# Mediane und Warehouse-Location-Probleme

Julian Boß

Seminar zu Methoden der Logistik
Fachhochschule Wedel
Sommersemester 2012
20.06.2012

# Inhaltsverzeichnis

**1 Einleitung** **3**

**2 Mediane** **3**
2.1 ungerichtete Graphen . . . . . 3
2.2 gerichtete Graphen . . . . . 4
2.3 p-Mediane . . . . . 5

**3 Warehouse-Location-Probleme** **6**
3.1 unkapazitierte, einstufige WLP . . . . . 6
3.2 kapazitierte, einstufige WLP . . . . . 7
3.3 kapazitierte, mehrstufige WLP . . . . . 8

**4 Heuristiken** **9**
4.1 ADD-Algorithmus . . . . . 9
4.1.1 unkapazitierte WLP . . . . . 10
4.1.2 kapazitierte WLP . . . . . 11
4.2 Verbesserungsverfahren . . . . . 12

**5 Exakte Verfahren** **12**
5.1 Branch & Bound-Verfahren . . . . . 12
5.2 Branch & Bound nach Erlenkotter . . . . . 13
5.2.1 Dual-Ascent-Methode . . . . . 14
5.2.2 Ganzzahlige Lösung . . . . . 16
5.2.3 Dual-Adjustment-Methode . . . . . 17
5.2.4 Verzweigung . . . . . 19

**6 Literaturverzeichnis** **20**

# 1 Einleitung

Bei diesem Dokument handelt es sich um eine schriftliche Ausarbeitung zum Thema Mediane und Warehouse-Location-Probleme, basierend auf dem gleichnamigen Seminarvortrag vom 07.06.2012. Zielsetzung dieser Ausarbeitung ist es, einen Überblick über Warehouse-Location-Probleme und gängige Lösungsansätze zu geben. Da hier lediglich Grundlagen vermittelt werden sollen, beschränke ich mich auf Beispiele aus dem Bereich der unkapazitierten WLP, allerdings wird im Literaturverzeichnis auch deutlich weiterführende Literatur angegeben.
Bei Warehouse-Location-Problemen handelt es sich um np-schwere Probleme[1], daher wird mit einer Einführung über Mediane und anschließend über Warehouse-Location-Probleme begonnen und danach sowohl eine Heuristik, als auch ein exaktes Verfahren detailliert vorgestellt.

# 2 Mediane

Der Median eines Graphen ist derjenige Knoten, für den die Summe der, mit den Knotenbewertungen, gewichteten Entfernungen zu allen anderen Knoten minimal ist. Es handelt sich also um ein Minsum-Problem.
In der Theorie sind auch absolute Mediane vorstellbar, hier wird die Bedingung aufgegeben, dass es sich bei dem Median um einen Knoten handelt. Der absolute Median kann also auch auf einem beliebigen Punkt einer Kante liegen. Allerdings ist sowohl für ungerichtete, wie auch für gerichtete Graphen, bewiesen, dass zu jedem Punkt eines Graphen immer auch einen Knoten existiert, dessen gewichtete Entfernung zu allen Knoten nicht schlechter, als die des Punktes ist. Daraus folgt, dass auch der absolute Median immer auf einem Knoten liegt, damit ist hier eine Fallunterscheidung überflüssig.

## 2.1 ungerichtete Graphen

Für ungerichtete Graphen ist die Berechnung eines Medians relativ trivial. Grundbedingung für die Berechnung eines Medians ist es, dass es sich um einen zusammenhängenden Graphen handelt, es ist also jeder Knoten von jedem Knoten aus erreichbar. Zur Berechnung wird ein $\sigma(i)$ eingeführt, welches sich wie folgt definiert:

$$\sigma(i) := \sum_{j \in V} d_{ij} b_j$$

Wobei $V$ die Menge der Knoten des Graphen ist. $d_{ij}$ ist die kürzeste Entfernung vom Knoten $i$ zum Knoten $j$, wie sie zum Beispiel durch den Algorithmus von Dijkstra bestimmt werden kann und bei $b_j$ handelt es sich um die Knotenbewertung von $j$.
Der Median des Graphen ist nun derjenige Knoten $i$ mit minimalen $\sigma(i)$.

[1] Discrete Location Theory, Kap. 3

## 2.2 gerichtete Graphen

Für gerichtete Graphen wird die Problemstellung schon etwas komplexer. Auch hier gilt die Bedingung, dass jeder Knoten von jedem Knoten erreicht werden können muss, der Graph muss also stark zusammenhängend sein. Es wird nun ein $\sigma_{out}(i)$ ein $\sigma_{in}(i)$ und ein $\sigma_{io}(i)$ definiert:

$$\sigma_{out}(i) := \sum_{j \in V} d_{ij} b_j$$

$$\sigma_{in}(i) := \sum_{j \in V} d_{ji} b_j$$

$$\sigma_{io}(i) := \sigma_{out}(i) + \sigma_{in}(i)$$

Das $\sigma_{out}(i)$ bezeichnet die gewichtete Entfernung vom Knoten $i$ zu jedem Knoten $j$, das $\sigma_{in}(i)$ bezeichnet nun wiederum die gewichtete Entfernung von jedem Knoten $j$ zum Knoten $i$. Nun wird der Knoten $i$ mit minimalem $\sigma_{io}(i)$ als Median bezeichnet, allerdings ist es auch möglich einen out-Median als minimales $\sigma_{out}(i)$ und einen in-Median mit dem minimalen $\sigma_{in}(i)$ zu bestimmen.
Betrachten wir die Berechnung des Medians für gerichtete Graphen an einem kurzen Beispiel. In Abb. 1 sehen wir einen gerichteten Graphen. In Tab. 1 ist die zugehörige

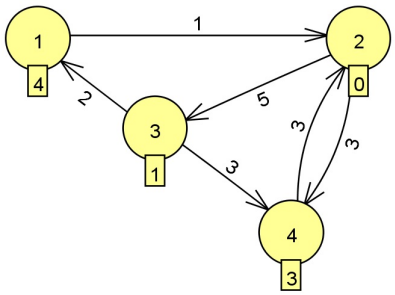


Abbildung 1: gerichteter Graph

| i \ j | **1** | **2** | **3** | **4** | **b** |
|---|---|---|---|---|---|
| **1** | 0 | 1 | 6 | 4 | 4 |
| **2** | 7 | 0 | 5 | 3 | 0 |
| **3** | 2 | 3 | 0 | 3 | 1 |
| **4** | 10 | 3 | 8 | 0 | 3 |
| **b** | 4 | 0 | 1 | 3 | |

Tabelle 1: Adjazenzmatrix von Abb. 1

Adjazenzmatrix zu sehen, unten und auf der rechten Seite um die Knotenbewertungen $b$ erweitert . Beginnen wir mit der Berechnung des $\sigma_{out}(i)$. Hierzu wird jedes Element einer Spalte mit der dazugehörigen Knotenbewertung $b$ multipliziert und anschließend die Zeilensumme gebildet. Das Ergebnis ist in Tabelle 2 zu sehen. Für unser Beispiel liegt also der out-Median in Knoten 3. Betrachten wir nun das $\sigma_{in}$, hierzu multiplizieren wir den Wert jeder Zeile mit dem zugehörigen $b$ und bilden anschließend die Spaltensumme, zu sehen in Tabelle 3. Der in-Median unseres Beispiels ist also Knoten 2.
Bilden wir nun die Summe aus $\sigma_{out}(i)$ und $\sigma_{in}(i)$, so erhalten wir $\sigma_{io}(i)$. Wir stellen

| i \ j | **1** | **2** | **3** | **4** | $\sigma_{out}(i)$ |
|---|---|---|---|---|---|
| **1** | 0 | 0 | 6 | 12 | 18 |
| **2** | 28 | 0 | 5 | 9 | 42 |
| **3** | 8 | 0 | 0 | 9 | **17** |
| **4** | 40 | 0 | 8 | 0 | 48 |

Tabelle 2: Berechnung des $\sigma_{out}$

| i \ j | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| **1** | 0 | 4 | 24 | 16 |
| **2** | 0 | 0 | 0 | 0 |
| **3** | 2 | 3 | 0 | 3 |
| **4** | 30 | 9 | 24 | 0 |
| $\sigma_{in}(i)$ | 32 | **16** | 48 | 19 |

Tabelle 3: Berechnung des $\sigma_{in}$

fest, dass Knoten 1 mit einem $\sigma_{io}$ von 50 der Median des Graphen ist.

## 2.3 p-Mediane

Nun ist es natürlich auch möglich, nicht nur einen einzelnen Median eines Graphen zu bestimmen, sondern auch eine p-elementige Menge $V^p$ als Teilmenge von $V$, wobei die Distanz aller Knoten zu einem Knoten der Lösungsmenge $V^p$ minimal werden soll. Es entsteht also folgende Definition für $\sigma$:

$$\sigma(V^p) := \sum_{j \in V} d(V^p, j) b_j$$

Der Median ist nun die Knotenmenge mit minimalem $\sigma(V^p)$.
Für kleine Mengen bzw. für ein kleines p ist hier eine Lösung eventuell noch durch ausprobieren aller Kombinationen möglich, im allgemeinen Fall ist dieses Vorgehen allerdings nicht praktikabel.
Um das Problem effizienter Lösen zu können, wird der Graph in einen bipartiten Graphen transformiert. Ein solcher bipartiter Graph ist in Abb. 2 ansatzweise dargestellt. Jeder Knoten tritt hier einmal als potentieller Median ($i = 1..m$) auf der linken Seite und einmal als Nachfrager ($j = 1..n$) auf der rechten Seite auf. Anschließend wird jeder Knoten der rechten Seite mit jedem Knoten der linken Seite verbunden, wobei die Kantenbewertung der kürzesten Distanz zwischen beiden Knoten entspricht.
Anschließend werden zwei Binärvariablen eingeführt $y_i$ wird mit 1 belegt falls $i$ Element

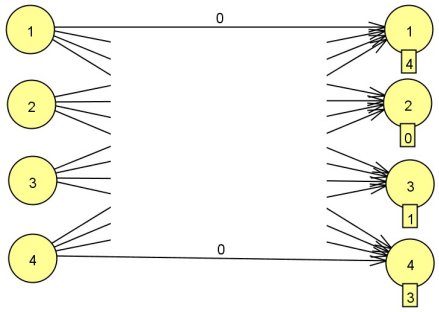


Abbildung 2: Bipartiter Graph

der p-Median-Menge ist, sonst mit 0. $x_{ij}$ wird nun mit 1 belegt falls der Nachfrager $j$ durch den Knoten $i$ abgedeckt wird, sonst mit 0. Ziel ist es nun folgende Zielfunktion zu minimieren:

$$Z(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} b_j d_{ij} x_{ij}$$

Hierbei gelten folgende Nebenbedingungen:

$$\forall j : \sum_{i=1}^{m} x_{ij} = 1$$

$$\forall ij : x_{ij} \leq y_i$$

$$\sum_{i=1}^{m} y_i = p$$

Wobei die erste Nebenbedingung aussagt, dass jeder Nachfrager durch ein Element der Lösungsmenge beliefert werden muss. Die Zweite stellt sicher, dass jeder Nachfrager nur durch einen Knoten beliefert werden kann, der auch Element der Lösungsmenge ist, und aus der dritten Nebenbedingung geht hervor, dass unsere Lösungsmenge genau p Elemente umfasst.

# 3 Warehouse-Location-Probleme

Kommen wir nun zu den Warehouse-Location-Problemen (kurz: WLP). Ziel der WLP ist es die kostengünstigste Standortplanung zu ermitteln. Hierfür steht eine Menge von potentiellen Standorten zur Verfügung, die durch entsprechende Vorarbeiten bereits herausgearbeitet wurden. Die Aufgabe besteht nun darin, den oder die potentiellen Standorte auszuwählen, von denen aus eine Menge von Kunden kostenminimal beliefert werden kann.
Es handelt sich hierbei also um eine spezielle Form der p-Median-Probleme. Wir betrachten wieder den bipartiten Graphen aus dem p-Median-Problem. Es treten nun die potentiellen Mediane als potentielle Standorte auf, wobei die Knotenbewertung $f_i$ als entstehende Standortfixkosten zu interpretieren sind. Die ehemaligen Nachfrager spiegeln nun die Kunden wieder, wobei deren Knotenbewertung $b_j$ als Nachfragemenge verstanden wird. Die Kantenbewertungen, im folgenden $c_{ij}$, spiegeln die Transportkosten von Standort $i$ zu Kunde $j$ wieder. Dieses ist in Abb. 3 noch einmal skizziert.
Die Binärvariable $y_i$ wird nun mit 1 belegt, wenn der potentielle Standort $i$ tatsächlich

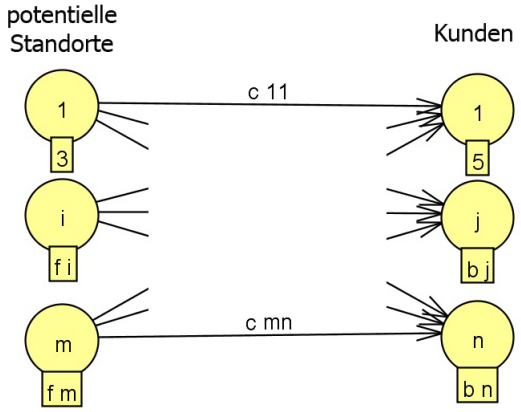


Abbildung 3: WLP als bipartiter Graph

zu errichten ist, die Bedeutung der Variable ändert sich also nicht. Auch die Bedeutung von $x_{ij}$ bleibt prinzipiell erhalten, sie kennzeichnet weiterhin, ob ein Standort $i$ einen Kunden $j$ beliefert, allerdings umfasst $x_{ij}$ bei verschiedenen WLP evtl. unterschiedliche Wertebereiche. Im folgenden werden kurz verschiedene WLP vorgestellt, aus denen sich allerdings auch weitere Probleme ableiten lassen.

## 3.1 unkapazitierte, einstufige WLP

Bei unkapazitierten, einstufigen Warehouse-Location-Problemen handelt es sich um die allgemeinste Form von WLP. Die Transportkosten $c_{ij}$ werden hier als absolute Kosten betrachtet, die anfallen um den gesamten Bedarf von Kunde $j$ durch Standort $i$ zu decken. Die ehemalige Binärvariable $x_{ij}$ stellt nun allerdings die anteilige Deckung des Bedarfes eines Kunden durch einen Standort dar, kann also Werte zwischen 0 und 1 annehmen. Somit lassen sich die Transportkosten als Produkt aus $c_{ij}$ und $x_{ij}$ bestimmen und es kann im weiteren Lösungsverfahren auf die Betrachtung der Nachfragemenge der Kunden verzichtet werden.

Es gilt nun folgende Zielfunktion als Summe aus Transportkosten und Fixkosten zu minimieren:

$$Z(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i$$

Folgende Nebenbedingungen sind dabei zu betrachten:

$$\forall ij : x_{ij} \leq y_i$$

$$\forall j : \sum_{i=1}^{m} x_{ij} = 1$$

$$\forall ij : 0 \leq x_{ij} \leq 1$$

Die Nebenbedingungen sind schnell erklärt. Aus der ersten Nebenbedingung geht hervor, dass ein Kunde nur durch einen Standort beliefert werden kann, der auch errichtet wird. Laut der zweiten Nebenbedingungen muss der gesamte Bedarf jedes Kunden befriedigt werden und die Dritte löst sich von der Vorgabe, dass $x_{ij}$ nur binäre Werte annehmen darf.

## 3.2 kapazitierte, einstufige WLP

Bei den kapazitierten, einstufigen WLP wird für jeden potentiellen Standort eine zusätzliche Knotenbewertung $a_i$ eingeführt. Dieses $a_i$ stellt die obere Kapazitätsgrenze eines Standortes dar und darf daher nicht überschritten werden. Um dieses Problem lösen zu können, werden die Transportkosten $c_{ij}$ hier als Kosten pro Transportierte Mengeneinheit interpretiert. Die Variablen $x_{ij}$ werden nun mit den absolut zu transportierenden Mengeneinheiten von Standort $i$ zu Kunde $j$ belegt. Die Zielfunktion

$$Z(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i$$

ändert sich damit im Vergleich zu den unkapazitierten WLP nicht. Allerdings treten einige Änderungen bei den Nebenbedingungen auf:

$$\forall i : \sum_{j=1}^{n} x_{ij} \leq a_i y_i$$

$$\forall ij : x_{ij} \leq b_j y_i$$

$$\forall j : \sum_{i=1}^{m} x_{ij} = b_j$$

Die Transportierte Menge eines Standortes darf maximal seiner Kapazitätsgrenze entsprechen, ein Standort darf nicht mehr als den Bedarf eines Kunden liefern und die Gesamtnachfrage eines Kunden muss gedeckt werden. In der Realität ist es nun natürlich häufig der Fall, dass sich die Transportkosten pro Mengeneinheit mit höherer Ausbringungsmenge reduzieren, auch solche Funktionen können für die Belegung der $c_{ij}$ verwendet werden, erhöhen allerdings entsprechend den Lösungsaufwand. Prinzipiell müssen zur Lösung der kapazitierten, einstufigen WLP für die einzelnen Standorte Transportprobleme gelöst werden, was den Rechenaufwand gegenüber den unkapazitierten WLP deutlich erhöht.

## 3.3 kapazitierte, mehrstufige WLP

In den bisher betrachteten WLP wurde quasi davon ausgegangen, dass Produktionswerk und Auslieferungslager identisch sind, diese Annahme muss natürlich nicht der Realität entsprechen. Für mehrstufige WLP wird eine komplexere Distributionsstruktur betrachtet. Transporte von einem Werk über Auslieferungslager zu einem Kunden sind hier denkbar. Weitere Stufen, beispielsweise zusätzliche Zwischenlager, wären hier ebenfalls vorstellbar, würden das Problem in diesem Rahmen allerdings unübersichtlich machen. Wir betrachten nun also eine Struktur aus bereits errichteten Werken, potentiellen Standorten für Auslieferungslager und Kunden, wie in Abb. 4 dargestellt. Die Werke

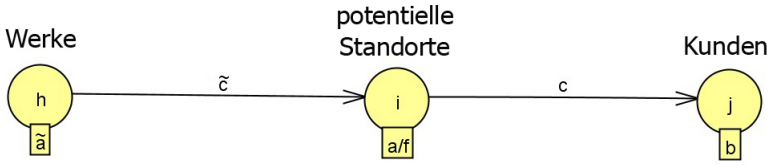


Abbildung 4: kapazitierte, mehrstufige WLP

($h = 1..k$) verfügen durch ihre Knotenbewertung über maximale Kapazitäten, dargestellt durch $\tilde{a_h}$. Auch die potentiellen Standorte verfügen durch ihre Knotenbewertung über Kapazitätsgrenzen bzw. Umladekapazitäten $a_i$, sowie die bereits bekannten Fixkosten. Die Transportkosten pro Mengeneinheit zwischen dem Werk h und dem Standort i werden durch $\tilde{c_{hi}}$ dargestellt. Entsprechend dazu wird eine weitere Variable $\tilde{x_{hi}}$ eingeführt, die die absolut transportierten Mengeneinheiten von Werk h zu Standort i angibt. Die Zielfunktion wird nun um die Transportkosten von den Werken zu den Standorten erweitert:

$$Z(\tilde{x}, x, y) = \sum_{h=1}^{k}\sum_{i=1}^{m} \tilde{c}_{hi}\tilde{x}_{hi} + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i$$

Folgende Nebenbedingungen sind dabei zu beachten:

$$\forall h : \sum_{i=1}^{m} \tilde{x_{hi}} \leq \tilde{a_h}$$

$$\forall i : \sum_{h=1}^{k} \tilde{x_{hi}} \leq a_i$$

$$\forall i : \sum_{h=1}^{k} \tilde{x_{hi}} - \sum_{j=1}^{n} x_{ij} = 0$$

Hier aufgeführt sind die Nebenbedingungen bezogen auf die Kapazitätsrestriktionen der Werke und den Transporten zu den Standorten, zusätzlich gelten die Nebenbedingungen, die bei den kapazitierten, einstufigen WLP aufgeführt wurden. Die letzte Nebenbedingung wird nur für den Fall betrachtet, dass gefordert wird, dass der Wareneingang im Auslieferungslager genau dem Warenausgang entspricht. Diese Forderung kann den Lösungsaufwand deutlich reduzieren und verschärft die zweite Nebenbedingung.
Grundsätzlich müssen bei dieser Form der WLP Umladeprobleme gelöst werden.

# 4 Heuristiken

Allgemein wird in diesem Kontext zwischen Eröffnungsverfahren zur Bestimmung einer gültigen Lösung, Verbesserungsverfahren zum Verbessern einer gefundenen Lösung, Kombinationen von Eröffnungs- und Verbesserungsverfahren, sowie Verfahren der begrenzten Enumeration, also modifizierten exakten Verfahren unterschieden.

## 4.1 ADD-Algorithmus

Im Folgenden wird der ADD-Algorithmus als Eröffnungsverfahren für unkapazitierte WLP detailliert beschrieben und an einem Beispiel vorgeführt, anschließend werden einige notwendige Modifikationen genannt um den Algorithmus auch für kapazitierte, einstufige WLP anwenden zu können.
Prinzipiell zielt der Algorithmus darauf ab, immer den kostengünstigsten Standort zu bestimmen, diesen in die Lösungsmenge aufzunehmen und anschließend zu prüfen, ob ein weiterer Standort die Gesamtkosten verringern kann. Dargestellt wird der Algorithmus in Abb. 5 als Flussdiagramm.
Zuerst ist jedoch die Definition einiger weiterer Variablen notwendig:

- $I$ als Menge der potentiellen Standorte
- $J$ als Menge der potentiellen Kunden
- $I0$ als Menge der verbotenen Standorte
- $I1$ als Menge der einbezogenen Standorte
- $I0_{vl}$ als Menge der vorläufig verbotenen Standorte
- $I1_{vl}$ als Menge der vorläufig einbezogenen Standorte

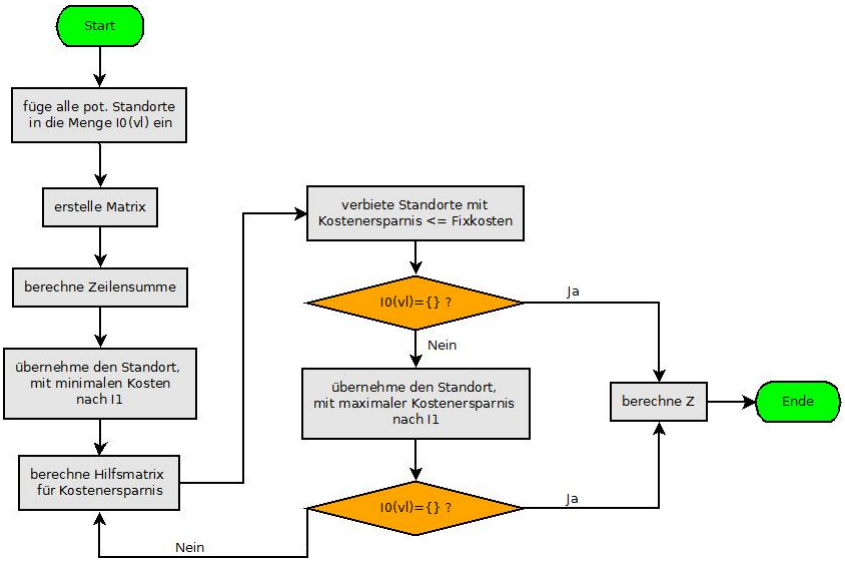


Abbildung 5: Flussdiagramm ADD-Algorithmus

### 4.1.1 unkapazitierte WLP

Der Algorithmus beginnt damit, dass alle potentiellen Standorte den vorläufig verbotenen Standorten zugeordnet werden. Anschließend betrachten die Adjazenzmatrix des bipartiten Graphen, wobei die einzelnen Werte die absoluten Transportkosten vom Standort $i$ zu Kunde $j$ repräsentieren. Diese Matrix erweitern wir um die Standortfixkosten $f_i$ und bilden die Zeilensumme $\Sigma$. Diese ersten Schritte sind in Tab. 4 zu sehen.
Dann wählen wir den Standort mit den geringsten Gesamtkosten, also dem minimalen $\Sigma$, aus, entfernen ihn aus der Menge der vorläufig verbotenen Standorte und fügen ihn in die Menge der einbezogenen Standorte ein. Diesen zuletzt eingefügten Standort bezeichnen wir mit k. Im Beispiel ist dies der Knoten 4 (in Tab. 4 grün markiert).
Im nächsten Schritt wird die Kostenersparnis berechnet, die wir erreichen würden, wenn

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $\sum$ |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 6 | 9 | 12 | 12 | 45 |
| **2** | 4 | 6 | 9 | 0 | 12 | 8 | 6 | 45 |
| **3** | 6 | 3 | 10 | 12 | 0 | 4 | 9 | 44 |
| **4** | 12 | 4 | 12 | 3 | 3 | 0 | 9 | 43 |
| **5** | 3 | 0 | 8 | 7 | 8 | 6 | 12 | 44 |

Tabelle 4: ADD-Algorithmus - Ausgangssituation

wir einen anderen Standort einbeziehen würden. Zu diesem Zweck berechnen wir für jedes $c_{ij}$ ein $\omega_{ij}$ als $max\{0, c_{kj} - c_{ij}\}$. Nun haben wir jedes $\omega_{ij}$ bestimmt und damit die Transportkosten, die wir sparen, wenn wir den Kunden j nicht mehr durch den Standort k sondern durch den Standort i beliefern. Wir bilden nun die Zeilensumme aller $\omega_{ij}$ und bezeichnen das Ergebnis als $\omega_i$. Diese weiteren Schritte sind in Tab. 5 dargestellt. Mit $\omega_i$ haben wir die Gesamttransportkostenersparnis für den Standort i bestimmt. Sollte die Transportkostenersparnis eines Standortes niedriger sein als dessen Fixkosten, können wir diesen Standort endgültig verbieten, da unsere Gesamtkosten steigen würden, wenn wir diesen Standort einbeziehen. In unserem Beispiel ist dieses jedoch noch nicht der Fall. Wir bestimmen nun den Standort mit der größten Ersparnis, also den Standort mit der größten Differenz zwischen $\omega_i$ und $f_i$. Im Beispiel ist das der Standort 1 (in Tab. 5 grün markiert). Wir beziehen nun den Standort 1 endgültig ein, entfernen ihn aus der Menge der vorläufig verbotenen Standorte und bezeichnen ihn als k, da er zuletzt einbezogen wurde.
Nun bestimmen wir wieder die Kostenersparnis für den nächsten Durchlauf. Diese Be-

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $\omega_i$ |
|---|---|---|---|---|---|---|---|---|
| **1** | 12 | 2 | 8 | 0 | 0 | 0 | 12 | 22 |
| **2** | 8 | 0 | 3 | 3 | 0 | 0 | 6 | 14 |
| **3** | 6 | 1 | 2 | 0 | 3 | 0 | 9 | 12 |
| **5** | 9 | 4 | 4 | 0 | 0 | 0 | 12 | 17 |

Tabelle 5: ADD-Algorithmus - 1. Iteration

stimmung ändert sich gegenüber dem Startschritt geringfügig. Da wir in unserer Matrix bereits eine Kostenersparnis betrachten, müssen wir das $\omega_{ij}$ nun als Ersparnis gegenüber der ersten Ersparnis berechnen. Hierzu belegen wir $\omega_{ij}$ mit $max\{0, w_{ij} - w_{kj}\}$. Anschließend bilden wir erneut $\omega_i$ und vergleichen die Werte mit den jeweiligen Fixkosten $f_i$. In unserem Beispiel stellen wir fest, dass nun die Fixkosten für jeden Standort höher

sind als die Transportkostenersparnis, die wir erzielen könnten. Daraus folgt, dass wir die verbliebenen Standorte endgültig verbieten können. Die Ergebnisse für die letzten Schritte sind in Tab. 6 dargestellt. Da wir nun alle Elemente aus unserer Menge der vorläufig verbotenen Standorte entfernt haben, endet der Algorithmus.
Als letztes gilt es nun die tatsächliche Ausgangslösung zu bestimmen. Es wird jedes $y_i$

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $\omega_i$ |
|---|---|---|---|---|---|---|---|---|
| **2** | 0 | 0 | 0 | 3 | 0 | 0 | 6 | 3 |
| **3** | 0 | 0 | 0 | 0 | 3 | 0 | 9 | 3 |
| **5** | 0 | 2 | 0 | 0 | 0 | 0 | 12 | 2 |

Tabelle 6: ADD-Algorithmus - 2. Iteration

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ |
|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 6 | 9 | 12 | 12 |
| **2** | 4 | 6 | 9 | 0 | 12 | 8 | 6 |
| **3** | 6 | 3 | 10 | 12 | 0 | 4 | 9 |
| **4** | 12 | 4 | 12 | 3 | 3 | 0 | 9 |
| **5** | 3 | 0 | 8 | 7 | 8 | 6 | 12 |

Tabelle 7: ADD-Algorithmus - Lösung

mit $i \in I1$ auf 1 gesetzt, die übrigen $y_i$ werden mit 0 belegt. Anschließend belegen wir für jeden Kunden j das $x_{ij}$ mit 1 mit minimalem $c_{ij}$, also den minimalen Transportkosten, wobei $i \in I1$ sein muss. Die restlichen $x_{ij}$ werden mit 0 belegt. Tabelle 7 zeigt noch einmal die Ausgangssituation, wobei hier die Daten der Lösung grün markiert wurden. Für unser Beispiel ergibt sich also folgende Lösung:

$$y_1 = y_4 = 1$$

$$x_{11} = x_{12} = x_{13} = x_{44} = x_{45} = x_{46} = 1$$

Hieraus lässt sich der Zielfunktionswert Z für unkapazitierte, einstufige WLP berechnen, er ergibt im Beispiel einen Wert von $Z = 33$

### 4.1.2 kapazitierte WLP

Der ADD-Algorithmus lässt sich nun ohne größeren Aufwand auch für kapazitierte, einstufige WLP erweitern. Hierfür muss zu Beginn jeder Iteration eine Fallunterscheidung durchgeführt werden. Betrachtet werden hierfür die Gesamtkapazität aller einbezogenen Standorte, sowie die Gesamtnachfrage. Ist die Kapazität größer als die Nachfrager, wird ein zusätzlicher fiktiver Kunde eingeführt, der die überschüssigen Mengen abnimmt. Die Transportkosten von jedem Standort zu dem fiktiven Kunden werden mit 0 belegt. Wenn die Nachfrage größer als die Kapazität ist, wird ein fiktiver Standort eingeführt, dieser deckt die fehlenden Kapazitäten. Die Transportkosten vom fiktiven Standort zu jedem Kunden werden mit unendlich belegt, damit ausgeschlossen werden kann, dass der Standort in die Lösung einbezogen wird. Der fiktive Standort wird in der Regel zu Beginn des ADD-Verfahren nötig sein, da zu diesem Zeitpunkt noch kein Standort einbezogen wurde und somit die Gesamtkapazität 0 ist.
Anschließend müssen für jeden Standort Transportprobleme zum Bestimmen der Kostenersparnis gelöst werden. Es wird wieder der Standort mit der höchsten Kostenersparnis in die Lösung einbezogen. Sollten sich für einen Standort nach der Lösung des Transportproblems negative Kostenersparnisse ergeben, so kann dieser Standort endgültig verboten werden.

### 4.2 Verbesserungsverfahren

Da es sich beim ADD-Algorithmus um eine relativ einfache Eröffnungsheuristik handelt, wird im Regelfall notwendig sein, die gefundene Lösung zu verbessern. Ein Pendant zum ADD-Algorithmus ist der DROP-Algorithmus, dieser geht ähnlich vor, nur mit dem Unterschied, dass zu Beginn jeder Standort vorläufig erlaubt wird und dann immer der Standort verboten wird, dessen Verbot die Kosten am stärksten senkt. Ein einfaches Verbesserungsverfahren wäre es nun, eine zulässige Ausgangslösung mit dem ADD-Algorithmus zu bestimmen. Anschließend wird nun mit dem DROP-Algorithmus derjenige Standort aus der Lösung entfernt, bei dem der Zielfunktionswert am stärksten sinkt, bzw. am wenigsten stark ansteigt. Anschließend wird wieder der ADD-Algorithmus angewendet. Dieses Verfahren wird solange wiederholt, bis genau der Standort durch das ADD-Verfahren hinzugefügt wird, der direkt vorher durch den DROP-Algorithmus entfernt wurde. Im Normalfall ergibt sich dadurch eine bessere Lösung.

## 5 Exakte Verfahren

Die ersten exakten Verfahren basierten ausschließlich auf vollständiger Enumeration über allen potentiellen Standorten. Ein solches Verfahren ist natürlich gerade für viele Standorte nicht effizient duchführbar. Aus diesem Grund werden häufig Branch & Bound-Verfahren verwendet.

### 5.1 Branch & Bound-Verfahren

Bei Branch & Bound-Verfahren handelt es sich um eine Methodik, die darauf abzielt obere und untere Schranken eines Problems zu bestimmen und anschließend dieses Problem in Teilprobleme zu verzweigen.
Durch die Verzweigung des ursprünglichen Problems entsteht eine Baumstruktur, in der man im Laufe der Zeit die optimale Lösung bestimmen kann. Zum Bestimmen der unteren Schranke wird hierbei eine Relaxation des eigentlichen Problems gebildet, jede Lösung dieser Relaxation bildet eine mehr oder weniger gute untere Schranke des WLP. Jede zulässige Lösung unseres ursprünglichen Problems bildet eine obere Schranke des Problems. Sollte die untere Schranke der Relaxation und die obere Schranke eines Teilproblems den gleichen Wert ergeben, so ist dieses Teilproblem optimal gelöst.
Kann keine optimale Lösung gefunden werden, merkt man sich die bestimmten oberen und unteren Schranken und verzweigt das aktuelle Problem. Dieses geschieht im Umfeld der Warehouse-Location-Probleme in der Regel dadurch, dass zwei Teilprobleme gebildet werden in denen ein $y_i$ zu 0 und zu 1 fixiert werden. Anschließend wird erneut versucht die Teilprobleme optimal zu lösen. Allerdings kann es vorkommen, dass vor dem Lösen von Teilproblemen gewisse Zweige des Problembaums durch logische Tests ausgeschlossen werden. Dieses geschieht, wenn man sicherstellen kann, dass sich die optimale Lösung des ursprünglichen Problems nicht in dem aktuellen Zweig des Verzweigungsbaums befindet.
Im ungünstigsten Fall ist allerdings auch bei Branch& Bound-Verfahren eine vollständige Enumeration notwendig.
Ein wichtiges Unterscheidungskriterium im Bereich der WLP ist die Wahl der Relaxation. Hierbei handelt es sich häufig um LP-Relaxationen[2], die auf Basis der aggregierten oder der disaggregierten Problemformulierungen für WLP gewählt werden. Liegt die

[2]Bei LP-Relaxationen wird in der Regel die Forderung nach Ganzzahligkeit einer Variablen aufgelöst

aggregierte Problemformulierung zugrunde, so wird von schwachen LP-Relaxationen gesprochen, bei der disaggregierten Problemformulierung wird von starken LP-Relaxationen gesprochen.
Bei den schwachen LP-Relaxationen wird üblicher Weise wenig Aufwand in die Bestimmung einer Lösung der Relaxation investiert, dafür sind viele Verzweigungen notwendig. Bei starken LP-Relaxationen wird in diesem Kontext relativ viel Aufwand in die Lösung eines Problems investiert, dafür muss kaum verzweigt werden. Im direkten Vergleich sind Branch & Bound-Verfahren auf Basis der starken LP-Relaxation in der Regel effizienter. Ein grober Überblick über das Vorgehen bei Branch & Bound-Verfahren wird im Flussdiagramm Abb. 6 gegeben.

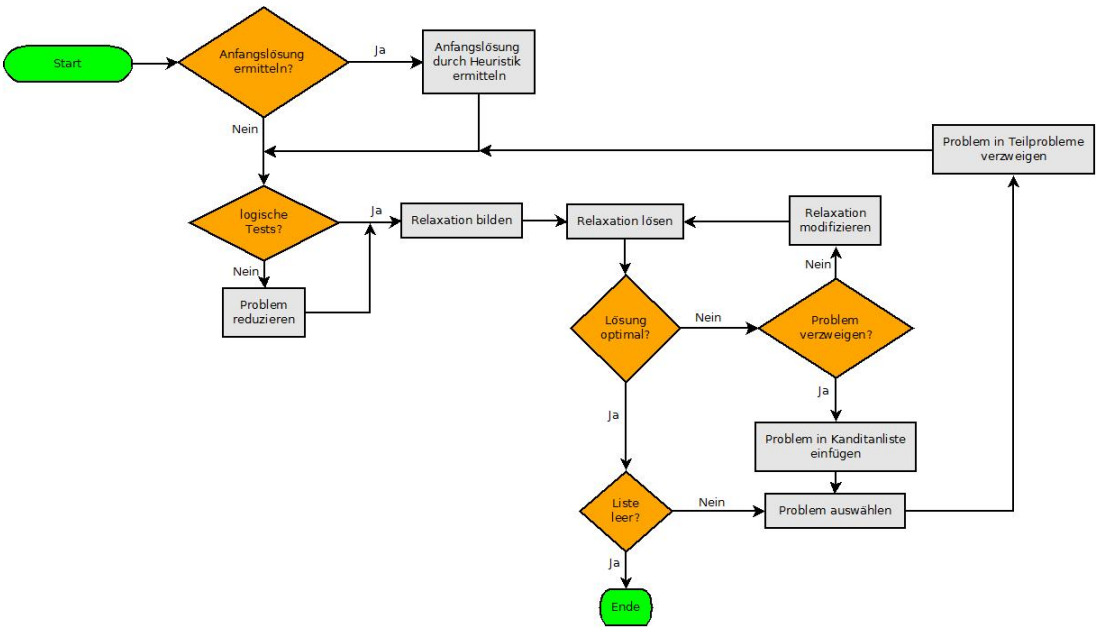


Abbildung 6: Flussdiagramm Branch & Bound-Verfahren

## 5.2 Branch & Bound nach Erlenkotter

Im Folgenden wird das Branch & Bound-Verfahren nach Erlenkotter genauer betrachtet und an einem ausführlichen Beispiel erläutert. Erlenkotter verwendet die starke LP-Relaxation und damit die disaggregierte Problemformulierung. Es wird also ein hoher Aufwand zur Lösung eines Problems zugunsten weniger Verzweigungen nötig. Die Relaxation besteht darin, dass die Binärbedingung $y_i \in \{0, 1\}$ zu $y_i \geq 0$ relaxiert wird.
Dadurch ergibt sich folgende relaxierte Problemstellung, die minimiert werden muss

$$Z(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i$$

mit den Nebenbedingungen:

$$\forall_{ij} - x_{ij} + y_i \geq 0$$

$$\forall_i \sum_{i=1}^{m} x_{ij} = 1$$

$$\forall_{ij} x_{ij}, y_i \geq 0$$

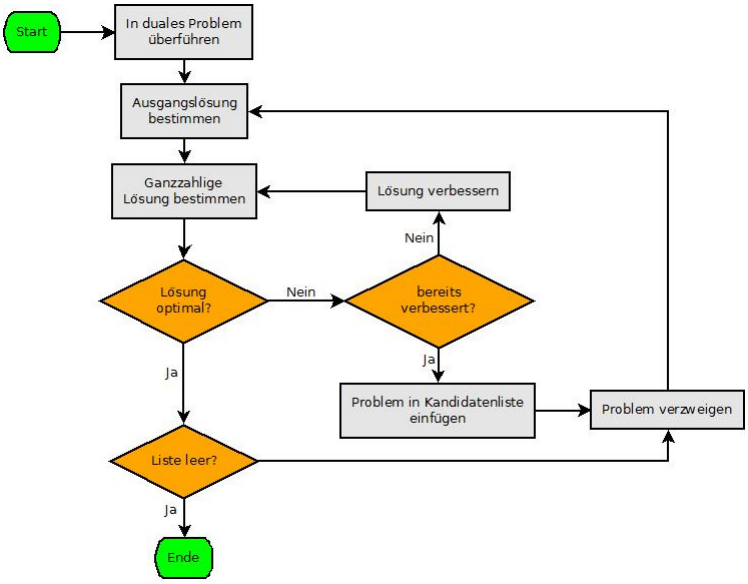


Abbildung 7: Flussdiagramm B &B nach Erlenkotter

Wie in Abb. 7 zu sehen, bildet Erlenkotter im ersten Schritt das duale Problem zu seinem unkapazitierten Warehouse-Location-Problem. Hierzu führt er eine Dualvariable $v_j$ ein. Die zu maximierende Zielfunktion für das duale Problem lautet damit:

$$ZD(v) = \sum_{j=1}^{n} v_j$$

mit der Nebenbedingung

$$\forall_i \sum_{j=1}^{n} max\{0, v_j - c_{ij}\} \leq f_i$$

Es gilt $Z(x, y) \geq ZD(v)$. Damit bildet das duale Problem eine untere Schranke für das behandelte Problem, nur bei der optimalen Lösung gilt $Z(x, y) = ZD(v)$.

### 5.2.1 Dual-Ascent-Methode

Anschließend verwendet Erlenkotter die Dual-Ascent-Methode, um eine zulässige Ausgangslösung zu bestimmen. Hierfür führt er für jeden Standort eine Schlupfvariable $s_i$. Diese Schlupfvariable ist definiert als

$$s_i := f_i - \sum_{j=1}^{n} max\{0, v_j - c_{ij}\}$$

Damit sorgt die Schlupfvariable dafür, dass die Nebenbedingung für das duale Problem stets eingehalten wird, deshalb muss stets darauf geachtet werden, dass $s_i$ nicht negativ wird.
Zu Beginn des Algorithmus wird nun für jeden Kunden ein Index $k(j)$ angelegt, der aufsteigend sortiert die Werte $c_{ij} \geq v_j$ enthält.
Ziel ist es nun die Summe der $v_j$ zu maximieren. Deshalb wird bei jeder Iteration versucht das $v_j$ auf den nächsthöheren Indexwert zu erhöhen. Der Algorithmus endet, wenn bei einer Iteration kein $v_j$ mehr erhöht werden kann.
Als Eröffnungsverfahren benötigt der Algorithmus eine gültige Startlösung. Diese lässt

sich leicht konstruieren, in dem für jedes $j$ das zugehörige $v_j$ mit dem niedrigsten $c_{ij}$ belegt und unser Index k um einen Wert erhöht wird. Unser Index zeigt damit in jeder Spalte auf das zweitkleinste $c_{ij}$. Diese duale Lösung ist auf jeden Fall gültig, da die Schlupfvariable $s_i$ zu diesem Zeitpunkt mit dem jeweiligen $f_i$ belegt ist. Darüber hinaus kann man davon ausgehen, dass die Lösung nicht optimal ist, da das $v_j$ hier mit minimalen Werten belegt ist. Wir betrachten nun das Beispiel, das bereits für den ADD-Algorithmus verwendet wurde, in der Tabelle 8 bereits um ergänzt die $v_j$ und $s_i$. Ziel ist

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ |
|---|---|---|---|---|---|---|---|---|
| **1** | $\underline{0}$ | 2 | $\underline{4}$ | 6 | 9 | 12 | 12 | 12 |
| **2** | 4 | 6 | 9 | $\underline{0}$ | 12 | 8 | 6 | 6 |
| **3** | 6 | 3 | 10 | 12 | $\underline{0}$ | 4 | 9 | 9 |
| **4** | 12 | 4 | 12 | 3 | 3 | $\underline{0}$ | 9 | 9 |
| **5** | 3 | $\underline{0}$ | 8 | 7 | 8 | 6 | 12 | 12 |
| $v_j$ | 0 | 0 | 4 | 0 | 0 | 0 | | |

Tabelle 8: Dual-Ascent-Methode - Ausgangssituation

es nun die $v_j$ größtmöglich zu erhöhen. Dafür wird versucht, die $v_j$ bei jeder Iteration auf den nächsten Indexwert zu erhöhen. Zu diesem Zweck wird für jeden Kunden ein $\sigma_j$ bestimmt. $\sigma_j$ wird nun mit dem kleinsten $s_i$ belegt, wobei nur diejenigen $s_i$ betrachtet werden, deren $c_{ij} \leq v_j$ gilt. Diese Werte sind in der Tabelle 8 unterstrichen. Wir beginnen mit dem ersten Kunden. In dieser Situation ist nur das $c_{11} < v_1$, unser $\sigma_1$ nimmt also vorerst den Wert 12 an.
Sollte $\sigma_j > c_{kj} - v_j$ sein, gilt $\sigma_j := c_{kj} - v_j$ und der Index k wird inkrementiert. Anschließend wird $s_i$ für jedes i mit $c_{ij} \leq v_j$ um $\sigma_j$ reduziert und $v_j$ um $\sigma_j$ erhöht.
Für unseren ersten Kunden bedeutet das, dass unser Index derzeit auf $i = 5$ zeigt damit gilt $\sigma_1 > c_{51} - v_1$ also $12 > 3 - 0$. Der Index zeigt nun für den ersten Kunden auf das drittkleinste $c_{i1}$ und wir setzen $\sigma_1 = 3 - 0 = 3$. Also reduzieren wir $s_1$ um 3 auf 9 und erhöhen $v_1$ auf 3.
Das Verfahren wiederholt sich für alle Kunden. Anschließend wird mit der nächsten Iteration begonnen. Die Situation nach der ersten Iteration ist in der Tabelle 9 mit den Veränderungen der $\sigma_j$ und $s_i$ dargestellt.
Wir betrachten nun wieder den ersten Kunden, die in Tabelle 9 unterstrichenen Werte

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ | $s_i$(1.It) |
|---|---|---|---|---|---|---|---|---|---|
| **1** | $\underline{0}$ | $\underline{2}$ | $\underline{4}$ | 6 | 9 | 12 | 12 | 12 | →9→5 |
| **2** | 4 | 6 | 9 | $\underline{0}$ | 12 | 8 | 6 | 6 | →3 |
| **3** | 6 | 3 | 10 | 12 | $\underline{0}$ | 4 | 9 | 9 | →6 |
| **4** | 12 | 4 | 12 | $\underline{3}$ | $\underline{3}$ | $\underline{0}$ | 9 | 9 | →5 |
| **5** | $\underline{3}$ | $\underline{0}$ | $\underline{8}$ | 7 | 8 | 6 | 12 | 12 | →10 |
| $v_j$ | 0 | 0 | 4 | 0 | 0 | 0 | | | |
| $\sigma_j$ | 12→3 | 6→2 | 9→4 | 6→3 | 9→3 | 9→4 | | | |
| $v_j$ | 3 | 2 | 8 | 3 | 3 | 4 | | | |

Tabelle 9: Dual-Ascent-Methode - 1.Iteration

sind wieder die $c_{ij} \leq v_j$, für den ersten Kunden also $c_{11}$ und $c_{51}$. Wir belegen nun vorerst unser $\sigma_1$ mit dem Minimum von $s_1 = 5$ und $s_5 = 10$ also mit 5. Unser Index zeigt aktuell auf $i = 2$. Wir prüfen also wieder $\sigma_i > c_{21} - v_1$, erneut ist diese Bedingung erfüllt, damit ändert sich $\sigma_1 = c_{21} - v_1 = 1$. In der Konsequenz reduzieren wir $s_1$ und $s_5$ um jeweils 1

und erhöhen $v_1$ um 1 auf 4.
Zwangsläufig werden irgendwann ein oder mehrere $s_i$ zu 0 reduziert. Diese $s_i$ können

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ | $s_i$(2.It) | $s_i$(3.It) |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 6 | 9 | 12 | 12 | 5 | →4→3→2 | → 1 |
| **2** | 4 | 6 | 9 | 0 | 12 | 8 | 6 | 3 | →0 | 0 |
| **3** | 6 | 3 | 10 | 12 | 0 | 4 | 9 | 6 | →4 | →3 |
| **4** | 12 | 4 | 12 | 3 | 3 | 0 | 9 | 5 | →2→0 | 0 |
| **5** | 3 | 0 | 8 | 7 | 8 | 6 | 12 | 10 | →9→8→7 | →6 |
| $v_j$ | 3 | 2 | 8 | 3 | 3 | 4 | | | | |
| $v_j$(2.It) | 4 | 3 | 9 | 6 | 5 | 4 | | | | |
| $v_j$(3.It) | 4 | 4 | 9 | 6 | 5 | 4 | | | | |

Tabelle 10: Dual-Ascent-Methode - 2.&3. Iteration

nicht weiter reduziert werden. Ist nun ein $c_{ij} \leq v_j$ in einer Zeile mit $s_i = 0$ so kann dieses $v_j$ nicht weiter erhöht werden, da $\sigma_j = 0$ gilt. Der Algorithmus endet, wenn in einer Iteration kein $v_j$ mehr erhöht werden kann. Bei dem Beispiel ist das nach der 3. Iteration der Fall. Die Veränderungen in der 2. und 3. Iteration wird in der Tabelle 10 dargestellt. Wir können einen dualen Zielfunktionswert $ZD = 33$ ermitteln, dieser stellt aktuell unsere untere Schranke dar. Im Anschluss bestimmt Erlenkotter eine ganzzahlige Lösung zu seiner dualen Lösung.

### 5.2.2 Ganzzahlige Lösung

Zum Bestimmen einer ganzzahligen Lösung betrachten wir nur jene Standorte, bei denen $s_i$ auf 0 reduziert werden konnte, in unserem Beispiel also die Standorte 2 und 4. Wir verwenden hier wieder die aus dem ADD-Algorithmus bekannte Menge I1 für die einbezogenen Standorte. Wir prüfen nun für jeden Kunden ob bereits ein Standort $i$ Element von I1 ist, für den gilt $c_{ij} < v_j$. Ist das nicht der Fall übernehmen wir von den Standorten mit $s_i = 0$ den Standort nach I1 mit $min\{c_{ij}\}$. Die $y_i$ mit $i \in I1$ werden mit 1 belegt, die Restlichen mit 0. Nun müssen noch die $x_{ij}$ bestimmt werden. Hierzu belegen wir für jeden Kunden $x_{ij}$ mit $min\{c_{ij}\}$ und $i \in I1$ mit 1, den Rest mit 0. Für unser Beispiel ergibt sich folgende Belegung:

$$y_2 = y_4 = 1$$

$$x_{21} = x_{42} = x_{23} = x_{24} = x_{45} = x_{46} = 1$$

Setzen wir die ermittelte Belegung in unsere Zielfunktion

$$Z(x,y) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i$$

ein so erhalten wir einen Zielfunktionswert $Z = 35$. Dieser Zielfunktionswert ist deutlich höher, als die gefundene duale Lösung, deshalb wendet Erlenkotter im Folgenden die Dual-Adjustment-Methode an.

### 5.2.3 Dual-Adjustment-Methode

Bevor die Dual-Adjustment-Methode angewendet werden kann sind einige Vorbereitungen nötig. Als erstes wird für jeden Kunden $j$ derjenige Standort mit den niedrigsten und den zweitniedrigsten Transportkosten bestimmt, wobei nur die Standorte betrachtet werden, deren Schlupfvariable $s_i$ aktuell mit 0 belegt ist. Diese Variablen werden mit $i_1(j)$ und $i_2(j)$ bezeichnet. Als nächstes wird für jeden Kunden eine Menge $I1_j$ gebildet. In diese Menge werden jeweils die Standorte eingefügt, deren $c_{ij} < v_j$ ist. Als letzten Vorbereitungsschritt wird für jeden Standort eine Menge $J_i^+$ benötigt. Wenn die Schlupfvariable für einen Standort mit 0 belegt ist, werden in die Menge $J_i^+$ für diesen Standort jene Kunden eingefügt, für die der Standort als einziger Lieferant in Frage kommt. Es darf also nur das $c_{ij}$ des aktuellen Standortes $\leq v_j$ gelten. Für unser Beispiel ist die Situation nach der Vorbereitung in Tab. 11 dargestellt.

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ | $J_i^+$ |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 6 | 9 | 12 | 12 | 1 | {} |
| **2** | 4 | 6 | 9 | 0 | 12 | 8 | 6 | 0 | {1,3} |
| **3** | 6 | 3 | 10 | 12 | 0 | 4 | 9 | 3 | {} |
| **4** | 12 | 4 | 12 | 3 | 3 | 0 | 9 | 0 | {2,5,6} |
| **5** | 3 | 0 | 8 | 7 | 8 | 6 | 12 | 6 | {} |
| $v_j$ | 4 | 4 | 9 | 6 | 5 | 4 | | | |
| $I1_j$ | {} | {} | {} | {2,4} | {4} | {4} | | | |
| $i_1$ | 2 | 4 | 2 | 2 | 4 | 4 | | | |
| $i_2$ | 4 | 2 | 4 | 4 | 2 | 2 | | | |

Tabelle 11: Dual-Adjustment-Methode - Vorbereitung

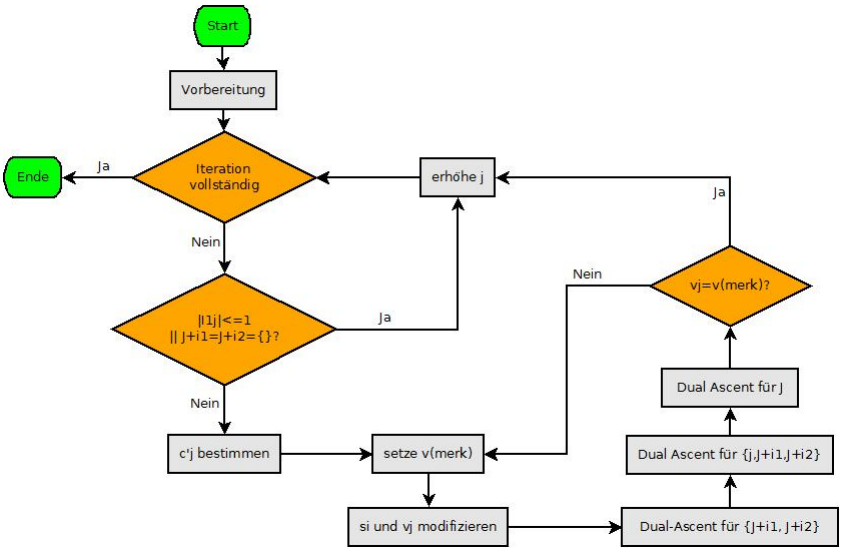


Abbildung 8: Flussdiagramm: Dual-Adjustment-Methode

Wie dem Flussdiagramm Abb. 8 zu entnehmen wird nun für jeden Kunden geprüft, ob $|I1_j| \leq 1$ oder $J_{i_1}^+ = J_{i_2}^+ = \{\}$ gilt. Ist dieses der Fall wird dieser Kunde nicht weiter betrachtet und der Algorithmus fährt mit dem nächsten Kunden fort.
In unserem Beispiel werden also die Kunden eins, zwei und drei vorerst nicht weiter

betrachtet. Erst für den vierten Kunden hat die Menge $I1_4$ mehr als ein Element, dieser Kunde wird also weiter betrachtet. Ziel ist es nun dieses $v_j$ also $v_4$ zu reduzieren, um dafür ein oder mehrere $v_j$ zu erhöhen und damit die untere Schranke nach oben zu verschieben. Wir bestimmen dafür ein $c^{'}$ als $max\{c_{ij}\}$ mit $c_{ij} < v_j$, im Beispiel würde also $c^{'} = 3$ gelten. Als nächstes führen wir eine Prüfvariable $v_{merk}$ ein. $v_{merk}$ wird mit dem aktuellen $v_j$ belegt, also im Beispiel $v_{merk} = v_4 = 6$. Danach wird jede Schlupfvariable $s_i$ um $c^{'}$ erhöht, sofern gilt $c_{ij} < v_j$, außerdem wird $v_j$ mit $c^{'}$ belegt. Diese Veränderung wird in Tabelle 12 dargestellt.

Aus dem Flussdiagramm für die Dual-Adjustment-Methode (Abb. 8) geht nun hervor,

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ | $s_i$ |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 6 | 9 | 12 | 12 | 1 | |
| **2** | $\underline{4}$ | 6 | $\underline{9}$ | $\underline{0}$ | 12 | 8 | 6 | 0 | →3 |
| **3** | 6 | 3 | 10 | 12 | 0 | 4 | 9 | 3 | |
| **4** | 12 | $\underline{4}$ | 12 | $\underline{3}$ | $\underline{3}$ | $\underline{0}$ | 9 | 0 | →3 |
| **5** | 3 | 0 | 8 | 7 | 8 | 6 | 12 | 6 | |
| $I1_j$ | {} | {} | {} | {2,4} | {4} | {4} | | | |
| $v_j$ | 4 | 4 | 9 | 6→3 | 5 | 4 | | | |

Tabelle 12: Dual-Adjustment-Methode - Veränderung bei Kunde 4

dass dreimal die Dual-Ascent-Methode ausgeführt wird. Im ersten Durchgang werden dabei nur die Kunden $J_{i_1}^{+} \cup J_{i_2}^{+}$ betrachtet. Im zweiten Durchgang wird die Kundenmenge des ersten Durchgang um den Kunden $j$ erweitert und im dritten Durchgang werden wieder alle Kunden betrachtet.

Die Veränderungen der Dual-Ascent-Methode im ersten Durchgang sind in der Tabelle

| i \ j | **1** | **2** | **3** | **5** | **6** | $f_i$ | $s_i$ | $s_i$ |
|---|---|---|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 9 | 12 | 12 | 1 | →0 |
| **2** | 4 | 6 | 9 | 12 | 8 | 6 | 3 | →2 |
| **3** | 6 | 3 | 10 | 0 | 4 | 9 | 3 | →0 |
| **4** | 12 | 4 | 12 | 3 | 0 | 9 | 3 | →0 |
| **5** | 3 | 0 | 8 | 8 | 6 | 12 | 6 | →5 |
| $v_j$ | 4→5 | 4 | 9 | 5→8 | 4 | | | |

Tabelle 13: Dual-Adjustment-Methode - 1. Dual-Ascent-Methode

13 dargestellt. Im zweiten Durchgang wird nun der Kunde 4 wieder in die Optimierung einbezogen.

Hier ergibt sich allerdings keine Veränderung mehr. Das Ergebnis wird in Tabelle 14 dargestellt. Es werden in unserem Beispiel bereits beim zweiten Durchgang alle Kunden betrachtet, damit muss der dritte Durchgang nicht mehr explizit durchgeführt werden.
Nachdem die Dual-Ascent-Methode dreimal durchgeführt wurde wird geprüft, ob

| i \ j | **1** | **2** | **3** | **4** | **5** | **6** | $f_i$ | $s_i$ |
|---|---|---|---|---|---|---|---|---|
| **1** | $\underline{0}$ | $\underline{2}$ | $\underline{4}$ | 6 | 9 | 12 | 12 | 0 |
| **2** | $\underline{4}$ | 6 | $\underline{9}$ | $\underline{0}$ | 12 | 8 | 6 | 2 |
| **3** | 6 | $\underline{3}$ | 10 | 12 | $\underline{0}$ | $\underline{4}$ | 9 | 0 |
| **4** | 12 | $\underline{4}$ | 12 | $\underline{3}$ | $\underline{3}$ | $\underline{0}$ | 9 | 0 |
| **5** | $\underline{3}$ | $\underline{0}$ | $\underline{8}$ | 7 | 8 | 6 | 12 | 5 |
| $v_j$ | 5 | 4 | 9 | 3 | 8 | 4 | | |

Tabelle 14: Dual-Adjustment-Methode - Ergebnis

$v_{merk} = v_j$ des aktuellen Kunden gilt. Wenn diese Bedingung zutrifft, wird der nächste Kunde betrachtet. Im Beispiel ist dieses nicht der Fall da $v_{merk} = 6 \neq v_4 = 3$, daher wird $v_{merk}$ wieder mit dem aktuellen $v_j$ belegt und die drei Durchgänge der Dual-Ascent-Methode starten erneut. In unserem Beispiel ergibt sich aber beim erneuten Durchlauf keine weitere Änderung damit gilt nun $v_{merk} = 3 = v_4$ und wir fahren mit dem nächsten Kunden fort. Für die Kunden 5 und 6 gilt nun wieder $|I1_j| \leq 1$, damit haben wir alle Kunden durchlaufen und die Dual-Adjustment-Methode endet.
Wir erhalten eine duale Lösung von $ZD = 33$ und bestimmen dazu, wie in Kap. 5.2.2 erläutert, eine ganzzahlige Lösung. Die hierfür nötigen Informationen finden wir in Tabelle 14, wo die Elemente $\leq v_j$ mit $s_i = 0$ unterstrichen sind. Daraus ergibt sich folgende primale Lösung:

$$I1 = \{1, 4\}$$

$$y_1 = y_4 = 1$$

$$x_{11} = x_{12} = x_{13} = x_{44} = x_{45} = x_{46} = 1$$

$$Z = 33 = ZD$$

Da unsere primale Lösung mit der dualen Lösung übereinstimmt, haben wir die optimale Lösung für unser Problem gefunden. In unserem Beispiel sind wir ohne Verzweigung des Problems ausgekommen, damit ist auch das gesamte Beispiel optimal gelöst. Darüber hinaus stellen wir fest, dass wir bereits beim ADD-Algorithmus die optimale Lösung für unser Beispiel bestimmen konnten, davon ist im Regelfall allerdings nicht auszugehen.

### 5.2.4 Verzweigung

Hätten wir auch nach Anwendung der Dual-Adjustment-Methode keine optimale Lösung bestimmen können, hätten wir unser Problem verzweigen müssen. Dafür bilden wir zwei Teilprobleme, in dem eine unserer Binärvariablen $y_i$ in einem Teilproblem zu null und in dem anderen Teilproblem zu eins fixiert wird.
Für die Auswahl des $y_i$, welches wir fixieren, betrachten wir unsere gefundene Lösung und wählen einen Kunden, für den gilt $|I1_j| > 1$. Aus dieser Menge selektieren wir den Standort mit $min\{c_{ij}\}$ und $i \in I1_j$.
Die Fixierung von $y_i$ zu 0 wird dadurch erreicht, dass die Fixkosten $f_i$ mit $\infty$ belegt werden. Für die Fixierung von $y_i$ zu 1 wird $f_i$ auf 0 gesetzt. Anschließend löst Erlenkotter die Probleme nach dem Last-In-First-Out-Prinzip, es wird also zuerst das Teilproblem gelöst, welches als letztes hinzugefügt bzw. verzweigt wurde.

# 6 Literaturverzeichnis


- Domschke / Drexl: Logistik: Standorte, Oldenbourg 1990 (3. Auflage), ISBN 3-486-21527-2
- P.B. Mirchandani: Discrete Location Theory, New York 1990, ISBN 0-471-89233-5