D I P L O M A R B E I T

Auktionsalgorithmus zum Lösen von Zuordnungsproblemen

Ausgeführt am Institut für
Wirtschaftsmathematik
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof.Dipl.-Ing.Dr.techn. Alexander Mehlmann

durch

Katharina Fabi
Eichbergstr. 11/b
2372 Gießhübl

Wien, 2010 _____

An dieser Stelle möchte ich Herrn Prof. Mehlmann für seine Hilfe und Betreuung meiner Diplomarbeit danken.

Meiner Familie und meinen Freunden danke ich für die Unterstützung während meines Studiums.

Kurzfassung

Auf Basis der Arbeit von Dimitri P. Bertsekas wird ein Auktionsalgorithmus zum Lösen von Zuordnungsproblemen erörtert. Zuerst wird die einfachste Form, das sogenannte *symmetrische Zuordnungsproblem*, betrachtet. Graphentheorie und Dualität liefern eine Brücke zwischen symmetrischen und asymmetrischen Problemen, sodass nur durch kleine Veränderungen der ursprüngliche Auktionsalgorithmus auf *asymmetrische* und *Mehrfachzuordnungsprobleme* angewendet werden kann. Alle Algorithmen sind in Matlab implementiert, um einige Probleme zu lösen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Diplomarbeit basiert auf Dimitri Bertsekas' Auktionsalgorithmus, den er in seinem Paper 'A Distributed Algorithm for the Assignment Problem' [1] im Jahr 1979 zum ersten Mal vorgestellt hat. In nachfolgenden Papers erweiterte er diesen ursprünglichen Algorithmus, um diesen auf ein breiteres Spektrum an Problemen anwenden zu können. In der Monographie 'Linear Network Optimization, Algorithms and Codes' [5] hat er im Jahr 1992 diese Algorithmen detailliert zusammengefasst. Wichtige Erweiterungsliteraturen sind unter anderem 'Network Optimization, Continuous and Discrete Models' [6] und 'Constrained Optimization and Lagrange Multiplier Methods' [4].

In einer kurzen Einleitung wird das Zuordnungsproblem präsentiert. Beim Vorstellen des Zuordnungsproblems kann man die Motivation für den Auktionsalgorithmus erkennen, welcher zur Lösung dieser spezifisch strukturierten Probleme herangezogen wird. Dieser Algorithmus wird auch zur Lösung anderer Probleme benutzt. Mehr dazu wird später erörtert.

1.1 Das Zuordnungsproblem

Das **Zuordnungsproblem** ist ein Spezialfall des *Verschiffungsproblems*. Es werden n Personen und n Objekte angenommen. Jede Person kann genau einem Objekt zugeordnet werden und jedes Objekt genau einer Person. Das Ziel ist es den Gesamtnutzen zu optimieren.

Für jede Person sind die Nutzenwerte gegeben, welche diese durch Wählen eines Objektes erhält. In der Menge A sind die Nutzenwerte aller möglichen Person-Objekt-Paare gespeichert. Wenn Person i Objekt j zugeordnet wird, so ist der Nutzen a_{ij}. Allerdings kann Person i nur dann Objekt j zugeordnet werden, wenn das Paar (i, j) in der Menge A aller möglichen Zuordnungen enthalten ist.
Mathematisch gesehen ist der Gesamtnutzen $\sum_{(i,j)\in A} a_{ij}x_{ij}$ zu maximieren, wobei man Person-Objekt-Paare $(1, j_1), ..., (n, j_n)$ aus A sucht, mit der Bedingung, dass sich alle $j_1, ..., j_n$ voneinander unterscheiden.
Das Zuordnungsproblem kann als lineares Optimierungsproblem dargestellt werden:

$$\max \sum_{(i,j)\in A} a_{ij}x_{ij}$$

unter den Nebenbedingungen

$$\begin{aligned}
\sum_{\{j|(i,j)\in A\}} x_{ij} &= 1, \forall i = 1, ..., n, \\
\sum_{\{i|(i,j)\in A\}} x_{ij} &= 1, \forall j = 1, ..., n, \\
0 \leq x_{ij} &\leq 1, \forall (i, j) \in A.
\end{aligned}$$

Wenn Person i Objekt j zugeordnet wird, so definiert man $x_{ij} = 1$. Andernfalls wird für alle $(i, j) \in A$ $x_{ij} = 0$ gesetzt.

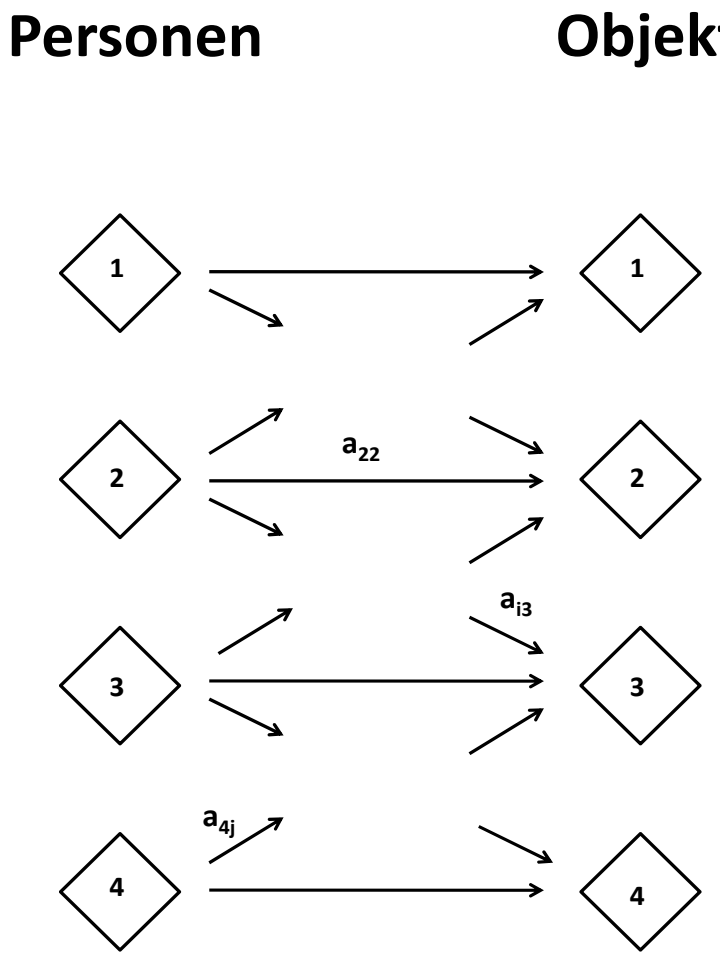


Abbildung 1.1: Graph eines symmetrischen Zuordnungsproblems

Kapitel 2

Auktionsalgorithmus für Symmetrische Probleme

In diesem Kapitel wird zuerst die Dualität erörtert, da diese in der linearen Optimierung eine wichtige Rolle spielt. Anschließend werden Algorithmen eingeführt, um symmetrische Zuordnungsprobleme zu lösen.

2.1 Dualität

Für alle linearen Optimierungsprobleme ist Dualität sehr wichtig. In diesem Abschnitt werden zwei Probleme betrachtet: das primale und das duale Problem. Um die Dualität besser zu verstehen, wird diese anhand des Zuordnungsproblems erörtert.
Nimmt man ein symmetrisches Zuordnungsproblem an, so gibt es n Personen, die n Objekten zugeordnet werden sollen. Jede Person versucht ihren eigenen Nutzen zu optimieren. Jedes Objekt hat einen Preis, der während des Prozesses verändert wird. Objekt j hat den Preis p_j, welchen eine Person, die auf dieses Objekt setzt, zu zahlen hat. Person i erhält ihren *Nettobetrag*, indem der Preis p_j des gewählten Objekts j vom Nutzen a_{ij} abgezogen wird. Jede Person möchte ihrem besten Objekt zugeordnet werden. Dies ist das Objekt, welches den jeweiligen Nettobetrag maximiert:

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{2.1}$$

mit

$$A(i) = \{j \mid (i,j) \in A\}.$$

$A(i)$ stellt die Menge der Objekte dar, auf welche Person i bereit ist zu setzen.
Die Gleichung (2.1) ist die sogenannte komplementäre Schlupfbedingung (KS). Sie stellt sicher, dass die bisherige Teilzuordnung für alle bereits zugeordneten Personen optimal ist. Das bedeutet, dass das System im Gleichgewicht ist. Das heißt, dass keine zugeordnete Person mit einem anderen Objekt besser gestellt wäre.
Das *duale Problem* des Zuordnungsproblems ist durch

$$\min_{p_j, j=1,\ldots,n} \{\sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j\} \tag{2.2}$$

gegeben.
Um den Zusammenhang zwischen dem Zuordnungsproblem und dem zugehörigen dualen Problem zu sehen, benötigt man folgenden Satz:

Satz 2.1.1 *Eine zulässige Zuordnung*[1] *und ein Preisvektor, welche die komplementäre Schlupfbedingung (2.1) erfüllen, sind eine optimale Lösung des Zuordnungsproblems. Gleichzeitig stellen die Preise eine optimale Lösung des dualen Problems dar. Weiters kann gezeigt werden, dass die optimale Lösung des Zuordnungsproblems mit der des dualen Problems übereinstimmt. Das bedeutet, dass der optimale Nutzen und die optimalen Kosten gleich sind.*
([9], S. 710/711, erweitert durch [5], S.33/34)

Beweis: $\{(i,k_i) \mid i = 1,\ldots,n\}$ ... zulässige Zuordnung

$$\Longrightarrow \sum_{i=1}^{n} a_{ik_i} \leq \{\sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j\} \tag{2.3}$$

für alle $\{p_j \mid j = 1,\ldots,n\}$,
da $\sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} \geq \sum_{i=1}^{n} (a_{ik_i} - p_{k_i})$

[1] Eine Zuordnung heißt zulässig, wenn jede Person einem Objekt und jedes Objekt einer Person zugewiesen ist.

und $\sum_{j=1}^{n} p_j = \sum_{i=1}^{n} p_{k_i}$.
Die gegebene Zuordnung und der gegebene Preisvektor - $\{(i, j_i) \mid i = 1, ..., n\}$ und $\{\overline{p_j} \mid j = 1, ..., n\}$ - erfüllen die komplementäre Schlupfbedingung.

$$\begin{aligned}
&\Longrightarrow a_{ij_i} - \overline{p_{j_i}} = \max_{j \in A(i)} \{a_{ij} - \overline{p_j}\},\ i = 1, ..., n \\
&\Longrightarrow \sum_{i=1}^{n} (a_{ij_i} - \overline{p_{j_i}}) = \sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - \overline{p_j}\} \\
&\Longleftrightarrow \sum_{i=1}^{n} a_{ij_i} = \sum_{i=1}^{n} \left(\max_{j \in A(i)} \{a_{ij} - \overline{p_j}\} + \overline{p_{j_i}} \right)
\end{aligned}$$

$\Longrightarrow \{(i, j_i) \mid i = 1, ..., n\}$ ist eine optimale Lösung der linken Seite der Gleichung (2.3) und daher eine optimale Lösung des Zuordnungproblems. Zugleich ist $\{\overline{p_j} \mid j = 1, ..., n\}$ eine optimale Lösung der rechten Seite der Gleichung (2.3) und stellt daher eine optimale Lösung des dualen Problems dar.

2.2 Naive Auktion

Zieht man das Zuordnungsproblem in Betracht und versucht einen natürlichen Weg zu finden dieses zu lösen, so kommt man auf den *naiven* Auktionsalgorithmus, welcher unter anderem in 'Auction Algorithms for Network Flow Problems: A Tutorial Introduction' [3] detailliert erklärt wird.
Dieser Algorithmus wird naiv genannt, da er einen schweren Fehler hat. Trotzdem ist es wichtig den naiven Auktionsalgorithmus zu erklären, um den tatsächlichen Algorithmus zu verstehen.
Der Algorithmus beginnt mit einer Teilzuordnung und versucht durch Iteration diese Zuordnung zu verbessern und zu vervollständigen. Das bedeutet, dass am Anfang nur ein Teil der Personen bereits zugeordnet ist. Nach jedem Iterationsschritt werden ein neuer Preisvektor und eine neue Zuordnung gebildet. Die optimale Zuordnung wird erreicht, wenn jede Person einem Objekt zugeordnet ist.
Vor jedem Iterationsschritt muss sichergestellt werden, dass die komplementäre Schlupfbedingung

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}$$

für alle Paare (i, j_i) der Teilzuordnung erfüllt ist.
Nun wählt man eine nicht zugeordnete Person i aus und versucht das Objekt j_i zu finden, welches den maximalen Nettobetrag liefert:

$$j_i = \arg \max_{j \in A(i)} \{a_{ij} - p_j\}.$$

Hat man das optimale Objekt gefunden, so wird nun die Person i ihrem optimalen Objekt j_i zugeordnet. Falls dieses Objekt bereits einer anderen Person zugeteilt war, so kommt diese Person wieder in die Menge der nicht zugeordneten Personen.
Nachdem die Zuordnung stattgefunden hat, muss noch der Preis definiert werden, den Person i bereit ist für ihr optimales Objekt zu zahlen. Dafür muss man feststellen, auf welchem Niveau die Person zwischen dem optimalen und dem zweitbesten Objekt indifferent ist.

Nachdem der Betrag des besten Objekts

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}$$

und der Betrag des zweitbesten Objekts

$$w_i = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}$$

ist, wird der Preis als

$$p_{j_i} + \gamma_i$$

gesetzt, wobei

$$\gamma_i = v_i - w_i$$

ist.
Das bedeutet, dass der Betrag $a_{ij_i} - p_{j_i}$, den das Objekt j_i der Person i liefert, während der Preis p_{j_i} ansteigt, soweit sinkt, dass j_i immer noch das beste Objekt für die Person i darstellt.
Diese Iteration wird wiederholt, bis alle Personen Objekten zugeordnet sind. Salopp erklärt ist dieser Prozess eine Auktion, bei der in jedem Iterationsschritt ein Bieter den Preis seines bevorzugten Objekts um ein Inkrement γ_i erhöht. Dieses Inkrement ist offensichtlich nichtnegativ, da $v_i \geq w_i$. Dadurch wird sichergestellt, dass die Preise steigen (siehe Abbildung (2.1)). Wie in einer echten Auktion macht der Bieter durch Erhöhung des Preises dieses Objekt für andere potentielle Bieter weniger attraktiv.

2.2.1 ϵ-Komplementärer Schlupf

Leider gibt es viele Beispiele, bei denen der naive Auktionsalgorithmus nicht funktioniert. Gibt es mehrere Objekte, die den Nettobetrag der bietenden Person i maximieren, so ist das Inkrement γ_i gleich Null. Daher steigen in diesem Fall die Preise nicht an. Wenn es nun eine Gruppe von Personen gibt, in welcher eine gewisse Menge von Objekten allen Personen gleich viel wert ist, so würde dies zu einer endlosen Schleife führen, da der Preis nicht steigen würde, obwohl sich der Eigentümer des Objektes ändert (siehe Abbildung (2.2)).
Um diese Schleife zu durchbrechen wird eine Regel definiert, die durch tat-

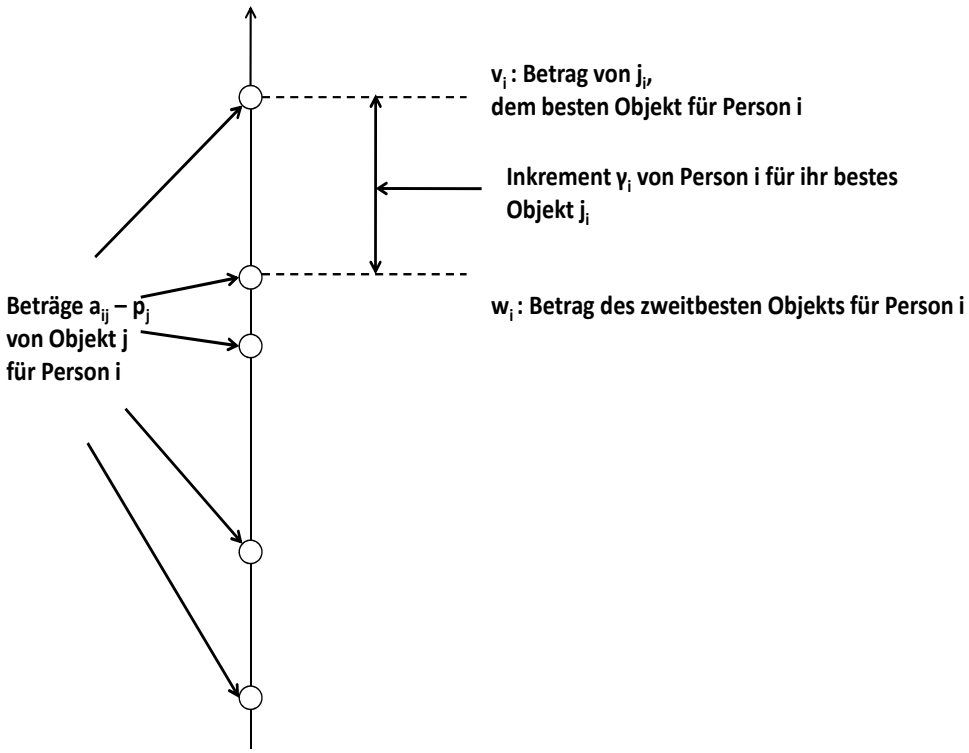
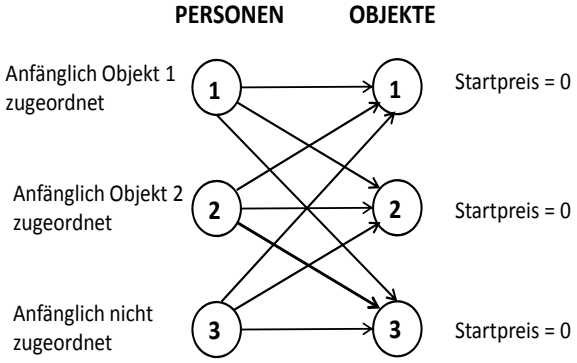


Abbildung 2.1: In der naiven Auktion ist Objekt j_i nach Erhöhung des Preises p_{j_i} noch immer die beste Wahl für Person i. Das bedeutet, dass die KS-Bedingung erfüllt ist. Das Inkrement γ_i kann allerdings auch Null sein, falls sich der Bieter zwischen zwei Objekten nicht entscheiden kann.

sächliche Auktionen motiviert wird. Wenn eine Person auf ein Objekt setzt, so muss der Preis erhöht werden. Um dies zu vereinfachen, wird ein positiver Skalar ϵ fixiert. Jede Teilzuordnung und ihr zugehöriger Preisvektor p müssen die *ϵ-komplementäre Schlupfbedingung* (kurz ϵ-KS)

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon \tag{2.4}$$

für alle zugeordneten Paare (i, j) erfüllen. Das heißt, dass jede zugeordnete Person der Teilzuordnung einem Objekt zugeordnet sein soll, dessen Nettobetrag um höchstens ϵ kleiner ist als der optimale Nettobetrag. Diese Bedingung wird unter anderem in 'Auction Algorithms for Network Flow Problems: A Tutorial Introduction' [3] definiert.



Hier ist $a_{ij} = C > 0$ für alle (i,j) mit i = 1,2,3 und j = 1,2 und $a_{ij} = 0$ für alle (i,j) mit i= 1,2,3 und j = 3

Zu Beginn von Iteration #	Objekt Preise	Zuordnungs-Paare	Bieter	Bevorzugtes Objekt	Inkrement γ
1	0,0,0	(1,1), (2,2)	3	2	0
2	0,0,0	(1,1), (3,2)	2	2	0
3	0,0,0	(1,1), (2,2)	3	2	0

Abbildung 2.2: Die naive Auktion versagt in diesem Beispiel. Die Objekte 1 und 2 bieten allen Personen denselben Betrag C. Das Objekt 3 bietet allen Personen Nutzen 0. Das heißt, dass sich alle Personen um die ersten beiden Objekte bemühen. Der Algorithmus bildet eine endlose Schleife, da die Personen 2 und 3 abwechselnd auf das Objekt 2 setzen ohne den Preis zu ändern, da sie zwischen den Objekten 1 und 2 indifferent sind ($\gamma_i = 0$).

2.3 Vorwärtsauktion

2.3.1 Einleitung

Man kann den tatsächlichen Auktionsalgorithmus als einen naiven Auktionsalgorithmus betrachten, mit dem einzigen Unterschied, dass das Inkrement γ_i statt

$$\gamma_i = v_i - w_i$$

nun

$$\gamma_i = v_i - w_i + \epsilon$$

beträgt.
Durch diese Definition wird sichergestellt, dass die ϵ-KS-Bedingung erfüllt ist (siehe Abbildung (2.3)). Das gewählte Inkrement $\gamma_i = v_i - w_i + \epsilon$ ist das größte, sodass sich der Nettobetrag des Objektes j_i um höchstens ϵ von dem des optimalen Objektes unterscheidet. Diese Aussage gilt, da das größtmögliche Inkrement, sodass das gewählte Objekt j_i optimal ist, $\gamma_i = v_i - w_i$ beträgt. Ein kleineres Inkrement würde ebenfalls die ϵ-KS-Bedingung erfüllen, solange $\gamma_i \geq \epsilon$ gilt. Benutzt man allerdings das größtmögliche Inkrement, so endet der Algorithmus schneller. Dies kann man auch in realen Auktionen beobachten. Setzen die Bieter die Preise rasch höher, so endet die Auktion schneller.

Es kann gezeigt werden, dass dieser Algorithmus mit einer zulässigen Zuordnung und einem Preisvektor endet, sodass die ϵ-KS-Bedingung erfüllt ist. Man kann sich einfach vorstellen, dass der Preis, falls ein Objekt m Gebote erhält, um mindestens $m\epsilon$ erhöht wird. Das heißt, dass dieses Objekt nach einer ausreichend großen Anzahl an Geboten weniger attraktiv wird als andere Objekte, die noch keine Gebote erhalten haben. Daher kann ein Objekt nur für eine begrenzte Anzahl an Iterationsschritten die „beste Wahl“ sein. Sobald jedes Objekt ein Gebot erhalten hat, endet der Algorithmus. In Abbildung (2.4) kann man sehen, wie die endlose Schleife durch das neue Inkrement durchbrochen wird.

Der Auktionsalgorithmus findet also eine Lösung für das Zuordnungsproblem. Nun bleibt noch zu zeigen, dass dies eine optimale Lösung ist. Die Optimalität hängt allerdings von der Größe von ϵ ab. Wird eine reale Auktion in Betracht gezogen, so ist ersichtlich, dass ein Bieter immer versuchen

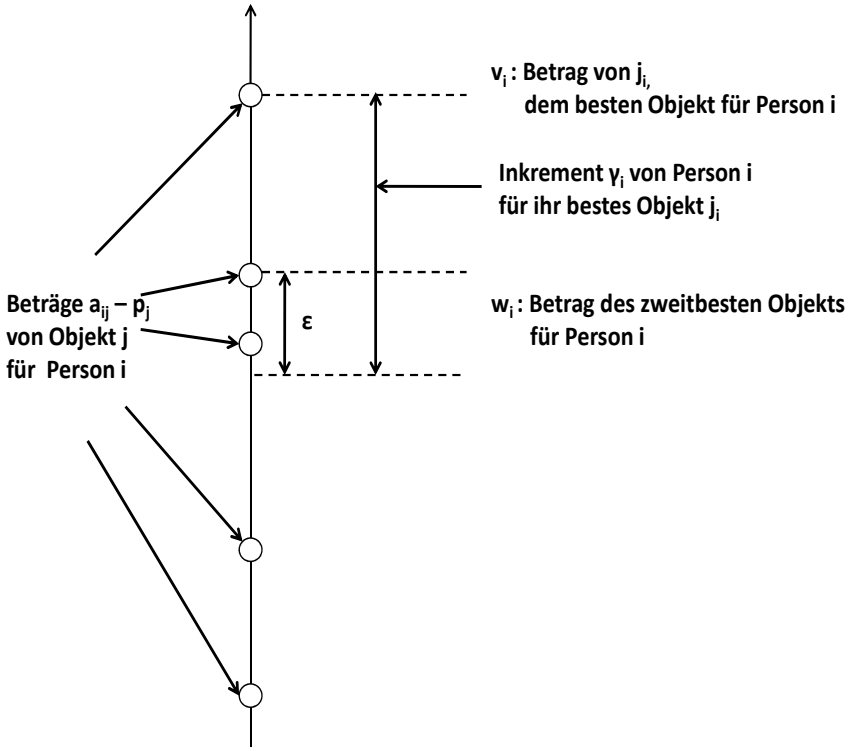
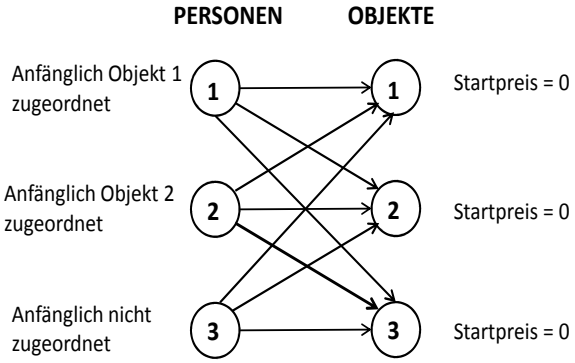


Abbildung 2.3: Der Nettobetrag, den das Objekt j_i der Person i liefert, ist immer noch um höchstens ϵ kleiner als der optimale, selbst wenn der Preis mit dem Inkrement $\gamma_i = v_i - w_i + \epsilon$ ansteigt, sodass die ϵ-KS-Bedingung erfüllt ist.



Hier ist $a_{ij} = C > 0$ für alle (i,j) mit i = 1,2,3 und j = 1,2 und $a_{ij} = 0$ für alle (i,j) mit i= 1,2,3 und j = 3

Zu Beginn von Iteration #	Objekt Preise	Zuordnungs-Paare	Bieter	Bevorzugtes Objekt	Inkrement γ
1	0,0,0	(1,1), (2,2)	3	2	ε
2	0,ε,0	(1,1), (3,2)	2	1	2ε
3	2ε, ε,0	(2,1), (3,2)	1	2	2ε
4	2ε,3ε,0	(1,2), (2,1)	3	1	2ε
5	4ε,3ε,0	(1,2), (3,1)	2	2	2ε
6	...	...	...	...	...

Abbildung 2.4: Das neue Inkrement $\gamma_i = v_i - w_i + \epsilon$ löst das Problem der naiven Auktion, da der Preis in jedem Iterationsschritt um mindestens ϵ steigt. Dadurch erhöhen die drei Personen die Preise der Objekte 1 und 2 so lange, bis sie C erreichen. Sobald die Preise der Objekte 1 und 2 den Wert C erreichen bzw. übersteigen, erhält auch Objekt 3 ein Gebot. Damit sind alle 3 Objekte zugeordnet und der Algorithmus endet.

wird sein Gebot nicht zu hoch zu setzen. Ist das Gebot zu hoch, so könnte es sein, dass er unnötigerweise zu viel für dieses Objekt bezahlt, da die anderen Personen eventuell schon bei einem niedrigeren Preis ausgestiegen wären. Es kann gezeigt werden, dass für ein ausreichend kleines ϵ die Zuordnung, die der Algorithmus ergibt, „beinahe optimal“ ist. Später wird gezeigt, dass der Gesamtnutzen um höchstens $n\epsilon$ geringer ist als der optimale Wert. Zur Erklärung wird wieder der Unterschied zwischen KS-Bedingung und ϵ-KS-Bedingung herangezogen. Erfüllen eine zulässige Zuordnung und ein Preisvektor die ϵ-KS-Bedingung, so kann man sagen, dass diese die KS-Bedingung für ein etwas anderes Problem erfüllen, wobei alle a_{ij} dem Originalproblem entsprechen mit Ausnahme der Werte der n zugeordneten Paare, die um höchstens ϵ verändert wurden.

Satz 2.3.1 *Ist eine Zuordnung zulässig und erfüllt gemeinsam mit einem Preisvektor die ϵ-KS-Bedingung, so ist der Unterschied zwischen dieser Zuordnung und der optimalen Zuordnung höchstens $n\epsilon$. Gleichzeitig ist der Unterschied zwischen dem Preisvektor und der optimalen Lösung des Problems ebenfalls höchstens $n\epsilon$.*
([3] S. 50/51 erweitert durch [5], S. 47/48)

Beweis:
Per Definition ist:
Optimaler Gesamtnutzen:

$$A^* = \max_{\substack{k_i, i=1,\ldots,n \\ k_i \neq k_m \text{ wenn } i \neq m}} \sum_{i=1}^{n} a_{ik_i}$$

Optimale duale Kosten:

$$D^* = \min_{p_j, j=1,\ldots,n} \left\{ \sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j \right\}$$

Nachdem wir eine zulässige Zuordnung $\{(i, j_i) \mid i = 1, ..., n\}$ annehmen, die gemeinsam mit einem Preisvektor $\overline{p}$ die ϵ-KS-

Bedingung erfüllt, gilt:

$$\max_{j \in A(i)} \{a_{ij} - \overline{p}_j\} - \epsilon \leq a_{ij_i} - \overline{p}_{j_i}.$$

Bildet man die Summe über alle i, beweist man den Satz:

$$D^* \leq \sum_{i=1}^{n} \left(\max_{j \in A(i)} \{a_{ij} - \overline{p}_j\} + \overline{p}_{j_i} \right) \leq \sum_{i=1}^{n} a_{ij_i} + n\epsilon \leq A^* + n\epsilon$$

Im Satz 2.1.1 wurde erörtert, dass $A^* = D^*$. Dies zeigt, dass die Aussage korrekt ist. Die zulässige Zuordnung und der Preisvektor $\overline{p}$ liefern optimale Werte, die sich um höchstens $n\epsilon$ von den Optima A^* und D^* unterscheiden.

Nimmt man an, dass alle Nutzenwerte a_{ij} ganzzahlig sind, was nicht unüblich ist, so kann man leicht eine Grenze finden. Nachdem sich der Gesamtnutzen um höchstens $n\epsilon$ vom optimalen Wert unterscheidet, muss lediglich sichergestellt werden, dass $n\epsilon < 1$. Unter dieser Voraussetzung ist die Zuordnung optimal. Das heißt dass man sich vergewissern muss, dass

$$\epsilon < \frac{1}{n}$$

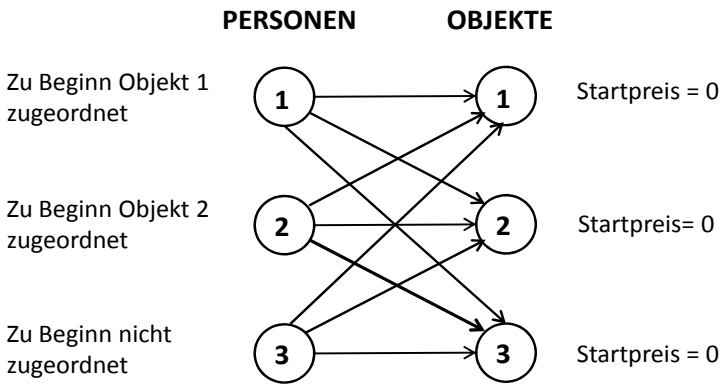
ist. Abbildung (2.5) zeigt die Folge der Preise p_1 und p_2 für das Beispiel in Abbildung (2.4).

2.3.2 Der Algorithmus

Die Iteration beginnt mit einer Teilzuordnung S und einem Preisvektor p, welche die ϵ-KS-Bedingung erfüllen:

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon, \ \forall (i,j) \in S.$$

Es kann ein beliebiger Preisvektor mit einer leeren Zuordnung gewählt werden, solange die ϵ-KS-Bedingung erfüllt ist.
Um den Algorithmus zu strukturieren, wird er in zwei Phasen eingeteilt: die Steigerungsphase und die Zuordnungsphase. Bertsekas erläutert diesen Algorithmus unter anderem in 'Linear Network Optimization, Algorithms and



Hier ist $a_{ij} = C > 0$ für alle (i,j) mit i = 1,2,3 und j = 1,2
und $a_{ij} = 0$ für alle (i,j) mit i= 1,2,3 und j = 3

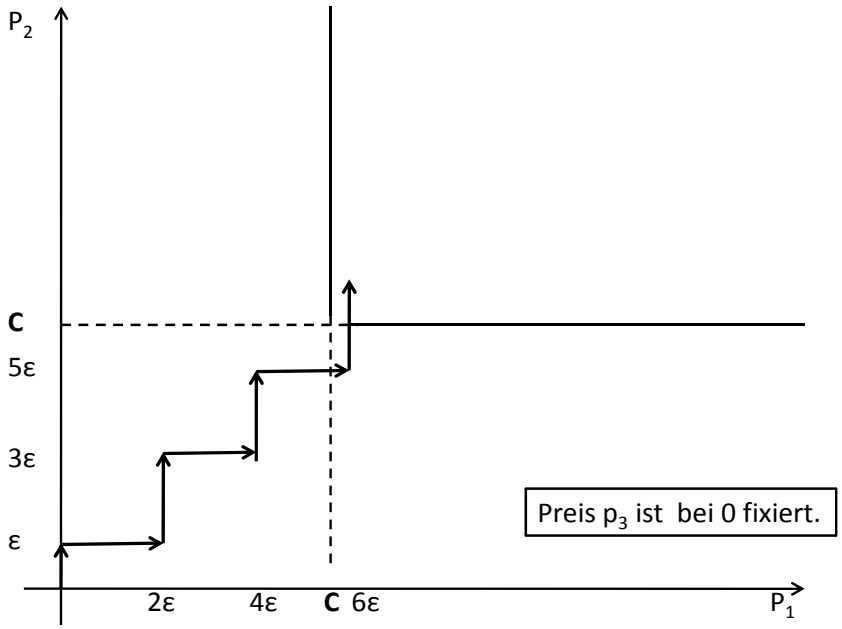


Abbildung 2.5: Der Algorithmus erzeugt eine Folge von p_1 und p_2. Die Preise steigen so lange, bis sie C erreichen.

Codes' [5].

Steigerungsphase

Angenommen I ist die nichtleere Untermenge der Personen, die bisher noch nicht zugeordnet sind. Für jede Person $i \in I$ müssen nun folgende Schritte durchgeführt werden:

1. Suche das beste Objekt j_i, welches der Person i den höchsten Nettobetrag liefert:

   $$j_i = \arg \max_{j \in A(i)} \{a_{ij} - p_j\}.$$

   Der zugehörige Nettobetrag wird durch

   $$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}$$

   dargestellt.
   Nachdem das beste Objekt gefunden wurde, sucht man nach dem zweitbesten und berechnet den zugehörigen Nettobetrag:

   $$w_i = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\}.$$

2. Nachdem ein Bieter gefunden wurde, muss noch das Gebot berechnet werden. Da der Bieter bereit ist den Preis höchstens um die Differenz zwischen dem besten und dem zweitbesten Objekt zu erhöhen, hat das Gebot folgenden Wert:

   $$b_{ij_i} = p_{j_i} + v_i - w_i + \epsilon = a_{ij_i} - w_i + \epsilon. \tag{2.5}$$

   Eigentlich könnte man jeden Wert zwischen $p_{j_i} + \epsilon$ und $p_{j_i} + v_i - w_i + \epsilon$ wählen. Allerdings endet der Algorithmus schneller, je größer die Inkremente sind.

Zuordnungsphase

Nachdem für die Objekte Bieter gefunden wurden, bleibt nun noch zu sehen, wer das höchste Gebot setzt. $P(j)$ wird als Menge

aller Personen angenommen, die auf Objekt j bieten. Wenn $P(j)$ eine nichtleere Menge ist, so kann man einfach das höchste Gebot finden:

$$p_j := \max_{i \in P(j)} b_{ij}.$$

Nun müssen alle Paare (i, j) aus der Teilzuordnung gelöscht werden, um das Paar (i_j, j) zur Menge der zugeordneten Person-Objekt-Paare hinzuzufügen.

Es gibt viele Möglichkeiten um die Teilmenge I der Personen, die während eines Iterationsschrittes bieten dürfen, auszuwählen. Im Anhang befindet sich die *Gauss-Seidel*-Version in Matlab programmiert. Mit diesen Programmcodes werden die Beispiele am Ende der jeweiligen Kapitel gelöst. Diese Version wird auf Grund ihrer Ähnlichkeit zur Gauss-Seidel-Methode so genannt, die für das Lösen nichtlinearer Gleichungen benutzt wird. Hierbei kann an jedem Iterationsschritt jeweils nur eine Person teilnehmen.
Während eines Iterationsschrittes ändern sich nur die Preise jener Objekte, die ein neues Gebot erhalten haben. Der Preis jedes Objektes, welches ein Gebot erhalten hat, erhöht sich um mindestens ϵ. In der folgenden Gleichung kann man sehen, dass diese Aussage gilt:

$$b_{ij_i} = a_{ij_i} - w_i + \epsilon \geq a_{ij_i} - v_i + \epsilon = p_{j_i} + \epsilon.$$

Nach jeder Iteration steigt also zumindest ein Preis. Das heißt allerdings nicht, dass auch die Menge der zugeordneten Person-Objekt-Paare zunimmt, da immer noch die Option besteht, dass eine Person auf ein Objekt setzt, welches davor bereits einer anderen Person zugewiesen war. Also kann die Menge der zugeordneten Person-Objekt-Paare entweder steigen oder gleich bleiben.
Mit den folgenden Aussagen kann die Gültigkeit des Algorithmus gezeigt werden:

1. Die ϵ-KS-Bedingung ist während der gesamten Durchführung des Algorithmus erfüllt. Da die ϵ-KS-Bedingung zu Beginn für die Ausgangszuordnung und den Ausgangspreisvektor erfüllt sein muss, reicht die spezielle Form der Inkremente $\gamma_i = v_i - w_i + \epsilon$, um zu versichern, dass die ϵ-KS-Bedingung auch während der Durchführung erfüllt bleibt.

Beweis: Objekt j^* erhält ein Gebot von Person i und ist während des gesamten Iterationsschrittes dieser Person i zugeordnet. Wenn p_j der Preis vor und p'_j der Preis nach der Zuordnungsphase ist, erhält man folgende Gleichung:

$$p'_{j^*} = a_{ij^*} - w_i + \epsilon.$$

Aus dieser Gleichung sieht man:

$$a_{ij^*} - p'_{j^*} = w_i - \epsilon = \max_{j \in A(i), j \neq j^*} \{a_{ij} - p_j\} - \epsilon.$$

Wie bereits erörtert gilt $p'_j \geq p_j$ für alle j, was Folgendes impliziert:

$$a_{ij^*} - p'_{j^*} = \max_{j \in A(i)} \{a_{ij} - p'_j\} - \epsilon.$$

Dies versichert, dass die ϵ-KS-Bedingung auch für alle neuen Paare (i, j^*), die erst in diesem Iterationsschritt hinzugekommen sind, erfüllt ist.
Für alle Preis-Objekt-Paare, die während des Iterationsschrittes nicht verändert wurden, gilt die ϵ-KS-Bedingung, da weder Zuordnung noch Preise verändert wurden.

([11], S. 9/10)

2. Ein Objekt, das während der Durchführung des Algorithmus einmal einer Person zugeordnet wurde, bleibt während der gesamten Durchführung zugeordnet, da nur Personen Objekte verlieren können, Objekte allerdings lediglich einen neuen Eigentümer erhalten. Nachdem Objekte, wenn sie einmal zugeordnet sind, nur den Eigentümer wechseln, jedoch immer zugeordnet bleiben, endet der Algorithmus, sobald alle Objekte zugeordnet sind. Das heißt, dass es bis zum letzten Iterationsschritt mindestens ein Objekt gibt, dass noch niemals zugeordnet wurde.
([3], S. 52)

3. Sobald ein Objekt ein Gebot erhält, steigt der Preis um mindestens ϵ.

Falls ein Objekt eine unendliche Anzahl von Geboten erhält, so steigt der Preis auf ∞.
([3], S. 52)

4. Der Wert von v_i, der durch

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}$$

definiert wird, sinkt alle $\mid A(i) \mid$ Gebote von Person i um mindestens ϵ. $\mid A(i) \mid$ stellt hier die Anzahl der Objekte der Menge $A(i)$ dar, wobei $A(i)$ die Menge aller Objekte ist, die der Person i zugeordnet werden können.
Wenn Person i steigert, nimmt v_i entweder um mindestens ϵ ab oder bleibt gleich, falls es mehr als ein Objekt gibt, welches maximalen Nutzen liefert. Nimmt v_i ab, so ist die Aussage klar. Bleibt v_i allerdings gleich, so steigt der Preis von Objekt j_i, welches das Gebot erhalten hat, um mindestens ϵ an. Objekt j_i wird jedoch kein weiteres Gebot von Person i erhalten, wenn der Wert v_i nicht um mindestens ϵ abnimmt. Nachdem der Wert v_i zumindest alle $\mid A(i) \mid$ Gebote abnimmt, sinkt der Wert v_i auf $-\infty$, falls Person i unendlich oft Gebote setzt.
([3], S. 52)

5. Die Existenz einer zulässigen Zuordnung versichert nicht nur die Beendigung des Algorithmus, sondern auch, dass sich die Lösung um höchstens $n\epsilon$ von der optimalen Zuordnung unterscheidet.

   Beweis: Diese Aussage kann durch Widerspruch gezeigt werden. Findet der Algorithmus keine zulässige Lösung, so ist die Teilmenge J^∞ der Objekte, die eine unendliche Anzahl an Geboten erhalten haben, nichtleer. Da es gleich viele Objekte wie Personen gibt und jedes Objekt nur einer Person zugeordnet werden kann, muss die Teilmenge I^∞ der Personen, die unendlich oft steigern, ebenfalls nichtleer sein. Laut 3. müssen die Preise der Objekte gegen ∞ gehen und laut 4. die Werte $v_i = \max_{j \in A(i)}\{a_{ij} - p_j\}$ gegen $-\infty$ divergieren.

Dies verlangt, dass

$$A(i) \subset J^{\infty}, \forall i \in I^{\infty}$$

ist. In 1. wurde allerdings gezeigt, dass die ϵ-KS-Bedingung $a_{ij} - p_j \geq v_i - \epsilon$ für alle zugeordneten Paare (i, j) gilt. Das bedeutet, dass nach einer endlichen Anzahl an Geboten die Objekte der Menge J^{∞} nur Personen der Menge I^{∞} zugeordnet werden können. Damit der Algorithmus nicht endet, muss es nach jedem Iterationsschritt zumindest eine nicht zugeordnete Person geben. Das bedeutet, dass nach einer endlichen Anzahl an Iterationsschritten zu Beginn jedes Iterationsschrittes zumindest eine Person von I^{∞} nicht zugeordnet ist, was bedeutet, dass die Anzahl der Personen in I^{∞} größer sein muss als die Anzahl der Objekte in J^{∞}. Das wiederum bedeutet, dass keine zulässige Zuordnung existieren kann, nachdem gezeigt wurde, dass jede Person von I^{∞} nur einer Person aus J^{∞} zugeordnet werden kann. Das heißt, dass der Algorithmus, wenn eine zulässige Zuordnung existiert, enden muss.
Laut 1. erfüllt die zulässige Zuordnung die ϵ-KS-Bedingung. Satz 2.3.1 zeigt, dass sich diese Zuordnung um höchstens $n\epsilon$ vom Optimum unterscheidet.

([3], S. 52/53)

2.3.3 Beispiele

Der Algorithmus wurde in Matlab implementiert. Die Preise werden Null gesetzt, da so die ϵ-KS-Bedingung erfüllt ist.

Beispiel 1

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 1.2500 & 1.2500 & 2.5000 \end{pmatrix}$$

Beispiel 2

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 & 1 \\ 2 & 2 & 1 & 4 \\ 3 & 2 & 4 & 3 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 2.4000 & 1.2000 & 1.2000 & 2.2000 \end{pmatrix}$$

2.4 Rückwärtsauktion

2.4.1 Einleitung

Beim Zuordnungsproblem geht es darum, für jede Person das beste Objekt zu finden. Nun betrachtet man das umgekehrte Problem, wobei die Objekte um die Personen konkurrieren.
Um den Auktionsalgorithmus auf dieses neue Problem anzupassen, wird eine neue Variable eingeführt. Die Profitvariable π_i gibt den Profit an, den ein Objekt macht, wenn es Person i wählt. Profite haben dieselbe Bedeutung für Objekte wie Preise für Personen.
Man kann das Problem also folgenderweise erklären:
Noch nicht zugewiesene Objekte steigern um Personen, wählen die beste Person aus und erlangen dadurch einen möglichst hohen Profit ohne die ϵ-KS-Bedingung zu verletzen.
Wichtig ist nun zu sehen, wie die ϵ-KS-Bedingung für dieses neue Problem aussieht:
Eine (Teil-)Zuordnung S und ein Profitvektor π erfüllen die ϵ-KS-Bedingung, wenn

$$a_{ij} - \pi_i \geq \max_{k \in B(j)} \{a_{kj} - \pi_k\} - \epsilon, \ \forall (i,j) \in S, \tag{2.6}$$

wobei $B(j)$ die Menge aller Personen darstellt, die Objekt j zugeordnet werden können:

$$B(j) = \{i | (i,j) \in A\}.$$

Um zu versichern, dass eine zulässige Lösung für das Problem existiert, nimmt man an, dass $B(j)$ für alle j keine leere Menge ist.

2.4.2 Der Algorithmus

Dieser Algorithmus wird unter anderem in 'Reverse Auction and the Solution of Inequality Constrained Assignment Problems' [12] detailliert erklärt.
Der Rückwärtsalgorithmus beginnt mit einer Teilzuordnung S. Mit jedem Iterationsschritt wird die Menge S entweder größer oder bleibt gleich. Der Algorithmus endet, wenn jedes Objekt einer Person zugeordnet ist. Die ϵ-KS-Bedingung ist zu Beginn jedes Iterationsschrittes erfüllt.

Steigerungsphase

Um mit der Steigerungsphase zu beginnen, darf die Teilmenge J der Objekte, die unter der Teilzuordnung S noch nicht zugeordnet sind, keine leere Menge sein. Für jedes $j \in J$ sind nun folgende Schritte durchzuführen:

1. Suche die beste Person i_j, die also dem Objekt j den höchsten Wert liefert:

$$i_j = \arg \max_{i \in B(j)} \{a_{ij} - \pi_i\}.$$

   Der zugehörige Wert wird durch

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\}$$

   dargestellt.
   Nachdem man die beste Person gefunden hat, wird nach dem Wert der zweitbesten Person gesucht:

$$w_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}.$$

2. Nachdem Bieter gefunden wurden, muss noch das Gebot berechnet werden. Vergleicht man Rückwärts- und Vorwärtsauktion, so sieht man, dass das optimale Gebot, welches Objekt $j \in J$ Person i_j bieten kann ohne die ϵ-KS-Bedingung zu verletzen, folgende Form hat:

$$b_{i_j j} = \pi_{i_j} + \beta_j - w_j + \epsilon = a_{i_j j} - w_j + \epsilon.$$

Zuordnungsphase

Nachdem Bieter für die Personen gefunden wurden, müssen noch die höchsten Gebote gefunden werden. Angenommen $P(i)$ ist die nichtleere Menge der Objekte, von denen Person i ein Gebot

erhalten hat, so steigt der Profit π_i auf das höchste Gebot:

$$\pi_i := \max_{j \in P(i)} b_{ij}.$$

Nun müssen alle Paare (i, j) aus der Teilzuordnung gelöscht werden, um das Paar (i, j_i) zur Menge der zugeordneten Person-Objekt-Paare hinzuzufügen.

Der Rückwärtsalgorithmus ist also identisch mit dem Vorwärtsalgorithmus, mit dem einzigen Unterschied, dass Objekte und Personen Rollen tauschen. Daher gilt folgender Satz:

Satz 2.4.1 *Existiert für ein Zuordnungsproblem zumindest eine zulässige Lösung, so unterscheidet sich die Lösung des Rückwärtsalgorithmus um höchstens $n\epsilon$ von der optimalen Lösung. (Sind alle Einträge der Matrix A ganzzahlig, so genügt $\epsilon < 1/n$, sodass die Lösung optimal ist.)*
([12], S. 7)

2.4.3 Beispiele

Der Algorithmus wurde in Matlab implementiert. Die Profite werden Null gesetzt, da so die ϵ-KS-Bedingung erfüllt ist.

Beispiel 1

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Profitvektor:

$$\begin{pmatrix} 1.2500 \\ 2.5000 \\ 3.2500 \end{pmatrix}$$

Beispiel 2

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 & 1 \\ 2 & 2 & 1 & 4 \\ 3 & 2 & 4 & 3 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Profitvektor:

$$\begin{pmatrix} 1.2000 \\ 3.4000 \\ 2.4000 \\ 1.2000 \end{pmatrix}$$

2.5 Kombinierte Auktion

2.5.1 Einleitung

Die kombinierte Auktion ist ein Algorithmus, der Vorwärts- und Rückwärtsauktion kombiniert. Hierbei wechselt man zwischen Vorwärts- und Rückwärtsauktion und führt sowohl einen Preisvektor p mit, der die ϵ-KS-Bedingung (2.4) erfüllt, als auch einen Profitvektor π, der die ϵ-KS-Bedingung (2.6) erfüllt. Um dieses neue Problem zu vereinfachen, müssen neue ϵ-KS-Bedingungen für das Paar (π, p) eingeführt werden, die sicherstellen, dass die anderen beiden ϵ-KS-Bedingungen erfüllt sind. Es ist sehr wichtig, dass diese ϵ-KS-Bedingungen während der gesamten Durchführung erfüllt sind. Andernfalls können beim Wechsel zwischen Vorwärts- und Rückwärtsauktion große Probleme auftreten. Bertsekas erklärt den kombinierten Algorithmus unter anderem in 'Linear Network Optimization, Algorithms and Codes' [5].
Die ϵ-KS-Bedingungen für die kombinierte Auktion sehen wie folgt aus: Eine (Teil-)Zuordnung S und ein Profit-Preis-Paar (π, p) erfüllen die ϵ-KS-Bedingungen, wenn

$$\pi_i + p_j \geq a_{ij} - \epsilon, \ \forall (i,j) \in A, \tag{2.7}$$

$$\pi_i + p_j = a_{ij}, \ \forall (i,j) \in S. \tag{2.8}$$

Satz 2.5.1 *Erfüllt eine Zuordnung S gemeinsam mit einem Profit-Preis-Paar (π, p) die ϵ-KS-Bedingungen (2.7) und (2.8), so gelten folgende Aussagen:*

1. *Die Zuordnung S gemeinsam mit dem Profitvektor π erfüllt die ϵ-KS-Bedingung der Rückwärtsauktion:*

$$a_{ij} - \pi_i \geq \max_{k \in B(j)} \{a_{kj} - \pi_k\} - \epsilon, \ \forall (i,j) \in S. \tag{2.9}$$

2. *Die Zuordnung S gemeinsam mit dem Preisvektor p erfüllt die ϵ-KS-*

Bedingung der Vorwärtsauktion:

$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon, \ \forall (i,j) \in S. \tag{2.10}$$

3. *Wenn S eine zulässige Zuordnung ist, dann unterscheidet sich S um höchstens $n\epsilon$ von der optimalen Lösung.*

([12], S. 8)

Beweis:

1.
$$(2.8) \Longrightarrow p_j = a_{ij} - \pi_i, \ \forall (i,j) \in S$$
$$(2.7) \Longrightarrow \pi_i + p_j \geq a_{ij} - \epsilon, \ \forall (i,j) \in A$$
$$\Longrightarrow a_{ij} - \pi_i \geq a_{kj} - \pi_k - \epsilon, \forall k \in B(j)$$

   Nachdem diese (Un-)Gleichungen für alle k gelten, sind sie auch für das Maximum über alle k erfüllt.

2. Tauscht man die Rollen von π und p, so ist der Beweis wie in *1.* zu zeigen.

3. Nachdem laut *2.* die ϵ-KS-Bedingung (2.4) erfüllt ist, gilt die Aussage laut Seite 20 (5.).

Der kombinierte Auktionsalgorithmus beginnt mit einer (Teil-) Zuordnung S und einem Profit-Preis-Paar (π, p), die die ϵ-KS-Bedingungen (2.7) und (2.8) erfüllen. Der Algorithmus endet, wenn alle Objekte und Personen zugeordnet sind.

2.5.2 Der Algorithmus

Schritt 1 (Vorwärtsauktion): Einige Iterationsschritte der Vorwärtsauktion werden durchgeführt. Nach jedem Iterationsschritt wird für alle Person-Objekt-Paare (i, j_i), die in diesem Iterationsschritt zu der Teilzuordnung S hinzugefügt wurden, der Profit

wie folgt angepasst:

$$\pi_i = a_{ij_i} - p_{j_i}. \tag{2.11}$$

Nachdem π aktualisiert wurde, kann willkürlich zur Rückwärtsauktion gewechselt werden.

Schritt 2 (Rückwärtsauktion): Einige Iterationsschritte der Rückwärtsauktion werden durchgeführt. Nach jedem Iterationsschritt wird für alle Person-Objekt-Paare (i, j_i), die in diesem Iterationsschritt zu der Teilzuordnung S hinzugefügt wurden, der Preis wie folgt angepasst:

$$p_j = a_{ij} - \pi_{i_j}. \tag{2.12}$$

Nachdem p aktualisiert wurde, kann willkürlich zur Vorwärtsauktion gewechselt werden.

Der Algorithmus wechselt zwischen Vorwärts- und Rückwärtsauktion bis eine zulässige Zuordnung gefunden wird.

Der Unterschied zwischen der kombinierten Auktion und der Vorwärts- bzw. Rückwärtsauktion ist sehr klein. Es müssen lediglich die zusätzlichen Gleichungen (2.11) nach der Vorwärtsiteration und (2.12) nach der Rückwärtsiteration eingefügt werden. Allerdings ist es sehr wichtig, dass die ϵ-KS-Bedingungen (2.7) und (2.8) für (π, p) erfüllt sind, damit auch die ϵ-KS-Bedingungen der Vorwärts- bzw. Rückwärtsauktion nicht verletzt werden.

Satz 2.5.2 *Erfüllen die (Teil-)Zuordnung und das Profit-Preis-Paar die ϵ-KS-Bedingungen (2.7) und (2.8) zu Beginn jedes Iterationsschrittes sowohl der Vorwärts- als auch der Rückwärtsauktion, so sind die Bedingungen auch am Ende des Iterationsschrittes erfüllt, solange π nach jedem Iterationsschritt der Vorwärtsauktion durch (2.11) und p nach jedem Iterationsschritt der Rückwärtsauktion durch (2.12) aktualisiert wird.*
([12], S. 9/10)

Beweis:
Angenommen die Vorwärtsauktion wird angewendet:

(π, p), (π^n, p^n) ... Profit-Preis-Paar vor und nach dem Iterationsschritt
S, S^n ... (Teil-)Zuordnung vor und nach dem Iterationsschritt

$$p_j^n \geq p_j, \ \forall j$$

(Erhält Objekt j während des Iterationsschrittes ein Gebot, so gilt $p_j^n > p_j$.)

$$(2.7) \Longrightarrow \pi_i^n + p_j^n \geq a_{ij} - \epsilon, \ \forall (i,j) \text{ mit } \pi_i = \pi_i^n$$

$$(2.8) \Longrightarrow \pi_i^n + p_j^n = \pi_i + p_j = a_{ij}, \ \forall (i,j) \text{ mit } (i,j) \in S \cap (i,j) \in S^n$$

$$(2.11) \Longrightarrow \pi_i^n + p_{j_i}^n = a_{ij_i}, \ \forall (i,j_i) \text{ mit } (i,j_i) \notin S \cap (i,j_i) \in S^n$$

Nun bleibt noch zu zeigen, dass

$$\pi_i^n + p_j^n \geq a_{ij} - \epsilon, \ \forall j \in A(i)$$

für alle Personen i gilt, die ein Gebot gesetzt haben und in diesem Iterationsschritt einem Objekt j_i zugeordnet wurden.
Für solch eine Person i gilt folgende Gleichung (siehe Formel der Inkremente der Vorwärtsauktion, (2.5)):

$$p_{j_i}^n = a_{ij_i} - \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\} + \epsilon$$

$$\Longleftrightarrow a_{ij_i} - p_{j_i}^n = \max_{j \in A(i), j \neq j_i} \{a_{ij} - p_j\} - \epsilon$$

Nachdem für alle Paare (i, j_i), die in diesem Iterationsschritt in die Teilzuordnung hinzugefügt wurden, $\pi_i^n + p_{j_i}^n = a_{ij_i}$ gesetzt wird, gilt äquivalent:

$$\pi_i^n = a_{ij_i} - p_{j_i}^n, \forall (i,j_i) : (i,j_i) \notin S \cap (i,j_i) \in S^n.$$

Berücksichtigt man nun, dass $p_j^n \geq p_j$ und setzt alle Gleichungen

und Ungleichungen zusammen, so erhält man:

$$\pi_i^n = a_{ij_i} - p_{j_i}^n \geq a_{ij} - p_j - \epsilon \geq a_{ij} - p_j^n - \epsilon, \ \forall j \in A(i),$$

was wiederum äquivalent ist zu

$$\pi_i^n + p_j^n \geq a_{ij} - \epsilon, \ \forall j \in A(i).$$

Wendet man die Vorwärtsauktion an, so steigen die Preise und die Profite nehmen ab. Umgekehrt steigen die Profite und die Preise nehmen ab, wenn die Rückwärtsauktion angewendet wird. Es kann vorkommen, dass der kombinierte Algorithmus nicht endet, obwohl sowohl Vorwärts- als auch Rückwärtsauktion eine zulässige Lösung finden. Daher benötigt man eine neue Regel, um sicherzustellen, dass der Algorithmus endet. Um das Ende des Algorithmus zu garantieren, genügt es anzunehmen, dass der Algorithmus erst dann zwischen Vorwärts- und Rückwärtsauktion wechseln darf, wenn die (Teil-)Zuordnung um mindestens ein Person-Objekt-Paar erweitert wurde.

2.5.3 Beispiele

Der Algorithmus wurde in Matlab implementiert.
Im Gegensatz zu Vorwärts- und Rückwärtsauktion können die Preise bzw. Profite nicht einfach Null gesetzt werden. Nachdem die ϵ-KS-Bedingung $\pi_i + p_j \geq a_{ij} - \epsilon, \forall (i,j) \in A$ zu Beginn für das Profit-Preis-Paar erfüllt sein muss, wird die Initialisierung in der Implementierung wie folgt durchgeführt:

1. Setze für jedes Objekt den Preis auf das Minimum der Nutzenwerte aller Personen.

2. Setze alle Profite, für die die ϵ-KS-Bedingung noch nicht erfüllt ist, $\pi_i = a_{ij} - p_j$.

Diese Initialisierung kann für eine entsprechende Bandbreite von Problemen angewendet werden.

Beispiel 1

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 1.2500 & 2.2500 & 0 \end{pmatrix}$$

Optimaler Profitvektor:

$$\begin{pmatrix} 0.7500 \\ 0,7500 \\ 4.0000 \end{pmatrix}$$

Beispiel 2

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 & 1 \\ 2 & 2 & 1 & 4 \\ 3 & 2 & 4 & 3 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 1.0000 & 2.2000 & 0 & 1.0000 \end{pmatrix}$$

Optimaler Profitvektor:

$$\begin{pmatrix} 3.0000 \\ 0.8000 \\ 4.0000 \\ 3.0000 \end{pmatrix}$$

Kapitel 3

Auktionsalgorithmus für Asymmetrische Probleme

In diesem Kapitel wird zunächst das minimale Kostenflussproblem - ein für die asymmetrische Auktion relevantes Problem der Graphentheorie[1]- vorgestellt. Mit Hilfe der Dualität wird der Auktionsalgorithmus für *Asymmetrische* und *Mehrfachzuordnungsprobleme* adaptiert.

3.1 Das Minimale Kostenflussproblem

Ein einfaches Beispiel: Ein Produkt des Produzenten ($p = 1$) soll zum Käufer ($k = 4$) transportiert werden. Für den Transport kann entweder Transportfirma ($t = 2$) oder ($t = 3$), oder beide in Anspruch genommen werden. Jede Transportfirma hat bestimmte Kosten für den jeweiligen Abschnitt (siehe Abbildung (3.1), S. 35). Das Problem in Abbildung (3.1) kann als lineares Optimierungsproblem dargestellt werden:

[1] Im Anhang werden einige Grundlagen der Graphentheorie erklärt.

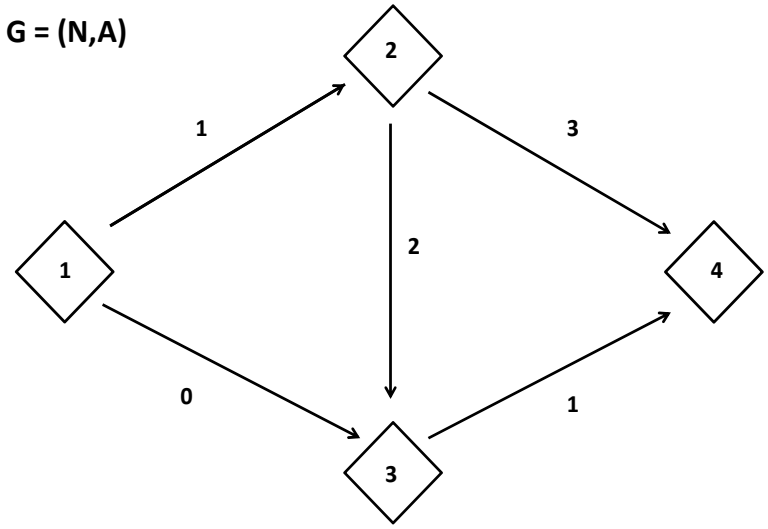


Abbildung 3.1: Minimales Kostenflussproblem

$$\min 1 \cdot x_{12} + 0 \cdot x_{13} + 2 \cdot x_{23} + 3 \cdot x_{24} + 1 \cdot x_{34}$$

unter den Nebenbedingungen

$$\begin{aligned} x_{12} + x_{13} &= 1 \\ -x_{12} + x_{23} + x_{24} &= 0 \\ -x_{13} - x_{23} + x_{34} &= 0 \\ -x_{24} - x_{34} &= -1 \end{aligned}$$

$$0 \leq x_{ij} \leq 1, \forall (i,j) \in A$$

Das *Minimale Kostenflussproblem* findet man in 'Linear Network Optimization, Algorithms and Codes' [5]. Genaueres darüber ist auch in 'Lineare Optimierung und Netzwerkoptimierung' [13] und 'Optimierung, Operations Research, Spieltheorie' [14] erklärt. Die verallgemeinerte Definition sieht folgendermaßen aus:

$$\min \sum_{(i,j)\in A} a_{ij}x_{ij}$$

unter den Nebenbedingungen

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i,\ \forall i \in N, \tag{3.1}$$

$$b_{ij} \leq x_{ij} \leq c_{ij},\ \forall (i,j) \in A, \tag{3.2}$$

wobei a_{ij}, b_{ij}, c_{ij} und s_i gegebene Skalare sind:
a_{ij} ... Kostenkoeffizient
b_{ij} und c_{ij} ... Flussgrenzen von (i,j)
$[b_{ij}, c_{ij}]$ ... zulässiger Flussbereich von (i,j)
s_i ... Angebot von Knoten i

Die Nebenbedingungen (3.1) und (3.2) werden *Kapazitätsbeschränkung* genannt. Gleichung (3.1) versichert, dass die Nachfrage zufriedengestellt wird, wobei (3.2) garantiert, dass der Fluss während des Prozesses die Kapazitäten nicht überschreitet.
Um Zulässigkeit zu gewährleisten, muss folgendes gelten:

$$\sum_{i\in N} s_i = 0.$$

Minimale Kostenflussprobleme werden gerne herangezogen, da sie eine vorteilhafte Struktur besitzen. Das *Zuordnungsproblem* ist ein Spezialfall des minimalen Kostenflussproblems. In den folgenden beiden Abschnitten werden noch zwei weitere Spezialfälle des minimalen Kostenflussproblems vorgestellt. Algorithmen zur Lösung des Minimalen Kostenflussproblems findet man in 'The Auction Algorithm for the Minimum Cost Network Flow Problem' [10] und 'Parallel Primal Dual Methods for the Minimum Cost Flow Problem' [8].

3.2 Transformierungen und Äquivalenzen

Minimales Kostenflussproblem

$$\min \sum_{(i,j)\in A} a_{ij}x_{ij}$$

unter den Nebenbedingungen

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i,\ \forall i \in N,$$

$$b_{ij} \leq x_{ij} \leq c_{ij},\ \forall (i,j) \in A.$$

Genaueres zu Transformierung und Äquivalenzen findet man unter anderem in 'Linear Network Optimization, Algorithms and Codes' [5].

Setzen der unteren Flussbeschränkungen auf Null

Dies kann man durch folgende Transformationen der Variablen erreichen:

$$x_{ij} \rightarrow x_{ij} - b_{ij}$$
$$c_{ij} \rightarrow c_{ij} - b_{ij}$$
$$b_{ij} \rightarrow b_{ij} - b_{ij}$$
$$(0 \leq x_{ij} - b_{ij} \leq c_{ij} - b_{ij})$$

$$\sum_{\{j|(i,j)\in A\}} (x_{ij} - b_{ij}) - \sum_{\{j|(j,i)\in A\}} (x_{ji} - b_{ji}) = s_i,\ \forall i \in N$$

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i + \sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji},\ \forall i \in N$$

$$s_i \rightarrow s_i + \sum_{\{j|(i,j)\in A\}} b_{ij} - \sum_{\{j|(j,i)\in A\}} b_{ji}$$

Eliminierung der oberen Flussbeschränkungen

Dafür wird eine Schlupfvariable eingeführt:

$$z_{ij} \geq 0 \rightarrow x_{ij} + z_{ij} = c_{ij}.$$

Weiters führt man für jede Kante (i,j) einen zusätzlichen Knoten mit Angebot c_{ij} und zwei den Flüssen x_{ij} und z_{ij} entsprechende von c_{ij} ausgehende Kanten ein (siehe Abbildung (3.2)).

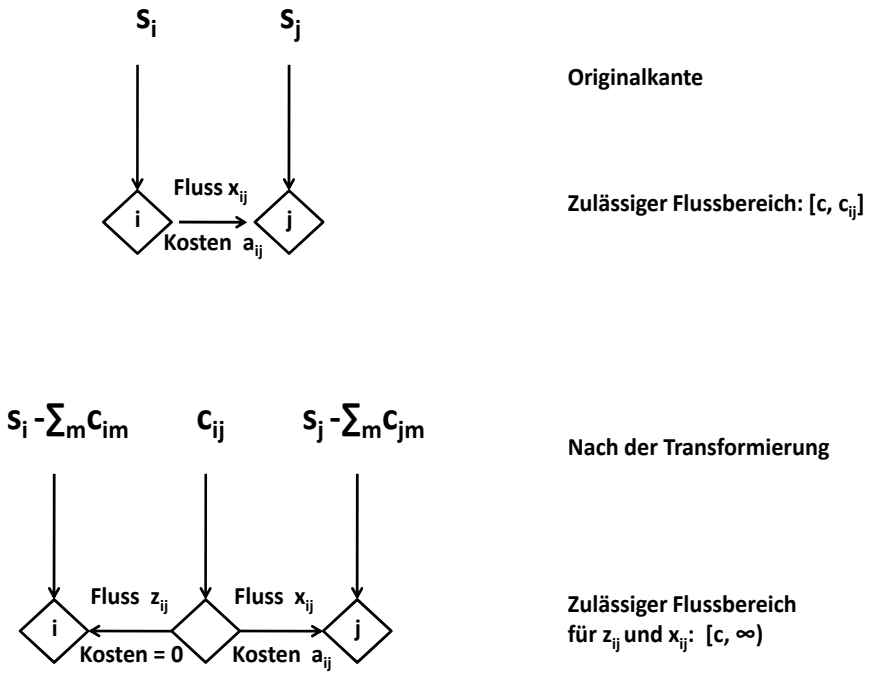


Abbildung 3.2: Transformierung, bei der die oberen Kapazitätsbeschränkungen durch Einführung eines neuen Knoten, der durch zwei Kanten die alte Kante ersetzt, eliminiert werden. Damit dies zulässig ist, muss $z_{ij} = c_{ij} - x_{ij}$ erfüllt sein. Dies erfordert, dass die künstliche Kante immer positiven Fluss haben muss ($0 \leq z_{ij}$), nachdem $x_{ij} \leq c_{ij}$.

Transformierung in eine Zirkulation

Eine Zirkulation impliziert, dass das Angebot aller Knoten Null ist.
Um ein minimales Kostenflussproblem in dieses spezielle Format zu bringen, führt man für alle Personen i mit $s_i > 0$ eine Kante (t, i) ein, wobei $0 \leq x_{ti} \leq s_i$ gilt, und für alle Personen i mit $s_i < 0$ eine Kante (i, t), wobei $0 \leq x_{it} \leq -s_i$ ist.
Die Kosten dieser künstlichen Kanten sollten sehr klein sein, um zu versichern, dass bei Erreichen des optimalen Flusses die Kapazitäten der künstlichen Kanten voll ausgenutzt werden (siehe Abbildung (3.3)).

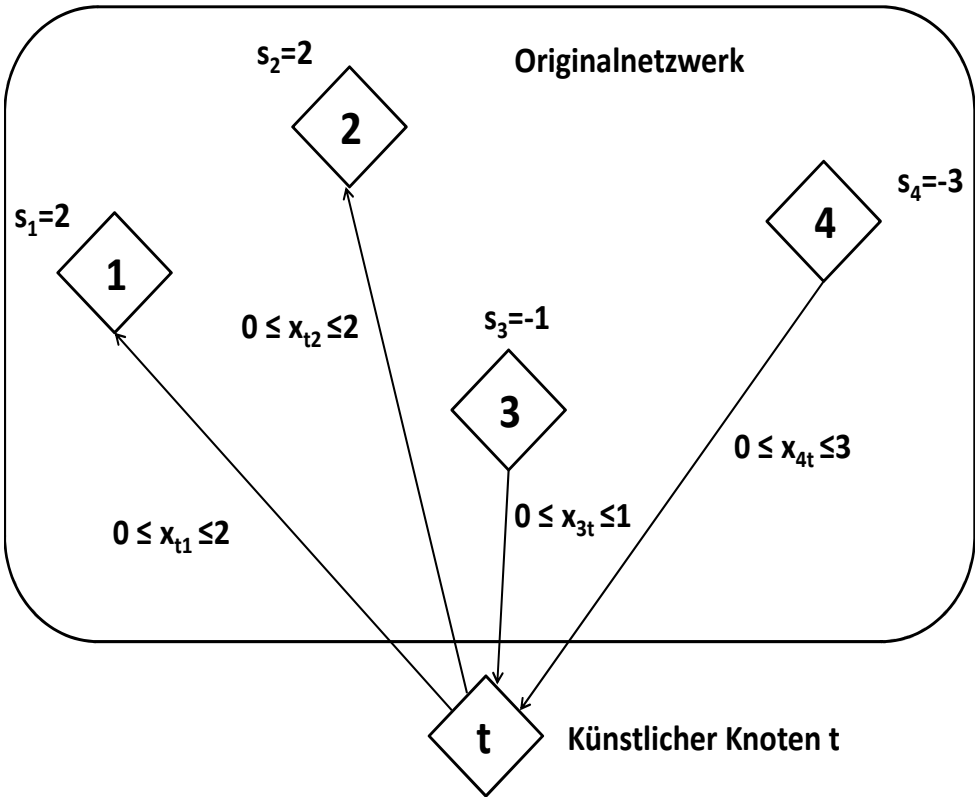


Abbildung 3.3: Transformierung in ein Zirkulationsformat

3.3 Dualität des Minimalen Kostenflussproblems

Die Dualität des minimalen Kostenflussproblems erläutert Bertsekas in 'Linear Network Optimization, Algorithms and Codes' [5]. In diesem Abschnitt wird das minimale Kostenflussproblem als primales Problem betrachtet. Um das duale Problem zu konstruieren, muss zuerst ein Preisvektor $p = \{p_j \mid j \in N\}$ eingeführt werden. Weiters gibt es nun eine komplementäre Schlupfbedingung für das Fluss-Preisvektor-Paar (x, p):
Ist der Flussvektor x zulässig (d.h. die Kapazitätsbedingungen sind erfüllt) und gelten folgende Bedingungen, so ist die KS-Bedingung des minimalen Kostenflussproblems erfüllt:

$$p_i - p_j \leq a_{ij},\ \forall (i,j) \in A \text{ mit } x_{ij} < c_{ij} \tag{3.3}$$

$$p_i - p_j \geq a_{ij},\ \forall (i,j) \in A \text{ mit } b_{ij} < x_{ij} \tag{3.4}$$

Diese Bedingungen indizieren, dass

$$p_i = a_{ij} + p_j,\ \forall (i,j) \in A \text{ mit } b_{ij} < x_{ij} < c_{ij} \tag{3.5}$$

ebenfalls erfüllt ist.
Äquivalent zu den KS-Bedingungen ist:

$$x_{ij} = \begin{cases} c_{ij} & \text{wenn } p_i > a_{ij} + p_j \\ b_{ij} & \text{wenn } p_i < a_{ij} + p_j \end{cases} \tag{3.6}$$

für alle $b_{ij} \leq x_{ij} \leq c_{ij}$.
Für den Spezialfall, dass x_{ij} nur Nichtnegativitätsbedingungen erfüllen muss $(0 \leq x_{ij})$, reichen folgende KS-Bedingungen aus:

$$p_i - p_j \leq a_{ij},\ \forall (i,j) \in A, \tag{3.7}$$

$$p_i - p_j = a_{ij},\ \forall (i,j) \in A \text{ mit } 0 \leq x_{ij}. \tag{3.8}$$

Für die Dualitätstheorie spielt die Methode der *Lagrange-Multiplikatoren* eine wichtige Rolle. Die Lagrange Theorie liefert eine Strategie Optimierungsprobleme mit Nebenbedingungen zu lösen. Die Lagrange Theorie wird in 'Constrained Optimization and Lagrange Multiplier Methods' [4] und 'Op-

timierung, Operations Research, Spieltheorie' [14] detailliert hergeleitet. In diesem Abschnitt werden die wichtigsten Ergebnisse angenommen.
Die Lagrange-Multiplikatoren stellen die dualen Variablen eines Optimierungsproblems dar. Daher wird nun p_i als Lagrange-Multiplikator des minimalen Kostenflussproblems angenommen. Die Lagrange-Funktion sieht folgendermaßen aus:

$$\begin{aligned} L(x,p) = \sum_{(i,j)\in A} a_{ij}x_{ij} + \sum_{i\in N}\left(s_i - \sum_{\{j|(i,j)\in A\}} x_{ij} + \sum_{\{j|(j,i)\in A\}} x_{ji}\right)p_i \\ = \sum_{(i,j)\in A}(a_{ij} + p_j - p_i)x_{ij} + \sum_{i\in N} s_i p_i. \end{aligned} \tag{3.9}$$

Die duale Funktion $d(p)$ ist durch

$$d(p) = \min_x \{L(x,p) \mid b_{ij} \le x_{ij} \le c_{ij}, (i,j) \in A\} \tag{3.10}$$

gegeben. Die Lagrange-Funktion $L(x,p)$ ist in x_{ij} trennbar, also kann die duale Funktion folgendermaßen angeschrieben werden:

$$d(p) = \sum_{(i,j)\in A} d_{ij}(p_i - p_j) + \sum_{i\in N} s_i p_i, \tag{3.11}$$

mit

$$\begin{aligned} d_{ij}(p_i - p_j) &= \min_{x_{ij}} \{(a_{ij} + p_j - p_i)x_{ij} \mid b_{ij} \le x_{ij} \le c_{ij}\} \\ &= \begin{cases} (a_{ij} + p_j - p_i)b_{ij} & \text{wenn } p_i \le a_{ij} + p_j \\ (a_{ij} + p_j - p_i)c_{ij} & \text{wenn } p_i > a_{ij} + p_j. \end{cases} \end{aligned} \tag{3.12}$$

Das duale Problem ist

$$\max d(p)$$
unter keinen Nebenbedingungen für p.

Satz 3.3.1 *Wenn man einen zulässigen Flussvektor x^* und einen Preisvektor p^* derartig wählt, dass die komplementären Schlupfbedingungen (3.7) und (3.8) erfüllt sind, so ist x^* die optimale Lösung des primalen und p^* die op-*

timale Lösung des dualen Problems. Es kann weiters gezeigt werden, dass die optimalen Kosten des primalen Problems den optimalen Kosten des dualen Problems entsprechen.
([5], S. 36-38)

Beweis:
Für jeden Preisvektor und jeden zulässigen Flussvektor gilt:

$$\begin{aligned} d(p) &\leq L(x,p) \\ &= \sum_{(i,j)\in A} a_{ij}x_{ij} + \sum_{i\in N}\left(s_i - \sum_{\{j|(i,j)\in A\}} x_{ij} + \sum_{\{j|(j,i)\in A\}} x_{ji}\right)p_i \\ &= \sum_{(i,j)\in A} a_{ij}x_{ij}. \end{aligned} \tag{3.13}$$

Die Zulässigkeit von x impliziert die letzte Gleichung. Dies zeigt, dass die primalen Kosten für jedes zulässige x nicht kleiner sind als die dualen Kosten von p.
Durch die Definition der dualen Funktion (3.10) erhält man:

$$\begin{aligned} d(p^*) = \min_x \{L(x,p^*) \mid b_{ij} \leq x_{ij} \leq c_{ij}, (i,j) \in A\} \\ =^{(1)} L(x^*,p^*) =^{(2)} \sum_{(i,j)\in A} a_{ij}x^*_{ij}. \end{aligned}$$

Nachdem das Optimum (x^*, p^*) den komplementären Schlupf erfüllt, gilt

$$x^*_{ij} = \underset{x_{ij}\in[b_{ij},c_{ij}]}{\operatorname{argmin}} \{(a_{ij} + p^*_j - p^*_i)x_{ij}\},\ \forall (i,j) \in A.$$

Dies impliziert Gleichung (1). Gleichung (2) folgt aus der Zulässigkeit von x^*.
x^* minimiert also die primalen Kosten der rechten Seite von (3.13), während p^* die linke Seite von (3.13) ($d(p)$) maximiert. Die optimalen Werte des primalen und des dualen Problems sind identisch.

3.4 Auktionsalgorithmus für Asymmetrische Zuordnungsprobleme

3.4.1 Einleitung

Bisher wurden nur symmetrische Zuordnungsprobleme betrachtet. In 'Linear Network Optimization, Algorithms and Codes' [5] und 'A Forward/Reverse Auction Algorithm for Asymmetric Assignment Problems' [7] wird auch ein Algorithmus zum Lösen asymmetrischer Probleme erklärt. Im symmetrischen Fall gibt es gleich viele Personen wie Objekte, sodass jede Person und jedes Objekt zugeordnet werden kann. Nun wird eine neue Möglichkeit betrachtet. Es wird angenommen, dass es mehr Objekte als Personen gibt. Das heißt, dass jede Person einem Objekt zugeordnet wird, allerdings nicht jedes Objekt einen Eigentümer erhält. Es gelten die selben Bedingungen wie bisher:
Zu Beginn gibt es eine Teilzuordnung S von Person-Objekt-Paaren (i, j). Für alle $(i, j) \in S$ gilt, dass $j \in A(i)$. Jede Person kann nur einem Objekt und jedes Objekt nur einer Person zugeordnet werden. Eine Lösung ist zulässig, wenn jede Person einem Objekt zugeordnet wurde.
Dieses Problem ist äquivalent zu folgendem linearen Optimierungsproblem:

$$\max \sum_{(i,j)\in A} a_{ij} x_{ij}$$

unter den Nebenbedingungen

$$\begin{aligned} \sum_{j\in A(i)} x_{ij} &= 1, \ \forall i = 1, ..., m, \\ \sum_{i\in B(j)} x_{ij} &\leq 1, \ \forall j = 1, ..., n, \\ 0 &\leq x_{ij}, \ \forall (i, j) \in A. \end{aligned}$$

Dieses Problem kann in ein minimales Kostenflussproblem transformiert werden:

$$\min \sum_{(i,j)\in A} (-a_{ij}) x_{ij}$$

unter den Nebenbedingungen

$$\sum_{j \in A(i)} x_{ij} = 1, \ \forall i = 1, ..., m,$$

$$\sum_{i \in B(j)} x_{ij} + x_{sj} = 1, \ \forall j = 1, ..., n,$$

$$\sum_{j=1}^{n} x_{sj} = n - m,$$

$$0 \leq x_{ij}, \ \forall (i,j) \in A,$$

$$0 \leq x_{sj}, \ \forall j = 1, ..., n.$$

Um das Originalproblem in dieses minimale Kostenflussproblem zu transformieren, wird lediglich minimiert anstatt maximiert und das Vorzeichen der Zielfunktion geändert. Weiters wird eine Superquelle mit künstlichen Kanten, die von der Superquelle s zu jedem Objekt j führen, hinzugefügt. Um die Zulässigkeit zu gewährleisten, müssen alle künstlichen Kanten Kosten Null und einen Flussbereich von $[0, \infty)$ haben (siehe Abbildung (3.4), S. 52).
Wendet man die Dualität an, wechselt von Maximierung zu Minimierung und verwendet $p_i = -\pi_i$, so erhält man folgendes duales Problem:

$$\min \ \sum_{i=1}^{m} \pi_i + \sum_{j=1}^{n} p_j - (n - m)\lambda$$

unter den Nebenbedingungen

$$\pi_i + p_j \geq a_{ij}, \ \forall (i,j) \in A, \tag{3.14}$$

$$\lambda \leq p_j, \ \forall j = 1, ..., n, \tag{3.15}$$

wobei λ den Preis der Superquelle s darstellt. Um den Auktionsalgorithmus diesem Problem anzupassen, muss man zuerst ϵ-KS-Bedingungen für dieses bestimmte Problem definieren.
Eine Zuordnung S und ein Paar (π, p) erfüllen die ϵ-KS-Bedingungen, wenn folgende Gleichungen gelten:

$$\pi_i + p_j \geq a_{ij} - \epsilon, \ \forall (i,j) \in A, \tag{3.16}$$

$$\pi_i + p_j = a_{ij}, \ \forall (i,j) \in S, \tag{3.17}$$

$$p_j \leq \min_{k\text{: zugeordnet unter } S} p_k, \ \forall j\text{: nicht zugeordnet unter } S. \tag{3.18}$$

Ähnlich wie für symmetrische Probleme gilt folgender Satz:

Satz 3.4.1 *Wenn eine Zuordnung zulässig ist und mit dem Paar (π, p) die ϵ-KS-Bedingungen erfüllt, so unterscheidet sich diese Zuordnung um höchstens $m\epsilon$ von der optimalen Lösung. Der Unterschied zwischen dieser Lösung und der optimalen Lösung des dualen Problems ist ebenfalls höchstens $m\epsilon$, wenn für (π^*, p^*, λ) folgende Bedingungen erfüllt sind:*

$$\lambda = \min_{k\text{: zugeordnet unter } S} p_k, \tag{3.19}$$

$$\pi_i^* = \pi_i + \epsilon, \ \forall i = 1, ..., m \tag{3.20}$$

$$p_j^* = \begin{cases} p_j, & \text{wenn } j \text{ unter } S \text{ zugeordnet ist,} \\ \lambda, & \text{wenn } j \text{ unter } S \text{ nicht zugeordnet ist,} \quad \forall j = 1, ..., n. \end{cases} \tag{3.21}$$

([5], S.183/184)

Beweis:
Erfüllen eine zulässige Zuordnung $\{(i, k_i) \mid i = 1, ..., m\}$ und $(\overline{\pi}, \overline{p}, \lambda\}$ die Bedingungen (3.14) und (3.15), so ergibt dies folgende Ungleichungen:

$$\sum_{i=1}^{m} a_{ik_i} \leq \sum_{i=1}^{m} \overline{\pi}_i + \sum_{i=1}^{m} \overline{p}_{k_i} \leq \sum_{i=1}^{m} \overline{\pi}_i + \sum_{i=1}^{n} \overline{p}_j - (n-m)\lambda.$$

Wird über alle zulässigen Zuordnungen $\{(i, k_i) \mid i = 1, ..., m\}$ maximiert und über alle dual-zulässigen $(\overline{\pi}, \overline{p}, \lambda)$ minimiert, ergibt dies

$$A^* \leq D^*.$$

Wird (π^*, p^*, λ) durch die Gleichungen (3.19),(3.20) und (3.21) definiert und stellt $S = \{(i, j_i) \mid i = 1, ..., m\}$ die optimale Zuordnung dar, die gemeinsam

mit (π, p) die ϵ-KS-Bedingungen erfüllt, so gilt:

$$\pi_i^* + p_{j_i}^* = a_{ij} + \epsilon, \ \forall i = 1, ..., m.$$

Nachdem (π^*, p^*, λ) für das duale Problem zulässig sind, gilt:

$$A^* \geq \sum_{i=1}^{m} a_{ij_i} = \sum_{i=1}^{m} \pi_i^* + \sum_{i=1}^{m} p_{j_i}^* - m\epsilon$$
$$\geq \sum_{i=1}^{m} \pi_i^* + \sum_{j=1}^{n} p_j^* - (n-m)\lambda - m\epsilon \geq D^* - m\epsilon.$$

$$\left.\begin{array}{l} A^* \leq D^* \\ A^* \geq D^* - m\epsilon \end{array}\right\} \Longrightarrow \begin{array}{l} A^* \text{ und } D^* \text{ unterscheiden sich um höchstens } m\epsilon \\ \text{von der optimalen Lösung} \end{array}$$

3.4.2 Der Algorithmus

Um dieses Problem mit Hilfe des Auktionsalgorithmus zu lösen, benutzt Bertsekas zuerst die Vorwärtsauktion - wie bereits für das symmetrische Problem definiert - um jeder Person ein Objekt zuzuordnen. All diese zugeordneten Objekte müssen unterschiedlich sein, da kein Objekt zwei Personen zugeordnet werden darf. Ist das Problem zulässig, so liefert dieser Prozess eine zulässige Lösung, die die ersten beiden ϵ-KS-Bedingungen (3.16) und (3.17) erfüllt. Diese Zuordnung muss allerdings nicht optimal sein, da die Preise der Objekte, die nicht zugeordnet wurden, nicht unbedingt minimal sind, was bedeutet, dass Bedingung (3.18) nicht unbedingt erfüllt ist.
Um eine Lösung zu finden, die alle drei ϵ-KS-Bedingungen erfüllt, muss in einem zweiten Schritt eine angepasste Rückwärtsauktion angewendet werden. In dieser Phase werden den Personen so lange andere Objekte zugewiesen, bis die Bedingung (3.18) ebenfalls erfüllt ist.
Um die angepasste Rückwärtsauktion anwenden zu können, muss zuerst λ als minimaler Preis aller bereits in der Vorwärtsauktion zugeordneten Objekte definiert werden:

$$\lambda = \min_{j\text{: zugeordnet unter der ursprünglichen Zuordnung } S} p_j.$$

Die Iteration der angepassten Rückwärtsauktion verläuft genauso wie die Rückwärtsauktion, die in Abschnitt 2.4 eingeführt wurde, mit dem einzigen Unterschied, dass hier nur jene Objekte an der Auktion teilnehmen dürfen, die bisher nicht zugeordnet waren und $p_j > \lambda$ erfüllen. Der Algorithmus endet, sobald alle Objekte entweder zugeordnet sind oder die Ungleichung $p_j \leq \lambda$ erfüllen. Das heißt, dass nun auch die Bedingung (3.18) erfüllt ist.

Angepasste Rückwärtsauktion

Steigerungsphase

Aus der Menge der nicht zugeordneten Objekte wird ein Objekt j ausgewählt, dass $p_j > \lambda$ erfüllt. Gibt es kein solches Objekt, so endet der Algorithmus.
Nun müssen folgende Schritte durchgeführt werden:
Suche die beste Person, die also dem Objekt j den höchsten Wert liefert:

$$i_j = \arg \max_{i \in B(j)} \{a_{ij} - \pi_i\}. \tag{3.22}$$

Der zugehörige Wert wird durch

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\} \tag{3.23}$$

dargestellt.
Nachdem die beste Person gefunden wurde, wird nach dem Wert der zweitbesten Person gesucht:

$$w_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{3.24}$$

Ist der Unterschied zwischen λ und β_j nicht größer als ϵ, so wird $p_j := \lambda$ gesetzt und ein neuer Iterationsschritt gestartet. Ist der Unterschied größer, so wird

$$\delta = \min\{\beta_j - \lambda, \beta_j - w_j + \epsilon\} \tag{3.25}$$

definiert.

Zuordnungsphase

Das Paar (i_j, j) wird nun zur Zuordnung S hinzugefügt, während das Objekt, welches bisher der Person i_j zugeordnet war, wieder in die Menge der nicht zugeordneten Objekte verschoben wird. Nun werden Preise und Profite angepasst:

$$p_j := \beta_j - \delta, \tag{3.26}$$

$$\pi_{i_j} := \pi_{i_j} + \delta. \tag{3.27}$$

Während der gesamten angepassten Rückwärtsauktion gilt $\delta \geq \epsilon$. Das heißt, dass die Preise monoton fallen, während die Profite monoton steigen.

Satz 3.4.2 *Die Lösung des asymmetrischen Zuordnungsproblems durch Anwendung der angepassten Rückwärtsauktion ist eine zulässige Zuordnung, die sich um höchstens $m\epsilon$ vom Optimum unterscheidet.*
([12], S.14-16 ,erweitert durch [5], S. 186/187)

Beweis:
Nachdem Satz 3.4.1 gilt, müssen noch folgende Aussagen geprüft werden:

1. Während der gesamten Durchführung des Algorithmus für asymmetrische Zuordnungsprobleme sind Bedingungen (3.16), (3.17) und
$$\lambda \leq \min_{k\text{: zugeordnet unter } S} p_j \tag{3.28}$$
erfüllt. Daher werden die ϵ-KS-Bedingungen (3.16), (3.17) und (3.18) am Ende des Algorithmus nicht verletzt.
2. Der Algorithmus endet.

Angenommen die Bedingungen (3.16), (3.17) und (3.28) sind zu Beginn eines bestimmten Iterationsschrittes erfüllt, so muss gezeigt werden, dass sie auch nach Durchführung des Iterationsschrittes erfüllt bleiben.
Es gibt zwei mögliche Iterationsschritte:

1. Die Zuordnung verändert sich nicht. Das heißt $\lambda \geq \beta_j - \epsilon$. Nachdem sich S nicht verändert, bleiben (3.17) und (3.28) erfüllt. Obwohl die Zuordnung gleich bleibt, ändert sich der Preis p_j zu $p_j := \lambda$. Also gilt

$$p_j := \lambda \geq \beta_j - \epsilon = \max_{i \in B(j)} \{a_{ij} - \pi_i\} - \epsilon.$$

Daher ist Bedingung (3.16) ebenfalls erfüllt.

2. Die Zuordnung verändert sich:
(π, p) ... Profit-Preis-Paar vor der Durchführung des Iterationsschrittes
$(\overline{\pi}, \overline{p})$ ... Profit-Preis-Paar nach der Durchführung des Iterationsschrittes
j, i_j ... Objekt/Person, das/die an der Iteration teilnimmt

$$\Longrightarrow \overline{\pi}_{i_j} + \overline{p}_j = a_{i_j j}$$

$$\forall i \neq i_j, k \neq j : \overline{\pi_i} = \pi_i \text{ und } \overline{p}_k = p_k$$

$\Longrightarrow$ (3.17) ist am Ende des Iterationsschrittes erfüllt.
Um zu zeigen, dass auch (3.16) am Ende des Iterationsschrittes erfüllt ist, sind zwei Fälle zu betrachten:

(a) $\underline{k \neq j :}$

$$\left.\begin{array}{l} \overline{p_k} = p_k \\ \overline{\pi_i} \geq \pi_i, \ \forall i \end{array}\right\} \Longrightarrow \overline{\pi_i} + \overline{p_k} \geq a_{ik} - \epsilon, \ \forall (i,k) \in A$$

(b) $\underline{k = j :}\ i = i_j :$

$$\overline{\pi_{i_j}} + \overline{p_j} = a_{i_j j} \Longrightarrow (3.16) \text{ ist erfüllt}$$

$\underline{k = j :}\ i \neq i_j :$
Nachdem aus Definition (3.25) ersichtlich ist, dass $\delta \geq \epsilon$ gilt, und der Algorithmus die Gleichungen (3.22) -

(3.25) beinhaltet, gilt

$$\overline{\pi_i} + \overline{p_j} = \pi_i + \overline{p_j} \geq \pi_i + \beta_j - (\beta_j - w_j + \epsilon) =$$
$$= \pi_i + w_j - \epsilon \geq \pi_i + (a_{ij} - \pi_i) - \epsilon = a_{ij} - \epsilon.$$

Nun wird noch Gleichung (3.28) betrachtet. Da der Fall betrachtet wird, in dem $\lambda \leq \beta_j - \epsilon$ gilt, erhält man

$$\epsilon \leq \beta_j - \lambda \leq \delta.$$

Addiert man noch Gleichung (3.26), so gilt

$$\overline{p_j} = \beta_j - \delta \geq \beta_j - (\beta_j - \lambda) = \lambda.$$

Nachdem gezeigt wurde, dass die ϵ-KS-Bedingungen erfüllt sind, muss noch gezeigt werden, dass der Algorithmus endet. Wie bereits erwähnt, gibt es zwei Möglichkeiten von Iterationsschritten:

1. Wenn $\lambda \geq \beta_j - \epsilon$ gilt, dann wird das Objekt nicht zu der Zuordnung hinzugefügt und der Preis p_j wird gleich λ gesetzt.

2. Wenn $\lambda < \beta_j - \epsilon$ gilt, dann gilt $\beta_j \geq w_j$ und daher $\delta \geq \epsilon$. In diesem Fall steigt π_{i_j} um mindestens ϵ.

Die Vorraussetzung für ein Objekt j, um an einem Iterationsschritt teilzunehmen, ist $p_j > \lambda$. Nachdem *1.* die Preise des teilnehmenden Objekts auf λ setzt, tritt Möglichkeit *1.* nur endlich oft auf. Daher würde der Profit einiger Personen gegen ∞ gehen, wenn der Algorithmus nicht endet. Laut Gleichungen (3.17) und (3.28) gilt

$$\pi_i = a_{ij} - p_j \leq a_{ij} - \lambda.$$

Das heißt, dass die Profite durch $\max_{(i,j)\in A} a_{ij} - \lambda$ beschränkt sind. Das wiederum bedeutet, dass die Profite nicht gegen ∞ gehen können und der Algorithmus endet.

3.4.3 Beispiele

Der Algorithmus wurde in Matlab implementiert.
Das Profit-Preis-Paar wurde wie bei der kombinierten Auktion initialisiert.

Beispiel 1

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 & 3 & 4 & 2 & 4 \\ 2 & 2 & 1 & 4 & 3 & 1 & 5 \\ 3 & 2 & 4 & 3 & 3 & 4 & 2 \\ 4 & 1 & 3 & 2 & 5 & 2 & 3 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 0 & 0 & 0.1250 & 0 & 0.8750 & 0 & 0.7500 \end{pmatrix}$$

Optimaler Profitvektor:

$$\begin{pmatrix} 3.1250 \\ 4.2500 \\ 3.8750 \\ 4.0000 \end{pmatrix}$$

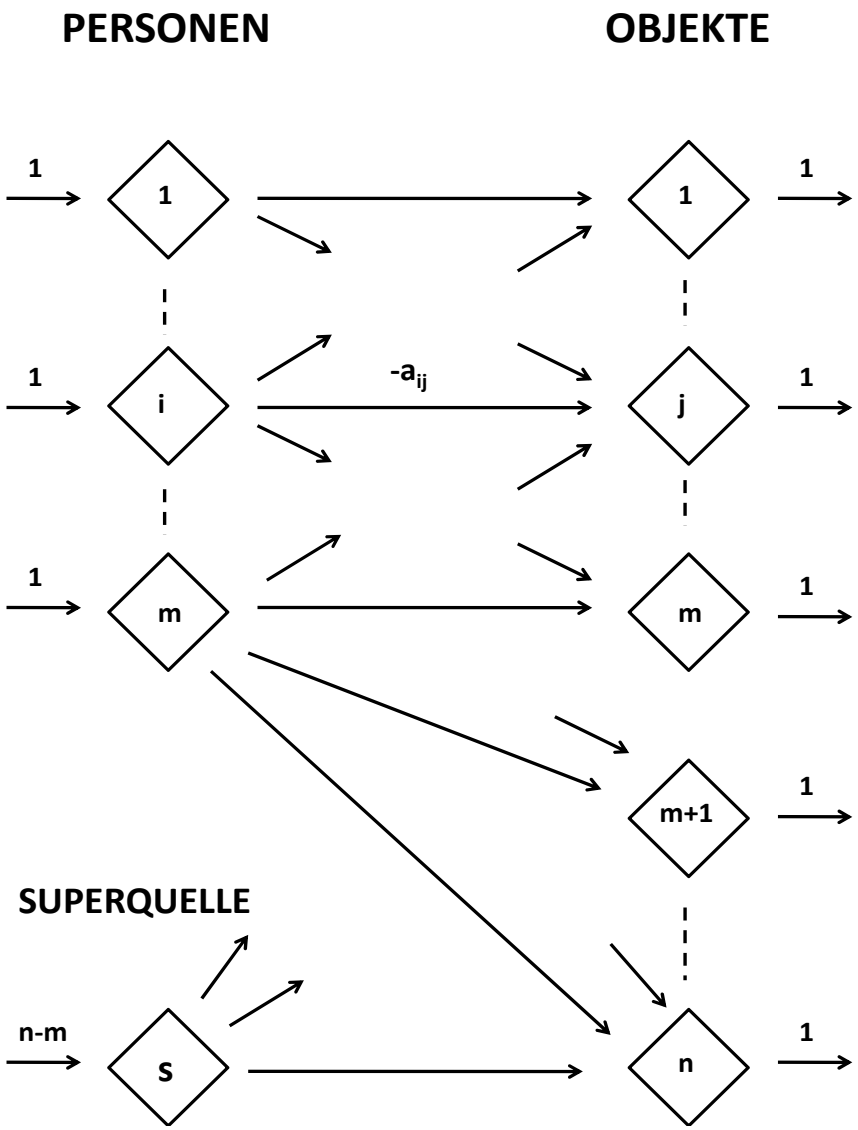


Abbildung 3.4: Durch das Einführen einer Superquelle s und künstlicher Kanten *(s,j)* mit Kosten gleich Null und einem zulässigen Flussbereich von $[0, \infty)$ für jedes Objekt j wird ein asymmetrisches Zuordnungsproblem in ein minimales Kostenflussproblem transformiert.

3.5 Auktionsalgorithmus für Mehrfachzuordnungsprobleme

3.5.1 Einleitung

In diesem Abschnitt werden ebenfalls asymmetrische Zuordnungsprobleme betrachtet. Das heißt, dass es mehr Objekte als Personen gibt. Der einzige Unterschied ist, dass eine Person nun mehrere Objekte ersteigern darf. Das heißt, dass das Problem nun folgendermaßen aussieht:

$$\max \sum_{(i,j)\in A} a_{ij}x_{ij}$$

unter den Nebenbedingungen

$$\sum_{j\in A(i)} x_{ij} \geq 1,\ \forall i = 1, ..., m,$$
$$\sum_{i\in B(j)} x_{ij} = 1,\ \forall j = 1, ..., n,$$
$$0 \leq x_{ij},\ \forall (i,j) \in A$$

mit $m < n$.
Um die Zulässigkeit sicherzustellen, müssen sowohl $A(i)$ als auch $B(j)$ für alle i und j nichtleer sein.
Dieses Problem kann in ein minimales Kostenflussproblem transformiert werden:

$$\min \sum_{(i,j)\in A} (-a_{ij})x_{ij}$$

unter den Nebenbedingungen

$$\sum_{j\in A(i)} x_{ij} - x_{si} = 1,\ \forall i = 1, ..., m,$$
$$\sum_{i\in B(j)} x_{ij} = 1,\ \forall j = 1, ..., n,$$
$$\sum_{i=1}^{m} x_{si} = n - m,$$

$$0 \leq x_{ij},\ \forall (i,j) \in A,$$

$$0 \leq x_{si},\ \forall i = 1, ..., n.$$

Um das Originalproblem in dieses minimale Kostenflussproblem zu konvertieren, muss die Maximierung durch Minimierung ersetzt und das Vorzeichen der Zielfunktion geändert werden. Weiters müssen eine Superquelle s und künstliche Kanten, die von der Superquelle zu allen Personen führen, eingeführt werden. Um die Zulässigkeit sicherzustellen, werden all diese künstlichen Kanten mit Kosten gleich Null und ein Flussbereich von $[0, \infty)$ definiert (siehe Abbildung (3.5)).
Wendet man die Dualitätstheorie an, so erhält man folgendes duale Problem:

$$\min \sum_{i=1}^{m} \pi_i + \sum_{j=1}^{n} p_j + (n-m)\lambda$$

unter den Nebenbedingungen

$$\pi_i + p_j \geq a_{ij},\ \forall (i,j) \in A, \tag{3.29}$$

$$\lambda \geq \pi_i,\ \forall i = 1, ..., m. \tag{3.30}$$

Um den Auktionsalgorithmus auf dieses Problem anzupassen, müssen ϵ-KS-Bedingungen für dieses spezielle Problem definiert werden. Eine Zuordnung S und ein Paar (π, p) erfüllen ϵ-KS, wenn folgende Bedingungen erfüllt sind:

$$\pi_i + p_j \geq a_{ij} - \epsilon,\ \forall (i,j) \in A, \tag{3.31}$$

$$\pi_i + p_j = a_{ij},\ \forall (i,j) \in S, \tag{3.32}$$

$$\pi_i = \max_{k=1,...,m} \pi_k,\ \forall i\text{: mehrfachzugeordnet unter S.} \tag{3.33}$$

Satz 3.5.1 *Wenn eine Zuordnung zulässig ist und gemeinsam mit dem Paar (π, p) die ϵ-KS-Bedingungen erfüllt, dann unterscheidet sich diese Zuordnung um höchstens $m\epsilon$ von der optimalen Lösung. Gleichzeitig ist der Unterschied zwischen dieser Lösung und der optimalen Lösung des dualen Problems eben-*

falls höchstens $m\epsilon$, wenn für (π^*, p, λ^*) folgende Bedingungen gelten:

$$\lambda^* = \max_{k=1,...,m} \pi_k^*, \tag{3.34}$$

$$\pi_i^* = \pi_i + \epsilon, \forall i = 1, ..., m. \tag{3.35}$$

([5], S. 190)

> Dieser Satz wird ohne Beweis angenommen, da die Beweisführung analog zu Satz 3.4.1 ist.

3.5.2 Der Algorithmus

In 'Reverse Auction and the Solution of Inequality Constrained Assignment Problems' [12] und 'Linear Network Optimization, Algorithms and Codes' [5] findet man einen Algorithmus, der dieses Mehrfachzuordnungsproblem löst. Um derartige Probleme mit dem Auktionsalgorithmus zu lösen, wird zuerst, wie bei der asymmetrischen Auktion, die Vorwärtsauktion solange angewendet, bis jede Person einem Objekt zugeordnet ist. Dabei ist es wichtig, dass die Anfangszuordnung und das initiale Profit-Preis-Paar (π, p) die ϵ-KS-Bedingungen (3.31) und (3.32) erfüllen. Die Zuordnung, die durch Anwendung der Vorwärtsauktion erreicht wird, ist nicht zulässig, da es noch nicht zugeordnete Objekte gibt. In einem zweiten Schritt müssen nun die optimalen Personen für diese Objekte gefunden werden. Dafür wird eine angepasste Rückwärtsauktion angewendet. Für diesen Algorithmus muss der maximale initiale Personenprofit λ definiert werden:

$$\lambda = \max_{i=1,...,m} \pi_i. \tag{3.36}$$

Dieser Skalar bleibt während der gesamten Durchführung des Algorithmus fixiert.
Die angepasste Rückwärtsauktion entspricht der Rückwärtsauktion für symmetrische Zuordnungsprobleme. Der einzige Unterschied ist, dass wenn ein Objekt einer Person zugeordnet wird, der bereits ein Objekt zugeordnet ist, dieses bereits zugeordnete Objekt erhalten bleiben kann und somit eine Person mehrere Objekte besitzen darf. Erreicht der Profit einer Person i die obere Grenze λ, so bleiben alle der Person i bereits zugeordneten Objekte

zugeordnet und teilen sich diese Person. Solange die obere Grenze λ nicht erreicht wird, erhält eine Person i immer nur ein Objekt. Der Algorithmus erfüllt die ϵ-KS-Bedingungen (3.31) und (3.32). Wenn alle Objekte zugeordnet sind, endet der Algorithmus. Am Ende des Algorithmus ist auch die Bedingung (3.33) erfüllt.

Angepasste Rückwärtsauktion

Steigerungsphase

Aus der Menge der nicht zugeordneten Objekte wird ein Objekt j ausgewählt und dessen beste Person i_j gesucht. Das ist die Person, die dem Objekt j den höchsten Wert liefert:

$$i_j = \arg \max_{i \in B(j)} \{a_{ij} - \pi_i\}. \tag{3.37}$$

Der zugehörige Wert wird durch

$$\beta_j = \max_{i \in B(j)} \{a_{ij} - \pi_i\} \tag{3.38}$$

dargestellt.
Nachdem die beste Person gefunden wurde, wird der Wert der zweitbesten Person gesucht:

$$w_j = \max_{i \in B(j), i \neq i_j} \{a_{ij} - \pi_i\}. \tag{3.39}$$

Definiere:

$$\delta = \min\{\lambda - \pi_{i_j}, \beta_j - w_j + \epsilon\}. \tag{3.40}$$

Zuordnungsphase

Nun wird das Paar (i_j, j) zu der Zuordnung S hinzugefügt und Preis und Profit folgendermaßen aktualisiert:

$$p_j := \beta_j - \delta, \tag{3.41}$$

$$\pi_{i_j} := \pi_{i_j} + \delta. \tag{3.42}$$

Das Objekt j' das bisher Person i_j zugeordnet war, wird von der Zuordnung entfernt, falls δ positiv ist.

Während der gesamten Durchführung des Algorithmus kann die Anzahl der zugeordneten Objekte entweder steigen oder gleich bleiben. Nur wenn δ gleich Null ist steigt die Anzahl($\delta = 0 \Longleftrightarrow \pi_{i_j} = \lambda$, weil $\beta_j - w_j + \epsilon \geq \epsilon, \forall j$).

Satz 3.5.2 *Die Lösung des Mehrfachzuordnungsproblems durch Anwendung der angepassten Rückwärtsauktion ist eine zulässige Zuordnung, die sich um höchstens $n\epsilon$ von der optimalen Lösung unterscheidet.*
([5], S. 192)

Beweis: Ähnlich wie im Beweis für Satz 3.4.2 ist Folgendes zu zeigen:

1. Während der gesamten Durchführung des Algorithmus für Mehrfachzuordnungsprobleme sind Bedingungen (3.31), (3.32) und
$$\lambda = \max_{i=1,\ldots,m} \pi_i \tag{3.43}$$
erfüllt. Daher werden die ϵ-KS-Bedingungen (3.31), (3.32) und (3.33) am Ende des Algorithmus nicht verletzt.
2. Der Algorithmus endet.

Um *1.* zu zeigen, muss gezeigt werden, dass, wenn die Bedingungen (3.31), (3.32), (3.33) und (3.43) am Anfang jedes Iterationsschrittes erfüllt sind, diese auch am Ende des Iterationsschrittes gelten.
Das einzige Preis-Profit Paar, welches sich ändert, ist:

$$p_j^n := \beta_j - \delta = a_{i_j j} - \pi_{i_j} - \delta$$

$$\pi_{i_j}^n := \pi_{i_j} + \delta$$

$$\Longrightarrow \pi_{i_j}^n + p_j^n = \pi_{i_j} + \delta + a_{i_j j} - \pi_{i_j} - \delta = a_{i_j j}$$

$\Longrightarrow$ (3.31) und (3.32) sind auch nach dem Iterationsschritt erfüllt.

Nachdem π_{i_j} der einzige Profit ist, der sich ändert, und entweder auf λ oder einen Wert niedriger als λ gesetzt wird, gilt nach dem Iterationsschritt $\lambda = \max_{i=1,\dots,m} \pi_i$. Das heißt, dass die Bedingungen (3.33) und (3.43) auch nach dem Iterationsschritt erfüllt sind.
Um zu zeigen, dass der Algorithmus endet, zieht man in Betracht, dass in jedem Iterationsschritt der Profit einer Person entweder um mindestens ϵ steigt oder auf λ gesetzt wird. Das heißt, dass nur endlich oft Profite vorkommen, die kleiner als λ sind. Erreicht der Profit einer Person allerdings λ, so bleiben alle Objekte, die dieser Person bereits zugeordnet wurden, zugeordnet. Das heißt, dass früher oder später alle Objekte zugeordnet sind. Andererseits sind die Profite der einzelnen Personen durch λ beschränkt. Das heißt, dass Personen nur eine endliche Anzahl an Geboten erhalten können.
Also findet der Algorithmus eine Mehrfachzuordnung und endet.

3.5.3 Beispiele

Der Algorithmus wurde in Matlab implementiert.
Das Profit-Preis-Paar wurde wie bei der kombinierten Auktion initialisiert.

Beispiel 1

Matrix der Nutzenwerte:

$$\begin{pmatrix} 1 & 3 & 2 & 3 & 4 & 2 & 4 \\ 2 & 2 & 1 & 4 & 3 & 1 & 5 \\ 3 & 2 & 4 & 3 & 3 & 4 & 2 \\ 4 & 1 & 3 & 2 & 5 & 2 & 3 \end{pmatrix}$$

Optimale Zuordnung:

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

Optimaler Preisvektor:

$$\begin{pmatrix} 1.1250 & 0.1250 & 1.1250 & 1.1250 & 2.1250 & 1.1250 & 2.1250 \end{pmatrix}$$

Optimaler Profitvektor:

$$\begin{pmatrix} 2.8750 \\ 2.8750 \\ 2.8750 \\ 2.8750 \end{pmatrix}$$

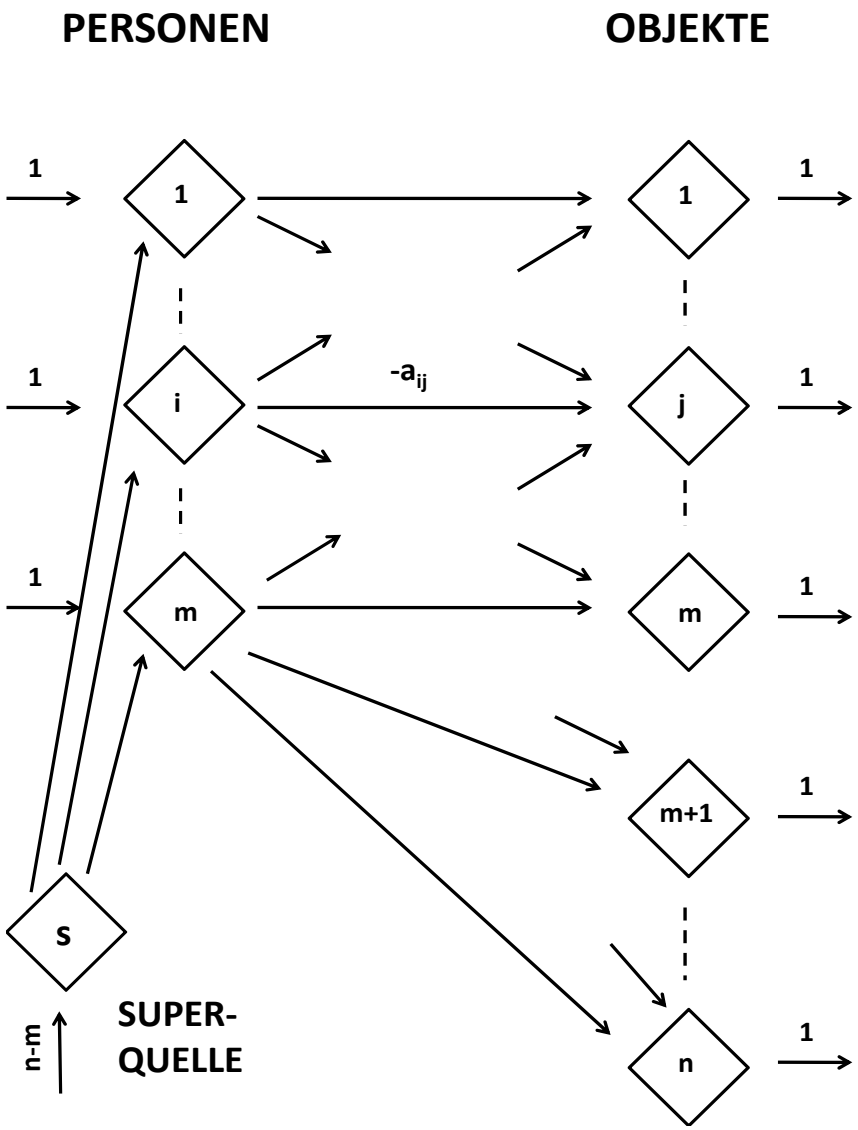


Abbildung 3.5: Durch das Einführen einer Superquelle s und künstlicher Kanten (s, i) mit Kosten gleich Null und einem zulässigem Flussbereich von $[0, \infty)$, wird das Mehrfachzuordnungsproblem in ein minimales Kostenflussproblem transformiert.

Kapitel 4

Zusammenfassung

In dieser Diplomarbeit wird das Zuordnungsproblem betrachtet. Bei diesem Problem werden Personen und Objekte einander zugeordnet und Preise bzw. Profite ermittelt. Dimitri P. Bertsekas hat einen sehr nützlichen Algorithmus konstruiert, der diese Art von Problemen löst.
Der einfachste Spezialfall ist das symmetrische Zuordnungsproblem, wobei es gleich viele Objekte wie Personen gibt, die einander zugeordnet werden sollen. In einer naiven Weise wird ein Algorithmus hergeleitet, der einen schweren Fehler beinhaltet. Für eine große Gruppe von Beispielen erzeugt dieser Algorithmus eine Endlosschleife. Daher wird die sogenannte ϵ-komplementäre Schlupfbedingung eingeführt. Mit dieser zusätzlichen Bedingung wird ein Algorithmus kreiert, der jegliche zulässige symmetrische Zuordnungsprobleme löst.
Durch Betrachten des Problems aus einer anderen Perspektive wird ein zweiter Algorithmus hergeleitet, der die Zuordnung und den Profit jeder Person bestimmt. Dieser Algorithmus wird als Rückwärtsauktion bezeichnet, da er das Problem umgekehrt löst.
Nachdem beide Algorithmen dasselbe Problem lösen, werden diese beiden in einem dritten Schritt kombiniert und dadurch ein neuer Algorithmus gebildet, der später auf eine größere Bandbreite an Problemen adaptiert werden kann. Durch die Anwendung von Graphentheorie und Dualität können asymmetrische Probleme derart transformiert werden, sodaß es möglich ist den Auktionsalgorithmus auf diese Probleme anzupassen. Allerdings müssen neue komplementäre Schlupfbedingungen eingeführt werden, um dies zu ermögli-

chen. Auf diese Art und Weise können auch asymmetrische und Mehrfachzuordnungsprobleme mit Hilfe eines angepassten Auktionsalgorithmus gelöst werden.

Anhang A

Grundlagen der Graphentheorie

In 'Linear Network Optimization, Algorithms and Codes' [5] und 'Lineare Optimierung und Netzwerkoptimierung' [13] findet man einige Grundlagen der Graphentheorie.
Ein Graph kann als $G = (N, A)$ dargestellt werden, wobei $N = \{n_1, ..., n_p\}$ eine Menge von Knoten und $A = \{a_1, ..., a_q\}$ eine Menge von Kanten darstellt. Jede Kante ist durch zwei Endpunkte definiert: $a = (n_i, n_j)$ bzw. $a = (i, j)$ (siehe Abbildung (A.1), S. 64).

Gibt man jeder Kante eine Richtung, so erhält man einen gerichteten Graphen. Nun hat jede Kante einen Ausgangspunkt $t(a)$ und einen Endpunkt $h(a)$. Daher ist es wichtig, dass die Kante (i, j) ein geordnetes Paar ist, da sich (i, j) von (j, i) unterscheidet.

Wenn man die Knoten eines Graphen in eine Menge T aller Startpunkte und eine Menge H aller Endpunkte teilen kann, spricht man von einem *bipartiten Graphen*. Ein Beispiel dafür ist das *Zuordnungsproblem* (siehe Abbildung (A.1), S. 64).

Zwei spezielle Formen von gerichteten Graphen sind *Wege* und *Kreise*.
Ein *Weg* W ist eine Folge von Knoten $(n_1, ..., n_p)$ und eine zugehörige Folge von Kanten $(a_1, ..., a_{p-1})$ mit $p \geq 2$. Hier kann a_i entweder (n_i, n_{i+1}) (Vorwärtskante) oder (n_{i+1}, n_i) (Rückwärtskante) sein. Besteht der gesamte Weg nur aus Vorwärts- bzw. Rückwärtskanten, so nennt man dies einen Vorwärts- bzw. Rückwärtsweg. Andernfalls kann man den Weg in W^+ und W^- teilen.

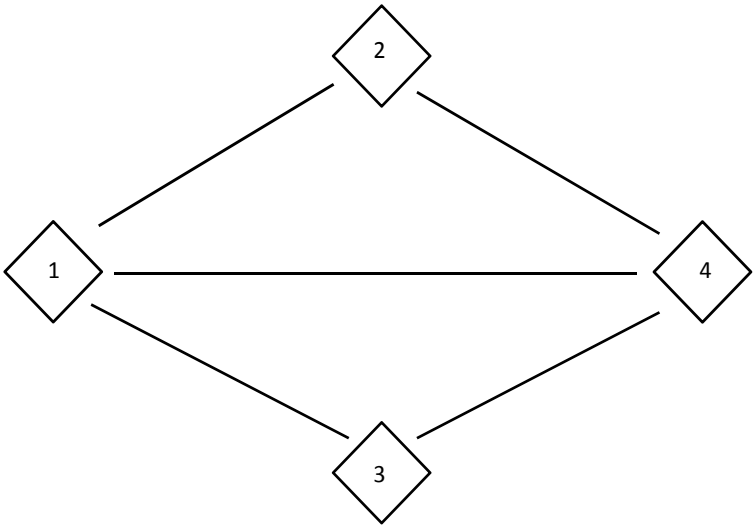


Abbildung A.1: Beispiel eines Graphen

Sind Anfangs- und Endknoten identisch ($n_1 = n_k$), spricht man von einem *Kreis* (siehe Abbildung (A.3), S. 66). Man spricht von einem azyklischen Graphen, wenn er keinen Kreis enthält. Ein Graph heißt zusammenhängend, wenn für jedes Knotenpaar (n_i, n_j) ein Weg existiert, der die beiden Knoten verbindet. Ein *Baum* ist ein zusammenhängender Graph, der keinen Kreis enthält. Ein *Untergraph* von $G = (N, A)$ ist ein Graph $G_1 = (N_1, A_1)$ mit $N_1 \subset N$ und $A_1 \subset A$. Ein *spannender Baum* B von G ist ein spannender Untergraph von G mit V' = V (siehe Abbildung (A.4), S. 67).

Angenommen jeder Kante (i, j) ist ein *Fluss* x_{ij} zugewiesen, so nennt man x einen *Flussvektor*. Dies ist ein Vektor, bei dem jeder Kante eine Zahl zugeordnet ist, die den Fluss der Kante darstellt. Im *Divergenzvektor* y sind die Werte des gesamten Flusses (ausgehender Fluss minus eingehendem Fluss)

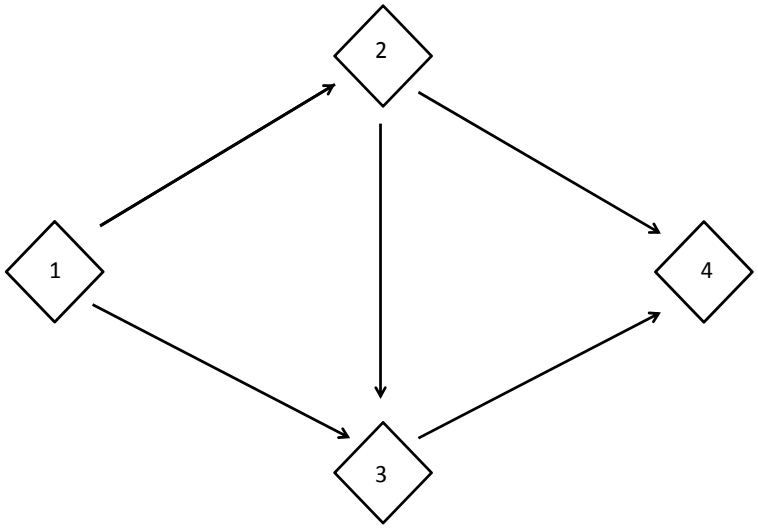


Abbildung A.2: Beispiel eines gerichteten Graphen

gespeichert:

$$y_i = \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji},\ \forall i \in N.$$

Beim *Zuordnungsproblem* ist der Fluss $x_{ij} = 1$ für alle Personen i, die einem Objekt j zugeordnet sind, andernfalls wird $x_{ij} = 0$ gesetzt. Nachdem jede Person nur einem Objekt zugeordnet werden kann, ist die Divergenz jedes Personenknoten $y_i = 1 - 0 = 1$ und die Divergenz jedes Objektknoten $y_j = 0 - 1 = -1$.
Ein Knoten mit $y_i > 0$ wird *Vorratsknoten* genannt. Knoten mit $y_i < 0$ bezeichnet man als *Bedarfsknoten*. Knoten mit $y_i = 0$ sind sogenannte *Durchflussknoten*. Ein Graph mit $y_i = 0$ für alle $i \in N$ heißt *Zirkulation*. Addiert man y_i über alle $i \in N$, ergibt dies $\sum_{i\in N} y_i = 0$ (siehe Abbildung (A.5), S. 68).

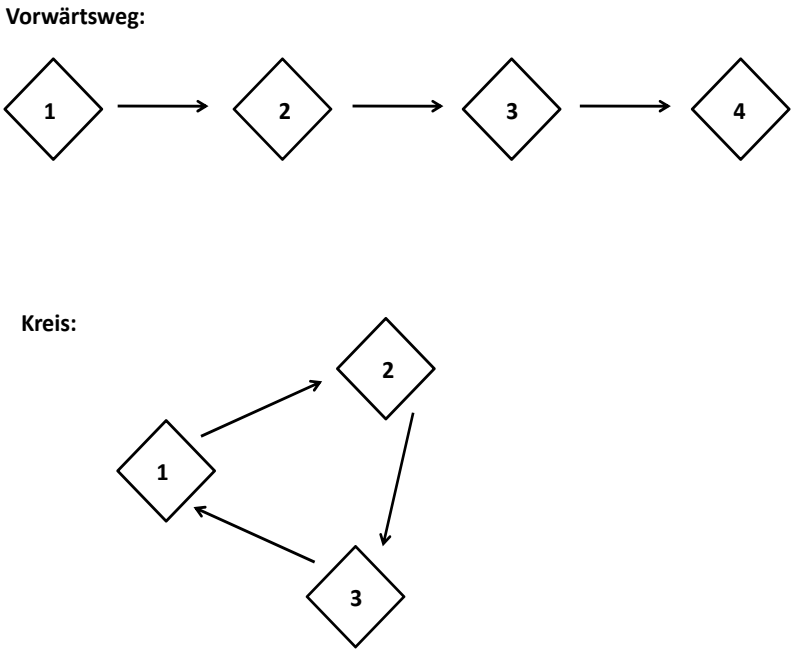


Abbildung A.3: Wege und Kreise

Ein negativer Fluss deutet an, dass der Fluss gegen die von der Kante angegebene Richtung geht. Im *Zuordnungsproblem* ist entweder $x_{ij} = 1$ oder $x_{ji} = -1$, wenn die Person i dem Objekt j zugeordnet ist. Das heißt, dass jeder negative Fluss durch Änderung der Richtung zu einem positiven Fluss transformiert werden kann. Also kann ohne Beschränkung der Allgemeinheit angenommen werden, dass alle Flüsse nichtnegativ sind.

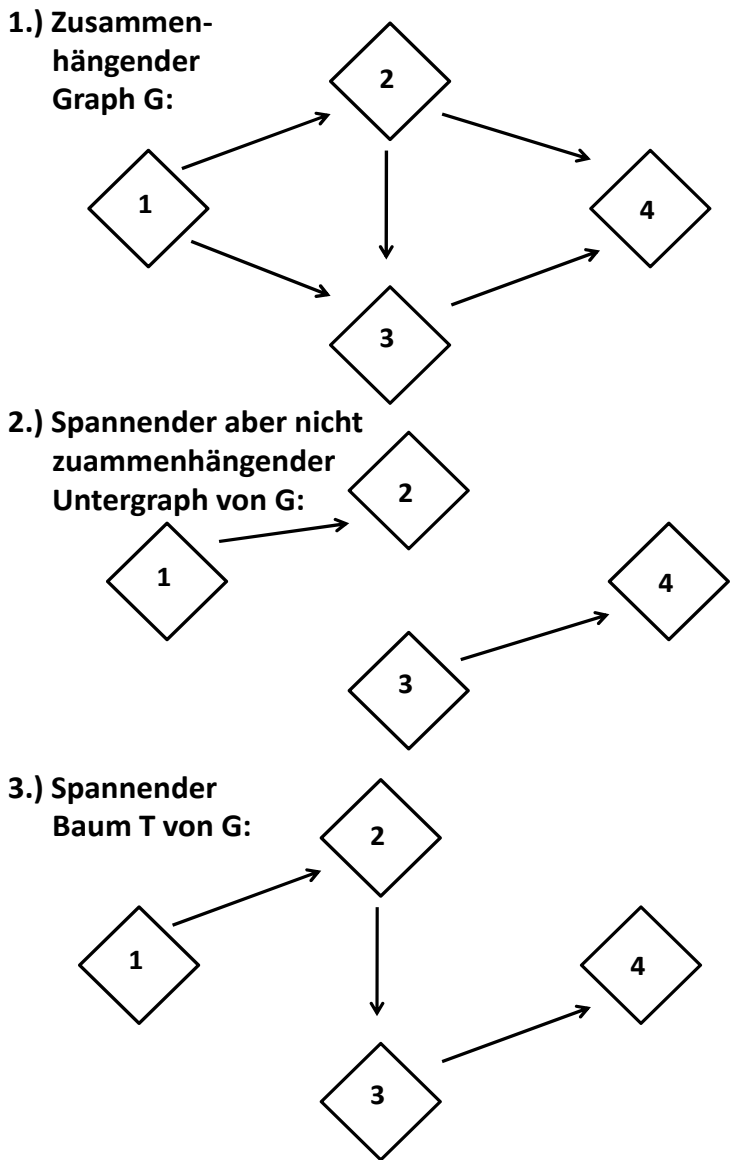


Abbildung A.4: 1.)Zusammenhängender aber nicht azyklischer Graph G 2.) Azyklischer, spannender aber nicht zusammenhängender Untergraph von G 3.) Spannender Baum T von G: Spannender Untergraph, der azyklisch und zusammenhängend ist

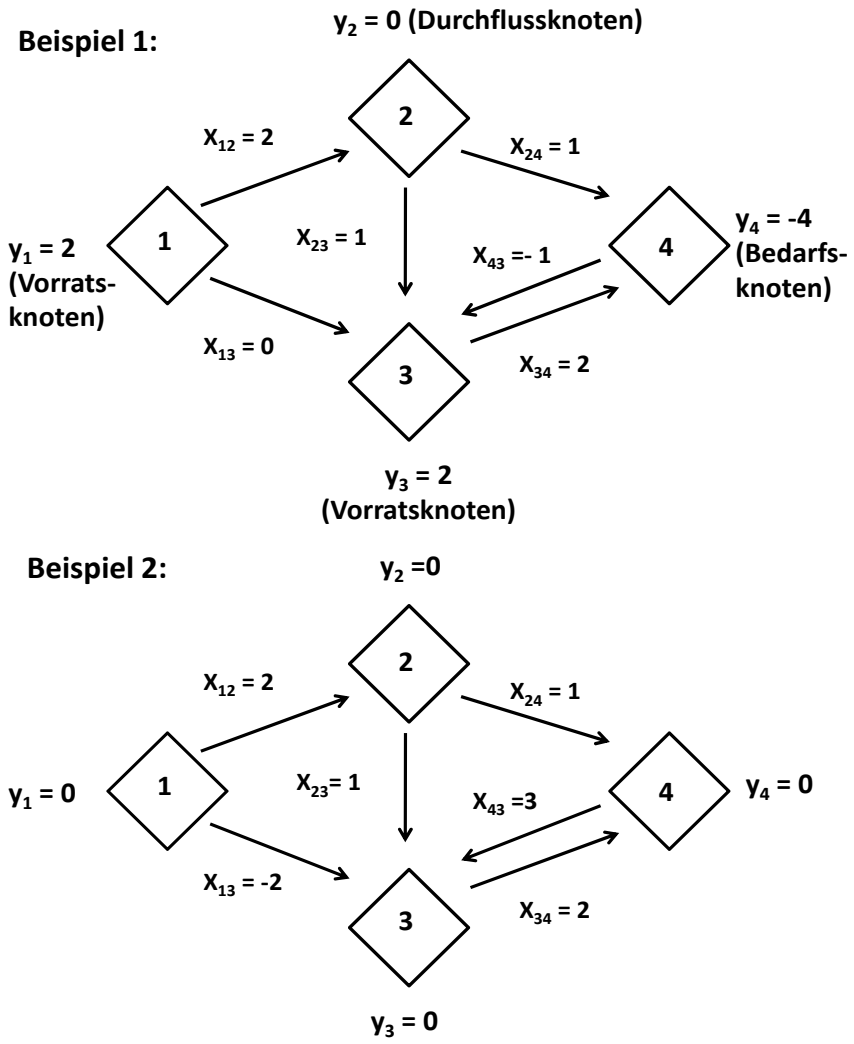


Abbildung A.5: *Beispiel 1*: Graph mit unterschiedlichen Flüssen *Beispiel 2*: Zirkulation

A.1 Das Maximale Flussproblem

In diesem speziellen Problem ist es das Ziel den maximalen Fluss von einer *Quelle* s zu einer *Senke* t zu erreichen. Während man versucht so viel Fluss wie möglich von der Quelle zur Senke zu verschieben, müssen die Kapazitätsbedingungen erfüllt sein.
Das *Maximale Flussproblem*, welches unter anderem in 'Linear Network Optimization, Algorithms and Codes' [5], 'Lineare Optimierung und Netzwerkoptimierung' [13] und 'Optimierung, Operations Research, Spieltheorie' [14] definiert ist, kann als lineares Optimierungsproblem dargestellt werden:

$$\max x_{ts}$$

unter den Nebenbedingungen

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = 0, \tag{A.1}$$

$$\forall i \in N \text{ mit } i \neq s \text{ und } i \neq t,$$

$$\sum_{\{j|(s,j)\in A\}} x_{sj} = \sum_{\{i|(i,t)\in A\}} x_{it} = x_{ts}, \tag{A.2}$$

$$b_{ij} \leq x_{ij} \leq c_{ij},\ \forall (i,j) \in A \text{ mit } (i,j) \neq (t,s), \tag{A.3}$$

$$\sum_{\{i|(i,t)\in A\}} b_{it} \leq x_{ts} \leq \sum_{\{i|(i,t)\in A\}} c_{it}. \tag{A.4}$$

Um das *maximale Flussproblem* mit dem *minimalen Kostenflussproblem* in Verbindung zu stellen, werden zuerst allen Kanten Kosten Null zugeordnet. Nun werden künstliche Kanten (t,s) mit Kosten -1 eingeführt, wobei die obere Flussgrenze entsprechend hoch und die untere Flussgrenze entsprechend tief gesetzt werden muss, sodass diese Grenzen während des Prozesses nie verletzt werden.
Nachdem die Kosten aller Kanten mit Ausnahme von (t,s) Null sind, wird nun $-x_{ts}$ minimiert. Durch einen Vorzeichenwechsel kann von der Minimierung zur Maximierung gewechselt werden. Daher kann x_{ts} maximiert werden, was die Divergenz von s darstellt. In dem Paper 'An Auction Algorithm for the Max-Flow Problem' [2], wird ein Algorithmus zum Lösen dieses speziellen Problems eingeführt. Da dies aber nicht in direktem Zusammenhang mit

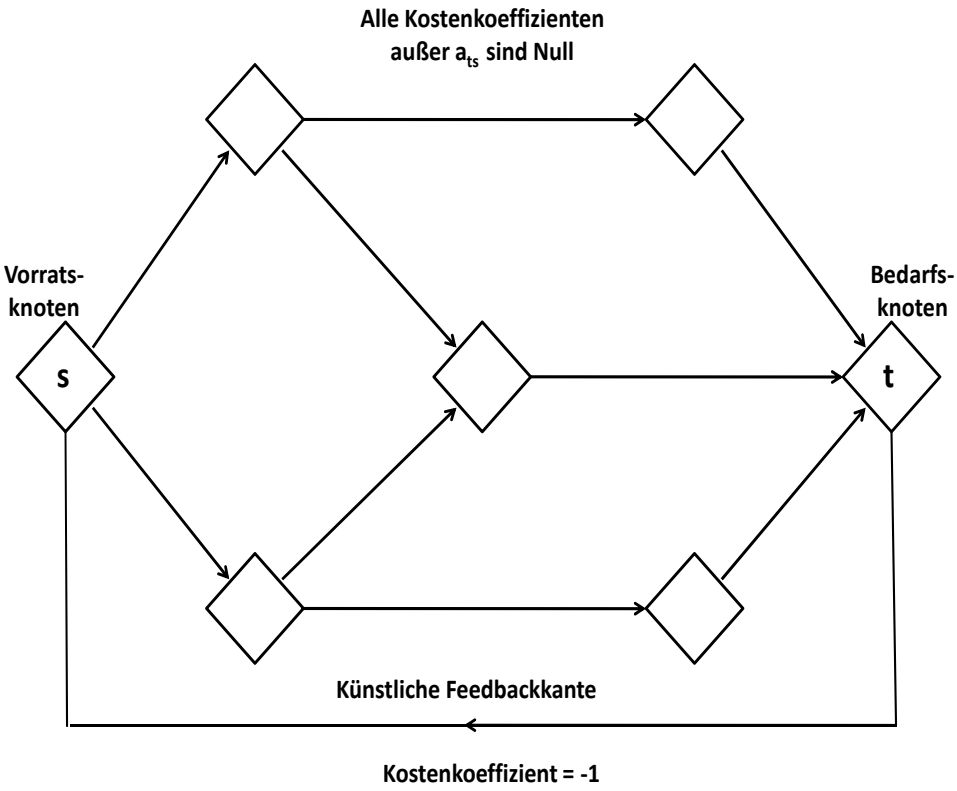


Abbildung A.6: *Minimale Kostenflussdarstellung* eines *maximalen Flussproblems*. Im Optimum ist der Fluss x_{ts} gleich dem maximalen Fluss von s nach t des Subgraphen ohne künstlicher Kante (t, s).

dem *Zuordnungsproblem* steht, wird dieser Algorithmus in dieser Diplomarbeit nicht weiter erörtert.

A.2 Das Transportproblem

Das *Transportproblem* ist dem *Zuordnungsproblem* ähnlich. Es kann ebenfalls durch einen bipartiten Graphen dargestellt werden.
Ein einfaches Beispiel (siehe Abbildung (A.7)): Eine Firma produziert ihre Produkte in drei unterschiedlichen Fabriken f_1, f_2 und f_3. Die Produkte sollen zu den Geschäften g_1, g_2 und g_3 transportiert werden. Dabei sollen die Transportkosten minimiert werden. Die Kante von Firma f_i zu Geschäft g_j wird als x_{ij} bezeichnet. Jede Fabrik kann eine gewisse Menge an Produkten herstellen ($= \alpha_i$) und jedes Geschäft kann eine gewisse Menge lagern ($= \beta_j$). Die Kosten des Transportes von Fabrik f_i zu Geschäft g_j werden als a_{ij} bezeichnet.
Das Transportproblem, welches Bertsekas in 'Linear Network Optimization, Algorithms and CodesLinear Network Optimization, Algorithms and Codes' [5] erklärt, kann verallgemeinert als lineares Optimierungsproblem betrachtet werden:

$$\min \sum_{(i,j)\in A} a_{ij}x_{ij}$$

unter den Nebenbedingungen

$$\sum_{\{j|(i,j)\in A\}} x_{ij} = \alpha_i,\ \forall i = 1, ..., m, \tag{A.5}$$

$$\sum_{\{i|(i,j)\in A\}} x_{ij} = \beta_j,\ \forall j = 1, ..., n, \tag{A.6}$$

$$0 \leq x_{ij} \leq \min\{\alpha_i\beta_j\},\ \forall (i,j) \in A. \tag{A.7}$$

Für Zulässigkeit muss

$$\sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j$$

erfüllt sein.
In dem Paper 'The Auction Algorithm for the Transportation Problem' [11], wird ein Algorithmus zum Lösen dieses speziellen Problems eingeführt. Da dies aber nicht in direktem Zusammenhang mit dem *Zuordnungsproblem* steht, wird dieser Algorithmus in dieser Diplomarbeit nicht weiter erörtert.

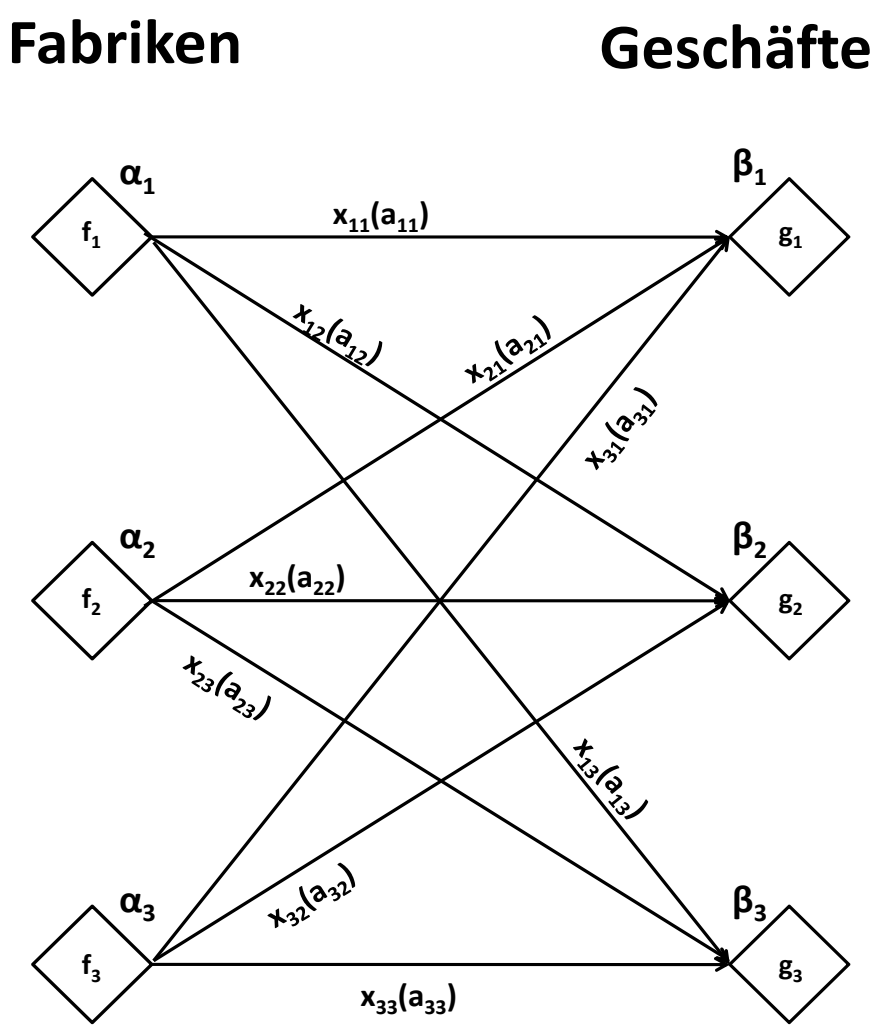


Abbildung A.7: Transportproblem

Anhang B

Implementierung des Algorithmus

Dies ist der Versuch die *Gauss-Seidel*-Version von Bertsekas' Auktionsalgorithmus in Matlab zu programmieren.

B.1 Vorwärtsauktion

Dieser Algorithmus löst das Zuordnungsproblem für symmetrische Matrizen der Nutzenwerte. Das Programm *main_ auction.m* fordert den Benutzer auf eine quadratische, positive Matrix reeller Nutzenwerte einzugeben. Dieses Hauptprogramm ruft die Funktion *auction(a)* auf, die den tatsächlichen Algorithmus beinhaltet.

main_ auction.m

```
% Auktionsalgorithmus für Symmetrische Zuordnungsprobleme

clear;
%Kurze Beschreibung des Programms:
disp('Dieses Programm löst ein symmetrisches Zuordnungsproblem
mittels Auktionsalgorithmus');
%Frage nach spezifischer Eingabe:
a = input('Geben Sie eine quadratische, positive Matrix
```

```
reeller Nutzenwerte ein, die vollen Rang hat:');
%Wiederholte Frage nach spezifischer Eingabe,
%falls Benutzer falsches Format eingibt:
while (isempty(a) | rank(a) ~= length(a(1,:)) | length(a(:,1))
~= length(a(1,:)) | ~ isnumeric(a) | imag(a) ~= 0)
a = input('Diese Eingabe erfüllt nicht die gefragten Bedingungen.
Bitte geben Sie eine quadratische, positive Matrix reeller
Nutzenwerte ein, die vollen Rang hat:')
end
%Das Ergebnis wird dargestellt:
disp('Die optimale Zuordnung ist durch b und der optimale
Preis durch p gegeben:')
[b,p]=auction(a)
```

auction.m

```
function[b,p]=auction(a)
%In dieser Funktion ist der tatsächliche Auktionsalgorithmus
%implementiert.
%n gibt die Anzahl der Personen an:
n = length(a(:,1));
%Die erste Zuordnung b und der Preisvektor p werden auf
%Null gesetzt, um die e-KS-Bedingung zu erfüllen.
b = zeros(n);
p = zeros(1,n);
%Der Parameter e muss kleiner als 1/n gewählt werden, damit
%die Lösung des Algorithmus optimal ist (angenommen die
%Nutzenwerte sind ganzzahlig, andernfalls ist die Lösung
%in einer gewissen Umgebung der optimalen Lösung).
e = 1/(n+1);
%Die Auktion läuft solange die Matrix b nicht vollen Rang
%hat. Sobald die Matrix vollen Rang erreicht, endet die
%Auktion.
while rank(b) < n
for i = 1:n
%Überprüfung, ob Person bereits zugeordnet:
```

```
if b(i,:)==0
%Suche das beste Objekt j:
%v ... zugehöriger Wert
[v,j]=max(a(i,:) - p);
%Suche das zweitbeste Objekt:
%kreiere dafür eine Matrix a1, wobei das beste Objekt gelöscht
%wird.
a1 = zeros(n-1);
for k = 1:(j-1)
a1(i,k)=a(i,k);
end
for k = (j+1):n
a1(i,k-1)=a(i,k);
end
%Kreiere einen Preisvektor p1, wobei der Preis des besten
%Objekts gelöscht wird:
p1 = zeros(1,(n-1));
for k = 1:(j-1)
p1(k)=p(k);
end
for k = (j+1):n
p1(k-1)=p(k);
end
%Der Wert, den das zweitbeste Objekt liefert, ist in w
% gespeichert:
[w,s]= max(a1(i,:)-p1);
%Berechne das Gebot:
d=p(j)+v-w+e;
%Speichere die neue Zuordnung in b:
b(:,j) = 0;
b(i,j) = 1;
%Aktualisiere den Preis des zugeordneten Objektes:
p(j) = d;
end end end
```

B.2 Rückwärtsauktion

Dieser Algorithmus löst das Zuordnungsproblem für symmetrische Matrizen der Nutzenwerte mittels Rückwärtsauktion. Das Programm *r_main_auction.m* fordert den Benutzer auf eine quadratische, positive Matrix reeller Nutzenwerte einzugeben. Dieses Hauptprogramm ruft die Funktion *r_auction(a)* auf, die den tatsächlichen Algorithmus beinhaltet.

r_main_auction.m

```
%Rückwärtsauktion für Symmetrische Zuordnungsprobleme
clear;
%Kurze Beschreibung des Programms:
disp('Dieses Programm löst ein symmetrisches Zuordnungsproblem
mittels Rückwärtsauktion');
%Frage nach spezifischer Eingabe:
a = input('Geben Sie eine quadratische, positive Matrix
reeller Nutzenwerte ein, die vollen Rang hat:');
%Wiederholte Frage nach spezifischer Eingabe,
%falls Benutzer falsches Format eingibt:
while (isempty(a) | rank(a) ~= length(a(1,:)) | length(a(:,1))
~= length(a(1,:)) | ~ isnumeric(a) | imag(a) ~= 0)
a = input('Diese Eingabe erfüllt nicht die gefragten Bedingungen.
Bitte geben Sie eine quadratische, positive Matrix reeller
Nutzenwerte ein, die vollen Rang hat:')
end
%Das Ergebnis wird dargestellt:
disp('Die optimale Zuordnung ist durch b und der optimale
Profit durch r gegeben:')
[b,r]=r_auction(a)
```

r_auction

```
function [b,r]=r_auction(a)
%In dieser Funktion ist der tatsächliche Auktionsalgorithmus
```

```
%implementiert.
%n gibt die Anzahl der Personen an:
n = length(a(:,1));
%Die erste Zuordnung b und der Profitvektor r werden auf
%Null gesetzt, um die e-KS-Bedingung zu erfüllen.
b = zeros(n);
r = zeros(n,1);
e = 1/(n+1);
%Die Auktion läuft solange die Matrix b nicht vollen Rang
%hat.
while rank(b) < n
for j = 1:n
%Überprüfung, ob Person bereits zugeordnet:
if b(:,j)==0
%Suche die beste Person: [v,i]=max(a(:,j) - r);
%Suche die zweitbeste Person
%kreiere dafür eine Matrix a1, wobei die beste Person
%gelöscht wird.
a1 = zeros(n-1);
for k = 1:(i-1)
a1(k,j)=a(k,j);
end
for k = (i+1):n
a1(k-1,j)=a(k,j);
end
%Kreiere einen Profitvektor r1, wobei der Profit der besten
%Person gelöscht wird:
r1 = zeros((n-1),1);
for k = 1:(i-1)
r1(k)=r(k);
end
for k = (i+1):n
r1(k-1)=r(k);
end
%Der Wert der zweitbesten Person ist w:
[w,s]= max(a1(:,j)-r1);
```

```
%Berechne das Gebot:
d=r(i)+v-w+e;
%Speichere die neue Zuordnung in b:
b(i,:)=0;
b(i,j)=1;
%Aktualisiere den Profit der zugeordneten Person:
r(i)=d;
end end end
```

B.3 Kombinierte Auktion

Dieser Algorithmus löst das Zuordnungsproblem für symmetrische Matrizen der Nutzenwerte mittels kombinierter Auktion. Das Programm *c_main_auction.m* fordert den Benutzer auf eine quadratische, positive Matrix reeller Nutzenwerte einzugeben. Dieses Hauptprogramm ruft die Funktion *c_auction(a)* auf, die den tatsächlichen Algorithmus beinhaltet.
Um zwischen Vorwärts- und Rückwärtsauktion zu wechseln, wird angenommen, dass die Vorwärtsauktion nur bei geradzahligem Rang der Nutzenwertmatrix und die Rückwärtsauktion nur bei ungeradzahligem Rang der Nutzenwertmatrix durchgeführt wird.

c_main_auction.m

```
clear;
%Kurze Beschreibung des Programms:
disp('Dieses Programm löst ein symmetrisches Zuordnungsproblem
mittels kombinierter Auktion');
%Frage nach spezifischer Eingabe:
a = input('Geben Sie eine quadratische, positive Matrix
reeller Nutzenwerte ein, die vollen Rang hat:');
%Wiederholte Frage nach spezifischer Eingabe,
%falls Benutzer falsches Format eingibt:
while (isempty(a) | rank(a) ~= length(a(1,:)) | length(a(:,1))
~= length(a(1,:)) | ~ isnumeric(a) | imag(a) ~= 0)
a = input('Diese Eingabe erfüllt nicht die gefragten Bedingungen.
Bitte geben Sie eine quadratische, positive Matrix reeller
Nutzenwerte ein, die vollen Rang hat:')
end
%Das Ergebnis wird dargestellt:
disp('Die optimale Zuordnung ist durch b, der optimale
Preis durch p und der optimale Profit durch r gegeben:')
[b,p,r]=c_auction(a)
```

c_auction.m

```
function [b,p,r]=c_auction(a)
%n gibt die Anzahl der Personen an:
n = length(a(:,1));
%Definiere Matrizen/Vektoren, die im Laufe des Algorithmus
%benötigt werden:
b = zeros(n);
p = zeros(1,n);
f1 = zeros(1,n);
r = zeros(n,1);
r1=zeros(n,1);
e = 1/(n+1);
%Initialisierung von Preis und Profit, damit die
%e-KS-Bedingungen erfüllt sind:
for i = 1:n
p(i) = min(a(:,i));
end
for j = 1:n
for i=1:n
if r(j)+p(i)<a(i,j)
r(j) = a(i,j)-p(i);
end end end
%Die Auktion läuft solange die Matrix b nicht vollen Rang
%hat.
while rank(b) < n
for i = 1:n
for j = 1:n
%Für geradzahligen Rang wende Vorwärtsauktion an:
if mod(rank(b),2)==0
if b(i,:)==0
[v,j]=max(a(i,:) - p);
a1 = zeros(n-1);
for k = 1:(j-1)
a1(i,k)=a(i,k);
end
for k = (j+1):n
a1(i,k-1)=a(i,k);
```

```
end
p1 = zeros(1,(n-1));
for k = 1:(j-1)
p1(k)=p(k);
end
for k = (j+1):n
p1(k-1)=p(k);
end
[w,s]= max(a1(i,:)-p1);
d=p(j)+v-w+e;
b(:,j)=0;
b(i,j)=1;
p(j)=d;
%Aktualisierung des Profits, damit e-KS nicht verletzt
%wird:
r(j)=a(i,j)-p(j);
end
%Für ungeradzahligen Rang wende Vorwärtsauktion an:
else
if b(:,j)==0
[v,i]=max(a(:,j) - r);
a1 = zeros(n-1);
for k = 1:(i-1)
a1(k,j)=a(k,j);
end
for k = (i+1):n
a1(k-1,j)=a(k,j);
end
r1 = zeros((n-1),1);
for k = 1:(i-1)
r1(k)=r(k);
end
for k = (i+1):n
r1(k-1)=r(k);
end
[w,s]= max(a1(:,j)-r1);
```

```
if s > (i-1)
s=s+1;
end
if a(s,j)== 0
w = v - e;
end
d=r(i)+v-w+e;
b(i,:)=0;
b(i,j)=1;
r(i)=d;
%Aktualisierung des Preises, damit e-KS nicht verletzt
%wird
p(i)=a(i,j)-r(i);
end end end end end
```

B.4 Auktionsalgorithmus für asymmetrische Zuordnungsprobleme

Dieser Algorithmus löst Zuordnungsprobleme mit asymmetrischen Matrizen der Nutzenwerte durch Verwendung einer angepassten Rückwärtsauktion. Das Programm *a_main_auction.m* fordert den Benutzer auf eine asymmetrische (mehr Zeilen als Spalten), positive Matrix reeller Nutzenwerte einzugeben. Dieses Hauptprogramm ruft die Funktion *a_auction(a)* auf, die den tatsächlichen Algorithmus beinhaltet.

a_main_auction.m

```
clear;
%Kurze Beschreibung des Programms:
disp('Dieses Programm löst ein asymmetrisches Zuordnungsproblem
mittels Auktionsalgorithmus');
%Frage nach spezifischer Eingabe:
a = input('Geben Sie eine asymmetrische (mehr Zeilen als
Spalten), positive Matrix reeller Nutzenwerte ein, die
vollen Rang hat');
%Wiederholte Frage nach spezifischer Eingabe,
%falls Benutzer falsches Format eingibt:
while (isempty(a) | rank(a) ~= length(a(:,1)) | length(a(:,1))
> length(a(1,:)) | ~isnumeric(a) | imag(a) ~= 0)
a = input('Diese Eingabe erfüllt nicht die gefragten Bedingungen.
Bitte geben Sie eine asymmetrische (mehr Zeilen als Spalten),
positive Matrix reeller Nutzenwerte ein, die vollen Rang
hat:')
end
%Das Ergebnis wird dargestellt:
disp('Die optimale Zuordnung ist durch b, der optimale
Preis durch p und der optimale Profit durch r gegeben:')
[b,p,r]=a_auction(a)
```

a_auction.m

```
function[b,p,r]=a_auction(a)
%m gibt die Anzahl der Objekte an:
m = length(a(1,:));
%n gibt die Anzahl der Personen an:
n = length(a(:,1));
%Definiere Matrizen/Vektoren, die im Laufe des Algorithmus
%benötigt werden:
b = zeros(n,m);
p = zeros(1,m);
r = zeros(n,1);
e = 1/(m+1);
%Initialisierung von Preis und Profit, damit die
%e-KS-Bedingungen erfüllt sind:
for i = 1:n
p(i) = min(a(:,i));
end
for j = 1:n
for i=1:n
if r(j)+p(i)<a(i,j)
r(j) = a(i,j)-p(i);
end end end

%%%%%%%%%Vorwärtsauktion%%%%%%%%%

%Vorwärtsauktion wird benutzt, um eine initiale Zuordnung
%zu finden. Die Auktion läuft solange die Zuordnungsmatrix
%b nicht vollen Rang hat.
while rank(b) < n
for i = 1:n
if b(i,:)==0
[v,j]=max(a(i,:) - p);
a1 = zeros(n-1,m-1);
for k = 1:(j-1)
a1(i,k)=a(i,k);
end
```

```
for k = (j+1):m
a1(i,k-1)=a(i,k);
end
p1 = zeros(1,(m-1));
for k = 1:(j-1)
p1(k)=p(k);
end
for k = (j+1):m
p1(k-1)=p(k);
end
[w,s]= max(a1(i,:)-p1);
d=p(j)+v-w+e;
b(:,j) = 0;
b(i,j) = 1;
p(j) = d;
%Aktualisierung des Profits, damit e-KS nicht verletzt
%wird:
r(i) = a(i,j) - p(j);
end end end

%%%%%%%%%%Angepasste Rückwärtsauktion%%%%%%%%%%

%Die angepasste Rückwärtsauktion liefert die optimale Zuordnung:
%Definiere:
%pi = Preise der Objekte, die in der initialen Zuordnung
%zugeordnet sind
%pc = Preise aller nicht zugeordneten Objekte
%l = Minimum von pi
for j = 1:m
if b(:,j) == 0
pi(j) = 0;
pc(j) = p(j);
else
pi(j) = p(j);
pc(j) = 0;
end end
```

```
l = min(pi);
while max(pc) > l
for j = 1:m
if b(:,j) == 0
if p(j) > l
[v,i]=max(a(:,j) - r);
a1 = zeros(n-1);
for k = 1:(i-1)
a1(k,j)=a(k,j);
end
for k = (i+1):n
a1(k-1,j)=a(k,j);
end
r1 = zeros((n-1),1);
for k = 1:(i-1)
r1(k)=r(k);
end
for k = (i+1):n
r1(k-1)=r(k);
end
[w,s]= max(a1(:,j)-r1);
%Ist der Wert der besten Person um nur höchstens e größer
% als l, so wird der Preis p(j) auf Null gesetzt und ein
%neuer Iterationsschritt gestartet.
if l >= v - e
p(j) = l;
else
d = min(v-l,v-w+e);
%Aktualisiere Preis und Profit:
p(j) = v-d;
r(i) = r(i)+d;
%Das Objekt, das bisher der Person i zugeordnet war,
%wird aus der Zuordnung entfernt und das Objekt j
%wird hinzugefügt.
b(i,:) = 0;
b(i,j) = 1;
```

```
end
%Aktualisiere die Preise der nicht zugeordneten Objekte:
for j = 1:m
if b(:,j) == 0
pc(j) = p(j);
else
pc(j) = 0;
end end end end end end
```

B.5 Auktionsalgorithmus für Mehrfachzuordnungsprobleme

Dieser Algorithmus löst Mehrfachzuordnungsprobleme durch Verwendung einer angepassten Rückwärtsauktion. Das Programm *m_main_auction.m* fordert den Benutzer auf eine asymmetrische (mehr Zeilen als Spalten), positive Matrix reeller Nutzenwerte einzugeben. Dieses Hauptprogramm ruft die Funktion *m_auction(a)* auf, die den tatsächlichen Algorithmus beinhaltet.

m_main_auction.m

```
clear;
%Kurze Beschreibung des Programms:
disp('Dieses Programm löst ein Mehrfachzuordnungsproblem
mittels Auktionsalgorithmus:');
%Frage nach spezifischer Eingabe:
a = input('Geben Sie eine asymmetrische (mehr Zeilen als
Spalten), positive Matrix reeller Nutzenwerte ein, die
vollen Rang hat');
%Wiederholte Frage nach spezifischer Eingabe,
%falls Benutzer falsches Format eingibt:
while (isempty(a) | rank(a) ~= length(a(:,1)) | length(a(:,1))
> length(a(1,:)) | ~isnumeric(a) | imag(a) ~= 0)
a = input('Diese Eingabe erfüllt nicht die gefragten Bedingungen.
Bitte geben Sie eine asymmetrische (mehr Zeilen als Spalten),
positive Matrix reeller Nutzenwerte ein, die vollen Rang
hat:')
end
%Das Ergebnis wird dargestellt:
disp('Die optimale Zuordnung ist durch b, der optimale
Preis durch p und der optimale Profit durch r gegeben:')
[b,p,r]=m_auction(a)
```

m_ auction.m

```
function[b,p,r]=m_auction(a)
%m gibt die Anzahl der Objekte an:
m = length(a(1,:));
%n gitb die Anzahl der Personen an:
n = length(a(:,1));
%Definiere Matrizen/Vektoren, die im Laufe des Algorithmus
%benötigt werden:
b = zeros(n,m);
p = zeros(1,m);
r = zeros(n,1);
e = 1/(m+1);
%Initialisierung von Preis und Profit, damit die
%e-KS-Bedingungen erfüllt sind:
for i = 1:n
p(i) = min(a(:,i));
end
for j = 1:n
for i=1:n
if r(j)+p(i)<a(i,j)
r(j) = a(i,j)-p(i);
end end end

%%%%%%%%%Vorwärtsauktion%%%%%%%%%

%Vorwärtsauktion wird benutzt, um eine initiale Zuordnung
%zu finden. Die Auktion läuft solange die Zuordnungsmatrix
%b nicht vollen Rang hat.
while rank(b) < n
for i = 1:n
if b(i,:)==0
[v,j]=max(a(i,:) - p);
a1 = zeros(n-1,m-1);
for k = 1:(j-1)
a1(i,k)=a(i,k);
```

```
end
for k = (j+1):m
a1(i,k-1)=a(i,k);
end
p1 = zeros(1,(m-1));
for k = 1:(j-1)
p1(k)=p(k);
end
for k = (j+1):m
p1(k-1)=p(k);
end
[w,s]= max(a1(i,:)-p1);
d=p(j)+v-w+e;
b(:,j) = 0;
b(i,j) = 1;
p(j) = d;
%Aktualisierung des Profits, damit e-KS nicht verletzt
%wird:
r(i) = a(i,j) - p(j);
end end end

%%%%%%%%%Angepasste Rückwärtsauktion%%%%%%%%%

%Die angepasste Rückwärtsauktion liefert die optimale Zuordnung:
%Definiere:
%ri = Profite der Personen, die in der initialen Zuordnung
%zugeordnet sind
%pc = Profite aller nicht zugeordneten Personen
%l = Maximum von ri
for i = 1:n
if b(i,:) == 0
ri(i) = 0;
rc(i) = r(i);
else
ri(i) = r(i);
rc(i) = 0;
```

```
end end
l = max(ri);
while sum(sum(b))< m
for j = 1:m
if b(:,j) == 0
[v,i]=max(a(:,j) - r);
a1 = zeros(n-1);
for k = 1:(i-1)
a1(k,j)=a(k,j);
end
for k = (i+1):n
a1(k-1,j)=a(k,j);
end
r1 = zeros((n-1),1);
for k = 1:(i-1)
r1(k)=r(k);
end
for k = (i+1):n
r1(k-1)=r(k);
end
[w,s]= max(a1(:,j)-r1);
d = min(l-r(i),v-w+e);
p(j) = v-d;
r(i) = r(i)+d;
%Wenn d>0 ist, wird das Objekt, das bisher Person
%i zugeordnet war, aus der Zuordnungsmatrix b
%gelöscht.
if d > 0
b(i,:) = 0;
end
b(i,j) = 1;
end end end end
```

Abbildungsverzeichnis

Literaturverzeichnis

[1] Dimitri P. Bertsekas: A Distributed Algorithm for the Assignment Problem, Lab. for Information and Decision Sciences Working Paper, M.I.T., March 1979.

[2] Dimitri P. Bertsekas: An Auction Algorithm for the Max-Flow Problem, Journal of Optimization Theory and Applications, Volume 87 , Issue 1, 69-101, Oktober 1995.

[3] Dimitri P. Bertsekas: Auction Algorithms for Network Flow Problems: A Tutorial Introduction, Computational Optimization and Applications, Volume 1, 7-66, 1992.

[4] Dimitri P. Bertsekas: Constrained Optimization and Lagrange Multiplier Methods, Athena Scientific, ISBN: 1-886529-04-3, 1982.

[5] Dimitri P. Bertsekas: Linear Network Optimization, Algorithms and Codes, The MIT Press, ISBN:0-262-02334-2, 1991.

[6] Dimitri P. Bertsekas: Network Optimization, Continuous and Discrete Models, Athena Scientific, ISBN: 1-886529-02-7, 1998.

[7] Dimitri P. Bertsekas und David A. Castanon: A Forward/Reverse Auction Algorithm for Asymmetric Assignment Problems, Computational Optimization and Applications, Volume 1, Number 3, 277-297, 1992.

[8] Dimitri P. Bertsekas und David A. Castanon: Parallel Primal Dual Methods for the Minimum Cost Flow Problem, Computational Optimization and Applications, Volume 2, 319-388, 1993.

[9] Dimitri P. Bertsekas und David A. Castanon: Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm, Alpha-

tech Report, Burlington, MA, Nov. 1989; also Parallel Computing, Vol. 17,707-732, 1991.

[10] Dimitri P. Bertsekas und David A. Castanon: The Auction Algorithm for the Minimum Cost Network Flow Problem, Laboratory for Information and Decision Systems Report LIDS-P-1925, M.I.T., Cambridge, MA, November 1989.

[11] Dimitri P. Bertsekas und David A. Castanon: The Auction Algorithm for the Transportation Problem, Ann. Op. Res. 20, 67-96, 1989.

[12] Dimitri P. Bertsekas, David A. Castanon und Haralampos Tsaknakis: Reverse Auction and the Solution of Inequality Constrained Assignment Problems, SIAM J. Optim. Volume 3, Issue 2, 268-297, Mai 1993.

[13] Horst W. Hamacher, Kathrin Klamroth: Lineare Optimierung und Netzwerkoptimierung, Friedr. Vieweg & Sohn Verlag, ISBN-10 3-8348-0185-2, 2006.

[14] Karl Heinz Borgwardt: Optimierung, Operations Research, Spieltheorie, Birkhäuser, ISBN-10 3-7643-6519-6, 2001.