Prof. Dr. Frank Werner

Fakultät für Mathematik

Institut für Mathematische Optimierung

http://math.uni-magdeburg.de/∼werner/sched-ma.html

Scheduling

Vorlesungsskript
(auszugsweise)

Sommersemester 2019

Bemerkungen:

1. Dieses Skript ersetzt nicht den Besuch der Vorlesung. Zahlenbeispiele werden ausschließlich in der Vorlesung präsentiert.
2. Das Symbol ✏ weist auf zusätzliche, detaillierte Ausführungen in der Vorlesung hin.
3. Für die Unterstützung bei der Aufbereitung dieses Vorlesungsskriptes bedanke ich mich bei Frau Julia Lange.

Inhaltsverzeichnis

Literatur **iii**

1 Einführung **1**

2 Zur Komplexität von Scheduling-Problemen **6**
2.1 Grundlagen . . . 6
2.2 $\mathcal{NP}$-complete und $\mathcal{NP}$-hard Probleme . . . 8
2.3 Einfache Reduktionskriterien zwischen Scheduling-Problemen . . . 10
2.4 Polynomial lösbare und $\mathcal{NP}$-complete Scheduling-Probleme . . . 10

3 Basisalgorithmen **12**
3.1 Branch and Bound Algorithmen . . . 12
3.2 Dynamische Optimierung . . . 14
3.3 Lokale Suche . . . 15
3.3.1 Iterative Verbesserung . . . 16
3.3.2 Simulated Annealing . . . 16
3.3.3 Tabu Suche . . . 17
3.3.4 Genetische Algorithmen . . . 18
3.4 Approximationsverfahren . . . 19

4 Einmaschinenprobleme **22**
4.1 Das Problem $1 \mid prec \mid f_{max}$ . . . 22
4.2 Das Problem $1 \mid tree \mid \sum w_i C_i$ . . . 23
4.3 Das Problem $1 \mid C_i \leq d_i \mid \sum w_i C_i$ . . . 24
4.4 Das Problem $1 \mid r_i \geq 0, pmtn \mid \sum C_i$ . . . 25
4.5 Das Problem $1 \mid\mid \sum U_i$ . . . 25

5 Parallelmaschinenprobleme **27**
5.1 Das Problem $P \mid\mid \sum C_i$ . . . 27
5.2 Das Problem $P \mid pmtn \mid C_{max}$ . . . 27
5.3 Das Problem $P \mid\mid C_{max}$ . . . 28
5.4 Das Problem $P \mid tree, t_i = 1 \mid C_{max}$ . . . 29

6 Flow-Shop Probleme **31**
6.1 Das Problem $F \mid\mid C_{max}$ . . . 31
6.2 Das Problem $F \mid prmu \mid C_{max}$ . . . 32
6.2.1 Branch and Bound Verfahren . . . 32
6.2.2 Heuristische Verfahren . . . 32

7 Job-Shop Probleme **35**

7.1 Das Problem $J2 \mid n_i \leq 2 \mid C_{max}$ 35
7.2 Das Problem $J \mid\mid C_{max}$ 35
7.2.1 Branch and Bound Verfahren 35
7.2.2 Heuristische Verfahren 37
7.3 Probleme mit zwei Aufträgen 38
7.3.1 Das Problem $J \mid n = 2 \mid C_{max}$ 38
7.3.2 Das Problem $J \mid n = 2 \mid f$ 40

8 Open-Shop Probleme **41**

Literatur

1. Andresen, M.; Bräsel, H.; Engelhardt, F.; Werner, F.: : LiSA - A Library of Scheduling Algorithms, Handbuch zu Version 3.0, TR 10-02, FMA, OvGU Magdeburg, 2010, 107 Seiten (Download siehe Internetseite).
2. Brucker, P.: Scheduling Algorithms, Springer Verlag, 1995 (1998, 2001, 2004, 2007).
3. Blazewicz, J.; Ecker, K.; Schmidt, G.; Pesch, E.; Weglarz, J.: Scheduling Computer and Manufacturing Processes, Springer Verlag, 1993 (1994, 1996, 2001).
4. Domschke, W.; Scholl, A.; Voss, S.: Produktionsplanung, Ablauforganisatorische Aspekte, Springer Verlag, 1993 (1997).
5. Pinedo, M.: Planning and Scheduling in Manufacturing and Services, Springer Verlag, 2004 (2009).
6. Pinedo, M.: Scheduling: Theory, Algorithms and Systems, Prentice Hall, 1995 (2002).
7. Pinedo, M.; Chao, X.: Operations Scheduling with Applications in Manufacturing and Services, Irwin, McGraw Hill, 1999.
8. Parker, G.R.: Deterministic Scheduling Theory, Chapman & Hall, 1995.
9. Tanaev, V.S.; Gordon, V.A.; Strusevich, V.A.: Theory of Scheduling. Single-Stage Systems, Kluwer Academic Publishers, 1994.
10. Tanaev, V.S.; Sotskov, Y.N.; Shafransky, Y.M.: Theory of Scheduling. Multi-Stage Systems, Kluwer Academic Publishers, 1994.
11. Werner, F.: Genetic Algorithms for Shop Scheduling Problems: A Survey, Preprint 31/11, FMA, OvGU Magdeburg, 2011, 65 S.

Kapitel 1

Einführung

SCHEDULING:

- optimale Zuordnung von (beschränkten) Ressourcen zu Aktivitäten
- Zugehörigkeit zum Bereich des Decision Making
- Teilbereich des Operations Research

Ressourcen	**Aktivitäten**
Maschinen	*Aufträge*
Start- und Landebahnen auf einem Airport	Starts und Landungen von Flugzeugen
Prozessoren	Tasks
Flugzeuge	Flüge

Scheduling-Probleme in Fertigung und Service

(a) Fertigung

- Projektplanung und -scheduling
  (z.B. CPM und PERT)
- *Maschinenbelegung*
  (Job Scheduling)
- Fließbandabstimmung
- Losgrößenplanung
  (Economic Lot Scheduling Problem;
  Berücksichtigung von Setup- und Lagerhaltungskosten)
- Planung und Scheduling in Supply Chains

(b) Service

- Intervall-Scheduling, Reservierungen und Timetabling
  (z.B. Stundenplanung an Schulen und Universitäten)
- Scheduling und Timetabling in Sport und Unterhaltung
  (z.B. Planung von Sportwettkämpfen)

– Planung, Scheduling und Timetabling im Transport
(z.B. Tanker-Scheduling, Fahrplangestaltung bei der Eisenbahn)

– Personaleinsatzplanung (Workforce Scheduling)

deterministische Scheduling-Probleme

alle Daten zu den Aufträgen sind deterministisch fixiert

(a) *offline scheduling:*
alle Daten zu den Aufträgen sind a priori bekannt
(b) *online scheduling:*
die Daten zu den Aufträgen sind nicht a priori bekannt

$\longleftrightarrow$

stochastische Scheduling-Probleme

Zufallsvariable treten auf
(z.B. Bearbeitungszeiten)

Annahmen der klassischen Schedulingtheorie

1. Zu jeden Zeitpunkt t wird jeder Auftrag J_i auf höchstens einer Maschine M_j bearbeitet.
2. Zu jedem Zeitpunkt t wird auf jeder Maschine M_j höchstens ein Auftrag J_i bearbeitet.

Problemstellung und Bezeichnungen

gegeben:

n **Aufträge (Jobs)** $J_i: \; J_1, J_2, \ldots, J_n$
m **Maschinen** $M_j : M_1, M_2, \ldots, M_m$

- (i, j) – **Operation**: Bearbeitung von J_i auf M_j
- t_{ij} – **Bearbeitungszeit** von Operation (i, j)
- $S = \{(i, j) \mid t_{ij} > 0\}$ – **Menge der Operationen**
- Auftrag J_i $(i = 1, 2, \ldots, n)$ besitzt häufig eine vorgegebene **technologische Reihenfolge** (TRF):
$$M_{j_1} \to M_{j_2} \to \ldots \to M_{j_k}$$
Beschreibung durch Permutation $q^i = (j_1, j_2, \ldots, j_k)$
- die **organisatorische Reihenfolge** (ORF) auf Maschine M_j $(j = 1, 2, \ldots, m)$ ist zu wählen:
$$J_{i_1} \to J_{i_2} \to \ldots \to J_{i_l}$$
Beschreibung durch Permutation $p^j = (i_1, i_2, \ldots, i_l)$
- c_{ij} – Bearbeitungsende (Fertigstellzeitpunkt) der Operation (i, j)
- $C_i = \max_j\{c_{ij}\}$ – Bearbeitungsende (Fertigstellzeitpunkt) von Auftrag J_i

SCHEDULING-PROBLEM: (Maschinenbelegung)

Bestimme auf allen Maschinen eine organisatorische Reihenfolge so, dass alle vorgegebenen Bearbeitungsbedingungen eingehalten und ein vorgegebenes Optimierungskriterium erfüllt wird.

Klassifikationsschema für Scheduling-Probleme

$\alpha \mid \beta \mid \gamma$

$\rightarrow$ α: Beschreibung des **Bearbeitungssystems**

$\rightarrow$ β: Charakterisierung der **Bearbeitungsbedingungen** und **Restriktionen**

$\rightarrow$ γ: **Zielfunktion**

Parameter $\alpha \in \{F, J, O, P, \ldots\}$

- F **(Flow-Shop)**:
  Die technologischen Reihenfolgen sind vorgegeben und für alle Aufträge identisch.
- J **(Job-Shop)**:
  Für jeden Auftrag ist eine spezifische technologische Reihenfolge vorgegeben.
  (auch mehrmaliges Bearbeiten eines Auftrags auf einer Maschine ist möglich)
- O **(Open-Shop)**:
  Es sind keine technologischen Reihenfolgen für die Aufträge vorgegeben.
- P **(Parallel-Shop)**:
  Jeder Auftrag besteht aus **einer** Operation, die auf genau einer von m identischen Maschinen auszuführen ist.

Modifikationen:

Q – uniforme Maschinen
R – heterogene Maschinen

$\rightarrow$ $F3, O2, \ldots$ – Probleme mit fester Maschinenanzahl

Parameter $\beta \in \{r_i \geq 0,\ C_i \leq d_i,\ t_{ij} = 1,\ prec,\ tree,\ chain,\ pmtn,$
$no - wait,\ prmu, n = k,\ n_i \leq k,\ res, \ldots\}$

- $r_i \geq 0$ – Jeder Auftrag hat einen **Bereitstellungstermin** (release date) $r_i \geq 0$.
- $C_i \leq d_i$ – Jeder Auftrag hat einen **Fälligkeitstermin** (deadline; strong due date) $d_i \geq 0$, der einzuhalten ist.
- $t_{ij} = 1$ – Alle Operationen haben **Einheitsbearbeitungszeit**.
- $prec$ – Es bestehen **Vorrangbedingungen** zwischen den Aufträgen.

- *tree* – wie *prec*, jedoch haben die Vorrangbedingungen **Baumstruktur** (z.B. *outtree*).
- *chain* – Spezialfall von *tree*: Die Vorrangbedingungen bestehen aus ein oder mehreren Wegen.
- *pmtn* – Die **Unterbrechung** von Operationen ist (beliebig oft) erlaubt.
- *no-wait* – **Wartezeiten sind verboten**, d.h. das Bearbeitungsende einer Operation muss mit dem Bearbeitungsbeginn der nachfolgenden Operation des Auftrages übereinstimmen.
- *prmu* (Permutation) – Auf allen Maschinen ist die **gleiche organisatorische Reihenfolge** zu wählen.
- $n = k$ – Die Anzahl der Aufträge ist **konstant** und gleich k.
- $n_i \leq k$ – Die Anzahl der Operationen je Auftrag J_i ist **höchstens gleich** k.
- *res* – Es gelten **Ressourcenbeschränkungen** (z.B. Verbrauch an Energie, Bedarf an Arbeitskräften usw.; es existieren verschiedene Typen).

Parameter $\gamma \in \{C_{max}, \sum w_i C_i, L_{max}, \sum w_i T_i, \sum w_i U_i, \sum f_i(C_i), \sum \mid C_i - d_i \mid, \ldots\}$

Reguläres Kriterium: Minimierung einer monoton nichtfallenden Zielfunktion

Reguläre Kriterien:

- $C_{max} = \max\{C_i \mid i = 1, \ldots, n\}$ – (Minimierung der) **Gesamtbearbeitungszeit** (Zyklusdauer, Makespan)
- $\sum w_i C_i$ – **gewichtete Summe der Bearbeitungsendtermine**
- $L_{max} = \max\{C_i - d_i \mid i = 1, \ldots, n\}$ – **maximale 'Verspätung'** (Lateness)
- $\sum w_i T_i = \sum w_i \max\{0, C_i - d_i\}$ – **gewichtete Summe der Terminüberschreitungen**
- $\sum w_i U_i$ – **gewichtete Summe der verspäteten Aufträge**
- $\sum f_i(C_i)$ – (Minimierung der) **Gesamtkosten**
  (f_i beliebig, monoton nichtfallend)

Beispiel für **nichtreguläres Kriterium**:

- $\sum \mid C_i - d_i \mid$ – (Minimierung der) Summe der zeitlichen Abstände der Bearbeitungsendtermine von den Fälligkeitsterminen (soft due dates) d_i
  (**MAD Kriterium**, just-in-time Produktion)

Darstellung von Plänen und Beispiele zum Klassifikationsschema für Scheduling-Probleme ✏

1. Prüfung der Zulässigkeit eines Plans
2. Maschinen- und auftragsorientiertes Ganttdiagramm
3. BEISPIEL 1.1: $1||\sum w_i T_i$
4. BEISPIEL 1.2: $J||C_{max}$
5. BEISPIEL 1.3: $P2|tree,\ r_i \geq 0|\sum w_i C_i$

Kapitel 2

Zur Komplexität von Scheduling-Problemen

Komplexität: größenordnungsmäßige Erfassung des Aufwands zur Abarbeitung eines Algorithmus bzw. zur Lösung eines Problems in Abhängigkeit vom Umfang der Eingangsdaten

→ Mittel zur Beurteilung der **Qualität von Algorithmen** bzw. des **Schwierigkeitsgrades eines Problems**

2.1 Grundlagen

Optimierungsproblem (Minimierung):

Bestimme einen Plan x^* mit

$$F(x^*) = \min\{F(x) \mid x \in S,\ S \text{ endlich}\}.$$

Entscheidungsproblem:

beinhaltet eine Fragestellung, die entweder mit *ja* oder *nein* beantwortet werden kann

Zuordnung eines Entscheidungsproblems zu einem Scheduling-Problem:

gegeben: Zahl k
Frage: Gibt es einen zulässigen Plan $x \in S$ mit

$$F(x) \leq k\ ? \qquad \Big(SP(k)\Big)$$

Die **Komplexität** wird in der Form $O(h(d))$ angegeben mit der Bedeutung

$$\lim_{d\to\infty} \frac{O(h(d))}{h(d)} = const \neq 0,$$

wobei d den Umfang der Eingangsdaten beschreibt.

Illustration: Komplexität ✏

$h(d)$ **polynomial beschränkt** $\rightarrow$ **Algorithmus polynomial** bzw. das **Problem ist polynomial lösbar** (andernfalls exponentiell)

$\Rightarrow$ 2 Fälle: polynomial beschränkt (n^a) und exponentiell beschränkt (a^n)

Veranschaulichung des Unterschiedes polynomialer bzw. exponentieller Algorithmus

Annahme: Computer führe 10^6 Operationen je Sekunde aus, Rechenzeiten in Abhängigkeit von n und der Zeitkomplexität eines Algorithmus:

Zeitkomplexität	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s	0,0036 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1,0 s	17,9 min	12,7 Tage	35,7 Jahre	366 Jhd.

Frage: Helfen zukünftigere schnellere Computer? (nicht wirklich bei exponentieller Zeitkomplexität!)

Zeitkomplexität	Größe auf einem heutigen Computer	100fach schnellerer Computer	1000fach schnellerer Computer
n	A	$100A$	$1000A$
n^2	B	$10B$	$31,6B$
n^5	C	$2,5C$	$3,98C$
2^n	D	$D + 6,64$	$D + 9,97$

$\mathcal{P}$: Klasse der mit **polynomialem Aufwand lösbaren Entscheidungsprobleme**

Nichtdeterministischer Algorithmus:

Neben Operationen von deterministischen Algorithmen werden weitere Operationen definiert:

(a) *choice(S)*: Aus der endlichen Menge S willkürlich ein Element ausgewählen.

(b) *success*: signalisiert eine erfolgreiche Beendigung des Algorithmus

(c) *failure*: signalisiert Beendigung von Rechnungen ohne Erfolg

Nichtdeterministischer Rechner:

fiktiver Rechner, der einen nichtdeterministischen Algorithmus wie folgt ausführt:

- Falls die *choice* - Operationen so durchgeführt werden können, dass man das *success* - Signal erreicht, so führt der Rechner die *choice* - Operationen so durch, dass das *success* - Signal auf dem schnellsten Wege erreicht wird.

- Ist es nicht möglich, das *success* - Signal zu erreichen, bricht der Rechner ohne Erfolg ab (Rechenaufwand 1).

BEISPIEL 2.1: $P2||C_{max}$ ✏

$\mathcal{NP}$: Klasse der mittels eines **nichtdeterministischen Algorithmus mit polynomialem Aufwand lösbaren Entscheidungsprobleme**

Bemerkung: Für ein Problem aus $\mathcal{NP}$ lässt sich in polynomialer Zeit überprüfen, ob eine Vermutung erfüllt ist, d.h. ob eine 'geratene' Lösung eine Ja-Antwort zu dem Entscheidungsproblem liefert. Insbesondere bedeutet dies, dass bei einem Problem aus $\mathcal{NP}$ der Zielfunktionswert in polynomialer Zeit berechnet werden kann.

klar: $\mathcal{P} \subseteq \mathcal{NP}$

aber offene Frage: $\mathcal{P} \neq \mathcal{NP}$?

2.2 $\mathcal{NP}$-complete und $\mathcal{NP}$-hard Probleme

Reduzierbarkeit von Problemen

Definition 1: Ein Problem P_1 heißt **reduzierbar** auf ein Problem P_2 (in Zeichen $P_1 \ \alpha \ P_2$), wenn zu **jedem** Beispiel B für P_1 mit Hilfe eines polynomial beschränkten deterministischen Algorithmus ein Beispiel $f(B)$ von P_2 erzeugt werden kann, so dass das zu B gehörige Problem **genau dann** lösbar ist, wenn das zu $f(B)$ gehörige Problem lösbar ist.

Definition 2: Zwei Probleme P_1 und P_2 heißen **polynomial äquivalent** (in Zeichen $P_1 \sim P_2$), wenn $P_1 \ \alpha \ P_2$ und $P_2 \ \alpha \ P_1$.

Definition 3: Mit **$\mathcal{NP}$-complete** wird die Klasse aller Probleme $Q \in \mathcal{NP}$ mit $P \ \alpha \ Q$ für alle $P \in \mathcal{NP}$ bezeichnet.

Theorem 1: Seien P_1, $P_2 \in \mathcal{NP}$. Aus $P_1 \in \mathcal{NP}$-complete und $P_1 \ \alpha \ P_2$ folgt $P_2 \in \mathcal{NP}$-complete.

Schritte zum Nachweis der Zugehörigkeit eines Problems P^* zur Klasse $\mathcal{NP}$-complete:

1. Zeige $P^* \in \mathcal{NP}$.
2. Suche ein bekanntes Problem $Q \in \mathcal{NP}$-complete mit

$$Q \; \alpha \; P^*.$$

Satisfizierbarkeitsproblem (SAT)

gegeben:

- endliche Menge U von Booleschen Variablen
- Familie F von Ausdrücken über U

Frage: Gibt es eine Zuordnung von wahr und falsch zu allen Elementen von U, so dass alle Ausdrücke von F erfüllt (d.h. wahr) sind?

Cook (1971)

Das Satisfizierbarkeitsproblem gehört zur Klasse $\mathcal{NP}$-complete.

Einige weitere Probleme

3-SAT: 3-Satisfizierbarkeitsproblem

3DM: 3-dimensionales Matching

PART: Partitionsproblem

$P \; \rightarrow \; Q$ entspricht $P \; \alpha \; Q$:

$$\boxed{\text{SAT}} \longrightarrow \boxed{\text{3-SAT}} \longrightarrow \boxed{\text{3DM}} \longrightarrow \boxed{\text{PART}}$$

Folgerung: Ist **ein** Problem aus $\mathcal{NP}$-complete mit Hilfe eines deterministischen Algorithmus polynomial lösbar, dann folgt

$$\mathcal{P} = \mathcal{NP},$$

d.h. dann wären alle Probleme aus $\mathcal{NP}$-complete polynomial lösbar, daher die **Vermutung**:

$$\mathcal{P} \neq \mathcal{NP}.$$

Beispiel 2.2: Reduzierbarkeit ✏

Definition 4: Ein allgemeines Problem P (nicht notwendig Entscheidungsproblem) gehört zur Klasse $\mathcal{NP}$**-hard**, wenn es mindestens so schwierig ist wie ein Problem der Klasse $\mathcal{NP}$-complete.

Illustration: Problemklassen ✏

Zusammenfassung

1. Die Problem aus $\mathcal{NP}$-complete sind polynomial äquivalent, sie stellen die **schwierigsten** Probleme in $\mathcal{NP}$ dar.
2. Für Probleme aus $\mathcal{NP}$-complete bzw. $\mathcal{NP}$-hard ist eine (optimale) Lösung häufig nicht mit vertretbarem Rechenaufwand erhältlich. Daher ist die **Entwicklung von Näherungsverfahren** für derartige Problem sinnvoll und notwendig.

2.3 Einfache Reduktionskriterien zwischen Scheduling-Problemen

Einem Problem $\alpha \mid \beta \mid \gamma$ wird ein 7-Tupel $(v_1, v_2, \ldots, v_7)$ mit $v_i \in G_i$ für $i = 1, \ldots, 7$ von jeweils einem Knoten aus den Graphen $G_1, G_2, \ldots, G_7$ zugeordnet, wobei

G_1 bzgl. α
$G_2, G_3, \ldots, G_6$ bzgl. β
G_7 bzgl. γ

Illustration: Graphen G_i ✏

Theorem 2: Gegeben sind die Probleme P und Q mit den zugeordneten 7-Tupeln $(v_1, v_2, \ldots, v_7)$ und $(v_1^*, v_2^*, \ldots, v_7^*)$. Dann ist das Problem P auf das Problem Q reduzierbar (in Zeichen $P\ \alpha\ Q$), wenn für alle $i = 1, 2, \ldots, 7$ entweder $v_i = v_i^*$ erfüllt ist oder ein Weg von v_i nach v_i^* in G_i existiert.

2.4 Polynomial lösbare und $\mathcal{NP}$-complete Scheduling-Probleme

Charakterisierung der **Trennlinie** zwischen "gut lösbaren" und "schwierigen" Problemen

dabei:

1. Angabe der "**einfachsten**" Probleme, die zur Klasse $\mathcal{NP}$**-complete** gehören
2. Angabe der "**schwierigsten**" Probleme, die noch polynomial lösbar sind

→ siehe Tabelle Seite **??**

Einige weitere Reduktionen

Theorem 3: Es gilt

(a) Rucksack $\alpha\ 1||\sum w_i U_i$
(b) PART $\alpha\ P2||C_{max}$
(c) PART $\alpha\ O3||C_{max}$

Beispiel 2.3: Anwendung der Reduktionskriterien ✏

(Entscheidungs-)Probleme in $\mathcal{P}$ und $\mathcal{NP}$-complete

$\mathcal{P}$	$\mathcal{NP}$-complete
(a) Einmaschinenprobleme	
$1 \mid prec, r_i \geq 0 \mid C_{max}$ $O(n^2)$ Lawler (1973)	$1 \mid r_i \geq 0 \mid L_{max}$
$1 \mid prec \mid f_{max}$ $O(n^2)$ Lawler (1973)	
$1 \mid tree \mid \sum w_i C_i$ $O(n \log n)$ Horn (1972)	$1 \mid prec, t_i = 1 \mid \sum w_i C_i$ $1 \mid r_i \geq 0 \mid \sum C_i$ $1 \mid C_i \leq d_i \mid \sum w_i C_i$
$1 \mid\mid \sum U_i$ $O(n \log n)$ Moore (1968)	$1 \mid tree, t_i = 1 \mid \sum U_i$ $1 \mid r_i \geq 0 \mid \sum U_i$ $1 \mid\mid \sum w_i U_i$
$1 \mid r_i \geq 0, t_i = 1 \mid \sum f_i(C_i)$ $O(n^3)$ Lawler (1974) für $f_i = w_i T_i$ (Lösung eines Zuordnungsproblems)	$1 \mid chain, t_i = 1 \mid \sum T_i$ $1 \mid\mid \sum T_i$
(b) Parallelmaschinenprobleme	
$P \mid tree, t_i = 1 \mid C_{max}$ $O(n)$ Hu (1961)	$P2 \mid prec, 1 \leq t_{ij} \leq 2 \mid C_{max}$ $P \mid prec, t_i = 1 \mid C_{max}$ $P2 \mid\mid C_{max}$
$Q2 \mid pmtn, prec \mid L_{max}$ $O(n^2)$ Lawler (1982)	
$P2 \mid prec, r_i \geq 0, t_i = 1 \mid L_{max}$ $O(n^3 \log n)$ Garey & Johnson (1977)	
$P2 \mid prec, t_i = 1 \mid \sum C_i$ $O(n^2)$ Coffman & Graham (1972)	$P2 \mid prec, t_i = 1 \mid \sum w_i C_i$
$R \mid\mid \sum C_i$ $O(n^3 m)$ Bruno et al. (1974)	$P2 \mid\mid \sum w_i C_i$
(c) Flow-Shop Probleme	
$F2 \mid C_{max}$ $O(n \log n)$ Johnson (1954)	$F2 \mid\mid L_{max}$ $F2 \mid\mid \sum C_i$ $F2 \mid r_i \geq 0 \mid C_{max}$ $F3 \mid\mid C_{max}$
(d) Job-Shop Probleme	
$J2 \mid n_i \leq 2 \mid C_{max}$ $O(n \log n)$ Jackson (1956)	$J2 \mid 1 \leq t_{ij} \leq 2 \mid C_{max}$ $J2 \mid\mid \sum C_i$
$J2 \mid t_{ij} = 1 \mid L_{max}$ $O(n^2)$ Timkovsky (1994)	$J3 \mid t_{ij} = 1 \mid C_{max}$
(e) Open-Shop Probleme	
$O2 \mid\mid C_{max}$ $O(n)$ Gonzalez & Sahni (1976)	$O2 \mid r_i \geq 0 \mid C_{max}$ $O2 \mid\mid L_{max}$ $O2 \mid\mid \sum C_i$ $O3 \mid\mid C_{max}$
$O \mid pmtn, r_i \geq 0 \mid L_{max}$ LP Cho & Sahni (1981) *Spezialfall:* $O \mid pmtn \mid C_{max}$ $O(r^2(n+m)^{0,5})$, r - Anz. d. Oper. mit $t_{ij} > 0$	

Kapitel 3

Einige Basisalgorithmen zur exakten und näherungsweisen Lösung von Scheduling-Problemen

S – **endliche** Menge von Lösungen

$f : S \to \mathbb{R}$

gesucht: $s^* \in S$ mit

$$f(s^*) = \min\{f(s) \mid s \in S\} \qquad (P)$$

kombinatorisches Optimierungsproblem

3.1 Branch and Bound Algorithmen

- **exaktes** Verfahren der impliziten Enumeration
- **intelligentes Enumerieren** der zulässigen Lösungen (bei $\mathcal{NP}$-hard Problemen)

(a) **Branching:**

Ersetze Problem P durch k Teilprobleme P_i $(i = 1, 2, \ldots, k)$, für deren Lösungsmengen S_i gilt:

$$\cup_{i=1}^{k} \; S_i = S$$

Illustration: Verzweigungsbaum ✏

(b) **Bounding:**

$V = \{P, P_1, P_2, \ldots\}$ – **Knotenmenge**
X – **Bogenmenge**

→ **untere Schrankenfunktion** $LB : V \to \mathbb{R} \cup \{\infty\}$ mit

1. $LB(P_i) \leq f_0(P_i)$ für $P_i \in V$
2. $LB(P_i) = f_0(P_i)$ falls $|S_i| = 1$
3. $LB(P_i) \leq LB(P_j)$ für $(P_i, P_j) \in X$

→ **obere Schranke** UB mit $UB \geq f_0(P)$

(man beachte: ZF → min!)

Zur Aufspaltung eines Teilproblems

- *Partiallösungskonzept*
  S hat eine solche Struktur, die eine Zerlegung in Komponenten erlaubt.
- *Alternativkonzept*
  Das Problem P_k wird in zwei Teilprobleme P_{k1} und P_{k2} aufgespalten, wobei $S_{k1} \cup S_{k2} = S_k$ und $S_{k1} \cap S_{k2} = \emptyset$

BEISPIEL 3.1: Veranschaulichung der Konzepte der Aufspaltung $F|prmu|C_{max}$ ✏

Zur Auswahl eines Teilproblems, das nicht von den weiteren Untersuchungen ausgeschlossen ist

- *Best Bound Search* (Minimalstrategie)
- *Depth First Search* (LIFO, Tiefensuche)
- *Breadth First Search* (Breitensuche)

Algorithmus *Branch & Bound*

1. $LIST := P$;
2. $UB :=$ Zielfunktionswert einer heuristischen Lösung $s \in S$; $BL := s$;
   WHILE $LIST \neq \emptyset$ **DO**
   BEGIN
3. wähle ein Teilproblem k aus $LIST$;
4. streiche k in $LIST$;
5. erzeuge die Nachfolger $i = 1, 2, \ldots, n_k$ und berechne die zugehörigen unteren Schranken $LB(i)$;
6. **FOR** $i := 1$ **TO** n_k **DO**
   IF $LB(i) < UB$ **THEN**
   IF enthält i ein $s \in S$ mit $f(s) = LB(i)$
   THEN
   BEGIN $UB := LB(i); BL := s$ **END**
   ELSE nimm i in $LIST$ auf
   END

3.2 Dynamische Optimierung

- **exaktes** Verfahren
- Zerlegung des Ausgangsproblems in Teilprobleme, die sukzessiv behandelt werden (**n-stufiger Entscheidungsprozess**)
- In jeder Stufe i $(1 \leq i \leq n)$ wird aus einem **Eingangszustand** z_i mittels einer zulässigen **Entscheidung** x_i ein **Ausgangszustand**

  $$z_{i-1} = t_i(z_i, x_i)$$

  erzeugt.
- $f_i(z_i, x_i)$ – der sich aus z_i und x_i ergebende **Beitrag der i-ten Stufe zur Gesamtzielfunktion**

Ziel:

Bestimme für einen fest vorgegebenen Anfangszustand z_n des Prozesses eine **optimale Entscheidungsfolge**

$$(x_n, x_{n-1}, \ldots, x_1) \ ,$$

so dass

$$F = \sum_{i=1}^{n} f_i(z_i, x_i) \rightarrow \min!$$

Illustration: Dynamische Optimierung ✏

Bellmansches Optimalitätsprinzip

Falls die Entscheidungsfolge

$$(x_n, x_{n-1}, \ldots, x_i, \ldots, x_1)$$

optimal ist, so ist auch die Folge der letzten Entscheidungen

$$(x_i, x_{i-1}, \ldots, x_1)$$

optimal für den aus den Stufen $i, i-1, \ldots, 1$ bestehenden Restprozess bezüglich des Eingangszustandes z_i, der aus den Entscheidungen

$$(x_n, x_{n-1}, \ldots, x_{i+1})$$

resultiert.

Bellmansche Rekursionsgleichungen

$$F_i(z_i) = \min_{x_i \in X_i(z_i)} \left\{ f_i(z_i, x_i) + F_{i-1}(t_i(z_i, x_i)) \right\}$$
$$\text{mit } F_0(z_0) = 0$$

$F_i(z_i)$ – **optimaler Zielfunktionswert** für den aus den Stufen $i, i-1, \ldots, 1$ bestehenden **Teilprozess** mit Eingangszustand z_i

$\Longrightarrow F_n(z_n)$ – **optimaler Zielfunktionswert** für den **Gesamtprozess** mit Eingangszustand z_n

Beispiel 3.2: Dynamische Optimierung am Problem $1 \mid\mid \sum w_i U_i$ ✏

3.3 Lokale Suche

- **iterative Näherungsverfahren**
- Einführung einer **Nachbarschaftsstruktur**

$$N : S \to 2^S$$

- Beschreibung der Nachbarschaftsstruktur durch einen **gerichteten (oder ungerichteten) Graphen** $G = (V, E)$ mit $V = S$ und

$$(s, t) \in E \qquad \Longleftrightarrow \qquad t \in N(s)$$

G – **Nachbarschaftsgraph**

Beispiele für Nachbarschaftsgraphen bei Permutationsproblemen:

- *Left Shift Nachbarschaft*
- *Right Shift Nachbarschaft*
- *Shift Nachbarschaft*
- *Pairwise Interchange Nachbarschaft*
- *Adjacent Pairwise Interchange (API) Nachbarschaft*

Beispiel 3.3: Nachbarschaften für Permutationsprobleme ✏

Lokale Suchalgorithmen:

- einfachste Variante:
  iterative Verbesserung (3.3.1)

- **Metaheuristiken:**
  Simulated Annealing (3.3.2)
  Threshold Accepting
  Tabu Suche (3.3.3)
- **populationsorientierte Verfahren:**
  genetische Algorithmen (3.3.4)
  Ameisenalgorithmen
  Partikelschwarmoptimierung

3.3.1 Iterative Verbesserung

Algorithmus *Iterative Verbesserung*

1. bestimme eine Startlösung $s \in S$;
   REPEAT
2. erzeuge eine Lösung $s' \in N(s)$;
3. **IF** $f(s') < f(s)$ **THEN** $s := s'$;
   UNTIL $f(s') \geq f(s)$ für alle $s' \in N(s)$.

3.3.2 Simulated Annealing

randomisiertes Verfahren, da

- $s' \in N(s)$ wird zufällig ausgewählt
- im i-ten Schritt wird s' mit Wahrscheinlichkeit

$$\min\left\{1, e^{\left(-\frac{f(s')-f(s)}{t_i}\right)}\right\}$$

als neue Startlösung akzeptiert

Algorithmus *Simulated Annealing*

1. $i := 0$; wähle t_0;
2. bestimme eine Startlösung $s \in S$;
3. $best := f(s)$;
4. $s^* := s$;
   REPEAT
5. erzeuge zufällig eine Lösung $s' \in N(s)$;
6. **IF** $rand[0,1] < \min\left\{1, e^{\left(-\frac{f(s')-f(s)}{t_i}\right)}\right\}$ **THEN** $s := s'$;
7. **IF** $f(s') < best$ **THEN**
   BEGIN $s^* := s'$; $best := f(s')$ **END**;
8. $t_{i+1} := g(t_i)$;
9. $i := i + 1$;
   UNTIL Abbruchkriterium ist erfüllt.

Modifikation:

Threshold Accepting (deterministische Variante von Simulated Annealing)

$\rightarrow$ akzeptiere $s' \in N(s)$ falls

$$f(s') - f(s) < t_i$$

t_i – *Threshold* im i-ten Schritt

3.3.3 Tabu Suche

Ziel: Vermeidung von 'Kurzzyklen'

$\Rightarrow$ nutze Attribute, um die zuletzt besuchten Lösungen zu charakterisieren und für eine gewisse Anzahl von Iterationen die Rückkehr zu solchen Lösungen auszuschließen

Bezeichnungen:

$Cand(s)$	–	enthält die Nachbarn $s' \in N(s)$, zu denen ein Übergang erlaubt ist
TL	–	Tabu-Liste
t	–	Länge der Tabu-Liste

Algorithmus *Tabu Suche*

1. bestimme eine Startlösung $s \in S$;
2. $best := f(s)$;
3. $s^* := s$;
4. $TL := \emptyset$;
   REPEAT
5. bestimme $Cand(x) = \{\ s' \in N(s) \mid$ der Übergang von s zu s' ist nicht tabu oder s' erfüllt das Aspiration-Kriterium $\}$;
6. wähle eine Lösung $\overline{s} \in Cand(s)$;
7. aktualisiere TL (so dass maximal t Attribute in TL sind);
8. $s := \overline{s}$;
9. **IF** $f(\overline{s}) < best$ **THEN**
   BEGIN $s^* := \overline{s}$; $best := f(\overline{s})$ **END**;
   UNTIL Abbruchkriterium ist erfüllt.

3.3.4 Genetische Algorithmen

- Nutzung von **Darwin's Evolutionstheorie**
- Genetische Algorithmen arbeiten mit einer **Population von Individuen** (Chromosomen), die durch ihre Fitness charakterisiert werden.
- Erzeugung von Nachkommen mittels **genetischer Operatoren** (Crossover, Mutationen)

Algorithmus *Gen-Alg*

1. setze Parameter Populationsgröße $POPSIZE$, maximale Generationenanzahl $MAXGEN$, Wahrscheinlichkeit P_{CO} für die Anwendung von Crossover und Wahrscheinlichkeit P_{MU} für die Anwendung einer Mutation;
2. erzeuge die Startpopulation POP_0 mit $POPSIZE$ Individuen (Chromosomen);
3. bestimme die Fitness aller Individuen;
4. $k := 0$;
   WHILE $k < MAXGEN$ **DO**
   BEGIN
5. $h := 0$;
   WHILE $h < POPSIZE$ **DO**
   BEGIN
6. wähle zwei Eltern aus POP_k aus (z.B. zufällig proportional ihrer Fitness-Werte oder gemäss Roulette Wheel Selection);
7. wende mit Wahrscheinlichkeit P_{CO} auf die ausgewählten Eltern ein Crossover an;
8. wende jeweils mit Wahrscheinlichkeit P_{MU} auf die entstandenen Individuen eine Mutation an;
9. $h := h + 2$;
   END;
10. $k := k + 1$;
11. wähle aus den erzeugten Nachkommen (bzw. ggf. auch den Eltern) $POPSIZE$ Individuen der k-ten Generation POP_k aus (z.B. proportional ihrer Fitness-Werte);
    END

Genetische Algorithmen bei Permutationsproblemen

→ oft Codierung einer Lösung mittels Job-Code

$$p = (2, 1, 3, 4, 5) \quad \longrightarrow \quad \text{ch: } \boxed{2}\boxed{1}\boxed{3}\boxed{4}\boxed{5}$$

Mutationen

Sei $\boxed{2}\boxed{1}\boxed{3}\boxed{4}\boxed{5}$ das Elternchromosom

(a) (i, j)-Austausch
$(2, 4)$-Austausch: $\boxed{2}\boxed{\mathbf{4}}\boxed{3}\boxed{\mathbf{1}}\boxed{5}$

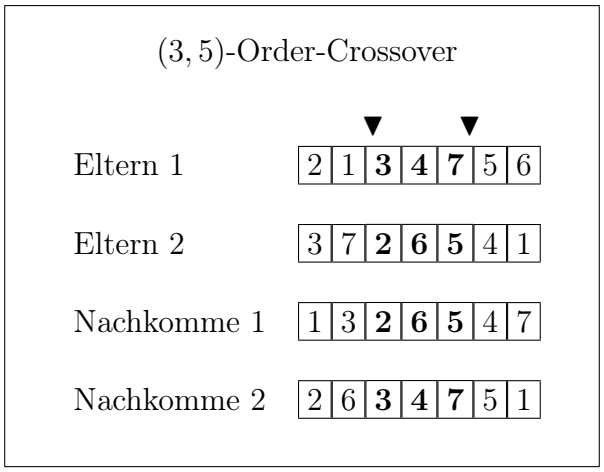
(b) (i,j)-Shift
$(1,4)$-Shift: | 1 | 3 | 4 | **2** | 5 |

(c) (i,j)-Inversion
$(3,5)$-Inversion: | 2 | 1 | **5** | **4** | **3** |

Crossover

(i,j)-Order-Crossover (LOX-Crossover)

Vertausche die Gene von den zwei ausgewählten Elternchromosomen auf den Positionen i bis j und erhalte die Reihenfolge der restlichen Gene des ursprünglichen Elternchromosoms. Somit wird die Zulässigkeit der Nachkommen gewährleistet.



3.4 Approximationsverfahren

→ Näherungsverfahren mit garantierter Genauigkeit

Bezeichnungen:

I	–	Beispiel ('Instance') eines Problems vom Typ (P)
$f^{opt}(I)$	–	optimaler Zielfunktionswert für das Beispiel I
$f^{H}(I)$	–	Zielfunktionswert für das Beispiel I bei Anwendung eines Algorithmus H

Annahme: $f^{opt}(I) > 0$ – für alle Beispiele I

Definition 1: Ein Algorithmus H heißt **ϵ-Approximationsalgorithmus** für Problem (P), wenn für **jedes** Beispiel I von (P) die Abschätzung

$$\frac{\mid f^H(I) - f^{opt}(I) \mid}{f^{opt}(I)} \leq \epsilon$$

gilt.

Frage: Existiert ein Algorithmus, so dass für jedes Beispiel die Abschätzung aus Def. 1 gilt?

Definition 2: Ein Algorithmus H heißt **ϵ-Approximationsschema** für Problem (P), wenn für **jedes** Beispiel I von (P) und **jedes** $\epsilon > 0$ die Abschätzung

$$\frac{\mid f^H(I) - f^{opt}(I) \mid}{f^{opt}(I)} \leq \epsilon$$

gilt.

Algorithmus H heißt **polynomiales Approximationsschema** (PAS), falls die Laufzeit von H für ein **festes** $\epsilon > 0$ durch ein Polynom in der Inputlänge des Problems beschränkt ist.

Ein polynomiales Approximationsschema heißt **vollpolynomiales Approximationsschema** (VPAS), falls zusätzlich die Laufzeit durch ein Polynom in $1/\epsilon$ beschränkt ist (d.h. die Laufzeit ist durch ein Polynom in der Inputlänge **und** $1/\epsilon$ beschränkt).

Bemerkungen:

1. Vollpolynomiale Approximationsschemata sind die 'besten' Näherungsverfahren, die man für ein Problem der Klasse NP-hard erwarten kann. Leider ist die Existenz solcher Verfahren im Fall $\mathcal{P} \neq \mathcal{NP}$ nicht für jedes Problem gegeben.
2. Die folgenden Komplexitäten verdeutlichen die Unterschiede zwischen polynomialen und vollpolynomialen Approximationsschemata:

$O(n^4/\epsilon)$	alle polynomial in n	$O(10n^4)$
$O(n^{1/\epsilon^2})$	für $\epsilon = 0,1$	$O(n^{100})$
$O(n^{2 \cdot 1/\epsilon})$	erhält man:	$O(n^{20})$

Nur $O(n^4/\epsilon)$ ist durch ein Polynom in n **und** $1/\epsilon$ beschränkt!

Zur Veranschaulichung eines vollpolynomialen Approximationsschemas wird in der Vorlesung folgendes **Einmaschinenproblem** betrachtet: ✏

Für jeden Auftrag J_i $(i = 1, 2, \ldots, n)$ ist gegeben:

- die Bearbeitungszeit t_i;
- der Due Date d_i und
- ein Gewinn g_i, der erzielt wird, falls $C_i \leq d_i$ (andernfalls wird kein Gewinn erzielt)

gesucht: optimale Reihenfolge mit maximalem Gewinn

Bemerkung:

Der Zusammenhang zu den bisher betrachteten Scheduling-Problemen ergibt sich durch Umformung der Zielfunktion:

$$f(x) = \sum_{i=1}^{n} g_i(1 - U_i) = \sum_{i=1}^{n} g_i - \sum_{i=1}^{n} g_i U_i.$$

Da $\sum_{i=1}^{n} g_i$ konstant ist, entspricht die Maximierung von $f(x)$ der Minimierung von

$$\tilde{f}(x) = \sum g_i U_i,$$

d.h. es liegt ein Problem vom Typ $1 \,||\, \sum w_i U_i$ vor.

Kapitel 4

Einmaschinenprobleme

4.1 Das Problem $1 \mid prec \mid f_{max}$

- f_i beliebige monoton nichtfallende Kostenfunktion $(i = 1, 2, \ldots, n)$
- $A = (a_{ij})$ mit

$$a_{ij} = \begin{cases} 1 & \text{falls Vorrangbedingung } i \to j \\ 0 & \text{sonst} \end{cases}$$

Algorithmus $1 \mid prec \mid f_{max}$ *(Lawler 1973)*

1. **FOR** $i := 1$ **TO** n **DO** $N_i := \sum_{j=1}^{n} a_{ij}$
2. $T := \sum_{i=1}^{n} t_i$;
3. $L := \{1, 2, \ldots, n\}$;
4. $k := n$;
   WHILE $L \neq \emptyset$ **DO**
   BEGIN
5. ermittle $i \in L$ mit $N_i = 0$ und minimalem $f_i(T)$;
6. $N_i := \infty$;
7. **FOR** $h := 1$ **T0** n **DO**
   IF $a_{hi} = 1$ **THEN** $N_h := N_h - 1$;
8. $L := L \setminus \{i\}$; $\overline{p}_k := i$;
9. $T := T - t_i$; $k := k - 1$;
   END

Komplexität des Algorithmus: $O(n^2)$

Theorem 1: Die mit dem Algorithmus $1 \mid prec \mid f_{max}$ konstruierte Lösung ist optimal.

Beispiel 4.1: $1|prec|f_{max}$ ✏

4.2 Das Problem $1 \mid tree \mid \sum w_i C_i$

Betrachtung der Variante: **outtree** (intree analog)

Bezeichnungen: ($U \subseteq \{1, 2, \ldots, n\}$)

$$\begin{aligned} t(U) &= \sum\{t_u \mid u \in U\} \\ w(U) &= \sum\{w_u \mid u \in U\} \\ q(U) &= \frac{w(U)}{t(U)} \text{ für } U \neq \emptyset \end{aligned}$$

Lemma 1: *(Blockvertauschungslemma)*

Seien

$$p = (p(U_1), p(U_2), p(U_3), p(U_4)) \qquad \text{und} \qquad p' = (p(U_1), p(U_3), p(U_2), p(U_4)).$$

Dann gilt im Fall $U_2, U_3 \neq \emptyset$:

$$f(p') \leq f(p) \qquad \Longleftrightarrow \qquad q(U_3) \geq q(U_2).$$

$$\frac{w(U_3)}{t(U_3)} \geq \frac{w(U_2)}{t(U_2)} \quad \rightarrow \text{ Optimalitätsbedingung}$$

Theorem 2: Sei

$$\underline{V}(i) = \emptyset \qquad \text{und} \qquad q(i) = \max\{q(j) \mid j = 1, 2, \ldots, n\}.$$

Dann existiert eine optimale Reihenfolge $\overline{p}$, die mit Auftrag J_i beginnt, d.h. $\overline{p}_1 = i$.

Theorem 3: Sei

$$\underline{V}(i) = k \qquad \text{und} \qquad q(i) = \max\{q(j) \mid j = 1, 2, \ldots, n\}.$$

Dann existiert eine optimale Reihenfolge $\overline{p}$, in der Auftrag J_k unmittelbar vor Auftrag J_i steht.

Folgerung: Unter den Voraussetzungen von Theorem 3 ist das Problem $1 \mid tree \mid \sum w_i C_i$ mit n Aufträgen äquivalent zu einem Problem mit $n - 1$ Aufträgen:

Ersetze die Aufträge J_k und J_i durch einen neuen Auftrag J_{ki} mit

$$t_{ki} = t_k + t_i \qquad \text{und} \qquad w_{ki} = w_k + w_i$$

und ändere den Graphen der Vorrangbedingungen entsprechend.

Ist

$$p' = (p(U_1), ki, p(U_2))$$

optimal für das reduzierte Problem, dann ist

$$p = (p(U_1), k, i, p(U_2))$$

optimal für das Ausgangsproblem.

Algorithmus $1 \mid tree \mid \sum w_i C_i$

1. $\overline{p} := \emptyset$;
2 $L := \{1, 2, \ldots, n\}$;
WHILE $L \neq \emptyset$ **DO**
BEGIN
3. ermittle $i \in L$ mit maximalem $q(i)$;
4. **IF** $\underline{V}(i) = \emptyset$ **THEN**
BEGIN
5. ordne Auftrag J_i beginnend an der ersten freien Position in $\overline{p}$ an;
6. $L := L \setminus \{i\}$;
END
ELSE
BEGIN
7. bilde Auftrag J_{ki} mit $k = \underline{V}(i)$ und $t_{ki} := t_k + t_i,\ w_{ki} := w_k + w_i$;
8. $L := L \setminus \{i, k\} \cup \{ki\}$;
END
9. aktualisiere $\underline{V}(j)$ für alle $j \in L$;
END

Komplexität: $O(n \log n)$

Beispiel 4.2: $1|tree| \sum w_i C_i$ ✏

4.3 Das Problem $1 \mid C_i \leq d_i \mid \sum w_i C_i$

Lemma 2: Falls $p = (1, 2, \ldots, n)$ mit $d_1 \leq d_2 \leq \ldots d_n$ nicht zulässig ist, dann hat das Problem $1 \mid C_i \leq d_i \mid \sum w_i C_i$ keine zulässige Lösung.

Annahmen:

- EDD (earliest due date) Reihenfolge zulässig
- $d_1 \leq d_2 \leq \ldots \leq d_n$

Algorithmus $1 \mid C_i \leq d_i \mid \sum w_i C_i$ (*Smith* Heuristik)

1. $T := \sum_{i=1}^{n} t_i$;
2. $L := \{1, 2, \ldots, n\}$;
3. $k := n$;
   WHILE $L \neq \emptyset$ **DO**
   BEGIN
4. $L' := \{i \in L \mid d_i \geq T\}$;
5. ermittle j mit $q(j) := \min\{q(i) \mid i \in L'\}$;
6. $L := L \setminus \{j\}$; $p_k := j$;
7. $T := T - t_j$; $k := k - 1$;
   END

Bemerkung: Algorithmus $1 \mid C_i \leq d_i \mid \sum w_i C_i$ konstruiert für den Spezialfall $w_i = 1$ für alle $i = 1, 2, \ldots, n$ eine optimale Lösung.

Beispiel 4.3: $1|C_i \leq d_i| \sum w_i C_i$ ✏

4.4 Das Problem $1 \mid r_i \geq 0, pmtn \mid \sum C_i$

SRPT-Regel: ('shortest remaining processing time' first)

- Starte mit dem Auftrag mit kleinstem r_i-Wert.
- Ordne zu jedem Zeitpunkt t, der Bereitstellungszeitpunkt oder Bearbeitungsendzeitpunkt ist, einen noch nicht fertiggestellten verfügbaren Auftrag mit kleinster Restbearbeitungszeit an.

Theorem 4: Das Problem $1 \mid r_i \geq 0, pmtn \mid \sum C_i$ wird durch die SRPT-Regel optimal gelöst.

Beispiel 4.4: $1|r_i \geq 0,\ pmtn| \sum C_i$ ✏

4.5 Das Problem $1 \mid\mid \sum U_i$

Vorgehen:

Konstruiere eine maximale Menge E pünktlicher Aufträge. Ordne die Aufträge in E nach nichtfallenden d_i-Werten und danach die verspäteten Aufträge in beliebiger Reihenfolge an.

Sei $d_1 \leq d_2 \leq \ldots \leq d_n$.

Algorithmus $1 \mid\mid \sum U_i$ *(Moore 1968)*

1. $E := \emptyset$; $T := 0$;
2. **FOR** $i := 1$ **TO** n DO
   BEGIN
3. $E := E \cup \{J_i\}$;
4. $T := T + t_i$;
5. **IF** $T > d_i$ **THEN**
   BEGIN
6. bestimme Auftrag $J_j \in E$ mit größtem t_j;
7. $E := E \setminus \{J_j\}$;
8. $T := T - t_j$;
   END
   END

Komplexität: $O(n \log n)$

Bemerkung:

In ähnlicher Weise läßt sich der Spezialfall $1 \mid t_i = 1 \mid \sum w_i U_i$ lösen.

Theorem 5: Algorithmus $1||\sum U_i$ konstruiert eine Menge E pünktlicher Aufträge in einem optimalen Plan.

Beispiel 4.5: $1||\sum U_i$ ✏

Kapitel 5

Parallelmaschinenprobleme

5.1 Das Problem $P \mid\mid \sum C_i$

List Scheduling Algorithmus:

Lege eine Prioritätsliste für die Aufträge fest und ordne in jedem Schritt auf der am frühesten verfügbaren Maschine den ersten einplanbaren Auftrag der Liste an.

Theorem 1: Das Problem $P \mid\mid \sum C_i$ wird durch den *SPT* ('shortest processing time') List Scheduling Algorithmus optimal gelöst.

Komplexität: $O(n \log n)$

BEISPIEL 5.1: $P|| \sum C_i$ ✏

5.2 Das Problem $P \mid pmtn \mid C_{max}$

untere Schranke bei erlaubten Unterbrechungen:

$$LB = \max\left\{\max(t_j);\ \frac{1}{m} \cdot \sum_{j=1}^{n} t_j\right\}$$

Algorithmus $P \mid pmtn \mid C_{max}$

Konstruiere einen zulässigen Plan mit $LB = C_{max}$ wie folgt:

Fülle die Maschinen nacheinander durch Anordnung der Aufträge in beliebiger Reihenfolge, splitte dabei einen Auftrag, sofern LB erreicht ist und setze die Bearbeitung auf der nächsten Maschine zur Zeit 0 fort.

Komplexität: $O(n)$

BEISPIEL 5.2: $P|pmtn|C_{max}$ ✏

5.3 Das Problem $P \,||\, C_{max}$

bekannt: $P2||C_{max} \in \mathcal{NP}$-complete $\Rightarrow$ nutze z.B. List Scheduling Algorithmus als Näherungsverfahren

Die Güte der Lösung hängt von der gewählten Prioritätsliste ab.

Theorem 2: Für den LPT ('longest processing time') List Scheduling Algorithmus gilt:

$$\frac{C_{max}^{LPT}}{C_{max}^{opt}} \leq \frac{4}{3} - \frac{1}{3m}.$$

Frage: Existiert ein Beispiel in dem die Ungleichung in Theorem 2 mit Gleichheit erfüllt ist? ✏

Theorem 3: Für einen beliebigen List Scheduling Algorithmus LS gilt:

$$\frac{C_{max}^{LS}}{C_{max}^{opt}} \leq 2 - \frac{1}{m}.$$

exakte Lösung von Problem $P \,||\, C_{max}$ mit Hilfe **dynamischer Optimierung**

Rothkopf (1966)

Einführung **Boolescher Variablen**

$$x_j(T_1, T_2, \ldots, T_m), \qquad j = 1, 2, \ldots, n; \quad T_i = 0, 1, \ldots, C,$$

wobei C eine obere Schranke für den optimalen Zielfunktionswert C_{max}^{opt} ist

$$x_j(T_1, T_2, \ldots T_m) = \begin{cases} \text{TRUE} & \text{falls die Aufträge } J_1, J_2, \ldots, J_j \text{ so auf} \\ & M_1, M_2, \ldots, M_m \text{ bearbeitet werden können, dass} \\ & M_i \text{ in } [0, T_i] \text{ Aufträge bearbeitet} \\ \text{FALSE} & \text{sonst} \end{cases}$$

Algorithmus Dynamic Programming für $P \mid\mid C_{max}$

1. **FOR** jedes Tupel $(T_1, T_2, \ldots, T_m) \in \{0, 1, \ldots, C\}^m$ **DO**
   $x_0(T_1, T_2, \ldots, T_m)$:=FALSE;
2. $x_0(0, 0, \ldots, 0)$:= TRUE;
3. **FOR** $j := 1$ **TO** n **DO**
   FOR jedes Tupel $(T_1, T_2, \ldots, T_m) \in \{0, 1, \ldots, C\}^m$ **DO**
   $x_j(T_1, T_2, \ldots, T_m) := \bigvee_{i=1}^{m} x_{j-1}(T_1, T_2, \ldots, T_{i-1}, T_i - t_j, T_{i+1}, \ldots, T_m);$ $(*)$
4. $C_{max}^{opt} := \min\{\max\{T_1, T_2, \ldots, T_m\} \mid x_n(T_1, T_2, \ldots, T_m) = \text{TRUE}\}$;
5. Ausgehend von C_{max}^{opt}, weise 'rückwärts' die Aufträge den Maschinen unter Nutzung von (*) zu.

Komplexität: $O(nC^m)$

m fest $\Rightarrow$ *pseudopolynomialer Algorithmus*

Beispiel 5.3: Anwendung von Algorithmus Dynamic Programming auf Problem $P||C_{max}$ ✏

5.4 Das Problem $P \mid tree, t_i = 1 \mid C_{max}$

intree-Vorrangbedingungen

Der Knoten i habe die **Stufe** k, wenn die Knotenanzahl auf einem Weg mit größter Knotenanzahl vom Knoten i zu einer Senke des *intree*-Graphen k ist, wobei Knoten i mitgezählt wird (k ist der Vorwärtsrang von Knoten i).

CP-Regel: ('critical path')

Ordne als nächsten Auftrag einen mit der höchsten Stufe aus der Menge der zur Verfügung stehenden Aufträge an.

Theorem 4: Die CP-Regel erzeugt eine optimale Lösung für das Problem $P \mid intree, t_i = 1 \mid C_{max}$.

outtree-Vorrangbedingungen:

LNS-Regel: ('largest number of successors')

Ordne als nächsten Auftrag einen zur Verfügung stehenden Auftrag mit größter Anzahl von (nicht notwendigerweise direkten) Nachfolgern an.

Theorem 5: Das Problem $P \mid outtree, t_i = 1 \mid C_{max}$ wird sowohl mit der CP- als auch der LNS-Regel optimal gelöst.

Bemerkungen:

1) Die CP- bzw. die LNS-Regel liefern nicht notwendig optimale Pläne, wenn allgemeine Vorrangbedingungen gegeben sind.

2) Sowohl die CP- als auch die LNS-Regel lassen sich auf den Fall beliebiger Bearbeitungszeiten verallgemeinern.

CP-Regel: Wähle den dem Auftrag zugeordneten Knoten zuerst, der den längsten ($\sum t_i$) oder kritischen Weg bezüglich der Knotenbewertungen zur Senke hat.

BEISPIEL 5.4: $P|tree, t_i = 1|C_{max}$ ✏

Kapitel 6

Flow-Shop Probleme

6.1 Das Problem $F \mid\mid C_{max}$

Theorem 1: Für das Problem $F \mid\mid C_{max}$ existiert ein optimaler Plan mit folgenden Eigenschaften:

1. Die ORF auf den ersten beiden Maschinen sind gleich.
2. Die ORF auf den letzten beiden Maschinen sind gleich.

Folgerung: Für Flow-Shop Probleme mit zwei bzw. drei Maschinen existert stets ein optimaler Permutationsplan (d.h. alle ORF sind gleich).

polynomial lösbarer Spezialfall: $F2 \mid\mid C_{max}$

Algorithmus $F2 \mid\mid C_{max}$ *(Johnson 1954)*

1. $L := \{1, 2, \ldots, n\}$;
2. $k := 1;\ l := n$;
   WHILE $L \neq \emptyset$ **DO**
   BEGIN
3. ermittle t_{i^*,j^*} mit $t_{i^*,j^*} = \min\{t_{ij} \mid i \in L; j \in \{1, 2\}\}$;
4. **IF** $j^* = 1$ **THEN**
   BEGIN
5. $\overline{p}_k := i^*;\ k := k + 1$
   END
   ELSE
   BEGIN
6. $\overline{p}_l := i^*;\ l := l - 1$
   END
7. $L := L \setminus \{i\}$;
   END

Theorem 2: Algorithmus $F2 \mid\mid C_{max}$ konstruiert eine optimale Reihenfolge $\overline{p}$.

Komplexität: $O(n \log n)$

BEISPIEL 6.1: $F2||C_{max}$ ✏

6.2 Das Problem $F \mid prmu \mid C_{max}$

6.2.1 Branch and Bound Verfahren

- Anwendung des **Partiallösungskonzeptes**:
  abwechselnde Fixierung von Aufträgen von vorn und hinten beginnend
  Illustration ✏
- **globale untere Schranke** (keine fixierten Aufträge → Wurzel P)
  Illustration ✏

  2-Maschine-Schranke LB_j für die Maschinen M_j und M_{j+1} $(1 \leq j \leq m-1)$:

  Bestimme eine optimale Lösung p^* für das $F2 \mid\mid C_{max}$ Problem für die Maschinen M_j und M_{j+1} mittels **Algorithmus von Johnson** und addiere jeweils die minimale Zeit vor dem Start der ersten Operation sowie die minimale Zeit nach Abschluss der letzten Operation

  $\Rightarrow$ wähle unter den erhaltenen $m-1$ Schranken die größte aus:

$$LB = \max\{LB_j \mid j = 1, 2, \ldots, m-1\}$$

- analog: untere Schranken bei **fixierten Anfangs- und Endfolgen**
  Illustration ✏

6.2.2 Heuristische Verfahren

(a) **<u>Konstruktionsverfahren</u>**

(i) **Lösung künstlicher 2-Maschinen-Probleme**

Dannenbring (1977)

Lösung eines künstlichen 2-Maschinen-Problems (mittels Algorithmus von Johnson) mit folgenden Bearbeitungszeiten:

$$t_{i1} = \sum_{j=1}^{m} (m - j + 1) \cdot t_{ij}; \qquad t_{i2} = \sum_{j=1}^{m} j \cdot t_{ij}, \qquad 1 \leq i \leq n$$

Campbell, Dudek & Smith (1970)

Lösung von $m-1$ künstlichen 2-Maschinen-Problemen (mittels Algorithmus von Johnson) und Auswahl von der Permutation mit dem besten Zielfunktionswert bezüglich des Ausgangsproblems.

Bearbeitungszeiten für Problem k:

$$t_{i1}^k = \sum_{j=1}^{k} t_{ij}, \qquad t_{i2}^k = \sum_{j=m-k+1}^{m} t_{ij}, \qquad 1 \leq i \leq n; \quad 1 \leq k \leq m-1$$

(ii) **Einfügeverfahren**

Nawaz, Enscore & Ham (1983)

Vorgehen:
In eine bestehende Teillösung wird ein weiterer Auftrag so eingefügt, dass der Zielfunktionswert der neuen Teillösung minimal wird.

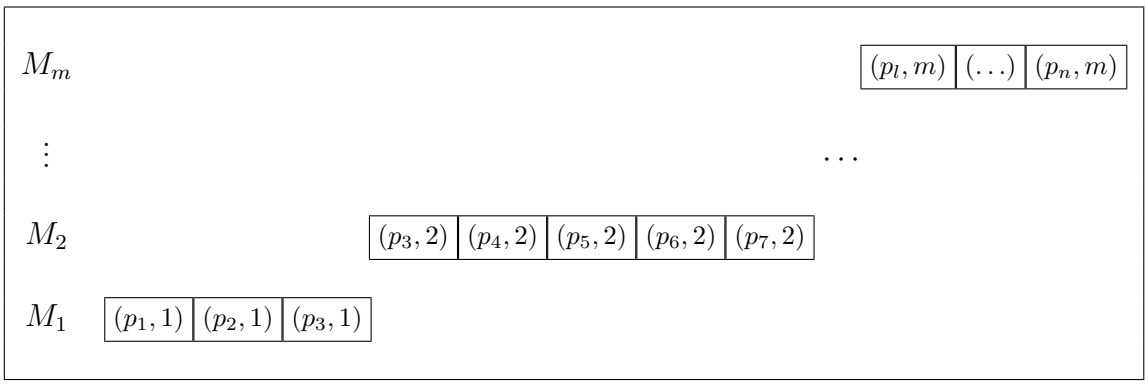
Einfügereihenfolge der Aufträge:
Sortiere die Aufträge ein nach monoton nicht wachsenden Summen der Bearbeitungszeiten der Operationen eines Auftrags

Komplexität: $O(n^2 m)$

(b) **<u>Iterative Verfahren</u>**

Vorgehen:

- Wende *lokale Suche* z.B. in der API Nachbarschaft, der Pairwise Interchange Nachbarschaft oder der Shift Nachbarschaft unter Nutzung von *Metaheuristiken* (z.B. Simulated Annealing, Tabu Suche) an.
- Nutze eine *notwendige Bedingung* für eine Zielfunktionswertverbesserung, betrachte 'Blöcke' des kritischen ('längsten') Weges. ✏



Definition 1: Ein **Block** ist eine maximale Menge aufeinanderfolgender Operationen auf der gleichen Maschine, die zum kritischen Weg gehören und mindestens aus zwei Operationen besteht.

$\Rightarrow$ '**Entferne**' Operationen aus dem kritischen Weg zur Erzeugung von Nachbarn, die nicht alle Knoten (Operationen) des ursprünglichen kritischen Weges enthalten.

Kapitel 7

Job-Shop Probleme

7.1 Das Problem $J2 \mid n_i \leq 2 \mid C_{max}$

Jackson (1956)

Verallgemeinerung des Algorithmus von Johnson

Konstruktion eines optimalen Plans wie folgt:

1. ermittle $p^{(12)}$ als optimale Lösung des $F2 \mid\mid C_{max}$ Problems mit der Auftragsmenge $M^{12} = \{J_i \mid q^i = (1,2)\}$;
2. ermittle $p^{(21)}$ als optimale Lösung des $F2 \mid\mid C_{max}$ Problems mit der Auftragsmenge $M^{21} = \{J_i \mid q^i = (2,1)\}$;
3. wähle $p^{(i)}$ als beliebige Permutation der Auftragsmenge M^i der nur auf M_i zu bearbeitenden Aufträge $(i = 1, 2)$;

Dann ist $P = (p^1, p^2)$ mit

$$p^1 = (p^{(12)}, p^{(1)}, p^{(21)}) \qquad \text{und} \qquad p^2 = (p^{(21)}, p^{(2)}, p^{(12)})$$

ein optimaler Plan.

Beispiel 7.1: $J2|n_i \leq 2|C_{max}$ ✏

7.2 Das Problem $J \mid\mid C_{max}$

7.2.1 Branch and Bound Verfahren

1. **Erzeugung aktiver Pläne** (Partiallösungskonzept)
2. **Orientierung von Kanten** (Alternativkonzept)
3. **Block Approach** und **Immediate Selection**

Zur Erzeugung aktiver Pläne:

aktiver Plan:

Plan, bei dem *kein* Bearbeitungsbeginn einer Operation verkleinert werden kann, ohne dass sich mindestens ein Bearbeitungsbeginn einer anderen Operation verzögert.

Vorgehen bei Anfügung einer weiteren Operation:

- Ermittle die Menge S aller Operationen, deren Vorgänger bereits eingeplant sind.
- Ermittle die Maschine M_l mit dem frühestmöglichen Bearbeitungsende einer Operation aus S. Dieses Bearbeitungsende liefert die Startzeitgrenze t.
- Ermittle alle Aufträge, die auf M_l **vor** der Startzeitgrenze t beginnen können. Die Anzahl dieser Aufträge liefert die Anzahl der Nachfolger des aktuellen Knotens des Verzweigungsbaumes.

Zur Berechnung unterer Schranken:

$G_s = (V, A \cup D_S)$ – gerichteter Graph

gesucht: untere Schranke für die Länge eines längsten Weges im Graphen G, der alle Bögen von A und D_S enthält

in der Regel: **Einmaschinen-Schranken**

Für die Operation $u = (i, j)$ werden folgende Bezeichnungen eingeführt:

r_u – **Head** der Operation u: Länge eines längsten Weges in G_S von Operation 0 zu Operation u (ohne Berücksichtigung von Operation u)

$t_u := t_{ij}$

q_u – **Tail** der Operation u: Länge eines längsten Weges in G_S von Operation u zu Operation $*$ (ohne Berücksichtigung von Operation u)

Einige Schranken bezüglich M_j $(j = 1, 2, \ldots, m)$:

- $LB_j^1(D_S) = \min_{u=(i,j)} \{r_u\} + \sum\{t_s \mid s \text{ ist Operation auf } M_j\} + \min_{u=(i,j)} \{q_u\}$
- $LB_j^2(D_S) = \max_{u=(i,j)} \{r_u + t_u + q_u\}$
- ermittle $LB_j^3(D_S)$ als optimalen Zielfunktionswert eines $1 \mid r_i \geq 0, q_i \geq 0 \mid C_{max}$ Problems, wobei q_i als Lieferzeit des Auftrags J_i nach Abschluss der Bearbeitung interpretiert wird, d.h.
$$C_i' := C_i + q_i$$
(Das Problem gehört zur Klasse $\mathcal{NP}$-complete, d.h. wende Enumerationsverfahren an.)

7.2.2 Heuristische Verfahren

(a) **Konstruktive Verfahren**

(i) **Erzeugung eines aktiven Plans**

(d.h. beim Branch and Bound Verfahren wird **keine** Verzweigung ausgeführt)
Wähle z.B. einen Auftrag gemäß einer der folgenden Regeln (oder anderer) aus:

- **FCFS**-Regel ('first come first served')
- **MOPR**-Regel ('most operations remaining')
- **LRPT**-Regel ('longest remaining processing time')
- **ECT**-Regel ('earliest completion time')

(ii) **Einfügeverfahren**

Werner & Winkler (1995)

Füge schrittweise die Operationen der Aufträge in bestehende Teilpläne so ein, dass

- alle technologischen Reihenfolgen eingehalten werden
- alle durch vorangegangene Einfügungen aufgestellten Vorrangbedingungen zwischen Operationen beibehalten werden und
- die Einfügung der aktuellen Operation zu keinem Zyklus im zugehörigen Graphen führt

Einfügereihenfolge:
Beginne mit dem vollständigen Auftrag mit der größten Summe der Bearbeitungszeiten und füge dann nacheinander die Operationen nach monoton nichtwachsenden Bearbeitungszeiten ein.

"*Bewertung*" eines Teilplans A mit der eingefügten Operation (i, j):

$$g(A) = r_{ij} + t_{ij} + q_{ij}$$

(entspricht dem längsten Weg "durch" Operation (i, j))

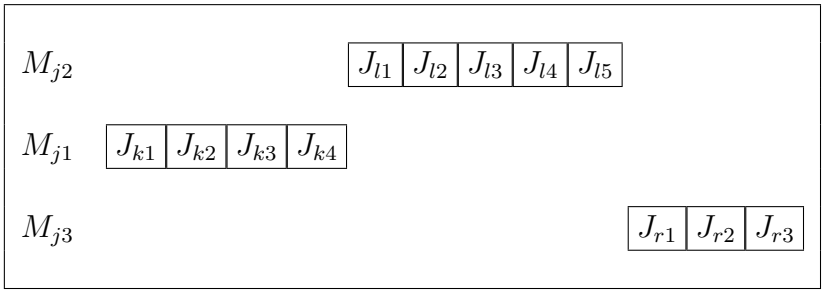
BEISPIEL 7.2: $J||C_{max}$ ✏

(b) **Iterative Verfahren**

Zur Wahl der Nachbarschaft:

– Erzeugung eines Nachbarn durch Austausch von zwei benachbarten Operationen auf einer Maschine (**API Nachbarschaft**)

Betrachte **Blöcke** des kritischen Weges des Planes P. ✏



$\Rightarrow$ Betrachte dabei nur solche Nachbarn, die eine **notwendige Bedingung** für eine Zielfunktionswertverbesserung erfüllen: tausche zwei benachbarte Aufträge am Rand eines **Blockes** aus (beim ersten Block ist nur der rechte und beim letzten Block nur der linke Rand zu betrachten).

– Streiche einen Auftrag J_i in allen organisatorischen Reihenfolgen und füge nacheinander alle Operationen dieses Auftrags analog zum Einfügeverfahren von Werner & Winkler wieder zulässig ein (**Reinsertion Nachbarschaft**).

7.3 Probleme mit zwei Aufträgen

7.3.1 Das Problem $J \mid n = 2 \mid C_{max}$

Akers & Friedman (1955)

Formulierung des Problems als **Kürzeste-Wege-Problem mit rechteckigen Hindernissen**

Bearbeitungszeiten der Operationen von J_1 (J_2):

Intervalle auf der x-Achse (y-Achse) gemäß technologischer Reihenfolge

Seien

$$0 = (0, 0) \qquad \text{und} \qquad F = (a, b),$$

wobei a (b) die Summe der Bearbeitungszeiten von Auftrag J_1 (J_2) ist.

Problem: Bestimme einen kürzesten Weg von $0 = (0, 0)$ nach $F = (a, b)$, der nur horizontale, vertikale und diagonale Segmente enthält, die **nicht** durch das Innere der verbotenen Regionen verlaufen.

Konstruktion des **Netzwerks**

$$N = (V,\ A,\ d)$$

Knotenmenge V:

- enthält $0, F$;
- Menge aller Nordwest-Ecken (NW) der Hindernisse;
- Menge alle Südost-Ecken (SO) der Hindernisse.

Bogenmenge A:

Jeder Knoten $i \in V \setminus \{F\}$ besitzt höchstens zwei von i fortführende Bögen.

Gehe diagonal von i in NO-Richtung bis

(a) der Rand des durch 0 und F bestimmten Rechtecks erreicht ist
$\Rightarrow F$ ist einziger Nachfolger von i;

(b) der Rand eines Hindernisses erreicht wurde
$\Rightarrow$ es gibt zwei Bögen (i, j) zur NW-Ecke sowie (i, k) zur SO-Ecke des betreffenden Hindernisses.

Länge $d(i, j)$ von Bogen (i, j):

Länge des vertikalen oder horizontalen Abschnittes plus Länge der Projektion des diagonalen Abschnittes auf die x- (bzw. y-)Achse.

Theorem 1: Ein kürzester Weg von 0 nach F in N entspricht einer optimalen Lösung des Kürzeste-Wege-Problems mit Hindernissen (d.h. einer optimalen Lösung des Problems $J \mid n = 2 \mid C_{max}$).

Seien $D_1 \prec D_2 \prec \ldots \prec D_r$ die lexikografisch anhand ihrer NW-Ecken geordneten Hindernisse

Algorithmus $J|n = 2|C_{max}$ *(Akers & Friedman 1955)*

1. **FOR** alle Knoten $i \in V$ **DO** $d(i) = \infty$;
2. **FOR** alle Nachfolger j von 0 **DO** $d(j) = d(0, j)$;
3. **FOR** $i := 1$ **TO** r **DO**
   BEGIN
4. **FOR** alle Nachfolger j der NW-Ecke k von D_i **DO**
   $d(j) := \min\{d(j),\ d(k) + d(k, j)\}$;
5. **FOR** alle Nachfolger j der SO-Ecke k von D_i **DO**
   $d(j) := \min\{d(j),\ d(k) + d(k, j)\}$;
   END;
6. $d^* := d(F)$.

Bemerkung: Der Algorithmus $J|n = 2|C_{max}$ ist leicht modifizierbar, so dass neben dem kürzesten Weg auch der zugehörige Plan ermittelt wird.

Komplexität von Algorithmus $J \mid n = 2 \mid C_{max}$: $O(r)$

Komplexität der Konstruktion des Netzwerks N: $O(r \log r)$

Gesamtkomplexität: $O(r \log r)$ $\quad (r \leq n_1 \cdot n_2)$

BEISPIEL 7.3: $J|n = 2|C_{max}$ ✏

7.3.2 Das Problem $J \mid n = 2 \mid f$

f - reguläres Kriterium $f(C_1, C_2)$ $\quad$ (monoton nichtfallend in jedem Argument)

$\Rightarrow$ Zielfunktionswert der zu einem Weg gehörenden Lösung hängt von dem zuerst erreichten Randpunkt P des durch 0 und F definierten Rechtecks ab:

- **"P auf nördlichem Rand"**
  $\Rightarrow$ beste Lösung zu einem Weg durch P hat den Zielfunktionswert

  $$f(d(P) + d(P, F),\ d(P)),$$

  wobei

  $d(P)$ – Länge eines kürzesten Weges von 0 zu P
  $d(P, F)$ – Länge des horizontalen Segments von P nach F

- **"P auf östlichem Rand"**
  $\Rightarrow$ beste Lösung zu einem Weg durch P hat den Zielfunktionswert

  $$f(d(P),\ d(P) + d(P, F))$$

Kapitel 8

Open-Shop Probleme

Das Problem $O2||C_{max}$

Gonzalez & Sahni (1976)

Bezeichnungen:

$$a_i := t_{i1}; \quad b_i := t_{i2};$$

$$K := \{J_i \mid a_i \geq b_i\}; \quad L := \{J_i \mid a_i < b_i\}$$

$$T_1 := \sum_{i=1}^{n} a_i; \qquad T_2 := \sum_{i=1}^{n} b_i$$

Auswahl zweier verschiedener Aufträge J_r und J_s derart, dass

$$a_r \geq \max_{J_j \in K} \{b_j\}; \qquad b_s \geq \max_{J_j \in L} \{a_j\}$$

Darüber hinaus seien

$$K' := K \setminus \{J_r,\ J_s\}; \qquad L' := L \setminus \{J_r,\ J_s\}$$

Vorgehen:

1. Es werden zulässige Pläne für

   $$L' \cup \{J_s\} \qquad \text{und} \qquad K' \cup \{J_r\}$$

   gebildet, wobei alle Aufträge zuerst auf M_1 und dann auf M_2 so bearbeitet werden, dass keine Stillstandszeiten zwischen den Bearbeitungen der Operationen in den Teilplänen auftreten.

2. Angenommen, es gilt
$$T_1 - a_s \geq T_2 - b_r$$
(der Fall $T_1 - a_s < T_2 - b_r$ ist symmetrisch).

Kombination beider Teilpläne wie folgt:

1. Aufträge in $L' \cup \{J_s\}$ werden auf M_2 nach rechts gesetzt (dabei tritt keine Stillstandszeit auf).
2. Ordne Auftrag J_r auf M_2 an erster Position an.

2 Fälle:

(a) $a_r \leq T_2 - b_r$
$\Rightarrow C_{max} = \max\{T_1, T_2\}$

(b) $a_r > T_2 - b_r$
$\Rightarrow C_{max} = \max\{T_1,\ a_r + b_r\}$

$\rightarrow$ *Für jeden zulässigen Plan gilt:*

$$C_{max} \geq \max\left\{T_1,\ T_2,\ \max_j\{a_j + b_j\}\right\}$$

$\Rightarrow$ In beiden Fällen wird als Zielfunktionswert die untere Schranke angenommen, d.h. **der erzeugte Plan ist optimal**.