweiterns von σ tatsächlich einen Vektor $\sigma \in \mathbb{N}^S$. Dieses σ ist, unabhängig von der Reihenfolge der Erweiterung, eine verallgemeinerte Permutation zu Vielfachheiten s_i, denn jeder Knoten aus $\{1, \ldots, m\}$ kommt in σ genau mit der Vielfachheit vor, die der Anzahl an Kreisen entspricht, in denen er enthalten ist. Diese Anzahl ist aber für jeden Knoten i wegen (1) gerade s_i. □

Satz 1.16.3 zeigt eine matriziell orientierte Charakterisierung einer Nachfolgermatrix. Satz 1.16.2 liefert dagegen eine einfache algorithmische Möglichkeit herauszufinden, ob eine Matrix Nachfolgermatrix einer verallgemeinerten Permutation zu gegebenen Vielfachheiten ist oder nicht.

1.17 Folgerung. Problem 1.15 ist (nur) $\mathcal{NP}$-schwer.
Beweis. Ein nichtdeterministischer Algorithmus kann eine positive Antwort für eine Instanz des zu Problem 1.7 gehörigen Entscheidungsproblems in polynomialer Zeit geben, indem er die Nachfolgermatrix einer geeigneten verallgemeinerten Permutation zu Vielfachheiten s_i und eine geeignete Partition P errät. Anschließend kann er in polynomialer Zeit die Kostenmatrix $\pi(C, P)$ berechnen, überprüfen, ob die erratene Matrix auch wirklich die Nachfolgermatrix einer verallgemeinerten Permutation von $\{1, \ldots, m\}$ zu Vielfachheiten s_i für alle i ist, indem er die Bedingungen (1) verfiziert und prüft, ob die der Nachfolgermatrix entsprechende Zirkulation eine aufspannende Arboreszenz mit Wurzel 1 enthält, die in Folgerung 1.14 beschriebene Transformation durchführen, die beiden resultierenden Vektoren skalar miteinander multiplizieren und den Vergleich mit dem angestrebten Zielfunktionswert durchführen. □

Obwohl man sich bei der Betrachtung von Problem 1.15 auf Nachfolgermatrizen beschränken kann, muß doch bei der praktischen Umsetzung einer gefundenen und als Nachfolgermatrix codierten Lösung eine Bearbeitungsreihenfolge generiert werden, also eine verallgemeinerte Permutation aus ihrer Nachfolgermatrix konstruiert werden. Hierzu ist der im Beweis von Satz 1.16 angegebene Algorithmus geeignet, der in "getunter" Form deshalb nochmals kurz dargestellt werden soll.

1.18 Algorithmus. Rekonstruktion einer verallgemeinerten Permutation aus einer Nachfolgermatrix.

Input. Nachfolgermatrix τ einer verallgemeinerten Permutation von $\{1, \ldots, m\}$ zu Vielfachheiten s_i für $i = 1, \ldots, m$ der Länge $S = \sum_{i=1}^{m}$.

Output. Verallgemeinerte Permutation σ von $\{1, \ldots, m\}$ zu Vielfachheiten s_i für $i = 1, \ldots, m$ mit Nachfolgermatrix τ.

Datenstrukturen. (Man beachte, daß die folgenden Variablen global zur Prozedur **walk** sind.) Liste zur Speicherung von m^2 gerichteten Kreisen in D_m als Listen der in ihnen in Reihenfolge enthaltenen Knoten und einer zu jedem Kreis gehörigen "Vielfachheit" aus $\mathbb{N}$.
Matrix aus ${\mathbb{N}_0}^{m \times m}$ zur Aufnahme der Matrix τ.
Vektor aus ${\mathbb{N}_0}^m$ zur Aufnahme der Vielfachheiten der verallgemeinerten Permutation s_i.
"Aktueller Knoten" v aus $\{1, \ldots, m\}$.
Zähler i und j.
Vektor σ aus $\{1, \ldots, m\}^S$ zur Aufnahme der verallgemeinerten Permutation.

Rekursive Prozedur walk.

Input. Kreis C aus der Liste der Kreise und seine Vielfachheit k.
Aktueller Knoten v.

Output. Knoten des Kreises C insgesamt k mal in der Reihenfolge ihres Auftretens in C, möglicherweise unterbrochen durch die zwischenzeitliche analoge Ausgabe anderer Kreise.

Datenstrukturen. Zähler i und j, jeweils lokal zur Prozedur walk.

begin
  { Buchhaltung }
  Ordne C als $(c_1, \ldots, c_{|\{C\}|})$ mit $c_1 = v$ an.
  Addiere zur Zirkulation τ die vom Kreis C induzierte Zirkulation so oft hinzu, wie die Vielfachheit von C angibt.
  Setze die Vielfachheit von C in der Liste der Kreise auf 0 und streiche so C aus der Liste.

  { Kreis C k-mal entlangwandern }
  for i=1 **to** k **do**
    for j=1 **to** $|\{C\}|$ **do**
    begin
      while $\sum_{k=1}^{n} \tau_{c_j k} < s_j$ **do** { noch ein anderer Kreis enthält c_j }
      begin
        Suche in der Liste der Kreise einen Kreis K mit positiver Vielfachheit l, der den Knoten c_j enthält.
        Rufe walk mit Parametern K, l und c_j auf.
      end;
      Erhöhe l um 1. Setze $\sigma(l) := c_j$.
    end;
end;

begin
  $\{\, \tau$ in höchstens m^2 gerichtete Kreise mit Vielfachheiten zerlegen $\}$
  Setze die Liste der Kreise gleich der leeren Liste.
  for i=1 to m **do**
    Setze $s_i = \sum_j \tau_{ij}$.
  while $\tau \neq 0$ **do**
  begin
    Finde in der Zirkulation τ einen gerichteten Kreis C.
    Trage ihn mit Vielfachheit $\min\{\tau_{ij} : ij$ ist ein Bogen von $C\}$ in die Liste der Kreise ein.
    Ziehe die von C induzierte Zirkulation mit der Vielfachheit von C von der Zirkulation τ ab.
  end;

  { Verallgemeinerte Permutation σ bestimmen }
  Setze $l := 0$.
  Rufe **walk** mit Parametern erster Kreis in der Liste der Kreise, seiner Vielfachheit und einem beliebigen in diesem Kreis enthaltenen Knoten auf.
  Gib σ aus.
end.

□

1.19 Hilfssatz. Algorithmus 1.18 funktioniert. Seine Komplexität ist bei geeigneter Implementierung der nicht spezifizierten Operationen

$$O\left(m^3 \max \langle \tau_{ij} \rangle + S\right)$$

Schritte.

Beweis. Am Beginn von Algorithmus 1.18 werden für alle i die Vielfachheiten s_i der verallgemeinerten Permutation aus der Nachfolgermatrix gemäß (1) berechnet. Dann wird eine Liste von Kreisen folgendermaßen initialisiert. τ ist eine Zirkulation. Solange diese verschieden von Null ist, enthält sie einen Kreis, sagen wir C, der vom Algorithmus gefunden wird. Die von ihm induzierte Zirkulation ist in der Zirkulation τ dann genau $\min\{\tau_{ij} : ij$ ist ein Bogen von $C\}$ mal enthalten. Diese Zahl wird als Vielfachheit des Kreises gespeichert. Abziehen der vom Kreis C induzierten Zirkulation von der Zirkulation τ mit der angegebenen Vielfachheit macht aus τ wieder eine Zirkulation, die Flußwerte auf allen Bögen von τ fallen dabei monoton. Tatsächlich "trocknet" ein Bogen sogar völlig aus. Da der zu τ gehörige Digraph D_m aber nur m^2 Bögen besitzt, ist

τ nach spätestens m^2 Schritten die Nullzirkulation, so daß in der Liste der Kreise höchstens m^2 Kreise mit positiven Vielfachheiten eingetragen sind. Die Superposition der von ihnen induzierten Zirkulationen mit einem Flußwert, der jeweils der Vielfachheit des induzierenden Kreises entspricht, ergibt nach Konstruktion aber gerade τ. Der Rest von Algorithmus 1.18 ist ein Spezialfall des im Beweis von Satz 1.16 verwendeten Algorithmus, wobei Kreise jeweils immer so weit wie möglich vorn in der entstehenden verallgemeinerten Permutation σ eingebaut werden; man beachte lediglich, daß zunächst bei der Initialisierung der Liste der Kreise die induzierten Zirkulationen iterativ aus der Zirkulation τ entfernt und nachher beim Durchlaufen wieder eingesetzt werden, so daß die Matrix τ am Ende des Algorithmus wieder völlig hergestellt ist.
Zum Auffinden von höchstens m^2 gerichteten Kreisen und zur Durchführung der entsprechenden Arithmetik auf der Matrix τ benötigt man bei Verwendung von zum Beispiel depth first search als Unterprogramm zum Aufspüren der Kreise

$$O\left(m^3 \cdot \max \langle \tau_{ij} \rangle\right)$$

Schritte, denn die Einträge in τ werden nie größer als die ursprünglich in τ enthaltenen Zahlen; zum Abwandern der Kreise mit ihrer Vielfachheit bei gleichzeitiger Ausgabe der verallgemeinerten Permutation benötigt man

$$O\left(S\right)$$

Schritte, insgesamt also die angegebene Anzahl. □

1.20 Beispiel. Wir betrachten die Zirkulation

$$\tau = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} & 2 & & & \\ & & 2 & 3 & \\ 2 & 3 & & & \\ & & 3 & & 2 \\ & & & 2 & \end{pmatrix} \end{array}$$

und finden die Kreise $\{(1,2),(2,3),(3,2)\}$, $\{(2,4),(4,3),(3,2)\}$ und $\{(4,5),(5,4)\}$ mit Vielfachheiten 2,3 und 2.
Der Algorithmus kann etwa wie folgt vorgehen. Er startet zum Beispiel in Kreis 1 bei Knoten 1 und gibt ihn aus. Dann geht er zu 2 und bemerkt, daß außer Kreis 1 auch noch Kreis 2 den Knoten 2 enthält. Das Abgehen von 1 wird (vor der Ausgabe von Knoten 2) unterbrochen und nun erst mal Kreis 2 bearbeitet. Da kein weiterer Kreis mehr 2 enthält, wird 2 zum ersten Mal für Kreis 2 ausgegeben. Weiter geht es zu Knoten 4, wo jetzt Kreis 3 bearbeitet wird. Beim Abwandern von Kreis 3 werden aber keine neuen Kreise mehr entdeckt, also wird Kreis 3 entsprechend seiner Vielfachheit zweimal abgegangen, wobei als letztes der Knoten 5 zum zweitenmal erreicht und ausgegeben wird. Nun wird mit Knoten 4 in Kreis 2 weitergemacht, Kreis 2 dreimal abgelaufen und dann Kreis 1 zuletzt zweimal umrundet. □

Offenbar ist es nicht möglich, einen Bestückungsplan in weniger als $O\left(S\right)$ Schritten in die Tat umzusetzen. Wichtig für die tatsächliche Ausführung eines solchen Planes ist daher nicht, daß die zur Steuerung benötigte verallgemeinerte Permutation eine Codierungslänge derselben Größenordnung besitzt, sondern ob es möglich ist, das jeweils nächste Element der verallgemeinerten Permutation schnell zu bestimmen. Algorithmus 1.18 ist hierzu in der Lage, denn er kann das jeweils nächste Element der verallgemeinerten Permutation durch Betrachtung von höchstens m^2 gerichteten Kreisen und entsprechender Vielfachheiten bestimmen. Er erscheint daher geeignet, online eine Steuerung einer Zuführungseinrichtung für Platinen zum Fließband unseres Produktionsmodells zu übernehmen.

2 Bearbeitungsreihenfolgen

2.1 Reihenfolgen

Angenommen, in unserem Produktionsmodell ist auf beliebige Weise eine Aufteilung der Bauteile auf die beiden am Fließband hintereinander aufgestellten Automaten vorgenommen worden, so daß die Kostenmatrix $\pi(C, P)$ bereits fest bestimmt ist; dann braucht nur noch eine optimale Bearbeitungsreihenfolge gefunden zu werden, so daß sich Problem 1.15 vereinfacht. Wir werden sehen, daß dieses Problem einen Bezug zu einem seit den 60er Jahren als polynomial lösbar bekannten Problem besitzt.

2.1 Problem. `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`.
Instanz. Gegeben seien m Zahlentripel (d_i, a_i, s_i) aus $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}$, wobei $d_1 \leq \ldots \leq d_m$ gelte.
Frage. Welche verallgemeinerte Permutation σ auf $\{1, \ldots, m\}$ mit Vielfachheiten s_i führt zu einer minimalen Gesamtbearbeitungszeit

$$\left(\{ \max d_i, a_j \}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}} \cdot \tau(\sigma)?$$

□

In Problem 2.1 repräsentieren die Zahlen d_i die zur Bestückung einer Leiterplatte vom Typ i auf Automat 1 benötigte Zeit, a_j entsprechend die auf Automat 2 benötigte.
`TWO PROCESSOR ASSEMBLY LINE SCHEDULING` ist kein **Flow Shop Scheduling Problem** (siehe [Garey & Johnson, 1979]), da bei einem solchen die Beschickung der einzelnen Prozessoren voneinander unabhängig ist und nicht wie hier im Takt erfolgt. Als Folge kann bei einem Flow Shop Scheduling Problem (auch und gerade bei Reihenfolgebedingungen für die "Tasks") ein schneller Job gegenüber einem langsamen Job "aufholen", was im von uns betrachteten Modell unmöglich ist.
Erlaubt man in Problem 2.1 nur einfache Vielfachheiten, das heißt $s_i = 1$ für alle i aus $\{1, \ldots, m\}$, so läßt sich das dann entstehende Problem als Spezialfall eines als polynomial lösbar bekannten Problems auffassen. Dieses Probem stellen wir nun vor.

2.2 Problem. `SINGLE STATE-VARIABLE MACHINE SEQUENCING PROBLEM` mit integrierbaren Funktionen f und g mit der Eigenschaft $f + g \geq 0$.
Instanz. Kostenmatrix C aus ${\mathbb{N}_0}^{m \times m}$ und m rationale Zweitupel (d_i, a_j) mit $d_1 \leq \ldots \leq d_m$ und mit der Eigenschaft

$$c_{ij} = \left(\left\{ \begin{array}{ll} \int_{d_i}^{a_j} f(x)\,dx, & \text{falls } d_i \leq a_j \\ \int_{a_j}^{d_i} g(x)\,dx, & \text{falls } d_i \geq a_j \end{array} \right\} \right)$$

für alle i und j aus $\{1, \ldots, m\}$.
Frage. Was ist die kürzeste Tour im vollständigen Digraphen mit m Knoten und durch C definierten Bogengewichten? □

Es soll implizit die Voraussetzung getroffen sein, daß die Funktionen f und g in einer Weise codiert sind, die eine Berechnung der auftretenden Integrale in zur Codierungslänge der Instanz polynomialer Zeit möglich macht, so daß ein Codierungsschema prüfen kann, ob ein String eine Instanz des Problems darstellt oder nicht.

2.3 Beobachtung. Sei eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` gegeben. Dann gilt für jede verallgemeinerte Permutation σ von $\{1, \ldots, m\}$ mit Vielfachheiten s_i

$$\begin{aligned}
\left(\max\{ d_i, a_j \}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}} \cdot \tau(\sigma) &= \left(d_i + \max\{ a_j - d_i, 0 \}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}} \cdot \tau(\sigma) \\
&= \left(d_i \mathbb{1}^t\right)_{i=1,\ldots,m} \cdot \tau(\sigma) + \left(\max\{ a_j - d_i, 0 \}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}} \cdot \tau(\sigma) \\
&= \sum_{i=1}^{m} d_i s_i + \left(\left\{ \begin{array}{ll} \int_{d_i}^{a_j} 1\,dx, & \text{falls } d_i \leq a_j \\ \int_{a_j}^{d_i} 0\,dx, & \text{falls } d_i \geq a_j \end{array} \right\} \right) \cdot \tau(\sigma).
\end{aligned}$$

Da die Zahl $\sum_{i=1}^{m} d_i s_i$ eine Konstante ist, sind das auf einfache Vielfachheiten eingeschränkte `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` Problem und das `SINGLE STATE-VARIABLE MACHINE SEQUENCING PROBLEM` mit Funktionen $(f, g) = (1, 0)$ polynomial äquivalent. □

Ein polynomialer Algorithmus zur Lösung des `SINGLE STATE-VARIABLE MACHINE SEQUENCING PROBLEM`s wurde von Gilmore und Gomory angegeben ([Gilmore & Gomory, 1964]). Gibt es auch einen polynomialen Algorithmus zur Lösung von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`, das heißt auch im Fall beliebiger Vielfachheiten? Diese Frage soll nun untersucht werden.

2.2 Zwei Bestückungsautomaten

Um einen polynomialen Algorithmus zur Lösung des `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` Problems darstellen und seine Korrektheit beweisen zu können, benötigen wir einige technische Vorbereitungen.
Zwischen Permutationen ϕ der m-elementigen Menge $\{1, \ldots, m\}$ und Lösungen $x = (x_{ij})_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}}$ des **Assignment Problems** in einem bipartiten Graphen mit auf jeder Seite m Knoten besteht vermöge der Beziehung

$$\phi(i) = j \iff x_{ij} = 1$$

ein eineindeutiger Zusammenhang; wir wollen deshalb im folgenden beide identifizieren und auch von einem Assignment ϕ sprechen.
Ist ein vollständiger Digraph $D = (V, A)$ mit m Knoten, $V = \{1, \ldots, m\}$ und Bogengewichten $(c_{ij})_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}}$ gegeben, so kann eine untere Schranke für die Länge einer kürzesten Tour τ durch die Bestimmung eines optimalen Assignments ϕ bezüglich der Kostenmatrix C gefunden werden; ϕ besteht jedoch im allgemeinen aus mehreren nichttrivialen Faktorn ϕ_i und ist daher in der Regel keine Tour. Nichtsdestotrotz besteht ein Zusammenhang zwischen ϕ und τ. Da die Permutationen auf m Elementen eine Gruppe bilden, existiert eine eindeutig bestimmte Permutation ψ mit

$$\tau = \phi\psi.$$

Bezeichnen wir die **Kosten des Assignments** mit $c(\phi) := \sum_{i=1}^{m} c_{i\phi(i)}$, so betragen die durch die Multiplikation von ϕ mit ψ von rechts entstehenden **Zusatzkosten** bezüglich ϕ

$$c\phi(\psi) := c(\phi\psi) - c(\phi).$$

Ein mögliche Idee zur Bestimmung einer optimalen Tour ist also die folgende. Bestimme zunächst ein optimales Assignment ϕ; dann bestimme eine Permutation ψ mit minimalen Zusatzkosten $c\phi(\psi)$, so daß dessen Multiplikation mit ϕ eine optimale Tour τ liefert
Was sind erfolgversprechende Permutationen ψ, die es sich zu untersuchen lohnt?

2.4 Satz. Seien ϕ und ψ Permutationen auf $V = \{1, \ldots, m\}$ mit Faktoren ϕ_i und ψ_j und sei $C \in \mathbb{N}_0{}^{m \times m}$. Dann gelten folgende Aussagen.

1. Ist $\phi\psi$ eine Tour, so ist der Hypergraph $H = \left(V, \left\{ \{\phi_i\}, \{\psi_j\} \right\}_{i,j}\right)$ zusammenhängend.
2. Ist der Hypergraph $H = \left(V, \left\{ \{\phi_i\}, \{\psi_j\} \right\}_{i,j}\right)$ ein Baum, so ist $\phi\psi$ eine Tour.
3. $c\phi(\psi) = \sum_j c\phi(\psi_j)$.

Beweis.

1. Angenommen, H hängt nicht zusammen. Dann gibt es einen nichttrivialen leeren Schnitt $\delta(W)$, also mit $\emptyset \neq W \neq V$. Für jeden Faktor ρ von ϕ oder ψ gilt dann aber $\rho(i) \in W \iff i \in W$, was das gleiche auch für ihr Produkt, also für $\phi\psi = \prod \phi_i \prod \psi_j$ impliziert. $\phi\psi$ ist dann aber keine Tour.
2. Induktion über die Anzahl der Kanten des Baumes. Für eine Kante ist die Behauptung klar. Sei sie bewiesen für bis zu k Kanten ausschließlich und habe H jetzt k Kanten. Ohne Beschränkung der Allgemeinheit sei ϕ_1 ein Blatt von H. Dann ist nach Voraussetzung $\prod_{i>1} \phi_i \prod \psi_j$ eine Tour τ auf der Knotenmenge $\bigcup_{i>1} \{\phi_i\} \cup \bigcup_j \{\psi_j\}$; $\{\phi_1\}$ hat mit dieser Menge genau einen Knoten gemeinsam, also ist $\left(V, \left\{ \{\phi_1\}, \{\tau\} \right\}\right)$ ebenfalls ein Baum (mit zwei Kanten) und damit $\phi\psi$ eine Tour.

3.

$$\begin{aligned} c\phi(\psi) &= \sum_{i=1}^{m} c_{i\phi\psi(i)} - \sum_{i=1}^{m} c_{i\phi(i)} \\ &= \sum_{j} \left(\sum_{i \in \{\, \psi_j \,\}} c_{i\phi\psi(i)} - \sum_{i \in \{\, \psi_j \,\}} c_{i\phi(i)} \right) . \\ &= \sum_{j} \left(\sum_{i=1}^{m} c_{i\phi\psi_j(i)} - \sum_{i=1}^{m} c_{i\phi(i)} \right) \\ &= \sum_{j} c\phi(\psi_j) \end{aligned}$$

□

Satz 2.4 nennt Beziehungen zwischen Assignments und Touren an sich und im bewerteten Fall; er liefert eine notwendige und eine hinreichende Bedingung für das Zusammensetzen einer Tour aus Permutationen ϕ und ψ; er zeigt, daß die Faktoren unabhängig voneinander Zusatzkosten verursachen. Kombiniert man Satz 2.4.11, 2 und 3, so kann man versuchen, einen Baum aus Faktoren mit minimalen Zusatzkosten zu finden. Da Satz 2.4.2 aber nur hinreichend ist, wird ein solcher Ansatz sicherlich nur zum Ziel führen können, wenn etwa an die Matrix C gewisse Anforderungen gestellt werden. Was für Anforderungen sind günstig? Betrachtung der Kosten $c\phi(\psi)$ einer Permutation ψ bezüglich eines optimalen Assignments ϕ zeigt, daß diese Kosten bis auf die Konstante $-c(\phi)$ gerade die Kosten des Assignments ψ bezüglich der Matrix $C^\phi := \left(c_{i\phi(j)}\right)_{ij}$, also bis auf eine Konstante mit $C^\phi(\psi)$ übereinstimmen, denn es gilt $c^\phi_{i\psi(j)} = c_{i\phi\psi(j)}$ für alle i und j.

2.5 Beispiel. Sei $\phi = (1,2,3)$ und $\psi = (1,2)$, dann ist $\phi\psi = (1,3)(2)$. Es ergeben sich etwa folgende Matrizen.

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad C^\phi = \begin{pmatrix} \mathbf{2} & 3 & 1 \\ 5 & \mathbf{6} & 4 \\ 8 & 9 & \mathbf{7} \end{pmatrix} \quad \text{und} \quad C^{\phi\psi} = \begin{pmatrix} \mathbf{3} & 2 & 1 \\ 6 & \mathbf{5} & 4 \\ 9 & 8 & \mathbf{7} \end{pmatrix},$$

es ist $c\phi(\psi) = (3+5+7) - (2+6+7) = 0$. □

Haben wir als Ziel vor Augen, daß es für ein optimales Assignment ϕ stets eine optimale Tour geben soll, so daß $\left(V, \left\{ \{\, \phi_i \,\}, \left\{ (\phi^{-1}\tau)_j \,\right\} \right\}_{i,j}\right)$ ein Baum ist (wobei die $(\phi^{-1}\tau)_j$ die Faktoren von $\phi^{-1}\tau$ seien), so erscheint es angebracht, diese Voraussetzungen als Voraussetzungen an C^ϕ zu formulieren. Im folgenden werden geeignete Voraussetzungen betrachtet.

2.6 Definition. Sei $D = (d_{ij})_{ij}$ eine Matrix aus $\mathbb{N}_0{}^{m\times m}$. Dann heißt die Matrix

$$\left(\sum_{k=i}^{m} \sum_{l=1}^{j} d_{ij} \right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}}$$

(kumulative) Verteilungsmatrix zur Dichte D; eine Matrix C ist eine **permutierte Verteilungsmatrix**, wenn es eine Permutation ϕ gibt, so daß C^ϕ eine Verteilungsmatrix ist. □

Für Zwecke der Bestimmung einer kürzesten Tour kann man die Klasse der permutierten Verteilungsmatrizen noch ein wenig erweitern.

2.7 Definition. Ganzzahlige Matrizen C und D heißen **vermöge einer additiv zulässigen Transformation äquivalent**, wenn C sich aus D durch Addieren von ganzen Zahlen zu Zeilen oder Spalten von C ergibt. ssigeÄquivalenzDefinition 2.7 □

2.8 Folgerung.

1. Additiv zulässige Äquivalenz ist eine Äquivalenzrelation.
2. Sind C und D vermöge einer additiv zulässigen Transformation äquivalent und quadratisch, so gibt es eine rationale Konstante k, so daß für alle Touren τ der Spaltenindizes von C und D

$$c(\tau) + k = d(\tau)$$

gilt, das heißt die Längen von Touren unterscheiden sich in Bezug auf C und D immer nur um eine Konstante. □

2.9 Satz.

1. Eine quadratische und ganzzahlige Matrix C ist vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix genau dann, wenn die Zahlen

$$d_{ij} := c_{ij} + c_{i+1j-1} - c_{i+1j} - c_{ij-1}$$

für alle $i \in \{1, \ldots, m-1\}$ und $j \in \{2, \ldots, m\}$ nichtnegativ sind.

2. Eine Matrix C ist vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix genau dann, wenn die Zahlen

$$c_{ij} + c_{kl} - c_{kj} - c_{il}$$

für alle $i < k$ und $j > l$ nichtnegativ sind.

Beweis.

1. $\Rightarrow$. Mit den Zahlen $z_i := \sum_{k=i+1}^{m} c_{k1}$ für i aus $\{1, \ldots, m\}$ und $s_j := \sum_{l=1}^{j-1} c_{ml}$ für j aus $\{1, \ldots, m\}$ ist die Matrix $C' := (c_{ij} + z_i + s_j)$ vermöge einer additiv zulässigen Transformation äquivalent zu C und es gilt ferner

$$d'_{ij} := c'_{ij} + c'_{i+1j-1} - c'_{i+1j} - c'_{ij-1} = d_{ij} \geq 0$$

für alle i aus $\{1, \ldots, m-1\}$ und j aus $\{2, \ldots, m\}$. Die Matrix

$$D' := \begin{pmatrix} (c_{i1})_{i=1,\ldots,m-1} & (d_{ij})_{\substack{i=1,\ldots,m-1 \\ j=2,\ldots,m}} \\ c_{m1} & (c_{mj})^t_{j=2,\ldots,m} \end{pmatrix}$$

ist dann wegen

$$\begin{aligned}
\sum_{\substack{k=i,\ldots,m \\ l=1,\ldots,j}} d'_{ij} &= \sum_{\substack{k=i,\ldots,m-1 \\ l=2,\ldots,j}} d_{ij} + \sum_{k=i}^{m-1} c_{k1} + \sum_{l=2}^{j} c_{ml} + c_{m1} \\
&= c_{ij} + c_{m1} - c_{i1} - c_{mj} + \sum_{i=1}^{m-1} c_{k1} + \sum_{l=2}^{j} c_{ml} + c_{m1} \\
&= c_{ij} - c_{i1} - c_{mj} + z_i + c_{i1} + s_j + c_{mj} \\
&= \begin{cases} c_{m1} & \text{, falls } i = m \text{ und } j = 1 \\ c_{i1} + z_i & \text{, falls } i < m \text{ und } j = 1 \\ c_{mj} + s_j & \text{, falls } j > 1 \text{ und } i = m \\ c_{ij} + z_i + s_j & \text{, falls } j > 1 \text{ und } i < m \end{cases} \\
&= c'_{ij}
\end{aligned}$$

für alle i und j die Dichtematrix von C', denn $z_m = s_1 = 0$.

$\Leftarrow$. Eine bei einer additiv zulässigen Transformation erlaubte Operation läßt die Nichtnegativität der betrachteten Größen invariant.

2.

$$c_{ij} + c_{kl} - c_{kj} - c_{il} = \sum_{\substack{p=i,\ldots,k-1 \\ q=l+1,\ldots,j}} (c_{pq} + c_{p+1q-1} - c_{p+1q} - c_{pq-1}) \geq 0.$$

□

Wollten wir aber nicht Bedingungen an die mittels eines optimalen Assignments permutierte Kostenmatrix eines vollständigen Digraphen stellen? Vermöge einer additiv zulässigen Transformation zu einer permutierten Verteilungsmatrix äquivalente Matrizen genügen bereits einer solchen Bedingung. Es existiert insbesondere ein optimales Assignment einer speziellen Form, so daß keine mehr oder weniger künstlichen Anforderungen in dieser Beziehung gestellt zu werden brauchen.

2.10 Satz. Ist C eine Matrix, so daß für eine Permutation ϕ die Matrix C^ϕ vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix ist, so ist ϕ ein optimales Assignment bezüglich der durch C definierten Kosten.
Beweis. ϕ ist offenbar genau dann ein optimales Assignment für C, wenn die identische Permutation ι ein optimales Assignment für die Matrix C^ϕ ist. Angenommen, das ist nicht der Fall und $\psi \neq \iota$ ist ein optimales Assignment für C^ϕ. Dann gibt es Indizes $i < j$ mit der Eigenschaft $\psi(i) > \psi(j)$. Es folgt mit Betrachtung der Transposition (i, j) und Satz 2.9.2

$$C^\phi\psi\big((i,j)\big) = C^\phi\big(\psi(i,j)\big) - C^\phi(\psi) = c^\phi_{i\psi(i,j)(i)} + c^\phi_{j\psi(i,j)(j)} - c^\phi_{i\psi(i)} - c^\phi_{j\psi(j)} = c^\phi_{i\psi(j)} + c^\phi_{j\psi(i)} - c^\phi_{i\psi(i)} - c^\phi_{j\psi(j)} \leq 0,$$

das heißt $\psi(i, j)$ ist ein nicht schlechteres Assignment als ψ. Ein endliche Anzahl solcher Vertauschungen liefert aber die identische Permutation, die damit doch optimal war. tsassignmentSatz 2.10 □

Wie der Leser vermutlich schon erraten hat, sind die uns speziell interessierenden Kostenmatrizen vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix.

2.11 Folgerung. Gegeben sei eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`. Dann ist mit dem Assignment ϕ mit der Eigenschaft $a_{\phi(1)} \leq \ldots \leq a_{\phi(m)}$ die Kostenmatrix $C^\phi := \big(\max\{\, d_i, a_{\phi(j)} \,\}\big)$ vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix.
Beweis. Nach Definition des Problems `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` gilt $d_1 \leq \ldots \leq d_m$. Es folgt

$$\begin{aligned}
&c_{i\phi(j)} + c_{i+1\phi(j-1)} - c_{i+1\phi(j)} - c_{i\phi(j-1)} \\
&\qquad = \max\{\, d_i, a_{\phi(j)} \,\} + \max\{\, d_{i+1}, a_{\phi(j-1)} \,\} \\
&\qquad\quad - \max\{\, d_{i+1}, a_{\phi(j)} \,\} - \max\{\, d_i, a_{\phi(j-1)} \,\} \\
&\qquad = \begin{cases}
a_{\phi(j)} + a_{\phi(j-1)} - a_{\phi(j)} - a_{\phi(j-1)} & \text{, falls } d_i \leq d_{i+1} \leq a_{\phi(j-1)} \leq a_{\phi(j)} \\
a_{\phi(j)} + d_{i+1} - a_{\phi(j)} - a_{\phi(j-1)} & \text{, falls } d_i \leq a_{\phi(j-1)} \leq d_{i+1} \leq a_{\phi(j)} \\
a_{\phi(j)} + d_{i+1} - d_{i+1} - a_{\phi(j-1)} & \text{, falls } d_i \leq a_{\phi(j-1)} \leq a_{\phi(j)} \leq d_{i+1} \\
a_{\phi(j)} + d_{i+1} - a_{\phi(j)} - d_i & \text{, falls } a_{\phi(j-1)} \leq d_i \leq d_{i+1} \leq a_{\phi(j)} \\
a_{\phi(j)} + d_{i+1} - d_{i+1} - d_i & \text{, falls } a_{\phi(j-1)} \leq d_i \leq a_{\phi(j)} \leq d_{i+1} \\
d_i + d_{i+1} - d_{i+1} - d_i & \text{, falls } a_{\phi(j-1)} \leq a_{\phi(j)} \leq d_i \leq d_{i+1}
\end{cases} \\
&\qquad = \begin{cases}
0 & \text{, falls } d_i \leq d_{i+1} \leq a_{\phi(j-1)} \leq a_{\phi(j)} \\
d_{i+1} - a_{\phi(j-1)} & \text{, falls } d_i \leq a_{\phi(j-1)} \leq d_{i+1} \leq a_{\phi(j)} \\
a_{\phi(j)} - a_{\phi(j-1)} & \text{, falls } d_i \leq a_{\phi(j-1)} \leq a_{\phi(j)} \leq d_{i+1} \\
d_{i+1} - d_i & \text{, falls } a_{\phi(j-1)} \leq d_i \leq d_{i+1} \leq a_{\phi(j)} \\
a_{\phi(j)} - d_i & \text{, falls } a_{\phi(j-1)} \leq d_i \leq a_{\phi(j)} \leq d_{i+1} \\
0 & \text{, falls } a_{\phi(j-1)} \leq a_{\phi(j)} \leq d_i \leq d_{i+1}
\end{cases} \\
&\qquad \geq 0
\end{aligned}$$

insbesondere für alle $i = 1, \ldots, m-1$ und $j = 2, \ldots, m$, was wegen Satz 2.9 die Behauptung beweist. □

Fassen wir unsere bisherigen Überlegungen zusammen. Angenommen, unsere Kostenmatrix C ist vermöge einer additiv zulässigen Transformation äquivalent zu einer permutierten Verteilungsmatrix. Dann ist zunächst eine Permutation ϕ zu bestimmen, so daß C^ϕ vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix ist; ϕ ist dann ein optimales Assignment für C. Jetzt kann versucht werden, gemäß Satz 2.4 individuell Faktoren einer Permutation ψ zu finden, so daß deren kostenminimale Komposition mit ϕ eine optimale Tour liefert; wir hatten bereits die Vorauswahl getroffen, nur solche Permutationen ψ zu betrachten, so daß $\big(V, \big\{\, \{\, \phi_i \,\}, \{\, \psi_j \,\} \,\big\}_{ij}\big)$ ein Baum ist. Die Kosten der Permutation ψ

beziehungsweise ihrer Faktoren können dann bezüglich der Verteilungsmatrix C^ϕ betrachtet werden, das heißt

$$c\phi(\psi) = C^\phi(\psi) - C^\phi(\iota),$$

wobei ι wieder die identische Permutation darstelle.
Nach was für Faktoren soll man Ausschau halten, wenn die Kostenmatrix (nach Permutation mit einem optimalen Assignment) vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix ist?

2.12 Definition. Ein Faktor ϕ einer Permutation ist eine **Pyramide** oder **pyramidal**, wenn er von der Form $(i_v, i_2, \ldots, i_p, i_{p+1}, \ldots, i_r)$ mit $i_1 < i_2 < \cdots < i_p > i_{p+1} > \cdots > i_r$ ist. Die Permutation selbst ist **pyramidal**, wenn jeder Faktor eine Pyramide ist. Ist ϕ eine Permutation und gilt für ein i die Beziehung $\phi^{-1}(i) < i > \phi(i)$, so ist p eine **Spitze** von ϕ, im Falle $\phi^{-1}(i) > i < \phi(i)$ heißt i eine **Antispitze**. Ist v eine Antispitze und p eine Spitze von ϕ, so heißt die Folge $\big(v, \phi(v), \phi^2(v), \ldots, p\big)$ eine **Flanke** von ϕ. Ein Faktor ρ von ψ mit Spitze p und Antispitze v heißt **dicht**, wenn $\{\,\rho\,\} = \{\,v, \ldots, p\,\}$ gilt, ψ selbst heißt **dicht**, wenn jeder Faktor dicht ist. □

Der folgende Hilfssatz faßt die Operationen zusammen, die im Falle einer vermöge einer additiv zulässigen Transformation zu einer Verteilungsmatrix äquivalenten Kostenmatrix C eine gegebene Assignment-Lösung verbessern oder jedenfalls nicht verschlechtern. Diese Operationen werden jeweils mit Hilfe von Transpositionen ausgeführt, die von rechts an das Assignment multipliziert werden. In Anbetracht der Tatsache, daß jede Permutation als Produkt von Transpositionen dargestelt werden kann, scheint diese Vorgehensweise jedenfalls nicht von vornherein zum Scheitern verurteilt zu sein.
Angenommen, eine Permutation ϕ ist gegeben und i und j seien verschieden. Dann gilt

$$c\phi\big((i,j)\big) = c_{i\phi(j)} + c_{j\phi(i)} - c_{i\phi(i)} - c_{j\phi(j)}.$$

Vergleichen wir mit Satz 2.9.2, so ist zu erkennen, daß für eine kostenminimale Permutation gewisse Bedingungen gelten müssen. Sind sie verletzt, so können wir transponieren und erhalten eine echt bessere Permutation; mit anderen Worten, ein Zweieraustausch oder **2Opt** liefert eine Verbesserung.
Zur einfacheren Darstellung des folgenden wollen wir einige Konventionen bezüglich der Notation treffen. Ist ρ ein pyramidaler Faktor einer Permutation ϕ, so schreiben wir bei der Darstellung von ρ etwa $\rho_i, \ldots, \rho_j$ und meinen damit den Teil der ansteigenden oder abfallenden Flanke von ϕ oder ψ, der von ρ_i und ρ_j begrenzt wird. Ist ρ_i eine Antispitze, so schreiben wir auch $\ldots, \rho_j$, ist ρ_j eine Antispitze, so wollen wir auch $\rho_i, \ldots$ schreiben.

2.13 Hilfssatz. Sei C vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix und ϕ eine Permutation mit Faktoren ϕ_i. Dann gelten die folgenden Aussagen.

1. Ist p eine Spitze von ϕ, sagen wir im Faktor ϕ_1, dann ist

   $$\phi \cdot \big(\phi^{-1}(p), p\big) = \big(\ldots, \phi^{-1}(p), \phi(p), \ldots\big)(p) \cdot \prod_{i \neq 1} \phi_i$$

   eine Faktorisierung von $\phi\big(\phi^{-1}(p), p\big)$ und

   $$c\phi\Big(\big(\phi^{-1}(p), p\big)\Big) \leq 0;$$

   war ϕ_1 pyramidal, so sind die beiden derart aus ϕ entstehenden Faktoren ebenfalls pyramidal. Eine analoge Aussage gilt für eine Antispitze v und die Transposition $(\phi^{-1}(v), v)$.
2. Ist (i) ein Faktor von ϕ und gibt es ein j, sagen wir im Faktor ϕ_1, mit $j < i < \phi(j)$, so gilt

   $$\phi \cdot (j,i) = \prod_{i \neq 1} \phi_i\big(\ldots, j, i, \phi(j)\big)$$

   und

   $$c\phi\big((j,i)\big) \leq 0;$$

war ϕ_1 pyramidal, so ist auch $\big(\dots, j, i, \phi(j)\big)$ pyramidal. Eine analoge Aussage gilt für j mit der Eigenschaft $j > i > \phi(j)$ mit der Transposition (j, i).

3. Sind ϕ_1 und ϕ_2 zwei pyramidale Faktoren von ϕ mit Antispitzen v_1 und v_2 und Spitzen p_1 und p_2 und es gilt
$$\phi^{-1}(p_1) < v_2 < p_1 < \phi^{-1}(v_2),$$
dann ist
$$\phi_1\phi_2\big(\phi^{-1}(v_2), \phi^{-1}(p_1)\big) = \big(v_1, \dots, \phi^{-1}(p_1), v_2, \dots, p_2, \dots, \phi^{-1}(v_2), p_1, \dots\big)$$
ein Faktor von $\phi\big(\phi^{-1}(v_2), \phi^{-1}(p_1)\big)$ und es gilt
$$c\phi\big(\phi^{-1}(v_2), \phi^{-1}(p_1)\big) \le 0;$$
waren ϕ_1 und ϕ_2 pyramidal, so ist auch $\big(v_1, \dots, \phi^{-1}(p_1), v_2, \dots, p_2, \dots, \phi^{-1}(v_2), p_1, \dots\big)$ pyramidal.

4. Ist $\phi_1 = (i_v, i_2, \dots, i_p, i_{p+1}, \dots, i_r)$ pyramidal mit Spitze i_p und Antispitze i_v, i aus der ansteigenden Flanke der Pyramide und j aus der absteigenden Flanke der Pyramide erfülle die Bedingung $j > i \ge \phi(j)$. Dann ist
$$\phi(i, j) = \big(i_v, \dots, i, \phi(j), \dots\big)\big(\phi(i), \dots, i_p, \dots, j\big)\prod_{i\neq 1}\phi_i$$
eine Faktorisierung von $\phi(i, j)$ und es gilt
$$c\phi\big((i, j)\big) \le 0;$$
war ϕ_1 pyramidal, so sind auch die beiden aus ihm entstehenden Faktoren pyramidal. Eine analoge Aussage gilt für i aus der absteigenden Flanke der Pyramide und j mit $j < i \le \phi(j)$

Beweis. Zu zeigen sind jeweils nur die Aussagen bezüglich der Zusatzkosten $c\phi(\cdot)$. Wir machen stets Gebrauch von Satz 2.9.2.

1.
$$\begin{aligned} c\phi\big(\phi^{-1}(p), p\big) &= C^\phi\big(\phi^{-1}(p), p\big) - C^\phi(\iota) \\ &= c_{\phi^{-1}(p)\phi(p)} + c_{p\phi\phi^{-1}(p)} - c_{\phi^{-1}(p)\phi\phi^{-1}(p)} - c_{p\phi(p)} \\ &= c_{\phi^{-1}(p)\phi(p)} + c_{pp} - c_{\phi^{-1}(p)p} - c_{p\phi(p)} \\ &\le 0. \end{aligned}$$

2.
$$c\phi\big((j, i)\big) = C^\phi(j, i) - C^\phi(\iota) = c_{j\phi(i)} + c_{i\phi(j)} - c_{j\phi(j)} - c_{i\phi(i)} = c_{ji} + c_{i\phi(j)} - c_{j\phi(j)} - c_{ii} \le 0.$$

3.
$$\begin{aligned} c\phi\big(\phi^{-1}(v_2), \phi^{-1}(p_1)\big) &= C^\phi\big(\phi^{-1}(v_2), \phi^{-1}(p_1)\big) - C^\phi(\iota) \\ &= c_{\phi^{-1}(p_1)\phi\phi^{-1}(v_2)} + c_{\phi^{-1}(v_2)\phi\phi^{-1}(p_1)} - c_{\phi^{-1}(p_1)\phi\phi^{-1}(p_1)} - c_{\phi^{-1}(v_2)\phi\phi^{-1}(v_2)} \\ &= c_{\phi^{-1}(p_1)v_2} + c_{\phi^{-1}(v_2)p_1} - c_{\phi^{-1}(p_1)p_1} - c_{\phi^{-1}(v_2)v_2} \\ &\le 0. \end{aligned}$$

4.
$$c\phi\big((i, j)\big) = C^\phi\big((i, j)\big) - C^\phi(\iota) = c_{i\phi(j)} + c_{j\phi(i)} - c_{i\phi(i)} - c_{j\phi(j)} \le 0.$$

□

Hilfssatz 2.13 besagt insbesondere, daß man auf einer pyramidalen Permutation im Falle, daß die Kostenmatrix vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix ist, ohne eine Steigerung der Kosten und unter Erhaltung der Pyramidalität die folgenden Operationen auf den Faktoren einer Permutation ausführen darf:

1. "Abschneiden" der Spitze eines Faktors einer Permutation; der Faktor zerfällt dadurch in zwei Faktoren, von denen einer nur die Ex-Spitze, der andere den Rest der Pyramide enthält.

2. Einsetzen eines einelementigen Faktors in die ansteigende oder abfallende Flanke einer Pyramide, sofern die Antispitze der Flanke kleiner und die Spitze größer als das Element ist, auf dem der einelementige Faktors "operiert".
3. Zusammensetzen zweier sich nur in jeweils einem Element "überlappenden" pyramidalen Faktoren zu einen Faktor.
4. "Abschneiden" nicht nur der Spitze, sondern einer ganzen Unterpyramide, so daß sich als neue Faktoren eine Teilpyramide mit alle Elementen der alten Pyramide oberhalb des "Schnittniveaus" (i in Hilfssatz 2.13) und ein Pyramidenstumpf mit allen Elementen unterhalb und einschließlich des Schnittniveaus ergeben.

2.14 Hilfssatz. Sei C^ϕ vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix und ψ eine Permutation, so daß $\left(V, \left\{ \{ \phi_i \}, \{ \psi_j \} \right\}_{i,j}\right)$ ein Baum ist, dann gelten folgende Aussagen.

1. Ist ψ nicht pyramidal, so kann jeder Faktor ψ_j zu einem pyramidalen Faktor ψ'_j umgestellt werden (das heißt $\{ \psi_j \} = \{ \psi'_j \}$), ohne das zusätzliche Kosten entstehen; insbesondere ist der Hypergraph $\left(V, \left\{ \{ \phi_i \}, \{ \psi'_j \} \right\}_{i,j'}\right)$ wieder ein Baum.
2. Ist ψ pyramidal aber nicht dicht, so gibt es eine Permutation ψ' derart, daß $c\phi(\psi') \leq c\phi(\psi)$ gilt und daß für jeden Faktor ψ_j von ψ ein Faktor ψ'_k von ψ' mit der Eigenschaft $\{ \psi_j \} \subseteq \{ \psi'_k \}$ existiert; insbesondere ist der Hypergraph $\left(V, \left\{ \{ \phi_i \}, \{ \psi'_j \} \right\}_{i,j'}\right)$ zusammenhängend.
3. Ist ψ pyramidal und dicht und der Hypergraph $\left(V, \left\{ \{ \phi_i \}, \{ \psi_j \} \right\}_{i,j}\right)$ zusammenhängend, so gibt es eine Permutation ψ', so daß auch ψ' pyramidal und dicht ist, für jeden Faktor ψ'_k von ψ' ein Faktor ψ_i von ψ mit der Eigenschaft $\{ \psi'_k \} \subseteq \{ \psi_i \}$ existiert, daß $c\phi(\psi') \leq c\phi(\psi)$ zutrifft und daß der Hypergraph $H = \left(V, \left\{ \{ \phi_i \}, \{ \psi'_j \} \right\}_{i,j}\right)$ ein Baum ist.

Beweis.

1. Ist ein Faktor ψ_j nicht pyramidal, so gibt es neben der (natürlich eindeutig bestimmten) größten Spitze p noch andere Spitzen und entsprechend außer der kleinsten Antispitze v noch weitere Antispitzen; offenbar ist sogar die Anzahl der Spitzen in einer Permutation immer gleich der der Antispitzen. Schneide iterativ gemäß Hilfssatz 2.13.1 die kleineren Spitzen so lange ab, bis sich eine pyramidale Permutation ergibt; diese besitzt die Spitze p und die Antispitze v. Setze nun die abgeschnittenen Spitzen mit Hilfssatz 2.13.2 eine nach der anderen an geeigneter Stelle in die ansteigende Flanke des sich jeweils ergebenden pyramidalen Faktors ein. Das Resultat ist ein pyramidaler Faktor mit nicht größeren Kosten.
2. Zunächst zeigen wir, wie man zwei sich "überlappende" pyramidale Faktoren zu einem pyramidalen Faktor verschmilzt. Seien also Faktoren ψ_1 und ψ_2 von ψ pyramidal und überlappend, das heißt ohne Beschränkung der Allgemeinheit gelte für die Spitzen p_1 und p_2 beziehungsweise die Antispitzen v_1 und v_2 von ψ_1 und ψ_2 die Beziehung $v_1 < v_2 < p_1 < p_2$.
   Angenommen, es gilt nicht die in Hilfssatz 2.13.3 geforderte Beziehung $\psi^{-1}(p_1) < v_2 < p_1 < \psi^{-1}(v_2)$; dann kann man entweder im Fall $v_2 < \psi^{-1}(p_1)$ die Spitze von ψ_1 oder im Fall $\psi^{-1}(v_2) < p_1$ die Antispitze von ψ_2 abschneiden. Im ersten Fall erhält man durch diese Operation drei Faktoren (p_1), ψ'_1 und ψ_2. ψ'_1 ist pyramidal und hat dieselbe Antispitze v_1 wie ψ_1. Sei seine Spitze p'_1. Die Größe $p'_1 - v_2$ hat sich aber im Vergleich zu $p_1 - v_2$ vermindert. Nach endlich vielen Schritten liegen daher pyramidale Faktoren ψ''_1 und ψ''_2 mit Antispitze v_1 beziehungweise Spitze p_2 und eine Reihe von abgeschnittenen ehemaligen Spitzen und Antispitzen vor, wobei ψ''_1 und ψ''_2 die Voraussetzungen von Hilfssatz 2.13.3 erfüllen und die ehemaligen Spitzen und Antispitzen alle größer als v_1 und kleiner als p_2 sind. ψ''_1 und ψ''_2 können jetzt nach Hilfssatz 2.13.3 zu einem pyramidalen Faktor verschmolzen werden; in diesen lassen sich nach Hilfssatz 2.132 die abgeschnittenen Ex-Spitzen und -antispitzen einsetzen. Der zweite Fall ist analog.
   Wir können also davon ausgehen, daß alle Faktoren von ψ pyramidal und nichtüberlappend sind. Ist ein Faktor ψ_j dennoch nicht dicht, so existiert ein dichter Faktor υ von ψ mit der Eigenschaft $\min\{ \psi_j \} < \min\{ \upsilon \} \leq \max\{ \upsilon \} < \max\{ \psi_j \}$. Auch dieser kann ohne Verursachung zusätzlicher Kosten nach Hilfssatz 2.13.2 in die aufsteigende Flanke von ψ_j eingesetzt werden, indem immer seine Spitze abgeschnitten und etwa in die aufsteigende Flanke von ψ eingesetzt wird.
3. Ist der Hypergraph H kein Baum, so enthält er aufgrund seines Zusammenhangs einen Kreis C, der notwendig eine Kante $\{ \psi_j \}$ mit mindestens zwei Elementen enthält. Seien j' und j'' zwei Elemente

dieser Kante, die in verschiedenen Kanten $\{\phi_{i'}\}$ beziehungsweise $\{\phi_{i''}\}$ enthalten sind; es gelte $j' < j''$. Ist j' die Antispitze oder j'' die Spitze von ψ_j, so spalten wir die Antispitze oder Spitze von ψ_j ab; ansonsten existitert ein k aus der absteigenden Flanke von ψ_j mit der Eigenschaft $\psi(k) \leq j' < k$. Wir spalten dann ψ_j mit Hilfe von Hilfssatz 2.13.4 in zwei pyramidale und dichte Faktoren, von denen der eine j' und der andere j'' enthält. In allen Fällen wird ψ_j in zwei pyramidale und dichte Faktoren ψ'_j und ψ''_j gespalten, von denen der eine j' und der andere j'' enthält. Da die Kante $\{\psi_j\}$ ein Element des Kreises C war, ist der Hypergraph $\left(V, \left\{\{\phi_i\}, \{\psi_j\}\right\}_{i,j\neq j',j''} \cup \left\{\left\{\psi'_j\right\}\right\} \cup \left\{\left\{\psi''_j\right\}\right\}\right)$ immer noch zusammenhängend, jedoch hat sich die Größe $\sum_{k\neq j}(|\{\psi_k\}|-1) + \left|\left\{\psi'_j\right\}\right| - 1 + \left|\left\{\psi''_j\right\}\right| - 1$ gegenüber $\sum_j \left(|\{\psi_j\}|-1\right)$ um eins vermindert, so daß wir nach einer endlichen Anzahl von Iterationen dieses Verfahrens eine Permutation ψ' mit den geforderten Eigenschaften produziert haben. □

Für eine vermöge einer additiven Transformation zu einer Verteilungsmatrix äquivalente Matrix C^ϕ reduziert Hilfssatz 2.14 die Anzahl der interessanten Permutationen ψ mit der Eigenschaft, daß $\phi\psi$ eine optimale Tour ist, gewaltig. Nichtsdestotrotz ist ihre Anzahl aber immer noch zu groß, um einfach enumerieren zu können; es ist im für uns speziell interessiernden Fall einer Kostenmatrix der Form $\left(\max\{d_i, a_j\}\right)$ jedoch möglich, die Faktoren der gesuchten Permutation ψ als Produkte von Transpositionen zu konstruieren.

2.15 Definition. Sei C^ϕ vermöge einer additiv zulässigen Transformation äquivalent zu einer Verteilungsmatrix. Dann heißt eine Menge von Transpositionen $\{\tau_k\}_k$ ein **Baum** bezüglich ϕ, wenn der Hypergraph $\left(V, \left\{\{\phi_i\}, \{\tau_k\}\right\}_{i,k}\right)$ ein Baum ist. Ein Baum bezüglich eines Assignments ϕ ist **minimal**, wenn er minimale Kosten

$$\sum_k c\phi(\tau_k)$$

besitzt. □

2.16 Satz. Sei C die Kostenmatrix einer Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` und es gelte $a_{\phi(1)} \leq \ldots \leq a_{\phi(m)}$. Dann gelten die folgenden Aussagen.

1. Ist ψ eine pyramidale und dichte Permutation mit Faktoren ψ_j, so daß der Hypergraph

   $$H = \left(V, \left\{\{\phi_i\}, \{\psi_j\}\right\}_{i,j}\right)$$

   ein Baum ist, so existiert ein Baum bezüglich ϕ von Tranpositionen τ_k für gewisse Indizes k der Form $(i, i+1)$ mit der Eigenschaft

   $$\sum_k c\phi(\tau_k) \leq c\phi(\psi).$$

2. Sind Transpositionen τ_k von der Form $(i, i+1)$ und bilden sie einen Baum bezüglich ϕ, so gibt es eine pyramidale und dichte Permutation ψ, so daß der Hypergraph $\left(V, \left\{\{\phi_i\}, \{\psi_j\}\right\}_{i,j}\right)$ ein Baum ist und daß $c\phi(\psi) = \sum_k c\phi(\tau_k)$ gilt.

Beweis. Sei D die Dichtematrix einer vermöge einer additiv zulässigen Transformation zu C^ϕ äquivalenten Verteilungsmatrix $V = (v_{ij})_{ij}$.

1. Seien $(j_1, \ldots, j_{p_j}, \ldots, j_{k_j}) = \psi_j$ die Faktoren von ψ mit Spitzen j_p und Antispitzen j_1 für alle j. Dann bildet

   $$\left\{(i, i+1) : i = j_1, \ldots, j_{p_j} - 1, j\right\}$$

   einen aufspannenden Baum bezüglich ϕ und es gilt

   $$\begin{aligned} c\phi(\psi) &= \sum_j c\phi(\psi_j) \\ &= \sum_j \left(C^\phi(\psi_j) - C^\phi(\iota)\right) \\ &= \sum_j \sum_{i=1}^{p_j-1} \left(c_{j_i\phi\psi(j_i)} - c_{j_i\phi(j_i)}\right) + \sum_j \sum_{i=p_j}^{k_j} \left(c_{j_i\phi\psi(j_i)} - c_{j_i\phi(j_i)}\right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_j \sum_{i=1}^{p_j-1} \big(v_{j_i\phi\psi(j_i)} - v_{j_i\phi(j_i)}\big) + \sum_j \sum_{i=p_j}^{k_j} \big(v_{j_i\phi\psi(j_i)} - v_{j_i\phi(j_i)}\big) \\
&= \sum_j \sum_{i=1}^{p_j-1} \sum_{\substack{k=j_i,\dots,m \\ l=j_i+1,\dots,\psi(j_i)}} d_{kl} - \sum_j \sum_{i=p_j}^{k_j} \sum_{\substack{k=j_i,\dots,m \\ l=\psi(j_i)+1,\dots,j_i}} d_{kl} \\
&\geq \sum_j \sum_{i=j_1}^{j_p-1} d_{ii+1} \\
&= \sum_j \sum_{i=j_1}^{j_p-1} \big(v_{i\phi(i+1)} + v_{i+1\phi(i)} - v_{i\phi(i)} - v_{i+1\phi(i+1)}\big) \\
&= \sum_j \sum_{i=j_1}^{j_p-1} \big(c_{i\phi(i+1)} + c_{i+1\phi(i)} - c_{i\phi(i)} - c_{i+1\phi(i+1)}\big) \\
&= \sum_k c\phi(\tau_k)
\end{aligned}
$$

mit gewissen Indizes k. (Für den Beweis ist es im übrigen nicht nötig vorauszusetzen, daß C durch eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` definiert wird; es genügt, wenn C^ϕ äquivalent zu einer Verteilungsmatrix ist.)

2. Wir betrachten eine Komponente des Graphen $G = \Big(V, \big\{\, \{\, \tau_k \,\} \,\big\}_k\Big)$. Sie ist von der Form

$$\big\{\, \{\, i, i+1 \,\}, \{\, i+1, i+2 \,\}, \dots, \{\, i+l-1, i+l \,\} \,\big\}.$$

Angenommen, für diese Komponente haben wir für ein $j < l$ bereits einen pyramidalen und dichten Faktor ψ mit Spitze $i+j$ und Antispitze i gefunden, der nicht größere Kosten $c\phi(\psi)$ als $\sum_{k=i}^{j-1} c\phi\big((k,k+1)\big)$ verursacht; im Falle $j = i+1$ ist dies offensichtlich möglich, indem wir einfach die erste Transposition zu unserem Faktor erklären. Wir wollen jetzt auch noch $j+1$ in den Faktor einbauen. Es ergeben sich zwei Fälle.

   a. $c^\phi_{jj} = d_j$. Dann ist $\psi\big(j, j+1\big) = \big(\dots, j, j+1, \psi(j), \dots\big)$ ein pyramidaler und dichter Faktor mit Spitze $j+1$ und Antispitze i und für seine Kosten erhalten wir

$$
\begin{aligned}
C\phi\psi\Big(\big(j, j+1\big)\Big) &= c^\phi_{jj+1} + c^\phi_{j+1\psi(j)} - c^\phi_{j\psi(j)} - c^\phi_{j+1j+1} \\
&= c^\phi_{jj+1} + d_{j+1} - d_j - c^\phi_{j+1j+1} \\
&= c^\phi_{jj+1} + c^\phi_{j+1j} - c^\phi_{jj} - c^\phi_{j+1j+1} \\
&= c\phi\big((j, j+1)\big).
\end{aligned}
$$

   b. $c^\phi_{jj} = a_j$. Dann ist $\psi\big(\psi^{-1}(j), j+1\big) = \big(\dots, \psi^{-1}(j), j+1, j\dots\big)$ ein pyramidaler und dichter Faktor mit Spitze $j+1$ und Antispitze i und für seine zusätzlichen Kosten errechnet sich

$$
\begin{aligned}
c\phi\psi\Big(\big(\psi^{-1}(j), j+1\big)\Big) &= c^\phi_{\psi^{-1}(j)j+1} + c^\phi_{j+1j} - c^\phi_{\psi^{-1}(j)j} - c^\phi_{j+1j+1} \\
&= a_{j+1} + c^\phi_{j+1j} - a_j - c^\phi_{j+1j+1} \\
&= c^\phi_{jj+1} + c^\phi_{j+1j} - c^\phi_{jj} - c^\phi_{j+1j+1} \\
&= C^\phi\big((j, j+1)\big).
\end{aligned}
$$

□

Man beachte, daß das im Beweis von Satz 2.162 angegebene Verfahren zur Konstruktion eines pyramidalen Faktors, der auf der Knotenmenge V einer Komponente eines Baumes bezüglich eines Assignments ϕ operiert, nur $O(|V|)$ Zeit benötigt.
Mit Hilfe von Satz 2.16 kann man jetzt das `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` Problem mit einfachen Vielfachheiten in polynomialer Zeit wie folgt lösen ([Gilmore & Gomory, 1964]).

1. Finde ein Assignment ϕ mit $a_{\phi(1)} \leq \ldots \leq a_{\phi(m)}$.
2. Finde die Faktoren ϕ_i des Assignments.
3. Berechne bezüglich der Kostenmatrix $C^\phi = \left(\max\{ a_i, d_{\phi(j)} \}\right)_{ij}$ einen bezüglich ϕ minimalen aufspannenden Baum; hierzu wird einfach im Graphen $\left(V, \{ \{ i, i+1 \} \}_{i=1,\ldots,m}\right) \prod_i \{ \phi_i \}$ ein minimaler aufspannender Baum berechnet, wobei die Kantengewichte der Kanten $\{ i, i+1 \}$ vor der Kontraktion natürlich $C^\phi\big((i, i+1)\big)$ waren.
4. Ist $\{ \tau_k \}_k$ der gerade berechnente minimale aufspannende Baum bezüglich ϕ, so bestimme man für jede Komponente des Graphen $\left(V, \{ \{ \tau_k \} \}_k\right)$ gemäß Satz 2.16.2 einen pyramidalen und dichten Faktor ψ_j, der genau auf den Knoten der zugehörigen Komponente operiert und der die gleichen Kosten verursacht.
5. Multipliziere ϕ mit $\psi = \prod \psi_j$ und erhalte so eine kürzeste Tour.

Obiger Algorithmus kann in einer Laufzeit von $O(m \log m(\max \langle d_i \rangle + \langle a_j \rangle))$ Schritten ausgeführt werden, denn die Bestimmung des optimalen Assigments benötigt mit Standardsortiermethoden genau die angegebene Anzahl Schritte, die Bestimmung der Faktoren ϕ_i kann in einfacher Weise in $O(m)$ Schtritten erfolgen, der Baum kann, da nur m Kanten betrachtet werden müssen, in $O(m \max \langle d_i \rangle + \langle a_j \rangle)$ Schritten gefunden werden, die Bestimmung der pyramidalen und dichten Faktoren kann in $O\left(m \max\left(\langle d_i \rangle, \langle a_j \rangle\right)\right)$ Zeit erfolgen und die Multiplikation von ϕ mit ψ erfordert $O(m)$ Operationen.
Unmittelbar ist klar, daß dieser Algorithmus uns bereits ein Verfahren zur Lösung des `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` Problems in die Hand gibt, indem man kurzerhand sämtliche Vielfachheiten in entsprechend viele einfache Vielfachheiten zerlegt. Diese Idee läßt sich mit ein wenig Aufwand zu einem wirklich polynomialen Verfahren ausbauen, denn die betrachtete spezielle Klasse von Matrizen wird durch nur wenige Einträge bestimmt und hat viele identische Zeilen und Spalten.

2.17 Definition.

1. Sei eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` gegeben. Sie und die Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten, erklärt durch S Zahlentripel $(a'_i, d'_i, 1)$, die den Bedingungen $d'_1 \leq \ldots \leq d'_S$ und $\forall i \in \{ 1, \ldots, m \} : \exists k : (a'_k, d'_k) = \ldots = (a_{k+s_i-1}, d_{k+s_i-1}) = (a_i, d_i)$ genügen, heißen einander **entsprechend**; in diesem Fall definieren wir die Zahlen $S_i := 1 + \sum_{j=1}^{i-1} s_j$ für i aus $\{ 1, \ldots, m+1 \}$ und die Kostenmatrix $C' := \left(\max\{ d'_i, a'_j \}\right)_{i,j=1,\ldots,S}$.
2. Sei eine einer Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` entsprechende Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten und ein Assignment ϕ' auf $\{ 1, \ldots, S \}$ mit der Eigenschaft $a'_{\phi'(1)} \leq \ldots \leq a'_{\phi'(S)}$ und Faktoren ϕ'_i gegeben. Sei H der Hypergraph $\left(\{ 1, \ldots, S \}, \{ \{ \phi_i \} \}_i\right) \prod_{i=1}^{m-1} \{ S_i, \ldots, S_{i+1} - 1 \}$ mit Komponenten H_i. Ist dann für eine Menge $T = \{ \tau_k \}_k$ von Transpositionen von $\{ 1, \ldots, S \}$ der Graph

$$\left(\{ 1, \ldots, S \}, \{ \{ \phi_i \}, \{ \tau_k \} \}_{i,k}\right) \prod_{i=1}^{m-1} \{ S_i, \ldots, S_{i+1} - 1 \} \prod_i \{ H_i \}$$

   ein Baum und wird für keine der Transpositionen $\{ \tau_k \}$ zu einer Schleife kontrahiert, so heißt T ein **kleiner Baum** bezüglich ϕ und bezüglich der Vielfachheiten s_i. Ein kleiner Baum ist **minimal**, wenn minimale Kosten $\sum_k c'(\tau_k)$ besitzt. □

Es sei bemerkt, daß bei der in Definition 2.17.1 angegebenen Transformation zu identifizierende Indizes aus $\{ 1, \ldots, S \}$ unmittelbar aufeinanderfolgen. Ein kleiner aufspannender Baum enthält dagegen keine Transpositionen, die auf zu identifizierenden Indizes opererieren und verbindet nur Faktoren, die nicht nach Reidentifikation der durch die Entsprechung vervielfältigten Knoten ohnehin leicht zusammenzusetzen sind. Die in ihm enthaltenen Transpositionen sind stets von der Form $(S_{i+1} - 1, S_{i+1})$.

2.18 Folgerung. Sei eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` und die ihr entsprechende Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten gegeben. Ist dann ϕ' ein optimales Assignment bezüglich C' auf $\{1,\ldots,S\}$ mit der Eigenschaft $a'_{\phi'(1)} \leq \ldots \leq a'_{\phi'(S)}$ und T ein Baum bezüglich ϕ', so existiert eine Teilmenge T' von T, so daß T' ein kleiner Baum bezüglich ϕ' und der Vielfachheiten s_i ist; es gilt ferner $\sum_{\tau\in T'} c'\phi'(\tau) \leq \sum_{\tau\in T} c'\phi'(\tau)$.
Beweis. Man entferne solange Transpositionen aus T, die die in Definition 2.17.2 angegebenen Bedingungen verletzen, bis dies nicht mehr der Fall ist. Da ϕ' ein optimales Assignment ist, hat jede Transposition τ positive Kosten $c'\phi'(\tau)$. Beides zusammen beweist die Behauptung. □

Mit Hilfe eines kleinen aufspannenden Baumes wollen wir die Faktoren eines optimalen Assignments nun zu einer verallgemeinerten Permutation zusammensetzen. Wir wollen ganz analog zum Algorithmus von Gilmore und Gomory vorgehen und die Komponenten eines kleinen Baumes zu pyramidalen und dichten Faktoren einer Permutation ψ zusammensetzen, deren Multiplikation mit dem optimalen Assignment zwar keine Tour, aber ein solches Assignment liefert, daß bei Identifikation der beim Übergang von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` zu 2 Processor Assmbly Line Scheduling mit einfachen Vielfachheiten gesplitteten Knoten eine verallgemeinerte Permutation existiert. Der entscheidende Punkt ist hierbei, daß wir uns nach Satz 2.16 bei der Bestimmung eines minimalen aufspannenden Baumes bezüglich ϕ' auf Transpositionen der Form $(i, i+1)$ beschränken können.

2.19 Satz. Sei eine Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` und die ihr entsprechende Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten gegeben; für ϕ' gelte $a'_{\phi'(1)} \leq \ldots \leq a'_{\phi'(S)}$ und ϕ'_i seien für gewisse Indizes i die Faktoren von ϕ'. Sei $T = \{\tau_k\}$ ein kleiner Baum bezüglich ϕ' und der Vielfachheiten s_i und seien ψ'_j die nach Satz 2.16 konstruierten und auf den Knotenmengen der Komponenten des kleinen Baumes operierenden pyramidalen und dichten Faktoren einer Permutation ψ', so ist die Nachfolgermatrix von $\phi'\psi'$ nach Identifikation der Knoten in den Mengen $\{S_i, S_{i+1}-1\}$für alle i aus $\{1,\ldots,m\}$ die Nachfolgermatrix einer verallgemeinerten Permutation σ zu Vielfachheiten s_i. War T auch minimal, so ist σ eine kürzeste verallgemeinerte Permutation zu Vielfachheiten s_i.
Beweis. Ist

$$h = \left(\bigcup_{i'} \{\phi_{i'}\} \cup \bigcup_{k'} \{\tau_{k'}\}, \left\{\{\phi'_{i'}\}, \{\tau_{k'}\}\right\}_{i',k'}\right)$$

eine Komponente von

$$H' = \left(\{1,\ldots,S\}, \left\{\{\phi'_i\}, \{\tau_k\}\right\}_{i,k}\right)$$

und sind $\psi_{j'}$ für gewisse j' die auf den Knotenmengen der Komponenten von $\left(\bigcup_{k'}\{\tau_{k'}\}, \left\{\{\tau_{k'}\}\right\}_{k'}\right)$ operierenden und nach Satz 2.16 konstruierten pyramidalen und dichten Faktoren von ψ', so ist $\rho' := \prod_{i'}\phi'_{i'}\prod_{j'}\psi'_{j'}$, eingeschränkt auf $\bigcup_{i'}\{\phi'_{i'}\}\cup\bigcup_{k'}\{\tau_{k'}\}$ nach Satz 2.16 eine Tour auf dieser Knotenmenge. Bezeichnen wir für die Komponenten H'_i von H' mit ρ'_i für gewisse i jeweils die entsprechenden Touren auf den Knotenmengen dieser Komponenten H'_i, dann gilt $\{\rho'_i\} = \{H'_i\}$ für alle i. Da für jede Tranposition τ_k von T ferner eine Kante $\{\rho'_i\}$ mit $\{\tau_k\} \subseteq \{\rho'_i\}$ existiert und der Hypergraph

$$H = \left(V, \left\{\{\phi'_i\}, \{\tau_k\}\right\}_{i,k}\right) \cdot \prod_{i=1}^{m-1} \{S_i, S_{i+1}-1\}$$

zusammenhing, hängt auch $\left(\{1,\ldots,S\}, \left\{\{\rho'_i\}\right\}_i\right)\prod_{i=1}^m \{S_i, S_{i+1}-1\}$ zusammen. Identifizieren wir die Knoten in den Mengen $\{S_i, S_{i+1}-1\}$ für alle i so, daß allen Elementen aus der Menge $\{S_i, S_{i+1}-1\}$ jeweils das Element i zugeordnet wird, so wird aus ρ'_i eine verallgemeinerte Permutation σ'_i von $V(H_i)$ zu Vielfachheiten $|\{\rho'_i\}\cap\{S_j, S_{j+1}-1\}|$ für alle j, wobei nur diejenigen dieser Zahlen betrachtet seien, die größer als Null sind. σ'_i ist für jedes i die Superposition von (keineswegs eindeutig bestimmten) gerichteten Kreisen $C_{j_{l_j}}$ für gewisse l_j. Auch $\left(\{1,\ldots,m\}, \left\{\left\{C_{j_{l_j}}\right\}\right\}_{j,l_j}\right)$ ist dann zusammenhängend, darüberhinaus kommt jeder Knoten i in gerade s_i Kreisen vor. Dies und Satz 1.16 impliziert, daß die Nachfolgermatrix von $\phi'\psi'$ nach Identifikation der Knoten in den Mengen $\{S_i, S_{i+1}-1\}$ für alle i, das heißt Addition der entsprechenden Spalten und Zeilen der Nachfolgermatrix, die Nachfolgermatrix einer verallgemeinerten Permutation zu Vielfachheiten s_i für alle i ist.

Für die kürzeste verallgemeinerte Permutation σ zu Vielfachheiten s_i gibt es eine (keineswegs eindeutig bestimmte) ihr entsprechende Permutation σ' gleicher Länge in der Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten. Für die Permutation $\psi' = \sigma'\phi'^{-1}$ gibt es einen Baum bezüglich ϕ' kürzerer Länge, der einen kleinen Baum noch kürzerer Länge enthält, die von einem minimalen kleinen Baum sicherlich nicht überboten wird; eine auf die genannte Weise konstruierte Permutation σ verursacht daher keine größeren Kosten und ist somit bereits optimal. □

Satz 2.19 zeigt, wie man auf einfache Weise die Komponenten eines Assignments zu einer verallgemeinerten Permutation zusammensetzt; der Fortschritt dort ist, daß ein kleiner Baum, auf dessen Betrachtung wir uns nach diesem Satz beschränken dürfen, nur wenige, nämlich maximal $m-1$ Transpositionen enthält; nach Definition eines kleinen Baumes kommen nämlich nur die Transpositionen $(S_{i+1}-1, S_{i+1})$ für i aus $\{1,\ldots,m-1\}$ als Mitglieder eines kleinen Baumes in Frage.
Können wir damit einen Algorithmus zur Bestimmung einer minimalen verallgemeinerten Permutation angeben? Wenn das Verfahren polynomial sein soll, so können wir nicht direkt mit verallgemeinerten Permutationen wie in Satz 2.19 arbeiten. Wir wollen die Nachfolgermatrix und die Darstellung einer verallgemeinerten Permutation als Superposition (weniger) gerichteter Kreise mit Vielfachheiten (vergleiche Satz 1.16 und Algorithmus 1.18) benutzen. Es ist nun im wesentlichen eine Buchhaltungsfrage, die zur Konstruktion einer kürzesten verallgemeinerten Permutation nötigen Operationen der Bestimmung eines minimalen kleinen Baumes, der Konstruktion von pyramidalen und dichten Faktoren aus seinen Komponenten und der Multiplikation einer Assignment-Lösung mit der sich ergebenden Permutation ψ auf der Nachfolgermatrix mehr oder minder direkt auszuführen. Zur Erleichterung des Verständnisses werden bereits während der Darstellung des Algorithmus Eigenschaften der manipulierten Objekte genannt.

2.20 Algorithmus. Bestimmung einer kürzesten verallgemeinerten Permutation zu gegebenen Vielfachheiten.

Input. Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`.

Output. Nachfolgermatrix einer kürzesten verallgemeinerten Permutation zu Vielfachheiten s_i, $i = 1,\ldots,m$.

Datenstrukturen. Assignment ϕ von $\{1,\ldots,m\}$.
Zahlen S_i und T_i, $i = 1,\ldots,m+1$.
Teilmengen $\{H_i\}$ von $\{1,\ldots,m\}$, $i = 1,\ldots,m$.
Graphen G beziehungsweise G' mit Knotenmenge $\{1,\ldots,m\}$.
Insgesamt höchstens $2m$ Teilmengen V_k^i von $\{1,\ldots,m\}$.

1. Bestimme ein Assignment ϕ auf $\{1,\ldots,m\}$ mit $a_{\phi(1)} \le \ldots \le a_{\phi(m)}$ und der Eigenschaft, daß $i<j$ und $d_i = d_j$ stets $\phi(i) < \phi(j)$ impliziert.
2. Bestimme Zahlen $S_i := \sum_{j=1}^{i-1} s_j$ und $T_i := 1 + \sum_{j=1}^{i-1} s_{\phi(j)}$ für alle i aus $\{1,\ldots,m+1\}$.
3. Bestimme eine Matrix τ vermöge

$$\tau_{ij} := \Big(\max\big\{\min_i\{S_{i+1}-1, T_{\phi(j+1)}-1\} - \max_j\{S_i, T_{\phi(j)}\}, 0\big\}\Big)_{ij}$$

   für alle i und j aus $\{1,\ldots,m\}$.
4. Bestimme Mengen H_i für entsprechende i als die Knotenmengen der Komponenten der Zirkulation τ.
5. Bilde den Graphen

$$G = \Big(\{1,\ldots,m\}, \big\{\{i,i+1\} : \exists H_j \ne H_k : i \in H_j, i+1 \in H_k\big\}\Big)$$

   mit Kantengewichten

$$\begin{aligned} w_{\{i,i+1\}} :=& c^{\phi}_{i\,\max\{k:T_k \le S_{i+1}\}} + c^{\phi}_{i+1\,\max\{k:T_k \le S_{i+1}-1\}} \\ & - c^{\phi}_{i\,\max\{k:T_k \le S_{i+1}-1\}} - c^{\phi}_{i+1\,\max\{k:T_k \le S_{i+1}\}} \end{aligned}$$

   für alle in ihm enthaltenen Kanten. Kontrahiere die Mengen $\{H_i\}$, merke aber zu jeder entstehenden Kante ihr Urbild. Die Kosten der neuen Kanten seien die Kosten ihrer Urbilder. Berechne im entstehenden Graphen $G\prod\{H_i\}$ einen minimalen Baum $T = \{\tau_k\}$.

6. Entferne aus dem Graphen G diejenigen Kanten, die nicht Urbilder von Kanten aus T bezüglich der Kontraktion von G zum Graphen $G\prod_i \{ H_i \}$ sind und erhalte so einen Graphen

$$G' = \Big(\{ 1, \ldots, m \}, \{ \{ \tau_{k'} \} \}_{k'} \Big).$$

   G' zerfällt in Komponenten G'_i für gewisse Indizes i mit entsprechenden Knotenmengen $\{ G'_i \} = \big\{ \min \{ G'_i \}, \max \{ G'_i \} \big\}$.
7. Finde in $\{ G'_i \}$ für jedes i die Indizes j, für die die Mengen $\{ S_j, S_{j+1} - 1 \}$ mehr als zwei Elemente haben und $\min \{ G'_i \} < j < \max \{ G'_i \}$ gilt; seien $j_1 < \ldots < j_{k_j}$ die entsprechenden Indizes und $j_0 := \min \{ G'_i \}$ und $j_{k_j+1} := \max \{ G'_i \}$. Bilde dann die $k_j + 1$ Mengen $V_k^i := \{ j_{k-1}, \ldots, j_k \}$ für $k = 1, \ldots, k_j + 1$. Bestimme für alle i und k einen auf der Menge V_k^i operierenden pyramidalen und dichten Faktor ψ_i bezüglich der Kostenmatrix

$$\Big(c^{\phi}_{k \max\{ j : T_j \le S_{l+1} - 1 \}} \Big)_{\substack{k=1,\ldots,m \\ l=1,\ldots,m}}.$$

8. Führe für jeden pyramidalen und dichten Faktor ψ_i folgende Operationen aus. Für jedes j aus der Menge $\{ \psi_j \}$ bestimme

$$k' := \max \{ k : T_k \le S_{j+1} - 1 \} \text{ und } k'' := \max \big\{ k : T_k \le S_{\psi(j)+1} - 1, \big\}$$

   vermindere $\tau_{j\phi'(k')}$ um eins und erhöhe $\tau_{j\phi'(k'')}$ um eins.
9. Gib τ aus. □

2.21 Satz. Algorithmus 2.20 funktioniert; seine Laufzeit ist bei geeigneter Ausführung der nicht spezifizierten Schritte

$$O\Big(m^2 \log m \big(\max \langle d_i \rangle + \max \langle a_j \rangle + \max \langle s_k \rangle \big) \Big);$$

insbesondere ist Algorithmus 2.20 polynomial.

Beweis. Wir verifizieren zunächst die Korrektheit und gehen hierzu Algorithmus 2.20 schrittweise durch, wobei wir insbesondere die dort behaupteten Eigneschaften prüfen.

1. Zwischen den optimalen Assignments ϕ von $\{ 1, \ldots, m \}$ und ϕ' von $\{ 1, \ldots, S \}$ mit $a'_{\phi'(1)} \le \ldots \le a'_{\phi'(S)}$ besteht die Beziehnung, daß man sich die Sequenz $(a'_{\phi'(1)}, \ldots, a'_{\phi'(S)})$ in der Art entstanden denken kann, daß in der Sequenz $(a_{\phi(1)}, \ldots, a_{\phi(m)})$ jedes Element $a_{\phi(i)}$ durch die Folge $(a_{\phi(i)}, \ldots, a_{\phi(i)})$ mit $s_{\phi(i)}$ Komponenten ersetzt wird. Die Bedingung, daß aus $i < j$ und $d_i = d_j$ stets $\phi(i) \le \phi(j)$ folgen soll, legt lediglich fest, daß von allen Assignments ϕ, die der Bedingung $a_{\phi(1)} \le \ldots \le a_{\phi(m)}$ genügen, dasjenige ausgewählt wird, bei dem die Urbilder gleicher Bilder gemäß ihrer Indizes aufsteigend geordnet sind.
2. Das Assignment ϕ' hat die unter 1 beschriebene spezielle Eigenschaft, daß immer $s_{\phi(i)}$ gleiche Komponenten aufeinanderfolgen; T_i ist für jedes i der Startindex eines solchen "Blocks".
3. Die Matrix τ zählt im Eintrag ij für alle i und j, wie oft i aus der Menge $\{ S_i, S_{i+1} - 1 \}$ von ϕ' in die Menge $\{ T_j, T_{j+1} - 1 \}$ abgebildet wird.
4. Aus 3 folgt, daß τ eine Zirkulation im vollständigen Digraphen mit m Knoten ist und daß

$$\sum_{k=1}^{m} \tau_{ik} = \sum_{k=1}^{m} \tau_{ki} = s_i$$

   für alle i aus $\{ 1, \ldots, m \}$ gilt. Die Mengen H_i sind die Knotenmengen der Komponenten des Hypergraphen $\Big(\{ 1, \ldots, S \}, \{ \{ \phi'_i \} \}_i \Big) \prod_{i=1}^{m} \{ S_i, S_{i+1} - 1 \}$, wobei Knoten i aus $\{ 1, \ldots, m \}$ als durch Kontraktion von $\{ S_i, S_{i+1} - 1 \}$ entstanden gelte.

5. und 6. Ferner gilt für alle i aus $\{1,\ldots,m-1\}$

$$\begin{aligned}
w_{\{i,i+1\}} =& \max\Big\{ d_i, a_{\phi\left(\max\{ k:T_k\le S_{i+1}\}\right)} \Big\} + \max\Big\{ d_{i+1}, a_{\phi\left(\max\{ k:T_k\le S_{i+1}-1\}\right)} \Big\} \\
&- \max\Big\{ d_i, a_{\phi\left(\max\{ k:T_k\le S_{i+1}-1\}\right)} \Big\} - \max\Big\{ d_{i+1}, a_{\phi\left(\max\{ k:T_k\le S_{i+1}\}\right)} \Big\} \\
=& \max\Big\{ d'_{S_i}, a'_{\phi'(S_{i+1})} \Big\} + \max\Big\{ d'_{S_{i+1}}, a'_{\phi'(S_i)} \Big\} - \max\Big\{ d'_{S_i}, a'_{\phi'(S_i)} \Big\} - \max\Big\{ d'_{S_{i+1}}, a'_{\phi'(S_{i+1})} \Big\} \\
=& \max\Big\{ d'_{S_{i+1}-1}, a'_{\phi'(S_{i+1})} \Big\} + \max\Big\{ d'_{S_{i+1}}, a'_{\phi'(S_{i+1}-1)} \Big\} \\
&- \max\Big\{ d'_{S_{i+1}-1}, a'_{\phi'(S_{i+1}-1)} \Big\} - \max\Big\{ d'_{S_{i+1}}, a'_{\phi'(S_{i+1})} \Big\} \\
=& c'\phi'\big((S_{i+1}-1, S_{i+1})\big).
\end{aligned}$$

Ein kleiner aufspannender Baum bezüglich des Assignments ϕ' enthält nur Kanten $\{S_{i+1}-1, S_{i+1}\}$ für i aus $\{1,\ldots,m-1\}$ und nach Definition von diesen nur diejenigen, für die $\{i,i+1\}$ nicht in einer der Mengen H_j enthalten ist. Zwischen den Urbildern bezüglich der Kontraktion von Bäumen T in $G\prod_i \{H_i\}$ und kleinen Bäumen T' bezüglich ϕ' und der Vielfachheiten s_i mit gleichen Kosten besteht daher vermöge

$$\{i,i+1\}\in T \iff \{S_{i+1}-1, S_{i+1}\}\in T'$$

eine eineindeutige Beziehung.

7. Wir betrachten den G' nach 3 und 6 entsprechenden Baum $T'=\{\tau'_{k'}\}_{k'}$. Pyramidalen, dichten und auf den Knotenmengen der Komponenten von $\left(\bigcup\{\tau'_{k'}\}, \{\{\tau'_{k'}\}\}_{k'}\right)$ operierenden Faktoren ψ'_j einer Permutation ψ', so daß $\phi'\psi'$ eine optimale Tour bezüglich C' ist, entsprechen dann ebenfalls eineindeutig pyramidale und dichte (dies beachte man besonders) Permutationen ψ_j vermöge

$$\psi_j=(j_1,\ldots,j_{p_j},\ldots,j_{k_j}) \iff \psi'_j=(S_{j+1}-1,\ldots,S_{j_{p_j}},\ldots,S_{j_{k_j}}).$$

Wegen

$$c^{\phi}_{k\,\max\{ j:T_j\le S_{l+1}-1\}} = {c'}^{\phi'}_{S_kS_l} = {c'}^{\phi'}_{S_{k+1}-1S_l}$$

haben sie auch die gleichen Kosten. Operieren zwei der Permutationen ψ_j auf (höchstens) einem gemeinsamen Element, nämlich der Spitze der einen und der Antispitze der anderen Permutation, so besitzt die Menge $\{S_p, S_{p+1}-1\}$ mehr als ein Element.

8. Wir betrachten j aus $\{\psi_i\}$ und das entsprechende j' aus ψ'_j und die Nachfolgermatrix τ'' der verallgemeinerten Permutationen von $\phi'\psi'=\phi'\prod\psi'_i$ von $\{1,\ldots,S\}$ zu einfachen Vielfachheiten sowie die Nachfolgermatrix τ' von ϕ'. Es gilt

$$\tau''_{j'\phi'\psi'(j')}=1=\tau'_{j'\phi'(j')} \quad \text{und} \quad \tau''_{j'\phi'(j')}=0=\tau'_{j'\phi'\psi'(j')}.$$

Bei Identifikation der Indizes in den Mengen $\{S_i, S_{i+1}-1\}$ ergeben sich gerade die oben angegebenen Operationen auf τ.

Das Ordnen von m Elementen d_i und a_j dauert $O\Big(m\big(\max\langle d_i\rangle+\max\langle a_j\rangle\big)\Big)$ Schritte, die Berechnung der Zahlen S_i und T_i nimmt $O\big(m\log m\max\langle s_i\rangle\big)$ (das $\log m$ stammt von der m-fachen Summation), die Berechnung der Nachfolgermatrix $O\big(m^2\log m\max\langle s_i\rangle\big)$, die Bestimmung der Mengen H_i $O\left(m^2\right)$, die Berechnung des Graphen G' höchstens $O\big(m\log m(\max\langle d_i\rangle+\max\langle a_j\rangle)\big)$ (man beachte, daß wegen der speziellen Form der interessierenden Transpositionen die Berechnung aller Kanten in in m linearer Zeit erfolgen kann), das Expandieren $O(m)$, die Bestimmung der Kostenmatrix der Pyramiden auf keinen Fall mehr als $O\big(m^2\log m(\max\langle d_i\rangle+\max\langle a_j\rangle)\big)$ und die Berechnung der Pyramiden selbst benötigt nicht mehr als $O\big(m\log m(\max\langle d_i\rangle+\max\langle a_j\rangle)\big)$ (man beachte, daß eine Untermatrix einer permutierten Verteilungsmatrix wieder eine solche ist) und das implizite Multiplizieren der bestimmten Permutation mit ϕ' $O\big(m\max\langle s_i\rangle\big)$ Schritte in Anspruch, so daß sich insgesamt die behauptete Laufzeit ergibt. □

2.22 Beispiel. Wir betrachten die Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`, gegeben durch 5 Dreitupel

$$(1,2,2),(2,1,2),(3,3,1),(4,4,2) \text{ und } (5,5,2).$$

Es gilt $d_1 = 1 \leq d_2 = 2 \leq d_3 = 3 \leq d_4 = 4 \leq d_5 = 5$. Die dieser Instanz entsprechende Instanz von `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` mit einfachen Vielfachheiten, die wir zur Erleichterung des Verständnisses mit betrachten, besitzt die 9 Dreitupel

$$(1,2,1),(1,2,1),(2,1,1),(2,1,1),(3,3,1),(4,4,1),(4,4,1),(5,5,1) \text{ und } (5,5,1).$$

Ordnen ergibt die Assignments

$$\phi = (1,2) \quad \text{und} \quad \psi = (1,3)(2,4),$$

die Kostenmatrizen sind

$$C^{\phi} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 3 & 4 & 5 \\ 3 & 3 & 3 & 4 & 5 \\ 4 & 4 & 4 & 4 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix} \end{array} \quad \text{und} \quad C'^{\phi'} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 4 & 4 & 5 & 5 \\ 1 & 1 & 2 & 2 & 3 & 4 & 4 & 5 & 5 \\ 2 & 2 & 2 & 2 & 3 & 4 & 4 & 5 & 5 \\ 2 & 2 & 2 & \mathbf{2} & \mathbf{3} & \mathbf{4} & 4 & 5 & 5 \\ 3 & 3 & 3 & \mathbf{3} & \mathbf{3} & \mathbf{4} & 4 & 5 & 5 \\ 4 & 4 & 4 & \mathbf{4} & \mathbf{4} & \mathbf{4} & 4 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 & 4 & \mathbf{4} & \mathbf{5} & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & \mathbf{5} & \mathbf{5} & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix} \end{array}.$$

Wir berechnen

$$S_1 = T_1 = 1, S_2 = T_2 = 3, S_3 = T_3 = 5, S_4 = T_4 = 6, S_5 = T_5 = 8, S_6 = T_6 = 10.$$

(Der Autor möchte bemerken, daß dies nicht etwa ein besonders einfacher, sondern ein besonders verzwickter Fall ist, da die Zirkulationslösung viele kleine Komponenten besitzt.) und

$$\tau := \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} & 2 & & & \\ 2 & & & & \\ & & 1 & & \\ & & & 2 & \\ & & & & 2 \end{pmatrix} \end{array}.$$

Diese Zirkulation besitzt vier Komponenten mit Knotenmengen

$$H_1 = \{1,2\}, H_2 = \{3\}, H_3 = \{4\} \quad \text{und} \quad H_4 = \{5\}.$$

G ergibt sich als

$$G = \Big(\{1,2,3,4,5\}, \big\{\{2,3\},\{3,4\},\{4,5\}\big\}\Big)$$

mit Kantengewichten

$$w_{\{2,3\}} = 3+3-2-3 = 1, w_{\{3,4\}} = 4+4-3-4 = 1 \quad \text{und} \quad w_{\{4,5\}} = 5+5-4-5 = 1.$$

Kontraktion der Mengen H_i liefert den Graphen

$$G \prod_{i=1}^{4} H_i = \Big(\{12,3,4,5\}, \big\{\{12,3\},\{3,4\},\{4,5\}\big\}\Big)$$

mit Kantengewichten von eins überall. Es bleibt uns nichts anderes übrig, als als minimalen Baum die Bilder der Kanten $\{2,3\}$, $\{3,4\}$ und $\{4,5\}$ von G unter der Kontraktion zu wählen, so daß wir G' als

$$G' = \Big(\{1,2,3,4,5\}, \{ \{2,3\}, \{3,4\}, \{4,5\} \} \Big)$$

erhalten. Die Menge $\{S_4, S_{4+1} - 1\} = \{6,7\}$ hat zwei Elemente (man betrachte die Verbindungstelle der beiden fettgedruchten Blöcke der Matrix $C'^{\phi'}$), also erhalten wir, obwohl G' nur zwei Komponeten besitzt, drei Mengen

$$V_1^1 = \{1\}, V_2^1 = \{2,3,4\} \quad \text{und} \quad V_2^2 = \{4,5\}.$$

Pyramidale und dichte Faktoren auf V_1^1 und V_2^2 sind trivialerweise (1) und $(4,5)$, für V_2^1 bestimmen wir bezüglich der Kostenmatrix (vergleiche die Matrix $C'^{\phi'}$)

$$\begin{pmatrix} 2 & 3 & 4 \\ 3 & 3 & 4 \\ 4 & 4 & 4 \end{pmatrix}$$

einen auf V_2^1 operierenden, pyramidalen, dichten und kostenminimalen Faktor durch Erweiterung von $(2,3)$ gemäß Satz 2.16..2.a oder b auf $(2,4,3)$ oder $(2,3,4)$ (beides ist hier möglich) als $(2,4,3)$. Die Permutation ψ' ist damit implizit als $(4,6,5)(7,8)$ bestimmt. Wir multiplizieren jetzt implizit ϕ' mit ψ' "auf der Matrix τ" und erhalten

$$\tau = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} & 2 & & & \\ 1 & & & 1 & \\ 1 & & 1 & & \\ & & 1 & & 1 \\ & & & 1 & 1 \end{pmatrix} \end{array}$$

als Nachfolgermatrix einer optimalen verallgemeinerten Permutation. □

2.3 Vier Bestückungsautomaten

Nach der erfolgreichen Behandlung des `TWO PROCESSOR ASSEMBLY LINE SCHEDULING` Problems drängt sich die Frage nach der Verallgemeinerbarkeit der dortigen Vorgehensweise auf den drei oder mehr hintereinander am Fließband aufgestellten Automaten entsprechenden Fall auf. Wir wollen den Fall mit 4 und mehr Automaten untersuchen.

2.23 Problem. `MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS BEI VIER AUTOMATEN`.
Instanz. Gegeben seien m Zahlenquintupel $(a_i, b_i, c_i, d_i, s_i)$ aus $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}$, wobei $d_1 \leq \ldots \leq d_m$ zutreffe.
Frage. Welche verallgemeinerte Permutation σ auf $\{1, \ldots, m\}$ mit Vielfachheiten s_i führt zu einer minimalen Gesamtbearbeitungszeit

$$\begin{aligned} & a_{\sigma(1)} + \max\{a_{\sigma(2)}, b_{\sigma(1)}\} + \max\{a_{\sigma(3)}, b_{\sigma(2)}, c_{\sigma(1)}\} \\ & \quad + \sum_{i=1}^{S-3} \max\{a_{\sigma(i+3)}, b_{\sigma(i+2)}, c_{\sigma(i+1)}, d_{\sigma(i)}\} \\ & \quad + \max\{b_{\sigma(S)}, c_{\sigma(S-1)}, d_{\sigma(S-2)}\} + \max\{c_{\sigma(S)}, d_{\sigma(S-1)}\} + d_{\sigma(S)}? \end{aligned}$$

□

Analog zum Vorgehen im Zwei-Automaten-Fall ist es günstig, die lineare Bearbeitungsreihenfolge zyklisch zu verketten.

2.24 Problem. FOUR PROCESSOR ASSEMBLY LINE SCHEDULING.
Instanz. Gegeben seien m Zahlenquintupel $(a_i, b_i, c_i, d_i, s_i)$ aus $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}$, wobei $d_1 \leq \ldots \leq d_m$ gelte.
Frage. Welche verallgemeinerte Permutation σ auf $\{1, \ldots, m\}$ mit Vielfachheiten s_i führt zu einer minimalen Gesamtbearbeitungszeit

$$\sum_{i=1}^{S} \max\left\{ a_{\sigma\left((i+2) \bmod S+1\right)}, b_{\sigma\left((i+1) \bmod S+1\right)}, c_{\sigma(i \bmod S+1)}, d_{\sigma(i)} \right\}?$$

□

2.25 Beobachtung. Seien Instanzen von Problem 2.23 und Problem 2.24 durch m beziehungsweise $m+4$ Zahlenquintupel

$$\big((a_i, b_i, c_i, d_i, s_i)_{i=1,\ldots,m}\big)$$

und

$$\Big(\big((a_i, b_i, c_i, d_i, s_i)\big)_{i=1,\ldots,m}, (0,0,0,T,1), (0,0,T,0,1), (0,T,0,0,1), (T,0,0,0,1)\Big)$$

gegeben, wobei T als $\sum_{i=1}^{m}(a_i + b_i + c_i + d_i)s_i + 1$ vereinbart sei. Dann gelten die folgenden Aussagen.

1. Ist σ eine verallgemeinerte Permutation von $\{1, \ldots, m+4\}$ zu Vielfachheiten $(s, 1, 1, 1, 1)$, so gilt

$$\begin{aligned}
&\sum_{i=1}^{S+4} \max\left\{ a_{\sigma\left((i+2) \bmod (S+4)+1\right)}, b_{\sigma\left((i+1) \bmod (S+4)+1\right)}, c_{\sigma\left(i \bmod (S+4)+1\right)}, d_{\sigma(i)} \right\} < 2T \\
\Longrightarrow\ &\exists k \in \{1, \ldots, S+4\}: \\
&\quad \big(\sigma(k), \sigma\big(k \bmod (S+4)+1\big), \sigma\big((k+1) \bmod (S+4)+1\big), \sigma\big((k+2) \bmod (S+4)+1\big)\big) \\
&\quad\quad = (m+1, m+2, m+3, m+4).
\end{aligned}$$

   Nehmen wir ohne Beschränkung der Allgemeinheit $k = S$ an (sonst "shifte" man σ analog zu Folgerung 1.11), so ist $\big(\sigma(1), \ldots, \sigma(S)\big)$ eine verallgemeinerte Permutation von $\{1, \ldots, m\}$ zu Vielfachheiten $(s_i)_{i=1,\ldots,m}$ und es gilt

$$\begin{aligned}
&a_{\sigma(1)} + \max\{ a_{\sigma(2)}, b_{\sigma(1)} \} + \max\{ a_{\sigma(3)}, b_{\sigma(2)}, c_{\sigma(1)} \} + \sum_{i=1}^{S-3} \max\{ a_{\sigma(i+3)}, b_{\sigma(i+2)}, c_{\sigma(i+1)}, d_{\sigma(i)} \} \\
&\quad + \max\{ b_{\sigma(S)}, c_{\sigma(S-1)}, d_{\sigma(S-2)} \} + \max\{ c_{\sigma(S)}, d_{\sigma(S-1)} \} + d_{\sigma(S)} \\
&= \sum_{i=1}^{S+4} \max\left\{ a_{\sigma\left((i+2) \bmod (S+4)+1\right)}, b_{\sigma\left((i+1) \bmod (S+4)+1\right)}, c_{\sigma\left(i \bmod (S+4)+1\right)}, d_{\sigma(i)} \right\} - T.
\end{aligned}$$

2. Ist σ eine verallgemeinerte Permutation von $\{1, \ldots, m\}$ zu einem Vektor von Vielfachheiten $(s_i)_{i=1,\ldots,m}$, so ist $(\sigma, m+1, m+2, m+3, m+4)$ eine verallgemeinerte Permutation von $\{1, \ldots, m+4\}$ zu Vielfachheiten $(s, 1, 1, 1, 1)$ und es gilt

$$\begin{aligned}
&a_{\sigma(1)} + \max\{ a_{\sigma(2)}, b_{\sigma(1)} \} + \max\{ a_{\sigma(3)}, b_{\sigma(2)}, c_{\sigma(1)} \} \\
&\quad + \sum_{i=1}^{S-3} \max\{ a_{\sigma(i+3)}, b_{\sigma(i+2)}, c_{\sigma(i+1)}, d_{\sigma(i)} \} + \max\{ b_{\sigma(S)}, c_{\sigma(S-1)}, d_{\sigma(S-2)} \} \\
&\quad + \max\{ c_{\sigma(S)}, d_{\sigma(S-1)} \} + d_{\sigma(S)} + T \\
&= \sum_{i=1}^{S+4} \max\left\{ a_{\sigma\left((i+2) \bmod (S+4)+1\right)}, b_{\sigma\left((i+1) \bmod (S+4)+1\right)}, c_{\sigma\left(i \bmod (S+4)+1\right)}, d_{\sigma(i)} \right\} \\
&< 2T.
\end{aligned}$$

Problem 2.23 und Problem 2.24 sind damit polynomial äquivalent. □

Für die Untersuchung des Reihenfolgeoptimierungsproblems im Falle von vier hintereinander am Fließband aufgestellten Automaten ist es also ausreichend, das FOUR PROCESSOR ASSEMBLY LINE SCHEDULING Problem zu betrachten.

2.26 Satz. Das zu FOUR PROCESSOR ASSEMBLY LINE SCHEDULING gehörige Entscheidungsproblem ist $\mathcal{NP}$-vollständig, auch wenn nur einfache Vielfachheiten zugelassen werden.

Beweis. Daß das zu FOUR PROCESSOR ASSEMBLY LINE SCHEDULING gehörige Entscheidungsproblem mit einfachen Vielfachheiten zu $\mathcal{NP}$ gehört, ist klar, denn eine NDTM braucht lediglich eine (verallgemeinerte) Permutation (zu den Vielfachheiten 1) zu erraten und dann in polynomialer Zeit zu überprüfen, daß die resultierende Gesamtbearbeitungszeit kleiner als der angestrebte Wert ist.

Um die $\mathcal{NP}$-Vollständigkeit zu zeigen, führen wir THREE DIMENSIONAL MATCHING auf FOUR PROCESSOR ASSEMBLY LINE SCHEDULING zurück. Sei also eine Instanz von THREE DIMENSIONAL MATCHING mit Mengen X, Y und Z gleicher Mächtigkeit und einer Menge $W \subseteq X \times Y \times Z$ gegeben. Die entsprechende Instanz des zu FOUR PROCESSOR ASSEMBLY LINE SCHEDULING gehörenden Entscheidungsproblems besteht dann aus $5|W|$ Quintupeln, von denen die letzte Komponente (also die Vielfachheit) stets den Wert eins besitzt. Wir denken uns die Quintupel als Knoten eines geschichteten Netzwerkes mit 5 gleichmächtigen Schichten S_W, S_X, S_Y, S_Z und S_S angeordnet. Die Knoten aus S_W assoziieren wir in eineindeutiger Weise mit den Elementen von W, die Knoten aus S_X, S_Y und S_Z jeweils mit den x-, y- und z-Koordinaten dieser Elemente, und zwar in der Weise, daß zu jedem Element aus X, Y und Z jeweils so viele diesem entsprechende Knoten in den Mengen S_X, S_Y und S_Z enthalten sind, wie dieses Element in Dreitupeln aus W vertreten ist. In jeder der drei Mengen S_X, S_Y und S_Z zeichnen wir nun für jedes Element von X, Y oder Z einen diesem Element entsprechenden Knoten aus und bezeichnen die Mengen dieser Knoten mit S'_X, S'_Y und S'_Z, die verbleibenden Knoten jeweils mit S''_X, S''_Y und S''_Z. Die einfach gestrichenen Mengen enthalten dann für jedes Element aus den Mengen X, Y und Z genau einen Repräsentanten, der dem Auftreten dieses Elementes in einem Dreitupel eines Dreimatchings entsprechen soll. Ebenso partitionieren wir die Menge S_S in zwei Hälften S'_S und S''_S, wobei S'_S gerade $|X| = |Y| = |Z|$ Elemente enthalte, also genau so viele, wie in einem Dreimatching Dreitupel enthalten sind, so daß für S''_S gerade $|W| - |X|$ Knoten übrig bleiben.

Identifizieren wir Quintupel einer Instanz von FOUR PROCESSOR ASSEMBLY LINE SCHEDULING mit einfachen Vielfachheiten und Knoten des vollständigen Graphen mit m Knoten, so beobachten wir, daß FOUR PROCESSOR ASSEMBLY LINE SCHEDULING mit einfachen Vielfachheiten auch als das Problem betrachtet werden kann, in diesem Graphen eine kürzeste Tour bezüglich einer Zielfunktion zu finden, die bei Erreichen eines Knotens i Kosten $\max\{ a_i, b_j, c_k, d_l \}$ verursacht, die nicht nur von diesem Knoten i abhängen, sondern auch von seinem Vorgänger j, seinem Vorvorgänger k und auch noch von dessen Vorgänger l. Wir wollen die vier ersten noch unbestimmten Komponenten der Quintupel unserer Instanz von FOUR PROCESSOR ASSEMBLY LINE SCHEDULING mit einfachen Vielfachheiten in einer solchen Weise bestimmen, daß eine kürzeste Tour in obigem Sinne diese nur in ganz bestimmter Reihenfolge besuchen kann.

Wie sind diese ersten vier Komponenten der Quintupel nun jeweils zu wählen? Wir wollen folgende Idee benutzen. Angenommen, es sind bereits alle Komponenten bis auf a_i und b_j für zwei verschiedene Knoten i und j bestimmt. Setzen wir diese nun auf einen im Verhältnis zu den anderen Kosten sehr großen Wert M fest, so wird eine kurze Tour versuchen (müssen), als Vorgänger von i den Knoten j zu verwenden. Auf analoge Weise können auch noch ein Vorgänger des Vorgängers und dessen Vorgänger, also bis zu vier Generationen betrachtet werden. Was ist eine gute Wahl für den Parameter M? Wählt man für M die Summe aller übrigen bereits bestimmten Gewichte plus eins, so kann man eine Tour mit einer Länge von weniger als $2M + 2$ nur erhalten, wenn die Tour die vorgesehenen Reihenfolgebedingungen einhält. Die genannte Konstruktion läßt sich auch iterieren, indem man verschiedene Reihenfolgen spezifiziert und dann eine nach der anderen mit immer größeren Gewichten realisiert, so daß sich die Erfülltheit derselben vom größten zum kleinsten Gewicht nach unten fortpflanzt. Diese Konstruktion wollen wir nun in die Tat umsetzen.

Wir definieren zuerst die Zahlen a_i und b_j für alle Knoten i aus S_W und j aus S_S als die Summe aller noch im weiteren festzusetzenden ersten vier Komponenten der $5|W|$ Quintupel unserer Instanz plus 1; diese Zahl sei M_1. Eine Tour hat genau dann eine Länge von weniger als

$$|W|M_1 + M_1,$$

wenn jeder Knoten von S_W einen beliebigen Vorgänger aus S_S besitzt.

In einem zweiten Schritt legen wir fest, daß eine kurze Tour die Knoten in den Mengen S''_X, S''_Y und S''_Z stets in der Weise besucht, daß die Nachfolger der Knoten in S''_X genau die Knoten in S''_Y sind, die wiederum von Knoten in S''_Z abgelöst werden, auf die stets Knoten in S''_S folgen. Hierzu definieren wir, wenn $M_2 < M_1$ nun die Summe aller noch im weiteren zu definierenden ersten vier Komponeten der Quintupel bezeichnet

$$a_i = b_j = c_k = d_l = M_2 + 1$$

für alle i aus S''_S, j aus S''_Z, k aus S''_Y und l aus S''_X; eine Tour besitzt jetzt dann und nur dann eine Länge von weniger als

$$|W|M_1 + (|W| - |X|)M_2 + M_2,$$

wenn erstens die Vorgänger der Knoten aus S_W genau die Knoten aus S_S sind und zweitens die Nachfolger der Knoten aus S''_X genau die Knoten aus S''_Y sind, die wiederum gerade von den Knoten aus S''_Z abgelöst werden, die zuletzt Nachfolger exakt in S''_S haben. Die derart entstehenden Wegstücke durch die zweifach gestrichenen Mengen sollen die nicht für ein Dreimatching ausgewählten Koordinaten in den nicht für ein Matching gewählten Dreitupeln repräsentieren.
Auf analoge Weise bestimmen wir vermöge der Festsetzungen

$$a_i = b_j = c_k = d_l = M_3$$

für alle i aus S'_S, j aus S'_Z, k aus S'_Y und l aus S'_X, wobei M_3 wie immer die Summe der im weiteren noch anzugebenden ersten vier Komponeten von Quintupeln plus eins bezeichne, daß eine Tour, die kürzer als

$$|W|M_1 + (|W| - |X|)M_2 + |X|M_3 + M_3$$

ist, neben den bereits oben genannten Reihenfolgebedingungen auch noch von Knoten aus S'_X gerade zu den Knoten aus S'_Y, von dort zu den Mitgliedern von S'_Z und zuletzt zu den Elementen von S'_S wandern muß; eine unmittelbare Konsequenz hiervon ist aber jetzt, daß die Tour auch von den Knoten in S_W immer direkt zu den Knoten in S_X forschreitet.
Zusammenfassend läßt sich damit folgendes feststellen. Touren mit einer Länge von weniger als

$$|W|M_1 + (|W| - |X|)M_2 + |X|M_3 + M_3$$

genügen der folgenden Bedingung. Der Nachfolger eines Knotens aus S_S ist ein Knoten aus S_W, dessen Nachfolger ein Knoten v aus S_X ist; gehört v zu S'_X, so ist sein Nachfolger aus S'_Y und dessen Nachfolger aus S'_Z, von wo aus wieder nach S'_S gegangen wird; gehörte v dagegen zu S''_X, so ist sein Nachfolger in S''_Y, dessen Nachfolger in S''_Z und dessen Nachfolger wiederum in S''_S zu finden.
Beim Übergang von S_W zu S_X findet somit eine Entscheidung statt, den Weg durch die Schichten S_X, S_Y und S_Z entweder im einfach oder im zweifach gestrichenen Teil dieser Mengen zurückzulegen; der einfachgestrichene soll, wie der Leser bereits vermuten wird, das gesuchte Dreimatching repräsentieren.
In einem letzten Schritt zählen wir die Mengen X, Y und Z ab; für den einem Dreitupel (x_i, y_j, z_k) entsprechenden Knoten l aus S_W setzen wir dann

$$b_l = i, \quad c_l = j, \quad d_l = k$$

und ergänzen diese Definition dadurch, daß wir allen Repräsentanten j, k oder l eines beliebigen Knotens x_a, y_b oder z_c aus X, Y oder Z in S_X, S_Y oder S_Z die Komponente

$$a_j = a, \quad a_k = b, \quad a_l = c$$

zuweisen; alle bisher noch nicht berücksichtigten anderen Komponenten aller Quintupel erklären wir zu Null.
Rückwärtsrechnug liefert für die Konstanten M_1, M_2 und M_3 die Schranken

$$\begin{aligned} M_3 &\leq 4|W||X| + 1 \\ M_2 &\leq |X|M_3 + M_3 \\ M_1 &\leq (|W| - |X|)M_2 + M_2, \end{aligned}$$

so daß die durchgeführte Transformation polynomial ist.
Wir behaupten nun, daß es eine Tour der Länge

$$|W|M_1 + (|W| - |X|)M_2 + |X|M_3 + \sum_{(x_i,y_j,z_k)\in W} (i+j+k)$$

oder weniger genau dann gibt, wenn W ein Dreimatching enthält.
Enthalte W ein Dreimatching. Eine entsprechende Tour in der Instanz von `FOUR PROCESSOR ASSEMBLY LINE SCHEDULING` kann dann bei einem beliebigen Knoten aus S_W starten. Entspricht der Startknoten einem Dreitupel (x_i, y_j, z_k) des Matchings, so besucht man als nächstes in Reihenfolge die den Koordinaten x, y und z des Matchings entsprechenden Knoten in S'_X, S'_Y und S'_Z, wobei Kosten von $i + j + k$ entstehen; Anschließend wandert man zu einem beliebigen Knoten aus S'_S, was die Länge der Tour um M_3 anhebt. War der Startknoten dagegen nicht einem Dreitupel aus W zugeordnet, so wandert man auf analoge Weise unter Erzeugung von Kosten von $i + j + k + M_2$ durch die doppelt gestrichenen Mengen bis nach S''_S. Auf diese Weise enstehen gerade $|W|$ knotendisjunkte Wege, die alle Knoten in S_W mit allen Knoten in S_S verbinden, wobei zusätzlich noch alle Knoten aus den Mengen S_X, S_Y und S_Z genau einmal besucht werden. Zuletzt verbindet man noch die Knoten in S_S so mit den Knoten in S_W, daß eine Tour entsteht. Summation über alle von den Teilstücken verursachten Kosten liefert die Behauptung.
Sei umgekehrt eine Tour mit den genannten Kosten gegeben; diese schreitet dann, wie wir bereits bei der Konstruktion festgestellt haben, in der Weise durch den Graphen, daß von Knoten in S_W entweder durch die Menge S'_X S'_Y und S'_Z und anschließend S'_S angesteuert wird oder die analoge Sequenz in den zweigestrichenen Mengen durchlaufen wird. Von S_S aus wird dann stets wieder S_W aufgesucht. Durch diese Vorgehensweise entstehen an den Knoten aus S_S und S_W Kosten von insgesamt

$$|W|M_1 + (|W| - |X|)M_2 + |X|M_3,$$

an den Knoten aus den Mengen S_X, S_Y und S_Z aber ebenfalls Kosten von mindestens $\sum_{(x_i,y_j,z_k)\in W}(i+j+k)$, da dies die Summe der ersten Komponenten der den Knoten aus den drei Mengen X, Y und Z entsprechenden Quintupel ist. Will unsere Tour daher diese Kosten nicht überbieten, so muß sie von jedem Knoten $v = (x_i, y_i, z_i)$ aus S_W mit $(a_v, b_v, c_v, d_v) = (M_1, i, j, k)$ einen Knoten u aus S_X mit einer ersten Komponente seines Quintupels von nicht weniger als i ansteuern; die analoge Aussage gilt für die angesteuerten Knoten in den Mengen S_Y und S_Z. Da aber für jede Koordinate in jedem Dreitupel aus W nur ein Knoten in einer dieser drei Mengen enthalten ist, muß von v aus ein Knoten angesteuert werden, der genau der x-Koordinate des Dreitupels entspricht. In den einfach gestrichenen Mengen ist hiervon aber jeweils nur einer enthalten, so daß diejenigen Knoten aus S_W, die Knoten aus S'_X ansteuern, genau einem Dreimatching aus W entsprechen. □

Satz 2.26 impliziert, daß auch die entsprechenden Probleme mit noch mehr hintereinander aufgestellten Maschinen $\mathcal{NP}$-vollständig beziehungsweise -schwer sind; man setze einfach in den zusätzlichen Koordinaten die Kosten auf Null. Schmerzhaft ist allerdings die Ungewißheit über den Fall mit drei Maschinen, den der Autor (noch) nicht lösen konnte; er tröstet sich damit, daß für viele ähnliche Scheduling Probleme ebenfalls genau der Fall mit drei Prozessoren offen ist ([Garey & Johnson, 1979]).
Der Autor möchte noch bemerken, daß Satz 2.26 in jedem Fall zeigt, daß eine praktische Anwendung der in diesem Kapitel gewonnenen Erkenntnisse, wie in der Einleitung bereits angedeutet, stark in Zweifel gezogen wird, da dort wohl in der Regel mit mehreren Maschinen gearbeitet wird.

3 Mengenpartitionen

3.1 Partitionsprobleme

In Problem 1.15 wird über zwei verschiedene Objekte, nämlich Partitionen und verallgemeinerte Permutationen, optimiert. Dies soll für uns der Anlaß sein, uns allgemein mit bewerteten oder "gewichteten" Partitionen zu beschäftigen. Wie soll man eine Partition bewerten? Zwei alternative Möglichkeiten drängen sich auf.

3.1 Problem. `WEIGHTED SET SPLITTING`.
Instanz. Gegeben sei eine natürliche Zahl k, ein Hypergraph $H = \big(V, \{\, e_1, e_2 \,\}\big)$ mit n Knoten und m Kanten $e_1, \ldots, e_m$ sowie eine nichtnegative ganzzahlige Gewichtung c_{ij} für alle j aus V und i aus $\{\, 1, \ldots, m \,\}$ der Knoten von H in Abhängigkeit von der Zugehörigkeit zu den Kanten von H, so daß c_{ij} Null ist, wenn der Knoten j nicht in e_i enthalten ist.
Frage. Gibt es eine Partition $P = \big(V, \{\, f_1, f_2 \,\}\big)$, so daß die Beziehung

$$\min\big\{\, c_{i\cdot}\chi\,(f_1), c_{i\cdot}\chi\,(f_2) \,\big\} \geq k$$

für alle i aus $\{\, 1, \ldots, m \,\}$ gilt?
Der Bequemlichkeit halber soll auch das zugehörige Optimierungsproblem

$$\max_{\text{Partition } P} \min_{i=1}^{m} \min\big\{\, c_{i\cdot}\chi\,(f_1), c_{i\cdot}\chi\,(f_2) \,\big\}$$

als `WEIGHTED SET SPLITTING` bezeichnet werden; es geht jeweils aus dem Zusammenhang hervor, welches Problem gemeint ist. □

Problem 3.1 gestattet folgende Interpretation. Eine Partition von V induziert Partitionen der Kanten von H. Jede einzelne solche Partition wird gemäß der Gewichtung $c_{i\cdot}$ für die Kante e_i unabhängig von den restlichen Kanten mit dem Minimum der Summe der Gewichte der Elemente ihrer beiden Hälften bewertet. Die gesamte Partition wird nun in worst-case Manier nach der Bewertung der schlechtesten derartigen Partition beurteilt. Das gesamte Mengensystem H soll demnach möglichst "gleichmäßig" in zwei bezüglich der Gewichtung C gleich große Hälften aufgeteilt werden.
Es ist auch möglich, für jede Kante die erreichte "Gewichtsüberdeckung" zwischen ihren beiden Hälften ins Verhältnis zur überhaupt möglichen Überdeckung zu setzen und so eine prozentuale Maximierung zu erreichen. Hierzu betrachtet man für eine Instanz von `WEIGHTED SET SPLITTING` mit Kostenmatrix C die Instanz mit der Kostenmatrix

$$C' := \left(c_{ij} \prod_{k \neq i} c_{k\cdot} \mathbb{1} \right),$$

die sich in polynomialer Zeit aus C konstruieren läßt. Es gilt dann für alle Zeilen i von C' beziehungsweise Kanten von H $c'_{i\cdot}\mathbb{1} = \sum_j c_{j\cdot}\mathbb{1}$.

3.2 Definition. `HYPERGRAPH MAX CUT`.
Instanz. Wie bei `WEIGHTED SET SPLITTING`.
Frage. Gibt es eine Partition $P = \big(V, \{\, f_1, f_2 \,\}\big)$ von V mit der Eigenschaft

$$\sum_{i=1}^{m} \min\big\{\, c_{i\cdot}\chi\,(f_1), c_{i\cdot}\chi\,(f_2) \,\big\} \geq k?$$

Analog zu Problem 3.1 soll das zugehörige Optimierungsproblem

$$\max_{\text{Partition } P} \sum_{i=1}^{m} \min\big\{\, c_{i\cdot}\chi\,(f_1), c_{i\cdot}\chi\,(f_2) \,\big\}$$

ebenfalls mit `HYPERGRAPH MAX CUT` bezeichnet werden. □

Wie bei WEIGHTED SET SPLITTING werden die auf den Kanten induzierten Partitionen unabhängig voneinander mit der kleineren der Gewichtssummen der beiden Hälften bewertet. Im Unterschied zu WEIGHTED SET SPLITTING wird nun aber eine —wenn man so will— "globale" Betrachtung durchgeführt, indem alle Einzelbewertungen aufaddiert werden. Diese Summe kann trotz oder gerade wegen einzelner schlechter Komponenten durchaus "groß" sein.
Für beide gerade eingeführten Maximierungsprobleme gibt es je zwei äquivalente Minimierungsversionen. Für jede Komponente i der beiden Zielfunktionen gilt nämlich für jede Partition $P = \big(V, \{ f_1, f_2 \}\big)$ die (triviale) Beziehung

$$\min\{ c_i.\chi(f_1), c_i.\chi(f_2) \} + \max\{ c_i.\chi(f_1), c_i.\chi(f_2) \} = c_i.\mathbb{1},$$

woraus sich sofort

$$\begin{aligned} \max_P \min\{ c_i.\chi(f_1), c_i.\chi(f_2) \} &\iff \min_P \max\{ c_i.\chi(f_1), c_i.\chi(f_2) \} \\ &\iff \min_P \Big\{ \max\{ c_i.\chi(f_1), c_i.\chi(f_2) \} - \min\{ c_i.\chi(f_1), c_i.\chi(f_2) \} \Big\} \end{aligned}$$

ergibt. Eine obere beziehungsweise untere Schranke für die beiden erstgenannten Optimierungsprobleme ist jeweils $c_i.\frac{\mathbb{1}}{2}$ und die Zielfunktionswerte der beiden Probleme liegen jeweils symmetrisch zu dieser, während eine untere Schranke für das dritte Problem jeweils Null ist. Da wir später approximative Algorithmen für die eingeführten Probleme untersuchen wollen, erscheint die dritte Version als "faul", so daß der Autor darauf verzichtet, diesen Ansatz weiter zu verfolgen. Man beachte jedoch, daß diese Einschätzung kritisch davon abhängt, daß in der Kostenmatrix C keine negativen Zahlen auftreten; ist dies nämlich der Fall, so mag auch bei den beiden ersten Varianten ein Optimalwert von Null gerade in den interessanten Fällen durchaus auftreten. Ob man dagegen die erste oder die zweite Formulierung vorzieht, ist weitgehend eine Geschmacksfrage. Der Leser überzeugt sich sofort davon, daß die gemachten Aussagen ihre Richtigkeit nicht einbüßen, wenn die betrachteten Zielfunktionen der Teilprobleme vermöge Minimumbildung und Summierung zur Zielfunktion von WEIGHTED SET SPLITTING oder HYPERGRAPH MAX CUT zusammengesetzt werden.

3.3 Beobachtung. Die Codierungslänge einer Instanz von WEIGHTED SET SPLITTING und HYPERGRAPH MAX CUT ist

$$O(\langle C \rangle + \langle k \rangle).$$

Beweis. Eine Codierung einer Instanz beider Probleme kann als durch eine Codierung der Matrix C und der Zahl k gegeben angesehen werden, da sich der Hypergraph H aus der Matrix C ableiten läßt. □

3.4 Beispiel. Gegeben sei der Hypergraph

$$H = \Big(\{ 1, \ldots, 6 \}, \{ e_1 = \{ 1,2,3,4 \}, e_2 = \{ 2,4,5,6 \}, e_3 = \{ 1,2,3 \}, e_4 = \{ 4,5 \}, e_5 = \{ 1,3,4,5,6 \} \} \Big)$$

mit Gewichtung

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 4 & 2 & 9 & 0 & 0 \\ 0 & 8 & 0 & 1 & 7 & 7 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 5 & 0 \\ 1 & 0 & 2 & 9 & 3 & 9 \end{pmatrix} \end{array}$$

und eine Partition $P = \Big(\{ 1, \ldots, 6 \}, \{ \ f_1 = \{ 1,3,4,6 \}, f_2 = \{ 2,5 \} \} \Big)$. Die Kante e_1 wird dann sowohl in der den obigen Daten entsprechenden Instanz von WEIGHTED SET SPLITTING alsauch von HYPERGRAPH MAX CUT für jeweils beliebiges k mit $\min\{ 1+2+9, 4 \} = 4$ bewertet. Analog erhält e_2 die Bewertung 8, e_3 1, e_4 4 und e_5 3. Für WEIGHTED SET SPLITTING brechnet man als Minimum dieser 5 Zahlen 1, für HYPERGRAPH MAX CUT ergibt ihre Summe 20. Wählt man in beiden Entscheidungsproblemen etwa $k = 10$, so kann für WEIGHTED SET SPLITTING aufgrund der betrachteten Partition noch keine Entscheidung getroffen werden. Für HYPERGRAPH MAX CUT wäre die gefundene Lösung ein Beweis für die Antwort "Ja". □

Was haben nun die Probleme WEIGHTED SET SPLITTING und HYPERGRAPH MAX CUT mit dem Problem MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS zu tun? Die nächste Aussage ist die Antwort des Autors auf diese Frage.

3.5 Beobachtung. Sei eine Instanz von MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS in der Version von Problem 1.15 durch eine Kostenmatrix C und Zahlen s_i für i aus $\{1,\dots,m\}$ gegeben.

1. Wir betrachten die Instanz des Optimierungsproblems WEIGHTED SET SPLITTING mit gleicher Kostenmatrix und zugehörigem Hypergraphen $H = \big(\{1,\dots,n\}, E\big)$, definiert durch $j \in e_i \iff c_{ij} \neq 0$ für alle i und j. Ist $P = \big(\{1,\dots,n\}, \{f_1, f_2\}\big)$ eine Partition von $\{1,\dots,n\}$ mit Zielfunktionswert bezüglich WEIGHTED SET SPLITTING von $c(P)$, so gilt für jede verallgemeinerte Permutation σ von $\{1,\dots,m\}$ mit beliebigen Vielfachheiten s_i die Garantie

$$\pi(C,P)\cdot\tau(\sigma) \leq \sum_{i=1}^{m} s_i c_{i\cdot}\mathbb{1} - S\cdot c(P) = \sum_{i=1}^{m} s_i c_{i\cdot}\mathbb{1} - \sum_{i=1}^{m} s_i \min_{i=1}^{m} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\}.$$

2. Wir betrachten die Instanz des Optimierungsproblems HYPERGRAPH MAX CUT mit Kostenmatrix $C' := (s_i c_{ij})_{\substack{i=1,\dots,m\\ j=1,\dots,m}}$; sei $P = \big(\{1,\dots,n\}, \{f_1, f_2\}\big)$ eine Partition von $\{1,\dots,n\}$. Ist dann ϕ eine optimale Lösung für die Instanz von TWO PROCESSOR ASSEMBLY LINE SCHEDULING, gegeben durch die m Zahlentripel $\big(c_{i\cdot}\chi(f_2), c_{i\cdot}\chi(f_1), 1\big)$ für $i = 1,\dots,m$ mit Zielfunktionswert $c(P)$, so ist ϕ insbesondere eine Permutation von $\{1,\dots,m\}$, also eine Tour. Von allen verallgemeinerten Permutationen σ von $\{1,\dots,m\}$ zu Vielfachheiten s_i für alle i mit der Eigenschaft

$$\forall i \in \{1,\dots,m\} : \exists k \in \{1,\dots,S\} : \big(\sigma(k),\dots,\sigma(k+s_i-1)\big) = (i,\dots,i) \tag{1}$$

ist dann die verallgemeinerte Permutation σ^*, definiert durch

$$\Big(\big(\phi(1),\dots,\phi(1)\big),\dots,\big(\phi(m),\dots,\phi(m)\big)\Big),$$

diejenige mit den geringsten Kosten $\tau(\sigma)\pi(C,P)$. Ferner gilt

$$\tau(\sigma^*)\pi(C,P) - \min_{\substack{\text{Partition } P'\\ \sigma:(1)}} \tau(\sigma)\pi(C,P') \leq \sum_{i=1}^{m} \max\{c_{i\cdot}\chi(f_2), c_{\phi(i)\cdot}\chi(f_1)\} - \sum_{i=1}^{m} c_{i\cdot}\frac{\mathbb{1}}{2}$$

und

$$\tau(\sigma^*)\pi(C,P) = \sum_{i=1}^{m} s_i c_{i\cdot}\mathbb{1} - c(P) - \sum_{i=1}^{m} \min\{c_{i\cdot}\chi(f_2), c_{i\cdot}\chi(f_1)\} + \sum_{i=1}^{m} \min\{c_{i\cdot}\chi(f_2), c_{\phi(i)\cdot}\chi(f_1)\}.$$

□

Beobachtung 3.5 läßt sich wie folgt interpretieren.
Beobachtung 3.5.1 zeigt, wie es mit Hilfe einer Lösung des Problems WEIGHTED SET SPLITTING möglich ist, eine Aufteilung der Bauelemente auf die zwei Automaten zu finden, so daß sich bei beliebiger Reihenfolge des Aufbringens der Platinen auf das Fließband immer noch eine Mindestgüte der Lösung ergibt, die offensichtlich nur unter bestimmten Voraussetzungen gut ist. Eine Anwendung einer solchen Vorgehensweise ist etwa dann gegeben, wenn zwar bekannt ist, welche Anzahlen von welchen Typen von Leiterplatten im Laufe des Tages zu bestücken sein werden, nicht aber, wann etwa einzelne Platinen zur Bestückung vorliegen werden. Kurz gesagt, dem Autor schwebt eine Bedieneinrichtung vor, die zufällig eintreffende Bedienwünsche in FIFO-Manier bearbeitet und sich dabei wie unser Produktionsmodell verhält. Er will aber gern zugeben, daß eine solche Situation bei der Bestückung von Leiterplatten wohl kaum auftreten wird.
Beobachtung 3.5.2 zeigt dagegen, wie man bei losorientierter Fertigung mit Hilfe der Ergebnisse früherer Abschnitte eine Optimallösung auf einfache Weise errechnen kann, wobei für die Güte der Lösung eine Differenzgarantie gilt (in der keine Zahlen s_i auftreten). Losorientierung bedeute hierbei, daß man stets alle

Leiterplatten eines Typs unmittelbar nacheinander abarbeitet. Diese Taktik ist etwa dann angebracht, wenn eine intelligente Einrichtung zum Aufbringen von Platinen verschiedener Typen auf das Fließband nicht vorhanden ist oder die Bestückungsautomaten durch Menschen ersetzt werden, deren begrenzte Konzentrationsfähigkeit einen dauernden Wechsel der Platinentypen als nicht angezeigt erscheinen läßt. Unter realistischen Voraussetzungen wird man wohl davon ausgehen können, daß für eine Partition $P = \big(V, \{ f_1, f_2 \}\big)$ der Quotient

$$\frac{\sum_{i=1}^{m} \max\{ c_{i\cdot}\chi(f_2), c_{\phi(i)\cdot}\chi(f_1) \} - \sum_{ij} c_{ij}\frac{1}{2}}{\tau(\sigma^*)\pi(C,P)}$$

"klein" ist, da die s_i "groß" sind, so daß man in der Regel eine gute relative Gütegarantie

$$\tau(\sigma^*)\pi(C,P) \leq \min_{\substack{\text{Partition } P \\ \sigma:(1)}} \tau(\sigma)\pi(C,P)\left(1 + \frac{\sum_{i=1}^{m} \max\{ c_{i\cdot}\chi(f_2), c_{\phi(i)\cdot}\chi(f_1) \} - \sum_{ij} c_{ij}\frac{1}{2}}{\min_{\substack{\text{Partition } P \\ \sigma:(1)}} \tau(\sigma)\pi(C,P)}\right)$$

beweisen wird können. $\min_{\substack{\text{Partition } P \\ \sigma:(1)}} \tau(\sigma)\pi(C,P)$ kann man für "gute" Partitionen P durch

$$\sum_{i=1}^{m} s_i c_{i\cdot}\mathbb{1} - c(P) - \sum_{i=1}^{m} \min\{ c_{i\cdot}\chi(f_2), c_{i\cdot}\chi(f_1) \} + \sum_{i=1}^{m} \min\{ c_{i\cdot}\chi(f_2), c_{\phi(i)\cdot}\chi(f_1) \}$$

schätzen.
Die gemachten Aussagen verallgemeinern direkt zu einer Version, in der die Lose nicht jeweils alle Platinen eines bestimmten Typs umfassen, sondern jeweils nur kleinere Anzahlen, indem in der Matrix C Zeilen entsprechend der Anzahl der einem Platinentyp entsprechenden Lose vervielfacht werden und Vielfachheiten entsprechend aufgeteilt werden. Obwohl diese Vorgehensweise jedenfalls keine polynomiale Transformation ist, so bleiben doch die obigen Aussagen über die Güte der Lösungen bestehen.

3.2 Komplexität

Als wie schwierig sind nun die beiden im vorigen Abschnitt eingeführten Probleme im Sinne der Komplexitätstheorie anzusehen? Diese Frage wollen wir nun untersuchen. Wir prüfen die $\mathcal{NP}$-Vollständigkeit, Einschränkungen auf Spezialfälle mit Zusatzbedingungen an die Kostenmatrizen, stellen eine scharfe Grenze zwischen polynomial lösbaren und $\mathcal{NP}$-vollständigen Unterproblemen fest und betrachten die Frage der Existenz approximativer Algorithmen.

3.6 Satz. `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT` sind $\mathcal{NP}$-vollständig.
Beweis. Sicherlich sind `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT` in der Klasse $\mathcal{NP}$, denn eine `NDTM` muß nur eine entsprechende Partition erraten, die Bewertungen der Kanten einzeln berechnen, sie zur jeweiligen Gesamtbewertung durch Maximumbildung oder Summierung zusammensetzen und mit dem gewünschten k vergleichen; dies ist sicherlich in polynomialer Zeit möglich.
Die Einschränkungen der Problme `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT` auf solche Instanzen, in denen der Hypergraph H nur eine einzige Kante besitzt, definieren jeweils das gleiche Problem; schränkt man dieses Problem mit der Fragestellung

$$\exists P = (V,F) : \min\{ c_{1\cdot}\chi(f_1), c_{1\cdot}\chi(f_2) \} \geq k?$$

noch auf die Instanzen mit $k = \left\lceil c_{1\cdot}\frac{\mathbb{1}}{2} \right\rceil$ ein, so ist es mit dem $\mathcal{NP}$-vollständigen Entscheidungsproblem `PARTITION` identisch. □

Der Beweis von Satz 3.6 zeigt, daß ein ("kleines") Unterproblem der beiden betrachteten Probleme, nämlich `PARTITION`, bereits $\mathcal{NP}$-vollständig ist. Die Schwierigkeit von `PARTITION` beruht aber darauf, daß extrem große Zahlen in seinen Instanzen auftreten können. Ist dies für unsere beiden Probleme auch so? Wir wollen zeigen, daß dies nicht der Fall ist und schicken einige Ergebnisse der Komplexitätstheorie voraus.

3.7 Problem. `NOT ALL EQUAL 3SAT`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V, so daß jede Klausel genau drei verschiedene Literale über V enthält.
Frage. Gibt es ein **Satisfying Truth Assignment** für C, so daß in jeder Klausel mindestens ein Literal falsch ist? □

3.8 Satz ([Schaefer, 1978]). `NOT ALL EQUAL 3SAT` ist $\mathcal{NP}$-vollständig.
Beweis. Es ist offensichtlich, daß `NOT ALL EQUAL 3SAT` zu $\mathcal{NP}$ gehört, denn eine `NDTM` braucht lediglich ein Truth Assignment mit den gewünschten Eigenschaften zu erraten und diese in polynomialer Zeit überprüfen. Um auch die $\mathcal{NP}$-Vollständigkeit zu zeigen, reduzieren wir `3SAT` auf `NOT ALL EQUAL 3SAT`. Sei also eine Instanz von `3SAT` mit Klauseln C und Variablenmenge V gegeben. Die entsprechende Menge von Variablen V' für die zu konstruierende Instanz von `NOT ALL EQUAL 3SAT` umfaßt V, eine Variable f und für jede Klausel c aus C eine Variable x_c. Die zugehörige Menge von Klauseln enthält genau für jede Klausel (x, y, z) aus C die zwei Klauseln

$$(x_{(x,y,z)}, y, z) \quad \text{und} \quad (\overline{x}_{(x,y,z)}, x, f).$$

Wir behaupten, daß es genau dann ein den in `NOT ALL EQUAL 3SAT` genannten Bedingungen entsprechendes Satisfying Truth Assignment für C' gibt, wenn ein Satisfying Truth Assignment für C existiert.
Ist t ein Satisfying Truth Assignment für C, so definieren wir ein Truth Assignment t' über V' als

$$t'(u) := \begin{cases} t(u), & \text{falls } u \text{ ein Literal über } V \text{ ist} \\ \text{falsch}, & \text{falls } u = f \\ \text{wahr}, & \text{falls } u = \overline{f} \\ \overline{y \vee z}, & \text{falls } u = x_{(x,y,z)} \\ y \vee z, & \text{falls } u = \overline{x}_{(x,y,z)}. \end{cases}$$

Ist für eine Klausel (x, y, z) aus C $y \vee z$ wahr, so ist nach Definition $x_{(x,y,z)}$ falsch, also die erste Klausel mit dem falschen Literal $x_{(x,y,z)}$ erfüllt; $\overline{x}_{(x,y,z)}$ ist dann wahr, f aber immer falsch, also ist auch die zweite Klausel mit falschem Literal f erfüllt. Ist $y \vee z$ dagegen falsch, so ist nach Definition $x_{(x,y,z)}$ wahr, also die erste Klausel mit falschen Literalen y und z erfüllt; außerdem ist x wahr, also die zweite Klausel mit falschem Literal $\overline{x}_{(x,y,z)}$ erfüllt.
Sei umgekehrt eine Lösung t' der konstruierten Instanz von `NOT ALL EQUAL 3SAT` gegeben; ohne Beschränkung der Allgemeinheit dürfen wir annehmen, daß f falsch ist. Wir behaupten, daß die Einschränkung von t' auf die Literale über V ein Satisfying Truth Assignment für C ist. Betrachten wir eine Klausel (x, y, z) aus C; wären x, y und z alle drei falsch, so müßte $x_{(x,y,z)}$ wahr, also $\overline{x}_{(x,y,z)}$ falsch sein und die Klausel $(\overline{x}_{(x,y,z)}, x, f)$ aus C' wäre im Widerspruch zu unserer Annahme nicht erfüllt. Also ist eine der drei Variablen wahr und somit die Klausel (x, y, z) erfüllt.
Es bleibt noch zu überprüfen, daß die genannte Transformation von `3SAT` auf `NOT ALL EQUAL 3SAT` polynomial ist; dies ist offensichtlich. □

3.9 Problem. `SET SPLITTING`.
Instanz. Gegeben sei ein Hypergraph $H = (V, E)$.
Frage. Gibt es eine Partition $P = \big(V, \{ f_1, f_2 \}\big)$ auf V, so daß der Hypergraph $H \cdot f_1 \cdot f_2$ keine Schlinge enthält? □

3.10 Satz ([Lovasz, 1973]). `SET SPLITTING` ist $\mathcal{NP}$-vollständig, sogar wenn alle Kanten von H höchstens drei Elemente enthalten.
Beweis. Die Zugehörigkeit von `SET SPLITTING` zu $\mathcal{NP}$ ist klar, denn ein nichtdeterministischer Algorithmus braucht lediglich eine geeignete Partition P zu erraten und in polynomialer Zeit zu überprüfen, ob jede Kante e von H nichtleeren Durchschnitt mit den beiden Kanten von P besitzt.
Gelingt es uns, `NOT ALL EQUAL 3SAT` auf Problem 3.9 zu transformieren, so ist Problem 3.9 schwieriger als `NOT ALL EQUAL 3SAT`, also $\mathcal{NP}$-vollständig.
Sei daher eine Instanz von `NOT ALL EQUAL 3SAT` mit Klauseln C über V gegeben. Für jede Variable u aus V enthält die Menge der Knoten V' des zu konstruierenden Hypergraphen H' zwei Knoten u und $\overline{u}$, die den gleichnamigen Literalen entsprechen. Als Menge der Kanten von H' wählen wir

$$\big\{ \{ u, \overline{u} \} : u \in V \big\} \cup \big\{ \{ x, y, z \} : (x, y, z) \in C \big\}.$$

Ist nun t ein Satisfying Truth Assignment für C, so definieren wir eine Partition P durch Angabe ihrer Kanten f_1 und f_2 als

$$f_1 := \{\, u : u \text{ ist wahr} \,\} \quad \text{und} \quad f_2 := \{\, u : u \text{ ist falsch} \,\}.$$

Betrachten wir die Kanten $\{\, u, \overline{u} \,\}$. Ist der Literal u wahr und der Literal $\overline{u}$ falsch, so ist der Knoten u ein Element von f_1 und der Knoten $\overline{u}$ gehört zu f_2. Sehen wir uns nun die Kanten $\{\, x, y, z \,\}$ an. In der Klausel (x, y, z) ist ein Literal falsch, sagen wir x, und ein anderer wahr; dies sei y. Dann gehört aber der Knoten x zu f_1 und der Knoten y zu f_2. Insgesamt hat jede Kante von H' nichtleeren Durchschnitt sowohl mit f_1 alsauch mit f_2.
Sei umgekehrt eine Partition P eine Lösung der konstruierten Instanz mit dem Hypergraphen H'. Dann ist das Truth Assignment t über V, definiert durch

$$t(u) := \begin{cases} \text{wahr}, & \text{falls } u \in f_1 \\ \text{falsch}, & \text{falls } u \in f_2 \end{cases}$$

wohldefiniert, denn die den Literalen u und $\overline{u}$ entsprechenden Knoten befinden sich niemals zusammen in einer der Mengen f_1 oder f_2, da sonst die Menge $\{\, u, \overline{u} \,\}$ leeren Durchschnitt mit einer der beiden Kanten von P hätte; außerdem ist jede Klausel (x, y, z) aus C erfüllt, denn die Menge $\{\, x, y, z \,\}$ hat ein Element mit der "Menge der wahren Literale" f_1 gemeinsam.
Daß die durchgeführte Transformation von polynomialer Art ist, ist ohne Schwierigkeiten zu erkennen. □

Nach dieser Vorbereitung erhalten wir nun ohne Schwierigkeiten das gewünschte Resultat in sehr scharfer Form.

3.11 Satz. Die Teilprobleme von `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT`, in denen die Kostenmatrix nur Nullen oder Einsen als Einträge besitzt, sind $\mathcal{NP}$-vollständig, sogar, wenn in jeder Kante des Hypergraphen H höchstens drei Knoten enthalten sind.
Beweis. Als Einschränkungen $\mathcal{NP}$-vollständiger Probleme sind auch die genannten Probleme `WEIGHTED SET SPLITTING` beziehungsweise `HYPERGRAPH MAX CUT` mit Kostenmatrix jeweils aus $\{\, 0, 1 \,\}^{m \times m}$ in $\mathcal{NP}$.
Betrachten wir zunächst den genannten Spezialfall von `WEIGHTED SET SPLITTING`. Wir wollen auf ihn Problem 3.9 reduzieren, wobei wir die Einschränkung von Problem 3.9 auf solche Probleme mit höchstens drei Knoten pro Kante benutzen wollen. Sei also eine Instanz dieses Spezialfalles von Problem 3.9 durch einen Hypergraphen $H = (V, E)$ gegeben. Die entsprechende Instanz unseres eingeschränkten `WEIGHTED SET SPLITTING`-Problems besteht aus demselben Hypergraphen H, als Kostenmatrix C wollen wir die Inzidenzmatrix von H verwenden, als k wählen wir 1. Dann gilt aber, wenn H m Kanten e_i besitzt, wegen $|e_i| \leq 3$ für jede Partition $P = \big(V, \{\, f_1, f_2 \,\}\big)$ die Beziehung

$$\begin{aligned} &\min_{i=1}^{m} \min\{\, c_i \cdot \chi\,(f_1), c_i \cdot \chi\,(f_2) \,\} = \min_{i=1}^{m} \min\{\, |e_i \cap f_1|, |e_i \cap f_2| \,\} \geq 1 \\ \iff\ & e_i \cap f_1 \neq \emptyset \text{ und } e_i \cap f_2 \neq \emptyset \quad \forall i = 1, \ldots, m, \end{aligned}$$

das heißt P ist eine Lösung der Instanz von Problem 3.9 dann und nur dann, wenn P auch eine Lösung der konstruierten Instanz von `WEIGHTED SET SPLITTING` ist; letztere gehört jedoch zur interessierenden Klasse von Problemen mit Kostenmatrizen aus Nullen und Einsen, H hat nach Voraussetzung höchstens drei Knoten in jeder Kante. Die genannte Transformation ist natürlich auch poynomial, so daß die Behauptung bezüglich `WEIGHTED SET SPLITTING` bewiesen ist.
Im zweiten Schritt konzentrieren wir uns auf die genannte Einschränkung von `HYPERGRAPH MAX CUT`. Ist wieder eine Instanz von Problem 3.9 wie gerade gegeben, so konstruieren wir die Instanz von `HYPERGRAPH MAX CUT` wie gerade die von `WEIGHTED SET SPLITTING`, lediglich als k wählen wir $m = |E|$. Dann gilt wieder wegen $|e_i| \leq 3$

$$\begin{aligned} &\sum_{i=1}^{m} \min\{\, c_i \cdot \chi\,(f_1), c_i \cdot \chi\,(f_2) \,\} = \sum_{i=1}^{m} \min\{\, |e_i \cap f_1|, |e_i \cap f_2| \,\} \geq |E| = m \\ \iff\ & e_i \cap f_1 \neq \emptyset \text{ und } e_i \cap f_2 \neq \emptyset \quad \forall i = 1, \ldots, m, \end{aligned}$$

und man argumentiert wie im ersten Teil des Beweises weiter. □

Das in Satz 3.11 erzielte Resultat läßt sich für `HYPERGRAPH MAX CUT` noch ein wenig verschärfen, wozu wir allerdings wieder einige Vorbereitungen benötigen.

3.12 Problem. `MAXIMUM 2SATISFIABILITY`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V, so daß jede Klausel genau zwei verschiedene Literale über V enthält, und eine natürliche Zahl k.
Frage. Gibt es ein Truth Assignment für C, so daß mindestens k der $|C|$ Klauseln erfüllt sind? □

3.13 Satz ([Garey, Johnson & Stockmeyer, 1976]). `MAXIMUM 2SATISFIABILITY` ist $\mathcal{NP}$-vollständig.
Beweis. Daß `MAXIMUM 2SATISFIABILITY` zur Klasse $\mathcal{NP}$ gehört, ist leicht zu sehen, denn ein nichtdeterministischer Algorithmus kann eine Instanz von `MAXIMUM 2SATISFIABILITY` in polynomialer Zeit in der Weise lösen, daß er ein entsprechendes Truth Assignment für C errät und überprüft, ob die notwendige Anzahl an Klauseln erfüllt ist.
Um auch die $\mathcal{NP}$-Vollständigkeit zu zeigen, reduzieren wir `SAT` auf `MAXIMUM 2SATISFIABILITY`. Sei also eine Instanz von `SAT` mit Klauseln C über einer Menge logischer Variablen V gegeben. Die Variablenmenge V' der zu konstruierenden Instanz von `MAXIMUM 2SATISFIABILITY` enthält außer ganz V für jede Klausel $c = (x, y, z)$ auch noch Variablen X_c, Y_c und Z_c mit der Interpretation, daß die Klausel aufgrund der Wahrheit des Literals x, y oder z erfüllt ist, sowie weitere drei Variablen K_c^1, K_c^2 und K_c^3 mit noch zu erklärendem Zweck; hinzu kommt eine einzige Variable f. Vereinbaren wir, daß für Literale x und y über V' $(x \Longrightarrow y)$ die Klausel $(\overline{x}, y)$ bezeichnen soll, so enthält die Menge C' der Klauseln unserer Instanz von `MAXIMUM 2SATISFIABILITY` für jede Klausel $c = (x, y, z)$ aus C die folgenden 18 Klauseln.

$(X_c \Longrightarrow x)$	$(Y_c \Longrightarrow y)$	$(Z_c \Longrightarrow z)$
$(X_c \Longrightarrow \overline{Y}_c)$	$(Y_c \Longrightarrow \overline{X}_c)$	$(Z_c \Longrightarrow \overline{X}_c)$
$(X_c \Longrightarrow \overline{Z}_c)$	$(Y_c \Longrightarrow \overline{Z}_c)$	$(Z_c \Longrightarrow \overline{Y}_c)$
(X_c, f)	(Y_c, f)	(Z_c, f)
$(K_c^1, \overline{f})$	$(K_c^2, \overline{f})$	$(K_c^3, \overline{f})$
$(\overline{K}_c^1, \overline{f})$	$(\overline{K}_c^2, \overline{f})$	$(\overline{K}_c^3, \overline{f}.)$

Als k wollen wir zuletzt noch $16|C|$ festlegen.
Wir behaupten, daß es genau dann ein Satisfying Truth Assignment für C gibt, wenn es ein Truth Assignment für C' gibt, in dem mindestens $16|C|$ Klauseln erfüllt sind.
Sei also zunächst t ein Satisfying Truth Assignment für C. Wir betrachten eine einzelne Klausel $c = (x, y, z)$ aus C. Nehmen wir ohne Beschränkung der Allgemeinheit an, daß x erfüllt ist, so definieren wir

$t'(X_c) :=$ wahr	$t'(Y_c) :=$ falsch	$t'(Z_c) =$ falsch
$t'(\overline{X}_c) :=$ falsch	$t'(\overline{Y}_c) :=$ wahr	$t'(\overline{Z}_c) =$ wahr
$t'(K_c^1) :=$ wahr	$t'(K_c^2) :=$ wahr	$t'(K_c^3) :=$ wahr
$t'(\overline{K}_c^1) :=$ falsch	$t'(\overline{K}_c^2) :=$ falsch	$t'(\overline{K}_c^3) :=$ falsch

für jede solche Klausel aus C und vervollständigen diese Definition eines Truth Assignments über C' durch die Festsetzungen

$$t'(f) := \text{falsch}, t'(\overline{f}) := \text{ wahr} \quad \text{und} \quad t'(u) = t(u) \quad \forall \text{ Literale } u \text{ über } V.$$

Wie der Leser unschwer erkennt, erfüllt t' für jede Klausel $c = (x, y, z)$ aus C von den ihr entsprechenden 18 Klauseln in C' gerade zwei nicht, nämlich, falls wieder ohne Beschränkung der Allgemeinheit x wahr ist, die Klauseln (Y_c, f) und (Z_c, f); insgesamt erfüllt t' damit (genau) die verlangten $16|C|$ Klauseln und ist eine Lösung von `MAXIMUM 2SATISFIABILITY`.
Sei nun umgekehrt ein Truth Assignment t' über C' gegeben, das von den $18|C|$ Klauseln in C' mindestens $16|C|$ erfüllt; wir behaupten, daß seine Einschränkung auf die Literale über V ein Satisfying Truth Assignment für C ist. Betrachten wir für eine Klausel c aus C die ihr entsprechenden 18 Klauseln in der Instanz von `MAXIMUM 2SATISFIABILITY`. Wir zeigen zunächst, daß es für kein Truth Assignment möglich ist, von diesen mehr als 16 zu erfüllen.

Ist dies nämlich der Fall, so muß notwendig f falsch sein, K_c^1, K_c^2 und K_c^3 können dagegen beliebig besetzt werden; tatsächlich sind sie nur vorhanden, um den erstgenannten Schluß zu ermöglichen. Ferner dürfen nicht alle drei Literale X_c, Y_c und Z_c wahr sein. Sind jedoch genau zwei von diesen, sagen wir X_c und Y_c wahr, so sind bereits die Klauseln $(X_c \Longrightarrow \overline{Y}_c)$ und $(Y_c \Longrightarrow \overline{X}_x)$ im Widerspruch zur Voraussetzung nicht erfüllt. Also ist genau einer drei genannten Literale, ohne Beschränkung der Allgemeinheit X_c, wahr, denn wären alle falsch, dann wären die Klauseln (X_c, f), (Y_c, f) und (Z_c, f) nicht erfüllt. Jetzt aber sind (Y_c, f) und (Z_c, f) nicht erfüllt.
Das Truth Assignment t' muß daher für jede Klausel $c = (x, y, z)$ aus C von den dieser entsprechenden 18 Klauseln aus C' genau 16 erfüllen. Wir überlegen uns jetzt, daß hieraus folgt, daß einer der in ihr enthaltenen Literale x, y oder z wahr ist. Angenommen, das wäre nicht der Fall. Analog zu oben folgt zunächst, daß aus dem Erfülltsein von 16 Klauseln die Wahrheit von $\overline{f}$ folgt sowie, daß nicht alle drei Literale X_c, Y_c und Z_c wahr sind. Umgekehrt muß jedoch mindestens einer derselben erfüllt sein, denn sonst wären die drei Klauseln (X_c, f), (Y_c, f) und (Z_c, f) nicht erfüllt. Ist nur einer dieser Literale, etwa X_c erfüllt, so sind die Klauseln (Y_c, f) und (Z_c, f) bereits zwei nichterfüllte, so daß die Existenz der Klausel (X_c, x) die Wahrheit von x garantiert. Sind dagegen zwei Literale, etwa neben X_c noch Y_c wahr, so sind $(X_c \Longrightarrow \overline{Y}_c)$ und $(Y_c \Longrightarrow \overline{X}_c)$ nicht erfüllt. Es folgt, daß $(X_c \Longrightarrow x)$ erfüllt sein muß, so daß x wahr ist.
Wir haben damit gezeigt, daß das Truth Assignment t' für jede Klausel $c = (x, y, z)$ einen ihrer Literale als wahr definiert; die Einschränkung von t' auf die Literale über V ist somit ein Satisfying Truth Assignment für C.
Es bleibt zu zeigen, daß die genannte Transformation polynomial ist. Dies ist jedoch klar. □

3.14 Problem. `MAX CUT`.
Instanz. Graph $G = (V, E)$ mit natürlichen Kantengewichten c_e für alle e aus E und eine natürliche Zahl k.
Frage. Gibt es einen Schnitt mit größerem Gewicht als k, das heißt existiert eine Teilmenge W von V mit der Eigenschaft

$$\sum_{e \in \delta(W)} c_e \geq k?$$

□

3.15 Satz ([Karp, 1972]). `MAX CUT` ist $\mathcal{NP}$-vollständig, sogar wenn als einziges Kantengewicht die 1 erlaubt ist.
Beweis. Die Zugehörigkeit von `MAX CUT` zu $\mathcal{NP}$ folgt aus der Tatsache, daß eine `NDTM` eine geeignete Teilmenge W von V erraten und dann in polynomialer Zeit verifizieren kann, ob W einen Schnitt der geforderten Größe in G induziert.
Um auch die $\mathcal{NP}$-Vollständigkeit nachzuweisen, reduzieren wir `MAXIMUM 2SATISFIABILITY` auf das wie oben eingeschränkte Problem `MAX CUT`. Sei also eine Instanz von `MAXIMUM 2SATISFIABILITY` mit Klauseln C und Variablen V sowie einer natürlichen Zahl k gegeben; ohne Beschränkung der Allgemeinheit dürfen wir hierbei $k \leq |C|$ annehmen.
Der Graph $G' = (V', E')$ der zugehörigen Instanz von `MAX CUT` enthält für jeden Literal u über V einen Knoten, den wir der Einfachheit halber ebenfalls mit u bezeichnen wollen, sowie weitere $3|C|$ Knoten u_i und die gleiche Anzahl Knoten $\overline{u}_i$. Hinzu kommen noch für jede Klausel c aus C ein Knoten f_c sowie insgesamt $3|C|$ Knoten w_i. Die Menge der Kanten enthält zunächst für jedes Paar von Literalen u und $\overline{u}$ über V die Kantenmengen

$$\big\{\, \{\, u, u_i \,\} : 1 \leq i \leq 3|C| \,\big\} \cup \big\{\, \{\, u_i, \overline{u}_i \,\} : 1 \leq i, j \leq 3|C| \,\big\} \cup \big\{\, \{\, \overline{u}_i, \overline{u} \,\} : 1 \leq i \leq 3|C| \,\big\}.$$

Zusätzlich gibt es noch für jede Klausel $c = (x, y)$ aus C die drei Kanten

$$\{\, x, f_c \,\}, \quad \{\, y, f_c \,\} \quad \text{und } \{\, x, y \,\}. \tag{2}$$

sowie die Kanten

$$\big\{\, \{\, f_c, w_i \,\} : c \in C, 1 \leq i \leq 3|C| \,\big\}.$$

Als zu diesen Daten gehörige natürliche Zahl k' wählen wir $3|C|^2 + 9|V||C| + 2k$, als Kantengewichte 1.

Wir behaupten, daß es genau dann ein Truth Assignment für C gibt, das k oder mehr Klauseln erfüllt, wenn es in G' einen Schnitt vom Gewicht k' gibt.
Sei also ein Truth Assignment t für C gegeben, das k (oder mehr) der insgesamt $|C|$ Klauseln erfüllt. Wir konstruieren eine Teilmenge W' von V' mit leicht verständlicher Notation als

$$W' := \{\, u : t(u) = \text{wahr} \,\} \cup \{\, \overline{u}_i : t(u) = \text{wahr} \,\} \cup \{\, w_i \,\},$$

ihr Komplement F' ist dann

$$F' := \{\, \overline{u} : t(u) = \text{wahr} \,\} \cup \{\, u_i : t(u) = \text{wahr} \,\} \cup \{\, f_c : c \in C \,\}.$$

Der von W' induzierte Schnitt hat dann bezüglich der Kantengewichtung mit lauter Einsen ein Gewicht von

$$\begin{aligned}
\sum_{e \in \delta(W)} 1 =& |\delta(W)| \\
=& \Big|\big\{\, \{\, f_c, w_i \,\} : c \in C, 1 \le i \le 3|C| \,\big\}\Big| \\
&+ \Big(\Big|\big\{\, \{\, u, u_i \,\} \forall u, i \,\big\}\Big| + \Big|\big\{\, \{\, u_i, \overline{u}_i \,\} \forall u, i \,\big\}\Big| + \Big|\big\{\, \{\, \overline{u}_i, \overline{u} \,\} \forall u, i \,\big\}\Big|\Big) \\
&+ \Big|\big\{\, \{\, u, f_c \,\} : u \in W', u \text{ ist Literal in } c \,\big\}\Big| + \Big|\big\{\, \{\, x, y \,\} : (x, y) \in C, t(x) = t(\overline{y}) \,\big\}\Big| \\
=& 3|C|^2 + |V|\{\, 3|C| + 3|C| + 3|C| \,\} + 2|\{\, c \in C : t \text{ erfüllt } c \,\}| \\
\ge& 3|C|^2 + 9|V||C| + 2k;
\end{aligned}$$

für den Term $2|\{\, c \in C : t \text{ erfüllt } c \,\}|$ beachte man hierbei, daß der Graph

$$\Big(\big\{\, x, y, f_{(x,y)} \,\big\}, \Big\{\, \big\{\, x, f_{(x,y)} \,\big\}, \big\{\, y, f_{(x,y)} \,\big\}, \{\, x, y \,\} \Big\}\Big)$$

für jede Klausel (x, y) aus C ein Untergraph von G' ist und ein "Dreieck" bildet. Der Knoten $f_{(x,y)}$ ist außer zu den Knoten w_i zu keinen anderen Knoten außer x und y inzident und in F' (der Menge der "falschen" Knoten) enthalten. Für jedes solche Dreieck, das zu einer unter t erfüllten Klausel c gehört, sind aber einer oder beide der in ihr auftretenden Literale wahr und die zugehörigen Knoten Elemente von W', so daß sich im Dreieck in jedem Falle ein Schnitt vom Wert zwei ergibt. Gehört das Dreieck dagegen zu einer nicht unter t erfüllten Klausel, so sind alle drei Ecken Elemente von F' und der Beitrag zur Zielfunktion ist Null.
Sei umgekehrt ein Schnitt $\delta(W)$ in G' vom Wert k' oder größer gegeben; wir bezeichnen das Komplement von W mit F.
Wir zeigen als erstes, daß ohne Beschränkung der Allgemeinheit alle Knoten f_c in F liegen, während alle Knoten w_i Mitglieder von W sind. Wir bemerken zunächst, daß der Graph G' insgesamt nur

$$|C|3|C| + 9|V||C| + 3|C|$$

Kanten besitzt, der gegebene Schnitt von diesen also höchstens $3|C| - 2k$ nicht benutzen darf. Nehmen wir an, daß alle Knoten f_c in F enthalten sind, so können wir auch davon ausgehen, daß alle Knoten w_i in W sind, denn letztere sind in G' genau zu den Knoten f_c adjazenzt; ein Schnitt würde also durch Umordnen aller Knoten w_i nach W höchstens größer. Sei also $0 < f_F < C$ die Anzahl der in F enthaltenen Knoten f_c und $0 \le w_W \le 3|C|$ die Anzahl der in W enthaltenen Knoten w_i, so enthält der Schnitt $\delta(W)$ von den im vollständigen bipartiten Untergraphen mit Knotenmengen $\{\, f_c \,\}$ und $\{\, w_i \,\}$ enthaltenen Kanten nur

$$w_W f_F + (3|C| - w_W)(|C| - f_F) \le w_W(|C| - 1) + (3|C| - w_W)(|C| - 1) = 3|C|(|C| - 1) = 3|C|^2 - 3|C|,$$

also zuwenig. Somit muß also die Behauptung richtig sein und alle $3|C|^2$ Kanten $\{\, f_c, w_i \,\}$ sind im Schnitt $\delta(W)$ enthalten.
Wir betrachten nun für jede Variable u aus V die Knoten u_i und $\overline{u}_i$. u_i ist nur zu $\overline{u}_i$ und zu u inzident, $\overline{u}_i$ nur zu u_i und $\overline{u}$. Wie daher auch u und $\overline{u}$ auf die Menge W und ihr Komplement aufgeteilt werden, wir

verkleinern den Wert des Schnittes nicht, wenn wir jeweils u_i beziehungsweise $\overline{u}_i$ auf die u beziehungsweise $\overline{u}$ entgegengesetzte Seite des Schnittes bringen, so daß wir also ohne Beschränkung der Allgemeinheit

$$u \in W \iff u_i \in F \quad \text{und} \quad \overline{u} \in W \iff \overline{u}_i \in F$$

für jeweils alle i voraussetzen dürfen. Träte nun für zwei Knoten u und $\overline{u}$ der Fall ein, daß sie beide Elemente von W oder beide Elemente von F wären, so erhielten wir für diese als Wert des Schnittes im maximalen Untergraphen von G mit Knotenmenge $\{ u, \overline{u}, u_i, \overline{u}_i \}$

$$\left|\left\{ \{ u, u_i \} : 1 \leq i \leq 3|C| \right\}\right| + \left|\left\{ \{ \overline{u}_i, \overline{u} \} : 1 \leq i \leq 3|C| \right\}\right| = 6|C|,$$

zuwenig, denn es fehlen die $3|C|$ Kanten $\{ u_i, \overline{u}_i \}$. Also liegen u und $\overline{u}$ für alle Variablen u aus V stets auf verschiedenen Seiten des Schnittes, was alle $9|V||C|$ Kanten in $\left\{ \{ u, u_i \} \right\} \cup \left\{ \{ u_i, \overline{u}_i \} \right\} \cup \left\{ \{ \overline{u}_i, \overline{u} \} \right\}$ zu Mitgliedern des Schnittes $\delta(W)$ macht.
Die einzigen noch nicht betrachteten Kanten gehören zu den Dreiecken

$$\Big(\{ x, y, f_{(x,y)} \}, \{ \{ x, f_{(x,y)} \}, \{ y, f_{(x,y)} \}, \{ x, y \} \} \Big).$$

Diese bilden für jede Klausel (x, y) aus C einen Untergraphen von G', während uns andererseits noch genau $2k$ Kanten an der geforderten Anzahl von k' Kanten fehlen. Zu W oder F zuordbar sind jedoch nurmehr die den Literalen entsprechenden Knoten. Es ist nicht schwer zu sehen, daß in jedem Dreieck genau dann zwei Kanten in $\delta(W)$ enthalten sind, wenn mindestens einer der einem Literal entsprechenden Knoten in der Menge W enthalten ist, da alle Knoten f_c Mitglieder von F waren.
Es folgt, daß das Truth Assignment t, definiert durch

$$t(u) = \text{wahr} \iff u \in W$$

mindestens k Klauseln aus C erfüllt.
Ebenso leicht ist zu sehen, daß die durchgeführte Transformation in polynomialer Zeit ausgeführt werden kann. □

Nunmehr können wir eine sehr gute Charakterisierung der Komplexität unserer beiden Probleme geben.

3.16 Satz.

1. `WEIGHTED SET SPLITTING`, eingeschränkt auf Instanzen mit Hypergraphen H mit höchstens zwei Knoten pro Kante (also Graphen) und Kostenmatrix mit Einträgen aus $\{ 0, 1 \}$, ist polynomial lösbar.
2. `HYPERGRAPH MAX CUT`, eingeschränkt auf Instanzen mit Hypergraphen H mit höchstens zwei Knoten pro Kante (also Graphen) und Kostenmatrix mit Einträgen aus $\{ 0, 1 \}$, ist $\mathcal{NP}$-vollständig.

Beweis.

1. Der Hypergraph H ist nach Voraussetzung ein Graph; ferner ist nur der Fall $k = 1$ nichttrivial. Unter dieser Voraussetzung läßt sich die gestellte Frage aber offenbar auch so formulieren: "Ist der Graph H schlingenlos und bipartit?" Diese Frage ist offenbar in einfacher Weise in polynomialer Zeit entscheidbar.
2. Das betrachtete Problem ist gerade die Frage, in einem Graphen mit Kantengewichten von 1 einen maximalen Schnitt zu finden; dieses Problem ist nach Satz ([Karp, 1972]) 3.15 $\mathcal{NP}$-vollständig. □

Das Resultat von Satz 3.11 ist also in gewissem Sinne für `WEIGHTED SET SPLITTING` optimal, während es sich für `HYPERGRAPH MAX CUT` noch einmal in Satz 3.16 verschärfen läßt; dann ist auch dieses Ergebnis in diesem Sinne nicht mehr verbesserbar, denn für höchstens einen Knoten pro Kante wird `HYPERGRAPH MAX CUT` trivial.
Satz 3.16 ist von großer Wichtigkeit, denn der Satz klärt, daß die meisten Spezialfälle, die einem bei der Betrachtung von `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT` in den Sinn kommen, ebenfalls $\mathcal{NP}$-vollständig sind. Insbesondere hat es keinen Zweck, die Anzahl der Elemente in den Zeilen der Kostenmatrix C, also die Anzahl der Knoten in den Kanten des Hypergraphen H zu beschränken, auf der Gleichartigkeit der Einträge in C, also der Gewichte der Knoten in Abhängigkeit von ihrer Zugehörigkeit zu verschiedenen Kanten zu bestehen oder ähnliche Zusatzbedingungen zu stellen. Spezielle Formen des Hypergraphen H oder, wenn man so will, der Verteilung der Nichtnullelemente in der Kostenmatrix C helfen

uns natürlich auch nicht weiter, da beide Probleme schon für nur eine Kante $\mathcal{NP}$-vollständig sind. Auch eine Bedingung, die den Autor einige Zeit beschäftigt hat, vereinfacht das Problem nicht; der Autor will sie trotzdem kurz nennen. Betrachten wir `HYPERGRAPH MAX CUT`. Warum ist das Problem schwierig? Der Autor war der Meinung, daß die nichtlineare Zielfunktion Probleme verursacht; versucht man, einen Algorithmus zur (approximativen) Lösung zu entwickeln, so kämpft man stets mit der Schwierigkeit nicht zu wissen, welche "Seite" jeder Kante letztendlich in der Zielfunktion auftauchen wird. Der Autor glaubte einige Zeit, daß, wenn für jede Zeile der Kostenmatrix C ein Element mindestens die Hälfte der Gewichtssumme dieser Zeile ausmacht und man daher die Zielfunktion in Komponenten zerlegen kann, die sich jeweils darauf beziehen, ob ein Knoten sich in der gleichen oder entgegesetzten Hälfte einer Partition befindet wie der in obigem Sinne eine Kante (entspricht einer Zeile der Kostenmatrix) dominierende Knoten, das Problem einfacher würde. Satz 3.16.2 zeigt jedoch, daß dies nicht der Fall ist. Tatsächlich scheint die Quelle der Schwierigkeit der beiden Probleme in der Vielzahl der Kanten des Hypergraphen H zu liegen; diese Größe einschränkende Bedingungen werden wir noch diskutieren.
Die $\mathcal{NP}$-Vollständigkeit geklärt und keine einfachen, also polynomial lösbaren und relevanten Spezialfälle gefunden, ist der nächste Punkt auf der Komplexitätscheckliste die Frage nach der Existenz von Approximationsschemata.

3.17 Satz. Für das Optimierungsproblem `WEIGHTED SET SPLITTING` und beliebiges rationales $0 < \epsilon < 1$ existiert genau dann ein ϵ-approximativer Algorithmus, wenn $\mathcal{P}$ und $\mathcal{NP}$ gleich sind; insbesondere existiert auch genau in diesem Falle ein `PAS` oder ein `FPAS`.
Beweis. Angenommen, für ein ϵ größer als Null existiert ein solcher ϵ-approximativer Algorithmus A; dann liefert er auch für die Einschränkung von `WEIGHTED SET SPLITTING` auf Instanzen mit Hypergraphen mit höchstens drei Knoten pro Kante und Kostenmatrizen aus $\{0,1\}^{m\times n}$ stets eine Partition $P^\epsilon = (V, \{f_1^\epsilon, f_2^\epsilon\})$ mit der Eigenschaft

$$\min_{i=1}^{m} \min\{ c_{i\cdot}\chi(f_1^\epsilon), c_{i\cdot}\chi(f_2^\epsilon) \} \geq (1-\epsilon)c_{\text{opt}},$$

wo c_{opt} den optimalen Zielfunktionswert der betrachteten Instanz bezeichne. Für die genannte Einschränkung von `WEIGHTED SET SPLITTING` gilt aber sogar für jede Partition $P = (V, F)$

$$\min_{i=1}^{m} \min\{ c_{i\cdot}\chi(f_1^\epsilon), c_{i\cdot}f_2^\epsilon \} \in \{0,1\},$$

so daß eine ϵ-approximative Lösung stets bereits die optimale ist. □

Wie ist die Situation im Falle des Optimierungsproblems `HYPERGRAPH MAX CUT`? Unglücklicherweise muß der Autor die Antwort auf diese Frage weitgehend schuldig bleiben, denn es ist ihm nicht gelungen, ein `PAS` oder einen approximativen Algorithmus zu konstruieren, jedoch auch nicht zu beweisen, daß dies nicht möglich ist, und zuletzt auch nicht, daß die Frage in einem Zusammenhang zu anderen $\mathcal{NP}$-vollständigen (Optimierungs-)Problemen steht, für die die Situation die gleiche ist, wie etwa `INDEPENDENT SET`. Immerhin sind die folgenden Aussagen möglich.

3.18 Satz. Es gibt ein `FPAS` für das Optimierungsproblem `HYPERGRAPH MAX CUT` genau dann, wenn $\mathcal{P}$ gleich $\mathcal{NP}$ ist.
Beweis. `HYPERGRAPH MAX CUT` ist $\mathcal{NP}$-vollständig auch dann noch, wenn in der Kostenmatrix nur Einträge aus $\{0,1\}$ erlaubt sind. Ein `FPAS` kann dann aber in zur Codierungslänge der Instanz und $1/\frac{1}{mn+1} = mn+1$, also insgesamt in zur Codierungslänge der Instanz polynomialer Zeit eine $\frac{1}{mn+1}$-approximative, also optimale Lösung finden. □

3.19 Satz. Es gibt ein `FPAS` für `HYPERGRAPH MAX CUT` eingeschränkt auf Instanzen, in denen der Hypergraph $H = (V, E)$ ein Baum ist.
Beweis. Wir machen Gebrauch von der Tatsache, daß es für das Optimierungsproblem `SUBSET SUM` ein `FPAS` gibt ([Ibarra & Kim, 1975]). Wir wollen ein `FPAS` angeben, das diesen Algorithmus als Unterprogramm verwendet.
Sei also eine Instanz von `HYPERGRAPH MAX CUT` mit Baum $H = (V, E)$ und Kostenmatrix C und eine rationale Zahl $0 < \epsilon < 1$ gegeben. Für jede Kante e_i von H bestimmen wir jetzt mit Hilfe des `FPAS` von Ibarra und

Kim eine ϵ-approximative Lösung $P_i = (V, \{ f_1^i, f_2^i \})$ des Optimierungsproblems

$$\max_{\text{Partition } P} c_i \cdot \chi\left(f_1^i\right)$$
$$c_i \cdot \chi\left(f_1^i\right) \leq \left\lfloor c_i \cdot \frac{\mathbb{1}}{2} \right\rfloor;$$

seien außerdem jeweils c_ϵ^i die Zielfunktionswerte dieser approximativen, c_{opt}^i dagegen die optimalen Zielfunktionswerte dieser Probleme.
Wir zeigen nun mit Hilfe vollständiger Induktion über die Anzahl der Kanten von H, daß sich die approximativen Lösungen P_i in einfacher Weise in polynomialer Zeit zu einer Partition $P = (V, \{ f_1, f_2 \})$ zusammensetzen lassen, so daß die Beziehung

$$\sum_{i=1}^{m} c_\epsilon^i = \sum_{i=1}^{m} \min\{ c_i \cdot \chi\left(f_1^i\right), c_i \cdot \chi\left(f_2^i\right) \} = \sum_{i=1}^{m} \min\{ c_i \cdot \chi\left(f_1\right), c_i \cdot \chi\left(f_2\right) \} \tag{3}$$

zutrifft. Der Induktionsanfang $m = 1$ ist klar. Angenommen, für weniger als m Kanten kann man stets eine Partition mit der genannten Eigenschaft finden und H besitzt nun m Kanten. Nach Induktionsannahme existiert eine Partition $P' = (V, \{ f_1', f_2' \})$ mit der Eigenschaft

$$\sum_{i=1}^{m-1} c_\epsilon^i = \sum_{i=1}^{m-1} \min\{ c_i \cdot \chi\left(f_1\right), c_i \cdot \chi\left(f_2\right) \}. \tag{4}$$

Da H ein Baum ist, besitzt er ein Blatt, wir dürfen annehmen e_m. Dieses Blatt hat mit den übrigen Kanten des Baumes entweder keinen oder genau einen Knoten gemeinsam. Im ersten Falle ist der Unterhypergraph $(e_m, \{ e_m \})$ eine Komponente von H und für die Partition

$$P = \left(V, \left(f_1' \setminus e_m\right) \cup f_1^m, \left(f_2' \setminus e_m\right) \cup f_2^m\right)\Big)$$

folgt die Behauptung (3) unmittelbar, denn es gilt $(c_{ij})_{\substack{i=1,\ldots,m-1 \\ j \in e_m}} = 0$ und $(c_{mj})_{j \notin e_m} = 0$. Ansonsten haben e_m und $\bigcup_{i=1}^{m-1} \{ e_i \}$ genau einen Knoten, sagen wir 1, gemeinsam. Ohne Beschränkung der Allgemeinheit dürfen wir jetzt aber von $\{ 1 \} = f_1^m \cap f_1' \neq \emptyset$ ausgehen, denn eine Vertauschung der Indizes der Kanten von P' ändert nichts an der Gültigkeit von (4) im Fall $m - 1$. Jetzt können wir aber wieder P wie gerade erklären und (3) ist auch für m gültig.
Jetzt gilt jedoch, falls E m Kanten enthält

$$\begin{aligned}
&(1-\epsilon) \max_{\text{Partition } P=(V,\{ f_1, f_2 \})} \sum_{i=1}^{m} \min\{ c_i \cdot \chi\left(f_1\right), c_i \cdot \chi\left(f_2\right) \} \\
&\leq (1-\epsilon) \sum_{i=1}^{m} \max_{\text{Partition } P=(V,\{ f_1, f_2 \})} \min\{ c_i \cdot \chi\left(f_1\right), c_i \cdot \chi\left(f_2\right) \} \\
&= (1-\epsilon) \sum_{i=1}^{m} c_{opt}^i \\
&\leq \sum_{i=1}^{m} c_\epsilon^i \\
&= \sum_{i=1}^{m} \min\{ c_i \cdot \chi\left(f_1\right), c_i \cdot \chi\left(f_2\right) \},
\end{aligned}$$

das heißt P ist eine ϵ-approximative Lösung der Instanz von `HYPERGRAPH MAX CUT`, wobei H ein Baum war. Es bleibt nachzuweisen, daß der vorgeschlagene Algorithmus polynomial in der Codierungslänge der Instanz und $\frac{1}{\epsilon}$ ist. Dies ist jedoch klar, da im wesentlichen nur m mal der Algorithmus von Ibarra und Kim aufgerufen wird sowie m Blätter des Hypergraphen H bestimmt werden, was in polynomialer Zeit ebenso möglich ist wie das jeweilige Erweitern der Partition P. □

Praktische Anwendungen von (4) werden zwar wohl die Ausnahme sein, immerhin mag es aber in einigen Fällen, etwa wenn die verschiedenen Platinentypen nur wenige Bauteile gemeinsam haben, angehen, einfach festzulegen, daß diese auf jeden Fall von nur einem Automaten bestückt werden und so den Hypergraphen H des ursprünglichen Problems zu einem Baum zu kontrahieren.

3.3 Ein Algorithmus

In diesem Abschnitt wollen wir Fälle untersuchen, bei denen für das Optimierungsproblem `HYPERGRAPH MAX CUT` gewisse Einschränkungen in Bezug auf die Anzahl der Kanten des Hypergraphen $H = (V, E)$ beziehungsweise gleichbedeutend in Bezug auf die Anzahl an Zeilen der Kostenmatrix C gestellt werden. Wir wollen in diesem Fall einen einfachen approximativen Algorithmus entwickeln. Dieser Algorithmus wird auch dann funktionieren, wenn die getroffenen Einschränkungen nicht gelten, jedoch wird dann die Laufzeit stark anwachsen. Wir schicken ein Hilfsresultat voraus.

3.20 Beobachtung. Gegeben seien rationale Zahlen a und b und rationale Zahlen $0 \leq \epsilon, \eta, \theta$. Gelten dann für rationale Zahlen α, β und γ die Beziehungen

$$\begin{gathered} a - \epsilon \leq \alpha \leq a + \epsilon \\ b - \eta \leq \beta \leq b + \eta \\ \alpha + \beta - \theta \leq \gamma \leq \alpha + \beta + \theta, \end{gathered}$$

so folgt

$$a + b - (\theta + \eta + \epsilon) \leq \gamma \leq a + b + (\theta + \eta + \epsilon).$$

□

Beobachtung 3.20 kann folgendermaßen interpretiert werden. Gegeben zwei Resultate von Berechnungen, die garantiert eine gewisse Genauigkeit haben, so können wir diese Addieren und dabei noch einen gewissen Fehler machen und erhalten ein Ergebnis von genau bestimmter Güte.
Vergleichen wir mit dem Optimierungsproblem `HYPERGRAPH MAX CUT`. Im wesentlichen geht es dort darum, für verschiedene Partitionen $P = \big(V, \{ f_1, f_2 \}\big)$ Zeilensummen $c_{i\cdot}\chi(f_1)$ und $c_{i\cdot}\chi(f_2)$ zu berechnen und dann nach einem gewissen Schema aus diesen eine Gesamtsumme abzuleiten; die Partition mit der maximalen Gesamtsumme ist die beste Lösung. Wir wollen dieses Problem so angehen, daß wir einfach alle möglichen Zeilensummen bestimmen und für jede Möglichkeit die gesuchte Zielfunktion berechnen; unser bestes Ergebnis nehmen wir als Lösung. Selbstverständlich ist die maximal mögliche Anzahl an Möglichkeiten, die Einträge in den Zeilen der Kostenmatrix C aufzuaddieren, 2^n. Da wir aber nur eine approximative Lösung anstreben, brauchen wir uns nicht alle Ergebnisse zu merken, sondern unter Verwendung von Beobachtung 3.20 nur gewisse "charakteristische".

3.21 Algorithmus.

Input. Natürliche Zahl k.
Instanz des Optimierungsproblems `HYPERGRAPH MAX CUT` durch einen Hypergraphen $H = \big(V, \{ e_1, \ldots, e_m \}\big)$ und eine Kostenmatrix C mit der Eigenschaft $\log_2 n \in \mathbb{N}$.

Output. Matrix $A \in \{ 0, 1 \}^{((k+1)^m)}$ mit den Eigenschaften

$$\begin{gathered} \forall \text{ Partition } P = \big(V, \{ f_1, f_2 \}\big) : \exists r \in \{ 0, \ldots, k \}^m : \\ c_{l\cdot}\chi(f_1) - \tfrac{\log_2 n}{k} c_{l\cdot}\mathbb{1} \leq \tfrac{r_l}{k} c_{l\cdot}\mathbb{1} \leq c_{l\cdot}\chi(f_1) + \tfrac{\log_2 n}{k} c_{l\cdot}\mathbb{1} \quad \forall l = 1, \ldots, m \\ \text{und} \\ A(r) = 1 \end{gathered} \tag{5}$$

sung(5)und

$$\begin{gathered} \forall r \in \{ 0, \ldots, k \}^m : \mathrm{A(r)}{=}1 \\ \Longrightarrow \qquad \exists \text{ Partition } P = \big(\{ 1, \ldots, n \}, \{ f_1, f_2 \}\big) : \\ c_{l\cdot}\chi(f_1) - \tfrac{\log_2 n}{k} c_{l\cdot}\mathbb{1} \leq \tfrac{r_l}{k} c_{l\cdot}\mathbb{1} \leq c_{l\cdot}\chi(f_1) + \tfrac{\log_2 n}{k} c_{l\cdot}\mathbb{1} \quad \forall l = 1, \ldots, m. \end{gathered} \tag{6}$$

Datenstrukturen. Matrizen $A_j^i \in \{0,1\}^{((k+1)^m)}$ für i aus $\{0,\ldots,\log_2 n\}$ und j aus $\{1,\ldots,2^{\log_2 n - i}\}$. Matrizen $B_j^i \in \{\{0,\ldots,k\}\times\{0,\ldots,k\}\}^{((k+1)^m)}$ für i aus $\{1,\ldots,\log_2 n\}$ und j aus $\{1,\ldots,2^{\log_2 n - i}\}$. Hypergraphen H^i mit $2^{\log_2 n - i}$ Kanten $e_1^i,\ldots,e_{2^{\log_2 n - i}}^i$ und n Knoten für i aus $\{0,\ldots,\log_2 n\}$.

begin
{ Initialisierung }
Setze $H_0 := \Big(\{1,\ldots,n\},\{\{i\}\}_{i=1}^n\Big)$.
for all j aus $\{1,\ldots,2^n\}$ und p aus $\{0,\ldots,k\}^m$
Setze $A_j^0(p) := 1$, falls $p = 0$ oder $p = k\mathbb{1}$ und Null sonst.
for all i aus $\{1,\ldots,\log_2 n\}$ und j aus $\{1,\ldots,2^{\log_2 n - i}\}$
Setze $A_j^i := 0$.

{ Hauptschleife }
for $i := 1$ **to** $\log_2 n$ **do**
begin
Konstruiere $H^i = \Big(\{1,\ldots,n\},\{e_1^i,\ldots,e_{2^{\log_2 n - i}}^i\}\Big)$, indem immer je zwei Kanten $e_{j'}^{i-1}$ und $e_{j''}^{i-1}$ von H^{i-1} zu einer Kante e_j^i von H^i vereinigt werden.
for $j := 1$ **to** $2^{\log_2 n - i}$ **do**
for all p und q aus $\{0,\ldots,k\}^m$ **do**
begin
if $A_{j'}^{i-1}(p) = 1$ **and** $A_{j''}^{i-1}(q) = 1$ **then**
begin
for $l := 1$ **to** m **do**
Setze $r_l := \left\lceil k \cdot \left(\frac{p_l}{k} c_{l\cdot}\chi\left(e_{j'}^{i-1}\right) + \frac{q_l}{k} c_{l\cdot}\chi\left(e_{j''}^{i-1}\right)\right) \Big/ c_{l\cdot}\chi\left(e_j^i\right) - \frac{1}{2}\right\rceil$.
Setze $A_j^i(r) := 1$.
Setze $B_j^i(r) = (p,q)^t$.
end;
end;
end;

{ Ausgabe }
Gib $A_1^{\log_2 n}$ aus.
end.

□

3.22 Hilfssatz. Algorithmus 3.21 funktioniert. Seine Laufzeit ist bei geeigneter Implementierung der nicht näher spezifizierten Schritte und Datenstrukturen

$$O\Big(\log_2 n n (k^m)^2 m\big(\langle k\rangle + \log_2 n \max_{i,j}\langle c_{ij}\rangle\big).\Big)$$

Beweis. Es ist leicht zu sehen, daß für die Hypergraphen H^i für alle i aus $\{0,\ldots,\log_2 n\}$ die folgenden Beziehungen gelten.

a. H^i besitzt genau $2^{\log_2 n - i}$ Kanten e_j^i.
b. $\left|e_j^i\right| = 2^i$ für alle j aus $\{1,\ldots,2^{\log_2 n - i}\}$.
c. $\bigcup_{j=1}^{2^{\log_2 n - i}} e_j^i = \{1,\ldots,n\}$.
d. $e_{j'}^i \cap e_{j''}^i = \emptyset$ für alle $j' \neq j''$.

Wir zeigen jetzt durch Induktion über i die Gültigkeit der folgenden Aussagen für jeweils alle Indizes j aus der Menge $\{1,\ldots,2^{\log_2 n - i}\}$.

1.
$$\forall \text{ Partition } P = \left(V, \{ f_1, f_2 \}\right) : \exists r \in \{ 0, \ldots, k \}^m :$$
$$c_l.\chi\left(f_1 \cap e_j^i\right) - \tfrac{i}{k} c_l.\chi\left(e_j^i\right) \leq \tfrac{r_l}{k} c_l.\chi\left(e_j^i\right) \leq c_l.\chi\left(f_1 \cap e_j^i\right) + \tfrac{i}{k} c_l.\chi\left(e_j^i\right) \quad \forall l = 1, \ldots, m$$
und
$$A_j^i(r) = 1.$$

2.
$$\forall r \in \{ 0, \ldots, k \}^m : A_j^i(r) = 1$$
$$\Longrightarrow \quad \exists \text{ Partition } P = \left(\{ 1, \ldots, n \}, \{ f_1, f_2 \}\right) :$$
$$c_l.\chi\left(f_1 \cap e_j^i\right) - \tfrac{i}{k} c_l.\chi\left(e_j^i\right) \leq \tfrac{r_l}{k} c_l.\mathbb{1} \leq c_l.\chi\left(f_1 \cap e_j^i\right) + \tfrac{i}{k} c_l.\chi\left(e_j^i\right) \quad \forall l = 1, \ldots, m.$$

Für $i = \log_2 n$ ist dies dann aufgrund von $e_1^{\log_2 n} = \{ 1, \ldots, n \}$ die Behauptung.
Betrachten wir den Induktionsanfang $i = 0$. Sei zunächst eine Partition $P = \left(V, \{ f_1, f_2 \}\right)$ gegeben und sei j aus $\left\{ 1, \ldots, 2^{\log_2 n - 0} \right\}$. Enthält f_1 dann j, so erfüllt $r = k\mathbb{1}$ die Beziehung 1, denn nach Definition von H^0 ist $e_j^0 = \{ j \}$ und also

$$c_l.\chi\left(f_1 \cap e_j^i\right) - \frac{0}{k} c_l.\chi\left(e_j^i\right) = c_{lj} \leq \frac{k}{k} c_{lj} \leq c_{lj} = c_l.\chi\left(f_1 \cap e_j^i\right) + \frac{0}{k} c_l.\chi\left(e_j^i\right). \tag{7}$$

Analog zeigt man, daß im Fall, daß j nicht in der Kante f_1 enthalten ist, $r = 0$ die Beziehung 1 erfüllt. Ist umgekehrt $A_j^0(r)$ für ein j aus $\left\{ 1, \ldots, 2^{\log_2 n - 0} \right\}$ und r aus $\{ 0, \ldots, k \}^m$ verschieden von Null, so trifft nach Definition von A^0 entweder $r = k\mathbb{1}$ oder $r = 0$ zu. Im ersten Fall erfüllt aber jede Partition $P = \left(\{ 1, \ldots, n \}, \{ f_1, f_2 \}\right)$, die j in f_1 enthält, die Aussage 2, wie die Betrachtung von (7) zeigt. Analog geht der zweite Fall.
Gelte also die Behauptung für alle i' kleiner als das nunmehr betrachtete i. Wir beginnen wieder mit Aussage 1. Sei eine Partition $P = \left(V, \{ f_1, f_2 \}\right)$ und ein beliebiges j aus $\left\{ 1, \ldots, 2^{\log_2 n - i} \right\}$ gegeben. Nach Induktionsannahme gibt es dann p und q, so daß die Aussage von 1 für $i - 1$ und Indizes j' und j'' zutrifft, das heißt, für r wie im Algorithmus komponentenweise als

$$r_l := \left\lceil k \cdot \left(\frac{p_l}{k} c_l.\chi\left(e_{j'}^{i-1}\right) + \frac{q_l}{k} c_l.\chi\left(e_{j''}^{i-1}\right) \right) \Big/ c_l.\chi\left(e_j^i\right) - \tfrac{1}{2} \right\rceil$$

für alle l aus $\{ 1, \ldots, m \}$ definiert, p und q gelten die Beziehungen

$$\begin{aligned}
c_l.\chi\left(f_1 \cap e_{j'}^{i-1}\right) - \tfrac{(i-1)}{k} c_l.\chi\left(e_{j'}^{i-1}\right) &\leq \tfrac{p_l}{k} c_l.\chi\left(e_{j'}^{i-1}\right) \\
&\leq c_l.\chi\left(f_1 \cap e_{j'}^{i-1}\right) + \tfrac{(i-1)}{k} c_l.\chi\left(e_{j'}^{i-1}\right) \\
c_l.\chi\left(f_1 \cap e_{j''}^{i-1}\right) - \tfrac{(i-1)}{k} c_l.\chi\left(e_{j''}^{i-1}\right) &\leq \tfrac{q_l}{k} c_l.\chi\left(e_{j''}^{i-1}\right) \\
&\leq c_l.\chi\left(f_1 \cap e_{j''}^{i-1}\right) + \tfrac{(i-1)}{k} c_l.\chi\left(e_{j''}^{i-1}\right) \\
\tfrac{p_l}{k} c_l.\chi\left(e_{j'}^{i-1}\right) + \tfrac{q_l}{k} c_l.\chi\left(e_{j''}^{i-1}\right) & \\
-\tfrac{1}{k} c_l.\left(\chi\left(e_{j'}^{i-1}\right) + \chi\left(e_{j''}^{i-1}\right)\right) &\leq \tfrac{r_l}{k} c_l.\chi\left(e_j^i\right) \\
&\leq \tfrac{p_l}{k} c_l.\chi\left(e_{j'}^{i-1}\right) + \tfrac{p_l}{k} c_l.\chi\left(e_{j''}^{i-1}\right) \\
&\quad + \tfrac{1}{k} c_l.\left(\chi\left(e_{j'}^{i-1}\right) + \chi\left(e_{j''}^{i-1}\right)\right)
\end{aligned} \tag{8}$$

für jedes l aus $\{ 1, \ldots, m \}$, so daß Beobachtung 3.20 wegen $\chi\left(e_{j'}^{i-1}\right) + \chi\left(e_{j''}^{i-1}\right) = \chi\left(e_j^i\right)$ die Behauptung liefert.
Ist umgekehrt für beliebiges j aus $\left\{ 1, \ldots, 2^{\log_2 n - i} \right\}$ ein r aus $\{ 0, \ldots, k \}^m$ mit $A_j^i(r) = 1$ gegeben, so muß r in der Hauptschleife von Algorithmus 3.21 für i mit dem nunmehr interessierenden Wert auf 1 gesetzt worden sein. Seien p und q dann solche Vektoren aus $\{ 0, \ldots, k \}^m$, daß ebendieses bei ihrer Betrachtung von Algorithmus 3.21 getan wurde; möglich sind insbesondere p und q mit $B_j^i(r) = (p, q)^t$. Dann trifft

für gewisse Indizes j' und j'' $A^{i-1}_{j'}(p) = A^{i-1}_{j''}(q) = 1$ zu. Nach Induktionsannahme gibt jetzt Partitionen $P^p = \left(\{1,\ldots,n\}, \{f^p_1, f^p_2\}\right)$ und $P^q = \left(\{1,\ldots,n\}, \{f^q_1, f^q_2\}\right)$, so daß Aussage 2 für $i-1$, j', p und P_p sowie für $i-1$, j'', q und P_q zutrifft. Mit der Partition

$$P = \left(\{1,\ldots,n\}, \{f_1 := \left(f^p_1 \cap e^{i-1}_{j'}\right) \cup \left(f^q_1 \cap e^{i-1}_{j''}\right), f_2 := \{1,\ldots,n\} \setminus f_1\}\right) \tag{9}$$

zeigt dann aber die Betrachtung von (8) die Aussage 2, denn die Mengen $e^{i-1}_{j'}$ und $e^{i-1}_{j''}$ sind disjunkt.
Die Behauptung bezüglich der Laufzeit ergibt sich direkt aus der Betrachtung des Algorithmus; es sei lediglich bemerkt, daß die Addition von n Zahlen einer Codierungslänge von $\max\langle c_{ij}\rangle$, wie sie bei der Berechnung der Zahlen r_l vorkommt, unter Verbrauch von höchstens $\log_2 n \max\langle c_{ij}\rangle$ Einheiten Zeit und Speicherplatz erfolgen kann. □

Ausdrücklich sei noch bemerkt, daß (9) in einfacher Weise eine Konstruktion einer Partition P, die (6) erfüllt, in $O(n \log_2 n)$ Zeit (es ist sogar leicht $O(n)$ Zeit möglich) ermöglicht, wenn bei der Berechnung von r die beiden "Väter" p und q von r gemerkt werden, wie wir das in den Matrizen B getan haben.
Algorithmus 3.21 muß nun noch ein wenig "getunt" werden, damit er zu einem (theoretisch) einigermaßen brauchbaren approximativen Verfahren wird. Kein Problem ist die Forderung, daß $\log_2 n$ eine natürliche Zahl sein soll. Diese wurde nur zur glatteren Darstellung von Algorithmus 3.21 aufgestellt und kann leicht durch Ergänzen von C mit $2^{\lceil \log_2 n \rceil} - n$ Nullspalten erfüllt werden. Einen Zielfunktionswert einer Lösung von `HYPERGRAPH MAX CUT` erhalten wir aus dem Output von Algorithmus 3.21, indem wir die sich in den einzelnen Zeilen ergebenden Resultate untersuchen und dann geeignet addieren. Den unangenehmen Term $(k^m)^2$ in der Laufzeitgarantie, der Algorithmus 3.21 als leider nicht polynomial kennzeichnet, "entschärfen" wir, wie bereits angekündigt, indem wir nur Instanzen der Optimierungsversion von `HYPERGRAPH MAX CUT` betrachten, in der die Anzahl der Kanten des Hypergraphen H, also m, gewissen Bedingungen unterliegt, also etwa sagen wir nur k beträgt. Bleibt die Frage der Gütegarantie, die bis jetzt auf die Terme $c_l.\mathbb{1}$ für l aus $\{1,\ldots,m\}$, also auf die Zeilensummen bezogen ist, wohingegen wir doch eine auf eine Optimallösung bezogene Garantie aufstellen wollen. Wir untersuchen nun das letztgenannte Problem.

3.23 Beobachtung. Sei eine Instanz des Optimierungsproblems `HYPERGRAPH MAX CUT` durch einen Hypergraphen $H = (V, E)$ und eine Matrix C gegeben. Dann gelten die folgenden Aussagen.

1.
$$\begin{aligned} &\max_{i=1}^{m} \max_{\text{Partition } P=\left(V,\{f_1,f_2\}\right)} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} \\ \leq\ &\max_{\text{Partition } P=\left(V,\{f_1,f_2\}\right)} \sum_{i=1}^{m} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} \\ \leq\ &m \cdot \max_{i=1}^{m} \max_{\text{Partition } P=\left(V,\{f_1,f_2\}\right)} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\}. \end{aligned}$$

2. Gibt es ein i' aus $\{1,\ldots,m\}$ und ein j' aus $\{1,\ldots,n\}$, so daß $c_{i'j'} > \sum_{j\neq j'} c_{i'j}$ zutrifft, so gilt mit der Matrix C', definiert durch

$$c'_{ij} := \begin{cases} c_{ij} & \text{, falls } i \neq i' \text{ oder } j \neq j' \\ \sum_{j\neq j'} c_{i'j} & \text{, falls } i = i' \text{ und } j = j' \end{cases}$$

für jede Partition $P = \left(V, \{f_1, f_2\}\right)$

$$\sum_{i=1}^{m} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} = \sum_{i=1}^{m} \min\{c'_{i\cdot}\chi(f_1), c'_{i\cdot}\chi(f_2)\}.$$

3. Gilt für alle i aus $\{1,\ldots,m\}$ und alle j aus $\{1,\ldots,n\}$ stets $\sum_{k\neq j} c_{ik} \geq c_{ij}$, so gilt auch für alle i aus $\{1,\ldots,m\}$ immer
$$\max_{\text{Partition } P=\left(V,\{f_1,f_2\}\right)} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} \geq \tfrac{1}{3} c_{i\cdot}\mathbb{1}.$$

4. Unter den Voraussetzungen von 3 gilt

$$\frac{1}{3m}\sum_{i=1}^{m} c_{i\cdot}\mathbb{1} \leq \tfrac{1}{3}\max_{i=1}^{m} c_{i\cdot}\mathbb{1} \leq \max_{\text{Partition } P=\left(V,\{f_1,f_2\}\right)} \sum_{i=1}^{m} \min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} \leq \sum_{i=1}^{m} c_{i\cdot}\frac{\mathbb{1}}{2}.$$

Beweis. Nur Aussage 3 bedarf eines Beweises. Sei also i aus $\{1,\ldots,m\}$ gegeben. Angenommen, es gibt ein j' mit $\frac{1}{3}\sum_{j\neq j'} c_{ij'} \leq c_{ij'} \leq \frac{1}{2}\sum_{j\neq j'} c_{ij'}$, dann ist

$$\left(\{1,\ldots,n\},\{f_1,f_2\}\right) := \left(\{1,\ldots,n\},\{\{j'\},\{j : j\neq j'\}\}\right)$$

eine Partition mit der Eigenschaft $\min\{c_{i\cdot}\chi(f_1), c_{i\cdot}\chi(f_2)\} \geq \frac{1}{3}c_{i\cdot}\mathbb{1}$. Im anderen Falle besitzt aber mit $k := \min\{k \in \{1,\ldots,n\} : \sum_{j=1}^{k} c_{ij} \geq \frac{1}{3}c_{i\cdot}\mathbb{1}\}$ die Partition

$$\left(\{1,\ldots,n\},\{\{1,\ldots,k\},\{k+1,\ldots,n\}\}\right)$$

diese Eigenschaft. □

Beobachtung 3.23 zeigt, daß unter gewissen Umständen, die durch ein "Preprocessing" gemäß 2 immer erreicht werden können, eine Beziehung zwischen der Optimallösung einer Instanz von `HYPERGRAPH MAX CUT` und der Summe aller Einträge in der Kostenmatrix besteht. Die Beziehung basiert im wesentlichen auf der trivialen Garantie Beobachtung 3.23.1. Da der Autor jedoch nicht in der Lage war, einen approximativen Algorithmus, der den Parameter m durch eine Konstante ersetzt hätte, zu finden, muß er sich leider auf die Anwendung von Beobachtung 3.23.1 beschränken.
Falls der Leser sich gefragt hat, warum der Autor die Probleme `WEIGHTED SET SPLITTING` und `HYPERGRAPH MAX CUT` als Maximierungs- und nicht als Minimierungsprobleme formuliert hat, was doch in Anbetracht ihrer Herleitung aus einem Minimierungsproblem vielleicht natürlicher gewesen wäre, so mag er hier eine Antwort finden (falls er nicht wie der Autor der Meinung ist, daß eine Maximierungsversion "natürlicher" ist). Beobachtung 3.23.2 ist nämlich nur bei Betrachtung der Maximierungsversion von `HYPERGRAPH MAX CUT` in der angegebenen Weise formulierbar.
Nach diesen Vorbereitungen sind wir nun in der Lage, Algorithmus 3.21 so zu verbessern, daß er einsatzfähig wird.

3.24 Algorithmus. $\frac{1}{k}$-approximative Lösung von `HYPERGRAPH MAX CUT`, eingeschränkt auf Instanzen mit höchstens k Kanten im Hypergraphen H.

Input. Natürliche Zahl k.
Instanz von `HYPERGRAPH MAX CUT` durch einen Hypergraphen $H = (V, E)$ mit höchstens k Kanten und eine Matrix C mit entsprechend höchstens k Zeilen.

Output. $\frac{1}{k}$-approximative Lösung der Instanz.

Datenstrukturen. Matrix C' aus $\mathbb{N}_0^{m\times 2^{\lceil\log_2 n\rceil}}$. Ansonsten wie bei Algorithmus 3.21, lediglich das dortige k wird ersetzt durch $k \cdot 3k\lceil\log_2 n\rceil$.

begin
{ Preprocessing }
Setze $C' := (C|0)$.
Setze $H' := \left(V \cup \{n+1,\ldots,\lceil\log_2 n\rceil\}, E\right)$.
for $i := 1$ **to** m **do**
if $\exists j' \in \{1,\ldots,n\} : \sum_{j\neq j'} c'_{ij} < c'_{ij'}$ **then**
Setze $c'_{ij'} := \sum_{j\neq j'} c'_{ij}$.

{ Berechnung der benötigten Genauigkeit }
Setze $k' := k \cdot 3k\lceil\log_2 n\rceil$.

{ Aufruf von Algorithmus 3.21 }
Rufe Algorithmus 3.21 mit Eingabeparametern k', H' und C' auf.

{ Rückwärtsrechnung }.
Setze max := 0.
for all r aus $\{0,\dots,k'\}^m : A(r) = 1$ **do**
  if $\sum_{i=1}^m \min\left\{ \frac{r_i}{k'} c_i.\mathbb{1}, \frac{(k'-r_i)}{k'} c_i.\mathbb{1} \right\} > \max$ **then**
    Setze max := $\sum_{i=1}^m \min\left\{ \frac{r_i}{k'} c_i.\mathbb{1}, \frac{(k'-r_i)}{k'} c_i.\mathbb{1} \right\}$.
Für das r aus $\{0,\dots,k'\}^m : A(r) = 1$, bei dessen Betrachtung sich max ergab, bestimme rekursiv gemäß (9) und der Bemerkung nach (9) eine Partition $P = \big(V, \{f_1, f_2\}\big)$.

{ Ausgabe }
Gib $\big(\{1,\dots,n\}, \{f_1 \cap \{1,\dots,n\}, f_2 \cap \{1,\dots,n\}\}\big)$ aus.
end.

□

3.25 Satz. Algorithmus 3.24 funktioniert. Seine Laufzeit beträgt bei geeigneter Implementation der nicht näher spezifizierten Schritte und Datenstrukturen

$$O\Big(\lceil \log_2 n \rceil n \Big(k'^{k'}\Big)^2 k' \big(\langle k' \rangle + \log_2 n \max_{i,j} \langle c_{ij} \rangle\big)\Big).$$

□

3.26 Beispiel. Wir wollen wegen des formalen Aufwands nur den wesentlichen Ablauf von Algorithmus 3.24 an einem (bereits aufwendigen) Minibeispiel erläutern und betrachten deshalb eine Instanz des Optimierungsproblems `HYPERGRAPH MAX CUT`, in dem sich ein Preprocessing erübrigt, gegeben durch die Kostenmatrix C und den Hypergraphen H

$$C := \begin{pmatrix} 10 & 20 & 30 & 40 \\ 10 & 0 & 0 & 10 \end{pmatrix} \quad \text{und} \quad \Big(\{1,2,3,4\}, \{\{1,2,3,4\}, \{1,4\}\}\Big);$$

als k wollen wir 1 wählen, so daß wir eine 1-approximative Lösung suchen. Wir berechnen

$$k' := 2 \cdot 3 \cdot 2\lceil \log_2 4 \rceil = 6,$$

wir wollen zur Vereinfachung sogar 10 wählen. Wir erhalten

$$H^0 := \Big(\{1,2,3,4\}, \{\{1\}, \{2\}, \{3\}, \{4\}\}\Big),$$

betrachten also zunächst nur einzelne Spalten der Matrix C. In einer Partition $P = \big(\{1,2,3,4\}, \{f_1, f_2\}\big)$ von $\{1,2,3,4\}$ kann sich jeder Knoten entweder in f_1 befinden oder nicht, entsprechend befindet sich in jeder Spalte sein Gewicht in jeder Zeile gleichzeitig entweder zu $\frac{0}{10}$ oder $\frac{10}{10}$ auf der Seite von f_1, geht also entsprechend in $c_l.\chi(f_1)$, $l = 1,2$ ein. Diesen Tatbestand merken wir uns, indem wir a_j^0 für jede Spalte $j \in \{1,\dots,4\} = \{1,\dots,2^{\log_2 4 - 0}\}$ durch

$$A_j^0(p) := \begin{cases} 1 & \text{, falls } p = (0,0)^t \text{ oder } p = (1,1)^t \\ 0 & \text{, sonst} \end{cases}$$

festsetzen.
Jetzt fassen wir immer zwei Spalten zusammen, sagen wir 1 und 4 sowie 2 und 3, so daß sich

$$H^1 := \Big(\{1,2,3,4\}, \{\{1,4\}, \{2,3\}\}\Big)$$

ergibt. Betrachten wir die Spalten 1 und 4. Es sind insgesamt 4 Möglichkeiten der Verteilung von 1 und 4 auf die Kanten einer Partition P von $\{1,2,3,4\}$ denkbar. Entsprechend ergeben sich als Summe der Gewichte der möglicherweise in f_1 enthaltenen Spaltenindizes

$$(0,0),(10,10),(40,10) \quad \text{und} \quad (50,20)$$

in (Zeile 1, Zeile 2), die Gewichtsummen in den Spalten 1 und 4 sind entsprechend $(50,20)$. Wir unterteilen jetzt einfach die Intervalle $[0,50]$ und $[0,20]$ durch 11 Punkte in $k'=10$ gleich große Teile und schauen nach, zu welchem der Teilpunkte jede der oben berechneten möglichen 4 Gewichtssummen jeweils am nächsten liegt.

$$0,10,40 \quad \text{und} \quad 50$$

liegen jeweils am nächsten bei

$$0\frac{50}{10}, 2\frac{50}{10}, 8\frac{50}{10} \text{ und } 10\frac{50}{10},$$

was der Algorithmus zum Beispiel für 40 durch Berechnung von

$$\left\lceil \frac{0+40}{50}10 - 0,5 \right\rceil = \lceil 7,5 \rceil = 8$$

ermittelt. Analog ermitteln wir für die in der zweiten Zeile möglichen Ergebnisse

$$0\frac{20}{10}, 5\frac{20}{10}, 5\frac{20}{10} \text{ und } 10\frac{20}{10}.$$

Wir merken uns dies in der Matrix A_1^1, die wir als

$$A_1^1 := \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{pmatrix} 1 & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & 1 & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & 1 & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & 1 \end{pmatrix} \end{array}$$

mit Nullen and den jeweils freien Stellen festsetzen. Völlig analog kommen wir für die Menge von Spaltenindizes $\{2,3\}$ zu einer Matrix

$$A_2^1 := \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{pmatrix} 1 & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ 1 & & & & & & & & & & \\ & & & & & & & & & & \\ 1 & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ & & & & & & & & & & \\ 1 & & & & & & & & & & \end{pmatrix} \end{array}.$$

Man beachte, daß bei einer möglichen Gesamtsumme von Null in der zweiten Zeile für die Spaltenindizes 2 und 3 die tatsächlich der Menge f_1 einer Partition zuzuordnende Summe in der zweiten Zeile von ebenfalls Null als ein beliebiger Bruchteil der maximal möglichen Summe angesehen werden kann, sich der Algorithmus aber dafür entscheidet, sie als den nullten Teil zu betrachten.
Im letzten Schritt betrachten wir alle Spalten gleichzeitig und erhalten

$$H^2 := \big(\{\,1,2,3,4\,\},\{\,1,2,3,4\,\}\big).$$

Als Zeilensummen ergeben sich $(100, 20)$. Betrachten wir alle Möglichkeiten, die (hier nicht) gerundeten Teilsummen für die disjunkten Spaltenmengen des ersten Schrittes zu addieren. Bei ausschließlicher Betrachtung der durch die Matizen A_1^1 und A_2^1 codierten legalen Kombinationsmöglichkeiten ergeben sich die folgenden Summen der (hier nicht) gerundeten Zwischenergebnisse.

$+$	$0\cdot 50/10$	$4\cdot 50/10$	$6\cdot 50/10$	$10\cdot 50/10$
$0\cdot 50/10$	0	20	30	50
$2\cdot 50/10$	10	30	40	60
$8\cdot 50/10$	40	60	70	90
$10\cdot 50/10$	50	70	80	100

in der ersten und

$+$	$0\cdot 20/10$	$0\cdot 20/10$	$0\cdot 20/10$	$0\cdot 20/10$
$0\cdot 20/10$	0	0	0	0
$5\cdot 20/10$	10	10	10	10
$5\cdot 20/10$	10	10	10	10
$10\cdot 20/10$	20	20	20	20

für die zweite Zeile. Diese (hier nicht) gerundeten Zwischenergebnisse werden nun analog zum ersten Schritt erneut bezogen auf die Zeilensummen von 100 und 20 zum nächsten Punkt $i\frac{100}{10}$ beziehungsweise $i\frac{20}{10}$ für i aus $\{\,0,\ldots,10\,\}$ gerundet; die Ergebnisse halten wir nun für alle Spalten in der Matrix A_1^2 fest (man beachte, daß unterschiedliche Ergebnisse zusammenfallen).

$$A_2^1 := \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \left(\begin{array}{ccccccccccc} 1 & & & & & & & & & & \\ & & & & & 1 & & & & & \\ 1 & & & & & & & & & & \\ 1 & & & & & 1 & & & & & \\ & & & & & 1 & & & & & \\ 1 & & & & & & & & & & 1 \\ & & & & & 1 & & & & & 1 \\ & & & & & 1 & & & & & 1 \\ & & & & & & & & & & \\ & & & & & 1 & & & & & \\ & & & & & & & & & & 1 \end{array}\right) \end{array}.$$

Ihr Eintrag $A_1^2(i,j)$ gibt für alle Indizes i und j aus der Menge $\{\,0,\ldots,10\,\}$ jeweils an, ob es eine Partition $P = \big(\{\,1,2,3,4\,\},\{\,f_1,f_2\,\}\big)$ von $\{\,1,2,3,4\,\}$ mit

$$c_1{\cdot}\chi(f_1) - 1c_1.\mathbb{1} \le \frac{i}{10}c_1.\mathbb{1} \le c_1{\cdot}\chi(f_1) + 1c_1.\mathbb{1}$$
$$c_2{\cdot}\chi(f_1) - 1c_2.\mathbb{1} \le \frac{i}{10}c_2.\mathbb{1} \le c_2{\cdot}\chi(f_1) + 1c_2.\mathbb{1}$$

gibt, entsprechend gilt natürlich

$$c_1{\cdot}\chi(f_2) - 1c_1.\mathbb{1} \le \frac{10-i}{10}c_2.\mathbb{1} \le c_1{\cdot}\chi(f_2) + 1c_1.\mathbb{1}$$
$$c_2{\cdot}\chi(f_2) - 1c_2.\mathbb{1} \le \frac{10-i}{10}c_2.\mathbb{1} \le c_2{\cdot}\chi(f_2) + 1c_2.\mathbb{1}.$$

A_1^2 versichert uns daher, daß es Partitionen gibt, deren Zielfunktionswerte (bei Abwandern von A_2^1 von links oben nach rechts unten) "ungefähr"

$$0, 20, 20, 30, 40, 50, 50, 50, 50, 40, 40, 30, 20, 0$$

sind. Den besten derartigen Wert, nämlich 50 nehmen wir. Mit Rückwärtsrechnung könnten wir etwa die Partition $\Big(\{ 1, 2, 3, 4 \}, \{ \{ 1, 4 \}, \{ 2, 3 \} \} \Big)$ als eine mögliche Lösung finden.

Um keine Mißverständnisse aufkommen zu lassen, möchte der Autor klarstellen, daß die Matrizen A_i^j, die im Beispiel ziemlich leer sind, bei einigermaßen realistischen Größenordnungen sehr schnell voll werden. Es findet dann eine Reduktion der Menge der möglichen Zwischenergebnisse statt, indem immer mehrere Ergebnisse auf denselben Matrixeintrag abgebildet werden. Unglücklicherweise sind jedoch die Beispiele, wo dieser Effekt auftritt, entweder von so trivialer Gütegarantie oder so groß, daß der Autor es vorzieht, auf die Vorstellungskraft des Lesers zu vertrauen. □

Der Autor will nicht mutmaßen, ob Algorithmus 3.24 von praktischer Relvanz ist oder nicht, jedenfalls ist er sehr einfach. Unmittelbar erkennbar ist außerdem, daß er noch viel Raum für Implementationstricks läßt. So wird man natürlich in der Praxis selbstverständlich nicht alle Möglichkeiten für die dort bezeichneten Größen p und q prüfen, um die Größe r zuberechnen, sondern nur "aussichtsreiche". Weiterhin ist auch nicht einzusehen, warum man in der dort angegebenen Weise die Matrizen berechnen soll. Cleverer scheint es zu sein, schnell in die "Tiefe" der Summation vorzustoßen und einige Summen schnell auszurechnen. Dann kann man sehen, ob man sich bei weiterer Rechnung verbessert. Tut sich nichts, kann man aufhören. Wegen der Einfachheit und der Anpassungsfähigkeit von Algorithmus 3.24 hält der Autor ihn für nicht völlig ungeeignet, das Optimierungsproblem `HYPERGRAPH MAX CUT` anzugehen.

4 Nachwort

Obwohl die vorliegende Arbeit möglicherweise einige in Bezug auf die vorgestellte Problematik interessante Punkte beleuchten konnte, so mußten doch wichtige Fragen offen bleiben.

1. Wie kann man das Problem der Mehrfachzuordnung von Bauteilen zu verschiedenen Automaten im vorgestellten Produktionsmodell in geschickter Weise modellieren?
2. Wie realisiert man interessante Nebenbedingungen, etwa, daß auf allen Automaten gleich viele Bauteile zu plazieren sind? Ein Hauruck-Ansatz wäre, auf alle für die Bestückung mit Bauteilen benötigten Zeiten eine große Konstante zu addieren, aber das würde natürlich der Güte einer approximativen Lösung nicht gut bekommen.
3. Ist `Three Processor Assembly Line Scheduling` $\mathcal{NP}$-vollständig oder polynomial lösbar?
4. Gibt es ein `PAS` für `HYPERGRAPH MAX CUT`?
5. Wie optimiert man gleichzeitig über die Bearbeitungsreihenfolge und die Partition der Bauteile?

Während 3 und 4 sicherlich mehr von theoretischem Interesse sind, sind die restlichen Punkte die interessanten Fragestellungen der Praxis. Insbesondere der letzte Punkt ist sicherlich für jede praktische Behandlung des Problems von überragender Wichtigkeit. Die in dieser Arbeit gegebenen Lösungsansätze vernachlässigen in sträflicher Weise diesen ganzheitlichen Aspekt, indem immer nur über ein Teilproblem optimiert wird. Auch die Tatsache, daß der Autor versucht hat, sinnvolle Interpretationen und Anwendungsfälle für die vorgeschlagenen Verfahren zu geben, ändert nichts an diesem grundsätzlichen Manko. Der Autor weiß jedoch nicht recht, wie das geschilderte Problem anzupacken ist; es ist ihm nicht gelungen, eine auch nur ansatzweise erfolgversprechende Modellierung angeben zu können. Nichtsdestotrotz ist es nach Meinung des Autors nicht grundsätzlich verkehrt, in einer solchen Situation wenigstens Teilprobleme zu lösen; möglicherweise lassen sich unter Verwendung und Weiterentwicklung der gegebenen Resultate einmal weitergehende Aussagen treffen.

5 Appendix

5.1 Verschiedenes

5.1 Definition. Für eine endliche Menge M bezeichnet

$$2^M$$

ihre Potenzmenge, besitzt M die Elemente m Elemente $m_i, i = 1, \ldots, m$, so wir auch $\{ m_i \}_{i=1}^{m}$. schreiben. □

5.2 Definition. Ist $H = (V, E)$ ein Hypergraph mit n Knoten und W eine Teilmenge von V, so heißt der Vektor $\chi(W) \in \{ 0, 1 \}^n$ mit der Eigenschaft

$$\chi(W)_i = 1 \iff v_i \in W$$

Inzidenzvektor von W. □

5.3 Definition. Eine **Doppelfolge** σ ist eine Funktion von den ganzen Zahlen in eine Bildmenge; sie wird in der Form

$$(\ldots, \sigma_{-2}, \sigma_{-1}, \sigma_0, \sigma_1, \sigma_2, \ldots)$$

notiert. □

5.4 Definition. Eine **Permutation** ist eine bijektive Selbstabbildung einer endlichen Menge. Die Menge aller Permutationen von $\{ 1, \ldots, m \}$ ist bezüglich der Hintereinanderausführung von Abbildungen, hier auch **Produkt** genannt, eine Gruppe, die **symmetrische Gruppe** von $\{ 1, \ldots, m \}$. Das Produkt von k Permutationen ϕ_i, $i = 1, \ldots, k$ schreiben wir, falls die Reihenfolge der Berechnung keine Rolle spielt, in der Form $\prod_{i=1}^{k} \phi_i$. Ist σ eine Permutation von $\{ 1, \ldots, m \}$ und M eine bezüglich Inklusion minimale invariante Teilmenge von $\{ 1, \ldots, m \}$ (das heißt, M ist minimal mit $\sigma(M) \subseteq M$), so ist die Permutation ρ auf $\{ 1, \ldots, m \}$, definiert durch

$$\rho(i) = \begin{cases} \sigma(i) & \text{, falls } i \in M \\ i & \text{, falls } i \notin M \end{cases}$$

ein **Faktor** von σ; wir sagen, der Faktor **operiert** auf M und schreiben $M = \{ \rho(i) \}$. Die Faktoren, die auf zweielementigen Mengen operieren, heißen **Transpositionen**. Ein Faktor ρ einer Permutation wird gern in der Form $(\rho_1, \ldots, \rho_k)$ dargestellt, wobei dann $\rho(\rho_i) = \rho_{i \bmod k+1}$ für $i = 1, \ldots, k$ und $\rho(i) = i$ sonst vereinbart sei; wir wollen (mit bequemer, aber nicht ganz konsistenter Notation) auch "einelementige" Faktoren zulassen, was bedeute, daß im Kontext jeweils ein Element, auf dem die identische Permutation operiert, ausgezeichnet ist. Wir schreiben dann etwa (i), falls i ausgezeichnet war und auch $\{ (i) \} = \{ i \}$. Insbesondere bedeutet dies, daß kein anderer Faktor auf i operiert. Ferner sei (zur Vereinfachung der Notation) $(i_1, \ldots, i_j, i_j, \ldots, i_k) = (i_1, \ldots, i_j, \ldots, i - k)$ verabredet. Eine Permutation, die nur die zwei trivialen Faktoren besitzt, heißt **zyklisch**. □

5.5 Satz. Jede Permutation ist Produkt von Transpositionen. Jede Permutation ist das in beliebiger Reihenfolge berechnete Produkt eindeutig bestimmter Faktoren. □

5.6 Definition. Eine **logische Variable** ist eine Funktion $v : \Omega \to \{ \text{wahr}, \text{falsch} \}$, wobei Ω nicht näher bestimmt ist; ist v ein Mitglied einer Menge von logischen Variablen V, so sind die Funktionen v und $\overline{v}$ **Literale** über V; $\overline{\cdot}$ bezeichne hierbei die logische **Negation**. Eine **Klausel** über V ist ein Tupel von Literalen über V und wird auch gerne in der Form $(c_1 \vee c_2 \vee c_3)$ notiert; wir wollen ohne Beschränkung der Allgemeinheit stets davon ausgehen, daß die Literale in den Klauseln in einer durch eine Ordnung auf der Menge der Literale festgelegten Reihenfolge auftreten. Ein **Truth Assignment** über V ist eine Funktion von der Menge der Literale über V in die Menge $\{ \text{wahr}, \text{falsch} \}$, so daß für alle Variablen v aus V die Beziehung $t(v) = \overline{t(\overline{v})}$ gilt. Ein Truth Assignment t ist ein **Satisfying Truth Assignment** für eine Menge von Klauseln C, wenn für jede Klausel $c = (c_1, c_2, c_3)$ $t(c_1)$ oder $t(c_2)$ oder $t(c_3)$ wahr ist. □

5.7 Bemerkung. Vektoren und damit auch Matrizen sind Funktionen von einer endlichen Urbildmenge, so daß wir nicht zwischen Vektoren und geeigneten Funktionen unterscheiden (müssen/sollen); die jeweilige Schreibweise deutet lediglich an, ob wir mehr daran interessiert sind, daß eine Funktion auf einer Menge operiert, in welchem Fall wir etwa die Schreibweise $f(x)$ wählen, oder ob wir die Elemente der Bildmenge für wichtig halten, die lediglich durch die (beliebige) Urbildmenge indiziert werden; in diesem Fall schreiben wir dann f_x. Wo dies geeignet erscheint, wechseln wir jedoch auch zwischen den Darstellungen.
Ferner sei noch bemerkt, daß für eine Matrix $C = (c_{ij})$ $c_{i\cdot}$ ihren i-ten Zeilenvektor bezeichnet. Für Teilmengen I und J der Zeilen- beziehungsweise Spaltenindexmenge von C ist die Matrix $(c_{ij})_{i\in I, j\in J}$ ein **Minor** von C. Zerfällt für eine quadratische Matrix C die Menge ihrer Zeilen- und Spaltenindizes in Mengen I_i mit $\max\{I_1\} < \min\{I_2\} < \max\{I_2\} < \ldots < \min\{I_k\}$, so daß nur in den Minoren C_{I_i} von Null verschiedene Einträge vorhanden sind, so ist C eine **Blockdiagonalmatrix**, jeder Minor C_{I_i} ein **Block**. Der Vektor $\mathbb{1}$ bezeichnet stets einen m-dimensionalen Vektor aus lauter Einsen; m ergibt sich dabei immer aus dem Kontext. □

5.2 Hypergraphen, Graphen und Digraphen

5.8 Definition. Ein **Hypergraph** $H = (V, E)$ ist ein Zweitupel, wobei $V = \{v_1, \ldots, v_n\}$ eine n-elementige endliche Menge und $E = \{e_1, \ldots, e_m\}$ eine m-elementige Menge von Teilmengen von V ist; V heißt Menge der **Knoten** von H, E Menge der **Kanten** von H. Die Knotenmenge eines Hypergraphen H wollen wir auch als $V(H)$ bezeichnen, seine Kantenmenge auch als $E(H)$. Wir wollen auch zulassen, daß zwei Kanten von H gleich sein dürfen. Einelementige Kanten eines Hypergraphen heißen **Schlingen** oder **Schleifen**. Enthält eine Kante e einen Knoten v, so heißt e zu v **inzident**. Sind zwei Knoten u und v in einer Kante e enthalten, so sind u und v **adjazent**. Ist $\phi = (\phi_1, \ldots, \phi_r)$ ein Faktor einer Permutation auf V, so ist die Kante $\{\phi\}$, also die Menge der Elemente, auf der ϕ operiert, die von ϕ **induzierte**. Für jeden Hypergraphen $H = (V, E)$ mit n Knoten und m Kanten erklären wir die Matrix $(\chi(e)^t)_{e\in E}$ aus $\{0, 1\}^{m\times n}$ zu seiner **Inzidenzmatrix**. Ist für einen zweiten Hypergraphen $I = (U, F)$ U eine Teilmenge von V und F in E enthalten, so ist I ein **Unterhypergraph** von H; I ist in H **enthalten**. Gibt es für einen zweiten Hypergraphen $I = (U, F)$ dagegen bijektive Funktionen $u : V \to U$ und $f : E \to F$, so sind I und H **isomorph**. Ist W eine Teilmenge von V, so ist die Menge der "nach W zeigenden Kanten"

$$\delta(W) := \{\, e \in E : e \cap W \neq \emptyset,\ e \cap V \setminus W \neq \emptyset \,\}$$

der von W induzierte **Schnitt** in H und —$W \neq \emptyset$ vorausgesetzt— der Hypergraph

$$H \cdot W := \Big\{ \{\, v : v \notin W \,\} \cup \{\, w \,\}, \{\, e : e \cap W = \emptyset \,\} \\ \cup \{\, e \setminus W \cup \{\, w \,\} : \exists v \in e \cap W,\ \exists v \in e \cap (V \setminus W) \,\} \cup \{\, \{\, w \,\} : \exists e : e \subseteq W \,\} \Big\}$$

der durch **Kontraktion** von W entstandene Hypergraph. Werden mehrere disjunkte und nichtleere Teilmengen W_i der Knotenmenge eines Hypergraphen H nacheinander kontrahiert, so spielt die Reihenfolge der Kontraktion keine Rolle und wir bezeichnen den durch diese Kontraktionen entstehenden Hypergraphen als $H \prod_i W_i$. Für einen Hypergraphen $H = (V, E)$ und $W \subseteq V$ ist der Hypergraph

$$H - W := \big(V \setminus W, \{\, e \setminus W : e \in E \,\}\big)$$

aus H durch **Einschränkung** auf $V \setminus W$ hervorgegangen. Ein Hypergraph ist **zusammenhängend**, wenn nur die Schnitte $\delta(\emptyset)$ und $\delta(V)$ leer sind, er ist ein **Baum**, wenn er zusammenhängt und wenn

$$\sum_{e\in E} \big(|e| - 1\big) = n - 1 = |V| - 1$$

gilt (man beachte, daß ein Baum beliebige Schleifen enthalten kann). Eine Kante e eines Baumes $T = (V, E)$ ist ein **Blatt**, wenn $T - e := \left(\bigcup_{f \in E : f \neq e} \{ f \}, \{ f \in E : f \neq e \}\right)$ immer noch ein Baum ist. Bezüglich Inklusion maximale zusammenhängende Untergraphen eines Hypergraphen sind seine **Komponenten**. Ein **Kreis** ist ein Tupel der Form

$$(e_1, v_1, e_2, v_2, e_3, v_3, \ldots, v_k),$$

in dem immer abwechselnd auf eine Kante e_i ein Knoten v_j folgt,wobei k größer als eins ist, die e_i und v_i alle verschieden sind und jedes v_i sowohl in e_i alsauch in $e_{i \bmod k+1}$ enthalten ist. □

5.9 Beobachtung. Für einen Schnitt $\delta(W)$ gilt stets

$$\delta(W) = \delta(V \setminus W).$$

□

5.10 Beobachtung. Ein Hypergraph zerfällt in eindeutig bestimmte Komponenten. □

5.11 Folgerung. Ein Hypergraph ist genau dann ein Baum, wenn er zusammenhängt und keinen Kreis enthält. Jede Kante eines Baumes hat genau einen Knoten mit jeder Komponente des Graphen gemeinsam, der durch Entfernen der Kante aus dem Baum entsteht. Ein Baum enthält immer ein Blatt. Hat ein Baum mindestens zwei Kanten, so hat jedes Blatt genau einen Knoten mit dem übrigen Baum gemeinsam. □

5.12 Definition. Ein **Graph** $G = (V, E)$ ist ein Hypergraph, bei dem jede Kante höchstens zu zwei Knoten inzident ist. Analog zur Situation bei den Hypergraphen spricht man von einem **Untergraphen** eines Graphen. Ein **Weg** von einem Knoten u zu einem Knoten v in G ist eine Menge von Kanten von G der Form

$$\big\{ \{ u = w_1, w_2 \}, \{ w_2, w_3 \}, \{ w_3, w_4 \}, \ldots, \{ w_{k-1}, w_k = v \} \big\},$$

wobei k größer als 1 und die w_i alle verschieden sind. □

5.13 Folgerung. Für einen Graphen G sind die folgenden Aussagen äquivalent.

1. G ist zusammenhängend.
2. G enthält einen Baum.
3. Für je zwei verschiedene Knoten u und v enthält G einen Weg von u nach v. □

5.14 Definition. Ein **Digraph** $D = (V, A)$ ist ein Zweitupel, wobei $V = \{ v_1, \ldots, v_n \}$ eine n-elementige endliche Menge und $A = \{ a_1, \ldots, a_m \}$ eine m-elementige Teilmenge von $V \times V$ ist; V heißt Menge der **Knoten** von D, A Menge der **Bögen** von D. Bögen der Form (v, v) heißen **Schlingen** oder **Schleifen**. Für einen Bogen $a = (u, v)$ heißt $\mathrm{tail}(a) := u$ **Schwanzende**, $\mathrm{head}(a) := v$ **Kopfende** von a; $\mathrm{tail}(a)$ und $\mathrm{head}(a)$ sind zu a **inzident**, das Kopf- und das Schwanzende eines Bogens sind **adjazent**. Ist für einen zweiten Digraphen $C = (U, B)$ U eine Teilmenge von V und B in A enthalten, so ist C ein **Unterdigraph** von D; C ist in D **enthalten**. Ist W eine Teilmenge von V, so sind sowohl die Menge der "aus W herauszeigenden Bögen"

$$\delta^+(W) := \big\{ (u, v) \in E : u \in W,\ v \notin W \big\}$$

alsauch die Menge der "nach W hineinzeigenden Bögen"

$$\delta^-(W) := \big\{ (u, v) \in E : u \notin W,\ v \in W \big\}$$

von W induzierte **Schnitte** in D und —$W \neq \emptyset$ vorausgesetzt— der Digraph

$$D \cdot W := \Big\{ \{ v : v \notin W \} \cup \{ w \}, \big\{ (u, v) \in A : u, v \notin W \big\} \cup \big\{ (w, v) : \exists u \in W : (u, v) \in \delta^+(W) \big\} \\ \cup \big\{ (u, w) : \exists v \in W : (u, v) \in \delta^-(W) \big\} \cup \big\{ (w, w) : \exists u, v \in W : (u, v) \in A \big\} \Big\}$$

der durch **Kontraktion** von W entstandene Digraph. Ein **gerichteter Kreis** in einem Digraphen D ist eine Menge von Bögen von D der Form

$$\big\{ (c_1, c_2), (c_2, c_3), (c_3, c_4), \ldots, (c_k, c_1) \big\},$$

wobei k größer als 1 und die c_i alle verschieden sind. Für einen gerichteten Kreis C bezeichnen wir mit ein $\{ C \}$ die Menge der in ihm enthaltenen Knoten. Ein **Pfad** von einem Knoten u zu einem Knoten v in D ist eine Menge von Bögen von D der Form

$$\big\{ (u = w_1, w_2), (w_2, w_3), (w_3, w_4), \ldots, (w_k, w_1 = v) \big\},$$

wobei k größer als 1 und die w_i alle verschieden sind. Ein Digraph ist eine **Arboreszenz** mit **Wurzel** w, wenn

$$\left|\delta^- (v)\right| = 1$$

für alle v aus $V \setminus \{ w \}$ gilt und D keinen gerichteten Kreis enthält. Sind s und t zwei verschiedene Knoten in D, so heißt eine Funktion $x : A \to \mathbb{N}_0$ (häufig auch als Matrix oder Vektor betrachtet), die die Bedingungen

$$\sum_{a \in \delta^+(v)} x_a = \sum_{a \in \delta^-(v)} x_a$$

für alle $v \in V \setminus \{ s, t \}$ erfüllt, ein **Fluß** von s nach t, die Zahl $\sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a$ heißt **Wert** des Flusses. Ein Fluß vom Wert Null ist eine **Zirkulation**. Ist x eine Zirkulation und W eine bezüglich Inklusion minimale Teilmenge der Knotenmenge des Digraphen, in dem x fließt, mit der Eigenschaft $\sum_{\substack{i \in W \\ j \notin W}} \tau_{ij} = 0$, so heißt die Zirkulation y, definiert durch

$$y_{ij} := \begin{cases} x_{ij} & \text{, falls } i, j \in W \\ 0 & \text{, sonst} \end{cases}$$

eine **Komponente** von x. Ist C ein gerichteter Kreis in einem Digraphen D, so bezeichnen wir die Zirkulation y mit der Eigenschaft

$$y_{ij} := \begin{cases} 1 & \text{, falls } (i, j) \in C \\ 0 & \text{, sonst} \end{cases}$$

als von C **induziert**. Ist die Zirkulation x die Summe der von gerichteten Kreisen induzierten Zirkulationen, so sagen wir auch, daß x in diese gerichteten Kreise zerfällt oder ihre Summe ist. Eine Matrix $x \in \{ 0, 1 \}^{m \times m}$ ist eine **Lösung eines Assignment Problems**, wenn für alle i stets $\sum_{j=1}^{m} x_{ij} = 1$ gilt.

5.15 Folgerung. Ist A eine aufspannende Arboreszenz in einem Digraphen D mit Wurzel w, so enthält A für jeden von w verschiedenen Knoten u einen Pfad von w nach u. □

5.16 Folgerung. Ist x ein (s, t)-Fluß in einem Digraphen D und enthält eine Teilmenge W von V s, aber nicht t, so gilt

$$\sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a = \sum_{a \in \delta^+(W)} x_a - \sum_{a \in \delta^-(W)} x_a = \sum_{a \in \delta^-(V \setminus W)} x_a - \sum_{a \in \delta^+(V \setminus W)} x_a.$$

□

5.17 Folgerung. Ist x eine Zirkulation in einem Digraphen D und C ein in x enthaltener Kreis, so ist x auch nach Abzug der von C induzierten Zirkulation noch eine Zirkulation in D. Jede Zirkulation ist die Summe eindeutig bestimmter Komponenten und keineswegs eindeutig bestimmter Kreise und auch die Summe (keineswegs eindeutig bestimmter) Kreise. □

5.3 Komplexitätstheorie

5.18 Definition. Eine endliche Menge Σ von mindestens zwei Elementen, hier **Symbole** genannt, heißt **Alphabet**; Σ^* bezeichnet dann die Menge der endlichen **Strings**, die aus Symbolen des Alphabets Σ gebildet werden können, also die Menge

$$\bigcup_{k \in \mathbb{N}} \Sigma^k$$

aller k-Tupel von Elementen aus Σ. Die **Länge eines Strings** ist die Anzahl der in ihm enthaltenen Symbole. □

5.19 Beobachtung. Mit Hilfe eines gegebenen Alphabets können alle anderen Alphabete in der Weise simuliert werden, daß man ihre Symbole durch Strings des gegebenen Alphabets darstellt; in diesem Sinne sind alle Alphabete äquivalent. Ohne Beschränkung der Allgemeinheit kann man daher davon ausgehen, daß ein Alphabet ein ausgezeichnetes **Leerzeichen** $\flat$ enthält. Ferner ist Σ^* stets abzählbar. □

5.20 Definition. Eine **deterministische Turing-Maschine**, `DTM` oder ein **Algorithmus** besteht aus folgenden Komponenten.

1. Einem Alphabet Σ.
2. Einer Doppelfolge τ mit Bildbereich Σ, dem **Band** der Maschine. Die Komponenten des Bandes heißen auch **Felder**.
3. Einem **Schreib-/Lesekopf**, der jeweils eine Position über einem Feld des Bandes einnimmt, sich jeweils ein Feld vor oder zurückbewegen kann und das in diesem Feld enthaltene Symbol lesen oder ein Symbol auf dieses Feld schreiben kann (und damit das dort befindliche Symbol überschreibt).
4. Einer endlichen Menge Z von **Zuständen** mit drei ausgezeichneten Elementen **Start**, **Nein** und **Ja**.
5. Einem **Zustandsregister**, in dem sich jeweils ein aktuelle Zustand der Maschine aus Z befindet.
6. Einer Transitionsfunktion $\Sigma \times Z \setminus \{ \text{Ja}, \text{Nein} \} \to \Sigma \times Z \times \{ -1, 1 \}$.

Eine Turing-Maschine funktioniert auf folgende Weise. Zunächst ist das Band leer, das heißt alle Felder enthalten das Leersymbol $\flat$. Dann wird ein String aus Σ^* in die Felder $1, 2, \ldots$ des Bandes geschrieben, der **Input**. Der Schreib-/Lesekopf wird über Feld 1 in Stellung gebracht und das Zustandsregister mit Start initialisiert. Jetzt beginnt die Maschine in **Schritten** zu arbeiten. In jedem Schritt wird zunächst untersucht, ob der Zustand der Maschine Ja oder Nein ist. Ist das so, **hält** die Maschine an. Ansonsten wird das Zeichen unter dem Schreib-/Lesekopf gelesen, der Zustand der Maschine geholt und die Transitionsfunktion auf dieses Zweitupel angewendet. Die erste Komponente des Ergebnisses wird in das Feld, über dem sich der Kopf befindet, geschrieben, die zweite überschreibt das Zustandsregister und die dritte bestimmt, daß der Kopf sich entweder ein Feld auf dem Band vor- oder zurückbewegt.
Eine **nichtdeterministische Turing-Maschine**, `NDTM` oder ein **Nichtdeterministischer Algorithmus** besteht aus abzählbar unendlich vielen gleichen deterministischen Turing-Maschinen. □

5.21 Bemerkung. Turing-Maschinen können auf einen Input nicht nur mit Ja oder Nein oder gar nicht antworten, sondern auch Ergebnisse auf ihr Band schreiben, bevor sie in einen der Haltezustände übergehen. (Ohne Beschränkung der Allgemeinheit kann verlangt werden, daß die Maschine vorher den Teil des Bandes mit den nichtpositiven Indizes wieder löschen muß.) □

5.22 Definition. Die **Laufzeitfunktion einer** `DTM` ordnet jeder natürlichen Zahl k die maximale Anzahl an Schritten (möglicherweise ∞) zu, die die `DTM`, gestartet mit einem beliebigen String einer Länge von höchstens k Symbolen als Input ausführt, bis sie anhält. Gibt es ein Polynom, das auf den natürlichen Zahlen nie kleiner als die Laufzeitfunktion einer `DTM` ist, so heißt die `DTM` **polynomial**.

5.23 Definition. Ein **Codierungsschema** ist eine polynomiale `DTM`, die beim Übergang in einen der Haltezustände das Band genau in dem Zustand hinterläßt, wie sie ihn bei ihrem Start vorfand. Ein **Problem** ist die Menge der Strings, für die ein Codierungsschema eine Ja-Antwort gibt, es heißt von diesem Codierungsschema induziert. Jedes Element eines Problems ist eine **Instanz**, die **Codierungslänge** einer Instanz ist ihre Länge. Zwei Codierungsschemata beziehungsweise die von ihnen induzierten Probleme sind **polynomial**

äquivalent, wenn es zwei polynomiale deterministische Turing-Maschinen gibt, von denen die erste jede Instanz des vom ersten Schemas induzierten Problems in eine Instanz des vom zweiten Schema induzierten Problems im Sinne von Bemerkung 5.21 transformiert während die zweite genau dasselbe tut, jedoch mit vertauschten Rollen der Codierungsschemata. Ein **Entscheidungsproblem** ist eine Funktion von einem Problem in die Menge $\{\,\text{Ja}, \text{Nein}\,\}$. Ein Entscheidungsproblem P ist **reduzierbar** auf ein Entscheidungsproblem Q, wenn wenn es eine `DTM` gibt, die Ja-Instanzen von P auf Ja-Instanzen von Q und Nein-Instanzen von P auf Nein-Instanzen von Q im Sinne von Bemerkung 5.21 **transformiert**, es ist **polynomial reduzierbar**, wenn die `DTM` polynomial ist. Zwei Entscheidungsprobleme sind **äquivalent**, wenn sie wechselseitig aufeinander transformierbar sind, **polynomial äquivalent**, wenn die Reduktion wechselseitig polynomial ist. □

5.24 Beispiel. Das asymmetrische `TSP` kann durch eine deterministische Turing-Maschine codiert werden, die als Alphabet $\{\,0,\ldots,9,(),\text{“,”},\flat\,\}$ akzeptiert, wobei “,” bedeute, daß das Komma selbst ein gültiges Symbol sei, und die “prüft”, ob ihr Input folgendes Format besitzt. Zuerst wird eine Zahl, also eine endliche Ziffernfolge, erwartet (die “Anzahl der Städte”), dann ein Komma, und jetzt eine Anzahl an durch Kommata getrennten Zahlen, die der Anzahl an Bögen des vollständigen Diraphen entspricht, der gerade soviel Knoten hat, wie die zuerst gelesene Zahl der Städte angibt (“die Zahlen entsprechen in der Reihenfolge der lexikalischen Ordnung den Bögen des Digraphen”). Hierauf muß ein Leerzeichen folgen. Ist dies der Fall, so antwortet die Maschine mit Ja, das heißt sie geht in den Ja-Zustand über, sonst mit Nein, wobei vorher noch das Band wiederhergestellt wird. Der Autor versichert, daß es möglich ist, eine derartige `DTM` mit polynomialer Laufzeit zu konstruieren. □

Analog zum Beispiel werden in dieser Arbeit, wie dies allgemein üblich ist, Algorithmen in einem **Pseudocode** formuliert, der Operationen und Daten nicht vollständig spezifiziert. Implizit ist damit stets die Aussage verbunden, daß es möglich ist, die nötigen Verfeinerungen durchzuführen, ohne behauptete Eigenschaften wie zum Beispiel Laufzeiten zu verlieren. Es sei weiter ausdrücklich festgehalten, daß eine ganze Zahl k binär mit Hilfe von

$$\lceil \log_2 |k| + 1 \rceil + 1$$

Symbolen aus dem Alphabet $\{\,0,1,+,-\,\}$, ein Vektor und damit auch eine Matrix als Folge seiner Komponenten, ein Graph etwa durch seine Inzidenzmatrix und aus diesen oder ähnlichen Komponenten bestehende Tupel einfach als Folge ihrer Komponenten codiert werden können. Bezeichnen wir mit $\langle\cdot\rangle$ die Funktion, die bei Verwendung dieses erweiterten binären Codierungsschemas den codierten Objekten ihre Codierugslänge zuordnet, so gilt für ganze Zahlen k

$$\langle k \rangle = \lceil \log_2 |k| + 1 \rceil + 1,$$

für Vektoren $(v_1,\ldots,v_n)$

$$\langle v \rangle = \sum_{i=1}^{n} \langle v_i \rangle$$

und so fort. Alle “üblichen” Codierungsschemata sind zueinander polynomial äquivalent. Aus diesem Grunde wollen wir stets das genannte binäre Codierungsschema zugrundelegen und die Funktion $\langle\cdot\rangle$ schlicht als “die” Codierungslänge bezeichen.

5.25 Beobachtung. Seien P und Q zwei äquivalente Entscheidungsprobleme. Dann gibt es einen polynomialen Algorithmus zur Lösung von P genau dann, wenn es einen polynomialen Algorithmus zur Lösung von Q gibt. □

Ein entsprechender Pseudocode wird zur Beschreibung von Problemen verwendet. Wir wählen die folgende Bezeichnugsweise.
Instanz. Beschreibung einer “allgemeinen Probleminstanz”.
Frage. Eine auf die Instanz bezogene, mit Ja oder Nein zu beantwortende Frage, deren “richtige Beantwortung” mit Ja oder Nein der Zuordnung der Zustände Ja und Nein zu den Instanzen entspricht.

5.26 Beispiel. `TSP`.
Instanz. Vollständiger Digraph auf n Knoten, für jeden Bogen ein natürliches Gewicht, natürliche Zahl k.
Frage. Gibt es eine Tour der Länge $\leq k$? □

5.27 Definition. Ein nichtdeterministischer Algorithmus **rät** auf folgende Weise. Gegeben sei ein beliebiger Input. Alle Strings des zu den (gleichartigen) deterministischen Komponenten gehörigen Alphabets und die Komponenten selbst werden abgezählt. Für alle i wird auf das Band von `DTM` i der interessierende Input und dahinter String i geschrieben. Erst jetzt dürfen die Komponenten starten. Als Anzahl der von der `NDTM` benötigten Schritte zur Bearbeitung eines Input gilt die Minimalanzahl an Schritten, die eine seiner Komponenten benötigt, um nach dem Raten im Ja-Zustand anzuhalten. Die **Laufzeitfunktion einer** `NDTM` ordnet jeder natürlichen Zahl k die maximale Anzahl an Schritten (möglicherweise ∞) zu, die die `NDTM`, gestartet mit einem String einer Länge von höchstens k zur Bearbeitung desselben benötigt. Ein nichtdeterministischer Algorithmus **löst** ein Entscheidungsproblem, wenn für jede Instanz mit Ja-Antwort nach dem Raten eine Komponente der `NDTM` einmal die Antwort Ja gibt (das heißt sie hält im entsprechenden Zustand an), für jede Instanz mit Antwort Nein aber niemals eine Komponente die Antwort Ja gibt (möglicherweise aber gar keine Antwort produziert). Eine `NDTM` heißt **polynomial**, wenn es ein Polynom gibt, das für keine natürliche Zahl kleinere Werte als die Laufzeitfunktion annimmt.

5.28 Definition. Ein Entscheidungsproblem gehört zur Klasse $\mathcal{P}$, wenn es einen deterministischen polynomialen Algorithmus zu sein Lösung gibt. Ein Entscheidungsproblem gehört zur Klasse $\mathcal{NP}$, wenn es einen polynomialen nichtdeterministischen Algorithmus gibt, der es löst. Ein Entscheidungsproblem gehört zur Klasse Co-$\mathcal{NP}$, wenn das Entscheidungsproblem, das durch Vertauschen der Ja- mit den Nein Antworten entsteht, zur Klasse $\mathcal{NP}$ gehört. Ein Entscheidungsproblem P ist **$\mathcal{NP}$-vollständig**, wenn es für jedes Problem Q aus $\mathcal{NP}$ einen polynomialen Algorithmus gibt, der Ja-Instanzen von Q auf Ja-Instanzen von P und Nein-Instanzen von Q auf Nein-Instanzen von P transformiert.

5.29 Folgerung.

1. Gibt es einen polynomialen Algorithmus zur Lösung eines $\mathcal{NP}$-vollständigen Entscheidungsproblems, so folgt die Gleichheit von $\mathcal{P}$ und $\mathcal{NP}$; insbesondere gibt es für alle Probleme aus $\mathcal{NP}$ polynomiale Algorithmen.
2. $\mathcal{P}$ ist eine Teilmenge von $\mathcal{NP} \cap$ Co-$\mathcal{NP}$. ndigkeitFolgerung 5.29 □

5.30 Definition. Ein **Optimierungsproblem** ist eine Funktion von einem Problem in die Menge der rationalen Zahlen. Ein Algorithmus **löst** ein Optimierungsproblem, wenn er für jede Instanz nach endlich vielen Schritten anhält und auf seinem Band einen String, die **Lösung** hinterläßt, so daß es einen polynomialen Algorithmus gibt, der als Input die Aneinanderreihnug der Codierung der Probleminstanz und den gerade erwähnten String annimmt und damit die zur jeweiligen Instanz gehörige rationale Zahl berechnet. Ein Optimierungsproblem läßt sich auf ein zweites **polynomial reduzieren**, wenn es einen polynomialen Algorithmus zu seiner Lösung gibt, der als Unterprogramm einen Algorithmus zur Lösung des zweiten Optimierungsproblems verwendet, sie sind **polynomial äquivalent**, wenn sie wechselseitig polynomial aufeinander reduzierbar sind.
Analog zum Vorgehen bei Entscheidungsproblemen notieren wir Optimierungsprobleme in umgangssprachlicher Form. Speziell wollen ein kombinatorisches Minimierungs- beziehungsweise Maximierungsproblem beschreiben als das folgende Problem.
Instanz. (Meist implizite) Beschreibung einer endlichen **Grundmenge** und einer **Zielfunktion** von der Grundmenge in die rationalen Zahlen.
Frage. Was ist der maximale (minimale) Zielfunktionswert aller Elemente der Grundmenge der Instanz?
Ist P ein kombinatorisches Maximierungs- beziehungsweise Minimierungsproblem, so ist das **zu P gehörende** Entscheidungsproblem das Problem
Instanz. Grundmenge, Zielfunktion, rationale Zahl k.
Frage. Gibt es ein Element aus der Grundmenge mit einem Zielfunktionswert von mehr beziehungsweise im Minimierungsfall von weniger als k?
Ein kombinatorisches Optimierungsproblem ist **$\mathcal{NP}$-schwer**, wenn das ihm zugeordnete Entscheidungsproblem $\mathcal{NP}$-vollständig ist, es ist **$\mathcal{NP}$-leicht**, wenn es einen polynomialen Algorithmus zu seiner Lösung gibt, der als Unterprogramm einen nichtdeterministischen polynomialen Algorithmus zur Lösung eines Entscheidungsproblems aus $\mathcal{NP}$ verwendet und es ist **$\mathcal{NP}$-äquivalent**, wenn es sowohl $\mathcal{NP}$-schwer alsauch $\mathcal{NP}$-leicht ist. □

5.31 Folgerung. Gibt es einen polynomialen deterministischen Algorithmus zur Lösung eines $\mathcal{NP}$-schweren Optimierungsproblems, so folgt die Gleichheit von $\mathcal{P}$und $\mathcal{NP}$. Gibt es einen polynomialen Algorithmus zur

Lösung eines $\mathcal{NP}$-vollständigen Entscheidungsproblems, so gibt es auch für alle $\mathcal{NP}$-leichten Optimierungsprobleme polynomiale Algorithmen. □

5.32 Definition. Sei P ein kombinatorisches Maximierungsproblem (Minimierungsproblem) mit Zielfunktion c und rationalem $0 < \epsilon < 1$ $(0 < \epsilon)$ beliebig. Dann heißt ein Algorithmus A **ϵ-approximativ**, wenn er für jede Probleminstanz von P ein Element x^ϵ aus der Grundmenge als Lösung liefert, so daß im Vergleich mit dem optimalen Zielfunktionswert der Instanz c_{opt} die Beziehung

$$c_{\text{opt}}(1-\epsilon) \leq c(x^\epsilon) \quad \big(c_{\text{opt}}(1+\epsilon) \geq c(x^\epsilon)\big) \tag{1}$$

gilt. Akzeptiert Algorithmus A zusätzlich zur Probleminstanz auch noch ϵ als Input und liefert für alle ϵ wie oben angegeben und alle Probleminstanzen von P eine Lösung entsprechend (1), so ist A ein **Approximationsschema**; ist zusätzlich die Laufzeit von A für jedes feste, aber beliebige ϵ polynomial in der Codierungslänge der Probleminstanz ohne Berücksichtigung von ϵ, so heißt A ein **polynomiales Approximationsschema** oder kurz `PAS`. Ist die Laufzeit von A sogar polynomial in der Codierungslänge der Probleminstanz und in $\frac{1}{\epsilon}$, so ist nennt man A ein **voll polynomiales Approximationsschema** oder kurz `FPAS`. □

6 Index

6.1 Literatur

[Garey & Johnson, 1979] M.R. Garey, D.S. Johnson (1979). *A Guide to the Theory of $\mathcal{NP}$-Completness*, Freeman, San Francisco.

[Garey, Johnson & Stockmeyer, 1976] M.R. Garey, D.S. Johnson, L. Stockmeyer (1976). Some simplified $\mathcal{NP}$-complete Graph Problems, *Theor. Comput. Sci. 1, 237–267.*

[Gilmore & Gomory, 1964] P.C. Gilmore, R.E. Gomory (1964). Sequencing a one state-variable machine: a solvable case of the traveling salesman problem, *Oper. Res. 12, 655–679.*

[Ibarra & Kim, 1975] O.H. Ibarra, C.M. Kim (1975). Fast Approximation Algorithms for the Knapsack and Sum of the Subset Problems, *J. Assoc. Comput. Mach. 22, 463–468.*

[Karp, 1972] R.M. Karp (1972). Reducibility among combinatorial Problems, in R.E. Miller, J.W. Thatcher (eds.), *Complexity of computer computations, 85–103*, Plenum Press, New York.

[Lovasz, 1973] L. Lovasz, (1973). Coverings and Colorings of Hypergraphs, *Proc. 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing, 3–12*, Utilitas Mathematica Publishing, Winnipeg.

[Lenstra, Lawler, Rinnooy Kan & Shmoys, 1985] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (1985). *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, Wiley, New York.

[Schaefer, 1978] T.J. Schaefer (1978). The Complexity of Satisfiability Problems, *Proc. 10th Ann. ACM Symposium on Theory of Computing, 216–226*, Association for Computing Machinery, New York.

6.2 $\mathcal{NP}$-vollständige Probleme

6.1 Problem. `FOUR PROCESSOR ASSEMBLY LINE SCHEDULING`.
Instanz. Gegeben seien m Zahlenquintupel $(a_i, b_i, c_i, d_i, s_i)$ aus $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}$, wobei $d_1 \leq \ldots \leq d_m$ gelte.
Frage. Welche verallgemeinerte Permutation σ auf $\{1, \ldots, m\}$ mit Vielfachheiten s_i führt zu einer minimalen Gesamtbearbeitungszeit

$$\sum_{i=1}^{S} \max \left\{ a_{\sigma\left((i+2) \bmod S+1\right)}, b_{\sigma\left((i+1) \bmod S+1\right)}, c_{\sigma(i \bmod S+1)}, d_{\sigma(i)} \right\}?$$

□

6.2 Problem. `HYPERGRAPH MAX CUT`.
Instanz. Gegeben sei eine natürliche Zahl k, ein Hypergraph $H = \left(V, \{e_1, e_2\}\right)$ mit n Knoten und m Kanten $e_1, \ldots, e_m$ sowie eine nichtnegative ganzzahlige Gewichtung c_{ij} für alle j aus V und i aus E der Knoten von H in Abhängigkeit von der Zugehörigkeit zu den Kanten von H, so daß c_{ij} Null ist, wenn der Knoten j nicht in e_i enthalten ist.
Frage. Gibt es eine Partition $P = \left(V, \{f_1, f_2\}\right)$ von V mit der Eigenschaft

$$\sum_{i=1}^{m} \min\left(c_i \cdot \chi(f_1), c_i \cdot \chi(f_2)\right) \geq k?$$

□

6.3 Problem. `INDEPENDENT SET`.
Instanz. Gegeben sei ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Frage. Gibt es eine Teilmenge W von V mit mindestens k Elementen, so daß keine zwei Knoten von W adjazent sind? □

6.4 Problem. `MAX CUT`.
Instanz. Graph $G = (V, E)$ mit natürlichen Kantengewichten c_e für alle e aus E und eine natürliche Zahl k.
Frage. Gibt es einen Schnitt mit größerem Gewicht als k, das heißt existiert eine Teilmenge W von V mit der Eigenschaft

$$\sum_{e \in \delta(W)} c_e \geq k?$$

□

6.5 Problem. `MAXIMUM 2SATISFIABILITY`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V, so daß jede Klausel genau zwei verschiedene Literale über V enthält, und eine natürliche Zahl k.
Frage. Gibt es ein Truth Assignment für C, so daß mindestens k der $|C|$ Klauseln erfüllt sind? □

6.6 Problem. `MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS`.
Instanz. Gegeben sei ein Job, der die Herstellung von s_i Einheiten einer Leiterplatte vom Typ i für jeden von insgesamt m Typen verlangt. Auf einer Platine vom Typ i sind dabei $m_{ij} \in \mathbb{N}_0$ Einheiten eines Typs j von insgesamt n Typen von Bauteilen zu plazieren. Die Bestückung mit einer Einheit eines Bauteils vom Typ j nehme hierbei jeweils $c_j \in \mathbb{N}$ Zeiteinheiten in Anspruch. Es bezeichne $c_{ij} := m_{ij} c_j$ für alle Platinentypen i und alle Bauteiltypen j die zur Bestückung mit allen Bauteilen des Typs j auf einer Platine des Typs i insgesamt benötigte Zeit und S die Summe der Zahlen s_i.
Frage. Welche Partition $P = \big(\{\,1, \ldots, n\,\}, \{\,f_1, f_2\,\}\big)$ auf der Menge der Bauteiltypen $\{\,1, \ldots, n\,\}$ und welche verallgemeinerte Permutation σ von $\{\,1, \ldots, m\,\}$ der Länge S mit Vielfachheiten s_i minimieren die für die Ausführung des Jobs insgesamt benötigte Zeit

$$c_{\sigma(1)\cdot\chi(f_1)} + \sum_{i=1}^{S-1} \max\big\{\, c_{\sigma(i)\cdot\chi(f_2)}, c_{\sigma(i+1)\cdot\chi(f_1)} \,\big\} + c_{\sigma(S)\cdot\big(\chi(f_2)\big)}?$$

(χ bezeichnet den Inzidenzvektor einer Menge, vergleiche den Symbolindex.) □

6.7 Problem. `MINIMIERUNG DER GESAMTBEARBEITUNGSZEIT EINES JOBS`.
Instanz.Gegeben sei eine Matrix C aus $\mathbb{N}_0{}^{m \times n}$ und m positive Zahlen s_i mit $S = \sum_{i=1}^{m} s_i$.
Frage. Welche Partition P von $\{\,1, \ldots, n\,\}$ und welche verallgemeinerte Permutation σ von $\{\,1, \ldots, m\,\}$ der Länge S mit Vielfachheiten s_i minimieren

$$\pi(C, P) \cdot \tau(\sigma)?$$

□

6.8 Problem. `NOT ALL EQUAL 3SAT`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V, so daß jede Klausel genau drei verschiedene Literale über V enthält.
Frage. Gibt es ein **Satisfying ruth Assignment** für C, so daß in jeder Klausel mindestens ein Literal falsch ist? □

6.9 Problem. `PARTITION`.
Instanz. Gegeben seien m natürliche Zahlen a_i für i aus $\{\,1, \ldots, m\,\}$.
Frage. Gibt es eine Partition $P = (V, \{\,f_1, f_2\,\})$ auf $\{\,1, \ldots, m\,\}$ mit der Eigenschaft

$$\sum_{i \in f_1} a_i = \sum_{j \in f_2} a_j?$$

□

6.10 Problem. `SAT`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V.
Frage. Gibt es ein Satisfying Truth Assignment für C? □

6.11 Problem. Problem 3.9.
Instanz. Gegeben sei ein Hypergraph $H = (V, E)$.
Frage. Gibt es eine Partition $P = \big(V, \{ f_1, f_2 \}\big)$ auf V, so daß der Hypergraph $H \cdot f.f_2$ keine Schlinge enthält? □

6.12 Problem. `SUBSET SUM`.
Instanz. Gegeben seien m natürliche Zahlen a_i für i aus $\{ 1, \ldots, m \}$ und eine natürliche Zahl k.
Frage. Gibt es eine Partition $P = (V, \{ f_1, f_2 \})$ auf $\{ 1, \ldots, m \}$ mit der Eigenschaft

$$\sum_{i \in f_1} a_i = k?$$

Die Optimierungsversion von `SUBSET SUM`, die wir der Bequemlichkeit halber ebenfalls mit `SUBSET SUM` bezeichnen wollen, besitzt die gleichen Instanzen, jedoch betrachtet man eine andere Frage.
Frage.

$$\max_{\text{Partition} P=(V,\{ f_1,f_2 \})} a_{i \cdot \chi}(f_1)$$
$$a_{i \cdot \chi}(f_1) \leq k$$

□

6.13 Problem. `THREE DIMENSIONAL MATCHING`.
Instanz. Gegeben sei eine Teilmenge W des kartesischen Produkts dreier gleichmächtiger Mengen X, Y und Z mit jeweils m Elementen.
Frage. Gibt es m Elemente aus W, so daß nie zwei in einer Koordinate übereinstimmen? □

6.14 Problem. `3SAT`.
Instanz. Gegeben seien eine Menge C von Klauseln über einer Menge von logischen Variablen V, so daß jede Klausel genau drei verschiedene Literale über V enthält.
Frage. Gibt es ein Satisfying Truth Assignment für C? □

6.15 Problem. `TSP`.
Instanz. Gegeben sei ein vollständiger Digraph D_m mit m Knoten und nichtnegativen ganzzahligen Bogengewichten c_{ij} für alle Bögen und eine natürliche Zahl k.
Frage. Gibt es eine Rundreise mit einer Länge von höchstens k, das heißt, gibt es eine Permutation σ auf $\{ 1, \ldots, m \}$ mit der Eigenschaft

$$\sum_{i=1}^{m} c_{\sigma(i)\sigma(i \bmod m+1)} \leq k?$$

6.16 Problem. `VERTEX COVER`.
Instanz. Gegeben sei ein Graph $G = (V, E)$ und eine natürliche Zahl k.
Frage. Gibt es eine Teilmenge W von V, W von jeder Kante mindestens einen Knoten enthält aber insgesamt nicht mehr als k Elemente umfaßt? □

6.17 Problem. `WEIGHTED SET SPLITTING`.
Instanz. Gegeben sei eine natürliche Zahl k, ein Hypergraph $H = \big(V, \{ e_1, e_2 \}\big)$ mit n Knoten und m Kanten $e_1, \ldots, e_m$ sowie eine nichtnegative ganzzahlige Gewichtung c_{ij} für alle j aus V und i aus E der Knoten von H in Abhängigkeit von der Zugehörigkeit zu den Kanten von H, so daß c_{ij} Null ist, wenn der Knoten j nicht in e_i enthalten ist.
Frage. Gibt es eine Partition $P = \big(V, \{ f_1, f_2 \}\big)$, so daß die Beziehung

$$\min\big\{ c_{i \cdot \chi}(f_1), c_{i \cdot \chi}(f_2) \big\} \geq k$$

für alle i aus $\{ 1, \ldots, m \}$ gilt?

Der Bequemlichkeit halber soll auch das zugehörige Optimierungsproblem

$$\max_{P} \min_{i=1}^{m} \min\{\, c_i \cdot \chi\,(f_1), c_i \cdot \chi\,(f_2) \,\}$$

als `WEIGHTED SET SPLITTING` bezeichnet werden; es geht jeweils aus dem Zusammenhang hervor, welches Problem gemeint ist. □

6.3 Polynomial lösbare Probleme

6.18 Problem. `SINGLE STATE-VARIABLE MACHINE SEQUENCING PROBLEM` mit integrierbaren Funktionen f und g mit der Eigenschaft $f + g \geq 0$.
Instanz. Kostenmatrix C aus ${\mathbb{N}_0}^{m \times m}$ und m rationale Zweitupel (a_i, d_j) mit $d_1 \leq \ldots d_m$ und mit der Eigenschaft

$$c_{ij} = \left(\left\{ \begin{array}{ll} \int_{d_i}^{a_j} f(x)\,dx, & \text{falls } d_i \leq a_j \\ \int_{a_j}^{d_i} g(x)\,dx, & \text{falls } d_i \geq a_j \end{array} \right\} \right)$$

für alle i und j.
Frage. Was ist die kürzeste Tour im vollständigen Digraphen mit m Knoten und durch C definierten Bogengewichten? □

6.19 Problem. `TWO PROCESSOR ASSEMBLY LINE SCHEDULING`.
Instanz. Gegeben seien m Zahlentripel (d_i, a_i, s_i) aus $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}$, wobei $d_1 \leq \ldots \leq d_m$ gelte.
Frage. Welche verallgemeinerte Permutation σ auf $\{\, 1, \ldots, m \,\}$ mit Vielfachheiten s_i führt zu einer minimalen Gesamtbearbeitungszeit

$$\left(\{\, \max d_i, a_j \,\}\right)_{\substack{i=1,\ldots,m \\ j=1,\ldots,m}} \cdot \tau(\sigma)?$$

□

6.4 Symbolindex

$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_0$	Menge der natürlichen Zahlen und der Null
$\lvert M \rvert$	Kardinalität einer endlichen Menge M
$\chi(W)$	Inzidenzvektor einer Teilmenge W einer Grundmenge
2^M	Potenzmenge einer Menge M
$\setminus$	Mengendifferenz
$\{C\}$	Menge der Knoten eines gerichteten Kreises C
$\{\phi_i\}$	Menge, auf der ein Faktor ϕ_i einer Permutation operiert
$M = (m_{ij})$	Matrix
$m_{i\cdot}$	i-te Zeile der Matrix M
$\cdot^t$	Transposition
$H = (V, E)$	Hypergraph mit Knotenmenge V und Kantenmenge E
$H \cdot W$	durch Kontraktion von W entstandener Hypergraph
$H - W$	durch Einschränkung auf $V \setminus W$ entstandener Hypergraph
$\delta(W)$	von W induzierter Schnitt in einem Hypergraphen
$\delta^+(W,)\ \delta^-(W)$	von W induzierte Schnitte in einem Digraphen
$\mathrm{head}(a)$	Kopfende des Bogens a
$\mathrm{tail}(a)$	Schwanzende des Bogens a
D_m	vollständiger Digraph mit m Knoten
C^ϕ	Spaltenpermutierte Matrix $\left(c_{i\phi(j)}\right)$
$c(\phi)$	Kosten eines Assignments ϕ bezüglich einer Matrix C
$c\phi(\psi)$	Kosten einer Permutation bezüglich eines Assignments ϕ
ι	die identische Permutation
$\langle\cdot\rangle$	Codierungslänge bei binärer Codierung
$\flat$	Leerzeichen
$\mathcal{P}$	polynomial lösbare Entscheidungsprobleme
$\mathcal{NP}$	nichtdeterministisch polynomial lösbare Entscheidungsprobleme
Co-$\mathcal{NP}$	Verneinungen der Probleme aus $\mathcal{NP}$
$\wedge$	logisches und
$\vee$	logisches oder
$\overline{\cdot}$	logische Negation
$O(m)$	von der Ordnung m
$\lceil\cdot\rceil$	Größte-Ganze-Größergleich-Funktion
$f^{-1}(\{W\})$	Urbild der Menge W unter f
$(\cdot,\cdot)$	Skalarprodukt
$[\cdot,\cdot]$	abgeschlossenes Intervall

6.5 Stichwortindex