|  |
| --- |
| HTWG Konstanz |
| Innerbetriebliche Standortplanung 1.1 |
| Anwendung der linearen Optimierung |

|  |
| --- |
| Jenny Schwarz 288236  Sarah Amann 288182  30.06.2015 |

Inhaltsverzeichnis

[Einleitung 3](#_Toc423439593)

[Klassen 4](#_Toc423439594)

[Berechnung starten 5](#_Toc423439595)

[Initialisierung – Init() 6](#_Toc423439596)

[Daten einlesen – EingabeDaten() 7](#_Toc423439597)

[Zuordnungsmatrix berechnen – BerechneMatrix() 8](#_Toc423439598)

[Lösungsverfahren 10](#_Toc423439599)

[Enumeration – RechneEnumeration() 10](#_Toc423439600)

[Entscheidungsbaumverfahren – RechneEntscheidungsbaum() 16](#_Toc423439601)

[Abbildungsverzeichnis 34](#_Toc423439602)

# Einleitung

Da die Methode keinen Solver benutzt, erfolgt die Berechnung innerhalb der vorhandenen Klassen. Um die Berechnung eventuell zu einem späteren Zeitpunkt durch das Einbinden eines Solvers abzulösen, wird im Folgenden die Berechnung einmal genauer untersucht.

# Klassen

Die Methode besteht aus folgenden Klassen:

* Anzeige.pas
* Berechnung.pas
* Einstellungen.pas
* Entscheidungsbaum.pas
* Enumeration.pas
* Feld.pas
* Initform.pas
* Matrix.pas
* Standort.pas
* Status.pas
* Transportmodell.pas

# Berechnung starten

Die Methode „Innerbetriebliche Standortplanung“ startet mit dem Aufruf der Klasse *Standort.pas*. In dieser Klasse wird die Methode Berechnen(N) aufgerufen. Dort werden die Methoden der Klasse *Berechnung.pas* in folgender Reihenfolge in der Startklasse *Standort.pas* aufgerufen:

* Init(N) (siehe Zeile 283, Abbildung 1)
* EingabeDaten(Standorte, Maschinen) (siehe Zeile 284, Abbildung 1)
* BerechneMatrix() (siehe Zeile 285, Abbildung 1)
* RechneEnumeration() (siehe Zeile 292, Abbildung 1)

oder

* RechneEntscheidungsbaum() (siehe Zeile 294, Abbildung 1).

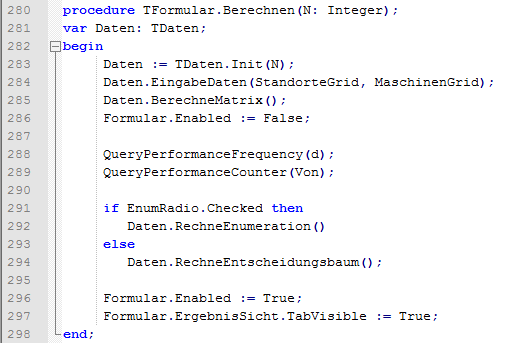


Abbildung 1: Berechnung starten in Startklasse Standort.pas

# Initialisierung – Init()

Als erster Schritt der Berechnung wird in der Klasse *Berechnung.pas* die Methode Init zur Initialisierung der Werte aufgerufen. Der Methode wurde die Tabellengröße der Eingabetabellen übergeben. Diese Größe wird an die Variable AnzOrte übergeben. AnzPaare wird in Zeile 34 (Abbildung 2) aus den AnzOrte berechnet. Diese Anzahl an Paaren werden als Übergabeparameter der Methode Dim(Size) aus der Klasse *Feld.pas* zur Erzeugung der eindimensionalen Felder StandortPaare und MaschinenPaare übergeben (siehe Zeile 35 - 36, Abbildung 2). Die zweidimensionalen Felder ZMatrix und UrMatrix werden in der Methode Dim(xSize, ySize) der Klasse *Matrix.pas* mit der Feldlängen AnzPaare gebildet (siehe Zeile 37 – 38, Abbildung 2).

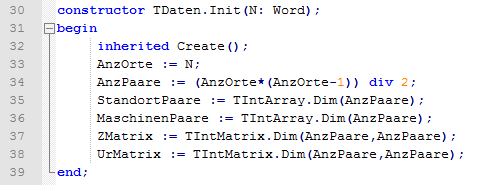


Abbildung 2: Initialisierung der Berechnungsklasse

# Daten einlesen – EingabeDaten()

In der Methode EingabeDaten() der Klasse *Berechnung.pas* werden die Daten aus der Eingabemaske eingelesen. Mit den Variablen Standorte und Maschinen werden in Zeile 49 (Abbildung 3) die zwei Eingabetabellen der Methode übergeben.

Innerhalb der Methode wird zweimal über die Variable AnzOrte mit einer Schleife iteriert, um die Zellen der Eingabetabellen auszulesen (siehe Zeile 54 – 55, Abbildung 3) und in das Feld StandortPaare mit dem Index z zu schreiben, welcher bei jedem Schleifendurchlauf hochgezählt wird (siehe Zeile 59, Abbildung 3).

Das Selbe wird auch für die Variable MaschinenPaare gemacht. Jedoch wird dabei die Transportmenge zwischen beispielsweise Maschine 1-2 und 2-1 (i=1 und j=2) aus der Eingabetabelle ausgelesen und addiert (siehe Zeile 67 und 68, Abbildung 3). Die Summe wird in das Feld MaschinenPaare gesichert (siehe Zeile 69, Abbildung 3).

Am Ende der Methode sind zwei Felder, einmal StandortPaare und einmal MaschinenPaare, mit den Werten aus der Eingabe entstanden.

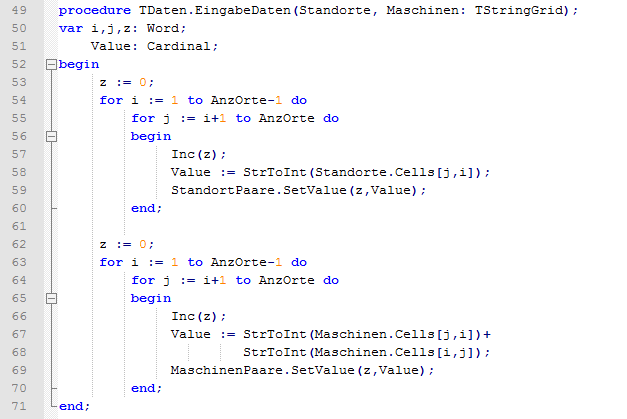


Abbildung 3: Eingabedaten in interne Felder einlesen

# Zuordnungsmatrix berechnen – BerechneMatrix()

Die Berechnung der Lösungsmatrix, der sogenannten Zuordnungsmatrix, erfolgt ebenfalls in der Klasse *Berechnung.pas.*

Die Zuordnungsmatrix gibt die einzelnen Transportaufwände (TAW) für Maschinen- und Standortpaare aus. In Feld 1 wird zum Beispiel der Transportaufwand Maschine 1 und 2 und Standort A und B ausgegeben. Berechnet wird der Transportaufwand wie folgt:

*( Transportmenge(1-2) + Transportmenge(2-1) ) \* Entfernung(A-B) = TAW*

Die Berechnung innerhalb der Methode erfolgt mit dem gleichen Prinzip, beginnend in der Methode BerechneMatrix() in Zeile 76 (Abbildung 4). Zunächst wird über das vorhandene Feld StandortPaare eine Schleife gelegt, und der Variablen ValOne den Wert des Feldinhaltes übergeben (siehe Zeile 79). In einer zweiten Schleife der gleichen Länge AnzPaare wird einer Variablen ValTwo den Wert der Zelle aus dem Feld MaschinenPaare übergeben (siehe Zeile 82, Abbildung 4). In Zeile 83 (Abbildung 4) wird in eine neue Variable Value das Produkt der Variablen ValOne und ValTwo übergeben. Dieser Wert entspricht dem zuvor beschrieben Transportaufwand. Der Wert Value wird abschließend mit der Methode SetValue(xIndex, yIndex, Val) der Klasse *Matrix.pas* in die richtige Zelle der Zuordnungsmatrix ZMatrix sowie in die UrMatrix geschrieben (siehe Zeile 84 – 85, Abbildung 4).

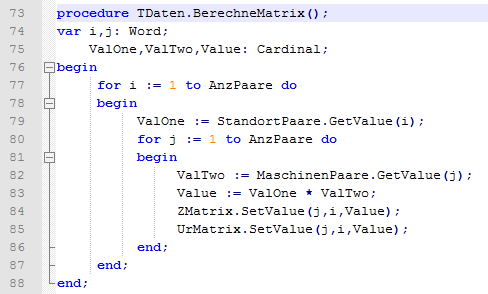


Abbildung 4: Berechnung Zuordnungsmatrix

# Lösungsverfahren

Innerhalb der Methode gibt es die Optionen zur Berechnung der Lösung mittels des Entscheidungsbaumverfahrens oder durch Enumeration.

## Enumeration – RechneEnumeration()

In der Methode RechneEnumeration() innerhalb der Klasse *Berechnung.pas* wird in Zeile 93 (Abbildung 5) die Methode Create(Matrix, Orte) der Klasse *Enumeration.pas* aufgerufen.

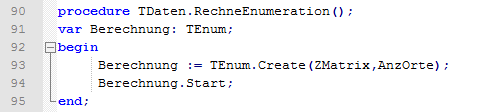


Abbildung 5: Berechnung durch Enumeration

Innerhalb der Methode Create(Matrix, Orte) wird durch den Aufruf von Create(True) in Zeile 48 (Abbildung 6) als aller erstes die Hauptmethode Execute aufgerufen.

Nach dem Aufruf der Hauptmethode werden verschiedene Variablen gesetzt. Darunter auch die AnzBerechnungen, welche durch die Fakultät der übergebenen Anzahl der Orte berechnet wird (siehe Zeile 54, Abbildung 6). In Zeile 59 bis 62 (Abbildung 6) wird die aus der Berechnung entstehende Zuordnungsmatrix ZMatrix kopiert.

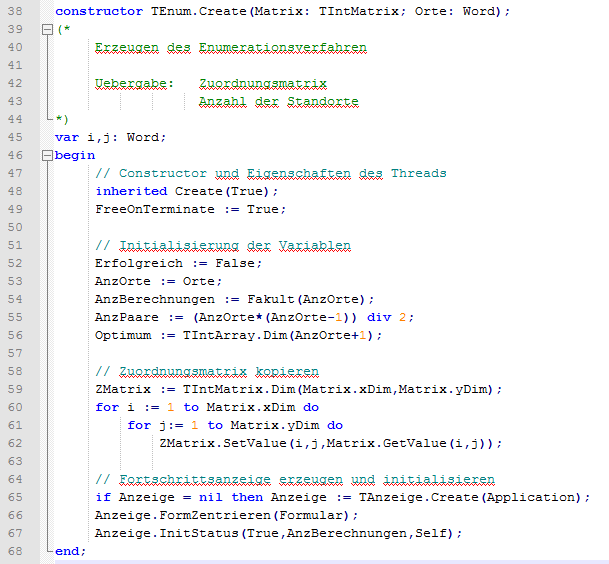


Abbildung 6: Klasse Enumeration.pas mit Methode Create()

In der Methode Execute() werden in Zeile 84 bis 88 (Abbildung 7) zuerst die Variablen initialisiert. In Zeile 91 (Abbildung 7) wird das optimale Ergebnis auf den höchstmöglichen Wert gesetzt. Danach wird die Anzeige des Fortschritts synchronisiert (siehe Zeile 94, Abbildung 7).

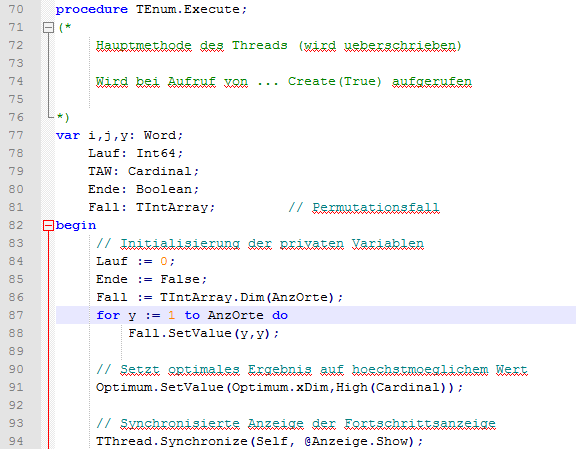


Abbildung 7: Methode Execute() in Klasse Enumeration.pas

In der Schleife von Zeile 97 bis 139 (Abbildung 8) werden alle Permutationen der Zuordnungsfälle so lange berechnet bis alle Möglichkeiten berechnet wurden und die Schleife mit einem break in Zeile 123 (Abbildung 8) beendet wird. Bei der Berechnung wird zunächst der Transportaufwand mit dem aktuellen Optimum verglichen und ggf. anschließend das Optimum neu gesetzt (siehe Zeile 102 - 107, Abbildung 8). Der Transportaufwand wird zuvor mit der Methode Rechne(Feld) berechnet (siehe Abbildung 9).

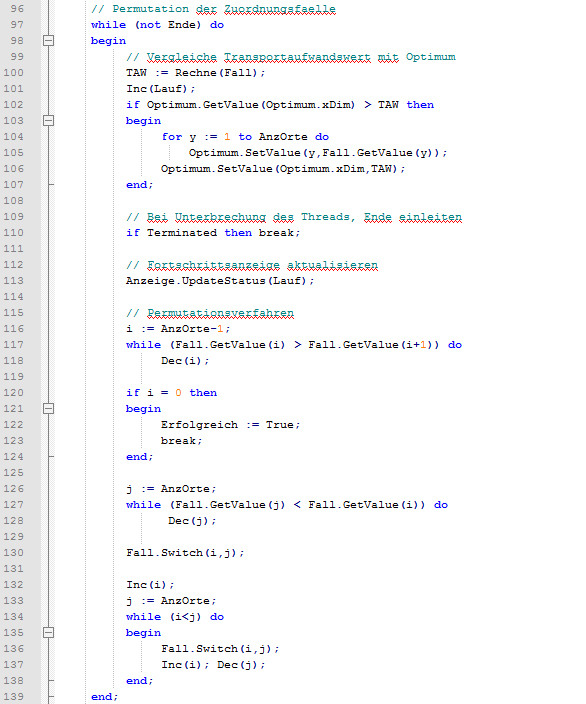


Abbildung 8: Schleife "Permutation der Zuordnungsfälle"

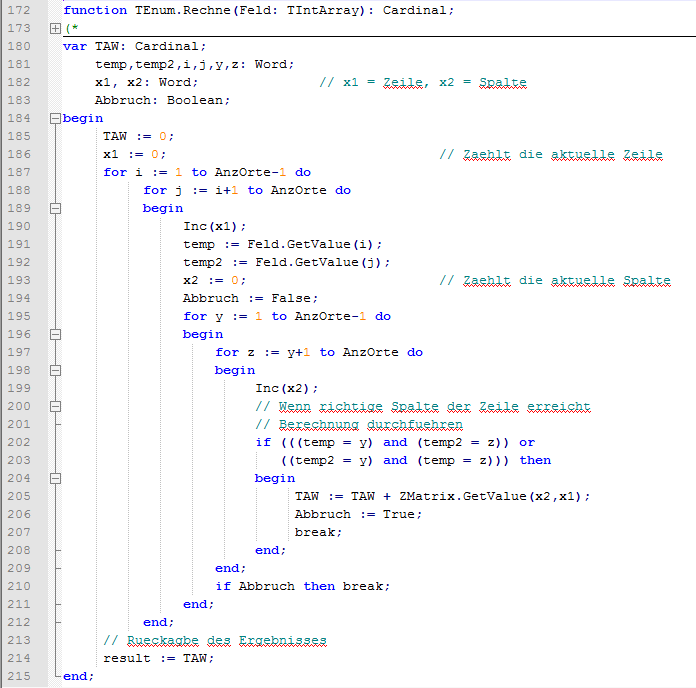


Abbildung 9: Methode Rechne() in Klasse Enumeration.pas

Sofern die Berechnung erfolgreich war, wird anschließend an die Schleife Zeile 159 bis 167 (Abbildung 10) ausgeführt und das Ergebnis ausgegeben. Bei einem Fehler in der Schleife wird eine Fehlermeldung ausgegeben (siehe Zeile 168, Abbildung 10).

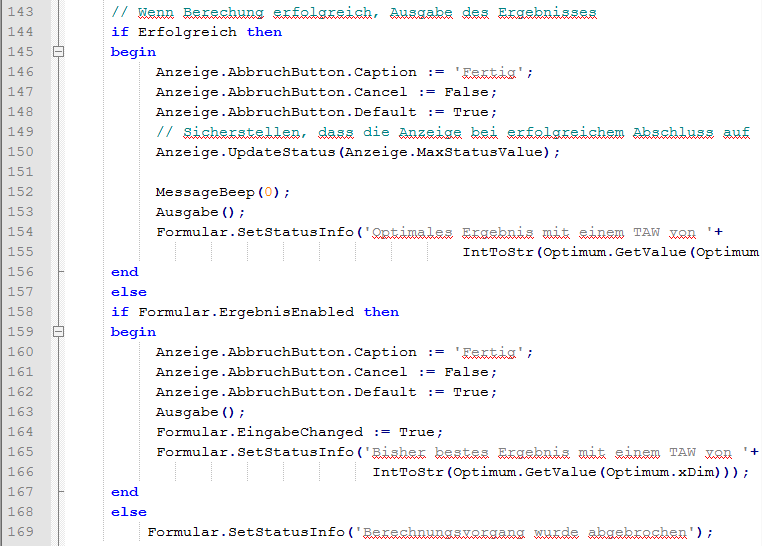


Abbildung 10: Ausgabe in Methode Execute()

## Entscheidungsbaumverfahren – RechneEntscheidungsbaum()

In der Methode RechneEntscheidungsbaum() in der Klasse *Berechnung.pas* wird zunächst überprüft ob innerhalb der Zuordnungsmatrix jede Zelle mit einem Wert ungleich 0 gesetzt ist (siehe Zeile 102 – 116, Abbildung 11). Ist dies nicht der Fall wird ein Zähler Counter hochgezählt. Ist der Counter größer als das Produkt der Zeilenanzahl minus einmal die Zeilenanzahl wird eine Fehlermeldung ausgegeben. Bei 4 Standorten und 4 Maschinen muss der Counter größer 30 sein. In diesem Fall wird eine Fehlermeldung ausgegeben.

In Zeile 118 (Abbildung 11) wird die Klasse *Entscheidungsbaum.pas* mit der Methode Create(Matrix, Orte) aufgerufen.

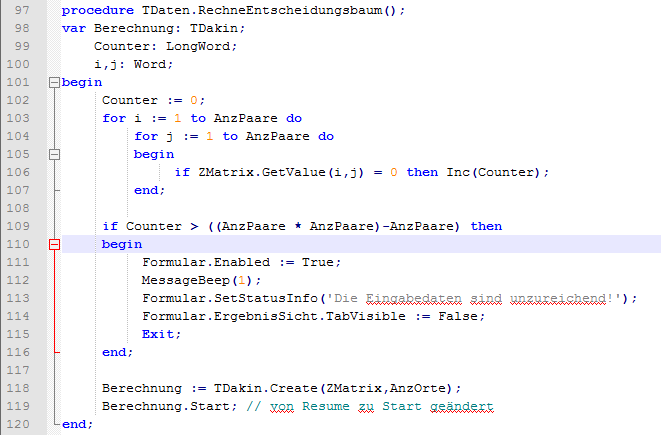


Abbildung 11: Aufruf des Entscheidungsbaumverfahrens

Innerhalb der Methode Create(Matrix, Orte) wird in Zeile 36 (Abbildung 12) als erstes die Methode Create(True) aufgerufen, welches die Hauptmethode Execute() aufruft. Anschließend werden verschiedene Variablen gesetzt. In Zeile 44 bis 51 (Abbildung 12) werden die Zuordnungsmatrix ZMatrix und eine MMatrix ebenfalls berechnet.

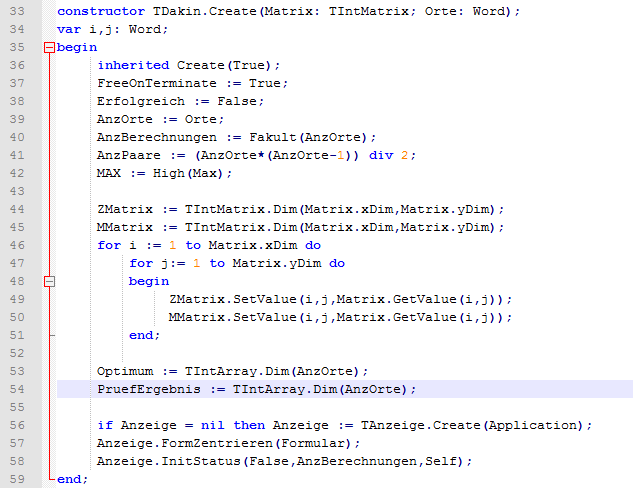


Abbildung 12: Berechnung Entscheidungsbaumverfahren

In der Execute()-Methode der Klasse *Entscheidungsbaum.pas* wird in Zeile 69 (Abbildung 13) zunächst das Modell durch den Aufruf der Create()-Methode der Klasse *Transportmodell.pas* initialisiert. Anschließend wird auf dieses Modell die Rechnen()-Methode der Transportmodell-Klasse aufgerufen und das Optimum berechnet (siehe Abbildung 14).

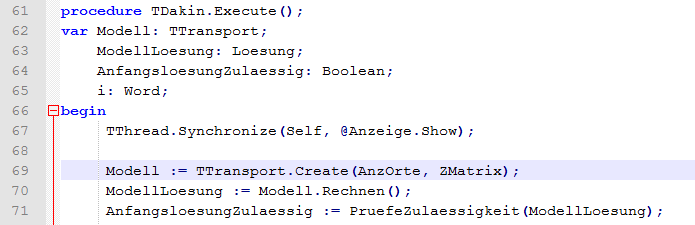


Abbildung 13: Execute()-Methode in Klasse Entscheidungsbaum.pas

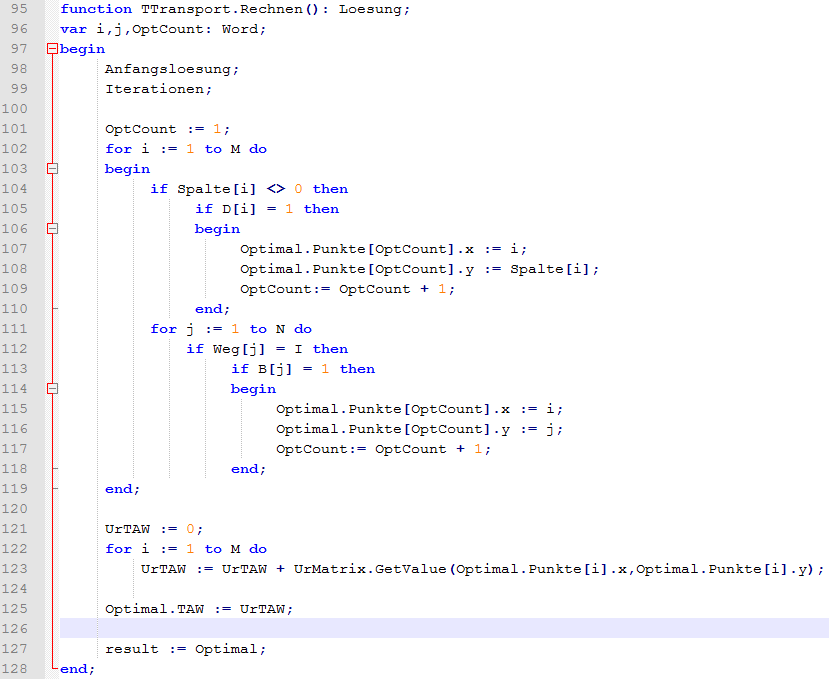


Abbildung 14: Methode Rechne() in Klasse Transportmodell.pas

Ist die Lösung der ersten Berechnung zulässig, wird dieses im Optimum gespeichert und das Ergebnis ausgegeben (siehe Zeile 73 bis 79, Abbildung 15). Andernfalls wird in Zeile 92 (Abbildung 15) der Entscheidungsbaum durchlaufen und die Methode RechneDakin() aufgerufen. Wurde nach dem Aufruf der Methode das Optimum gefunden wird es nochmal geprüft und anschließend in die Variable Optimum gespeichert (siehe Zeile 95 – 97, Abbildung 15). Abschließend wird es angezeigt (siehe Abbildung 16).

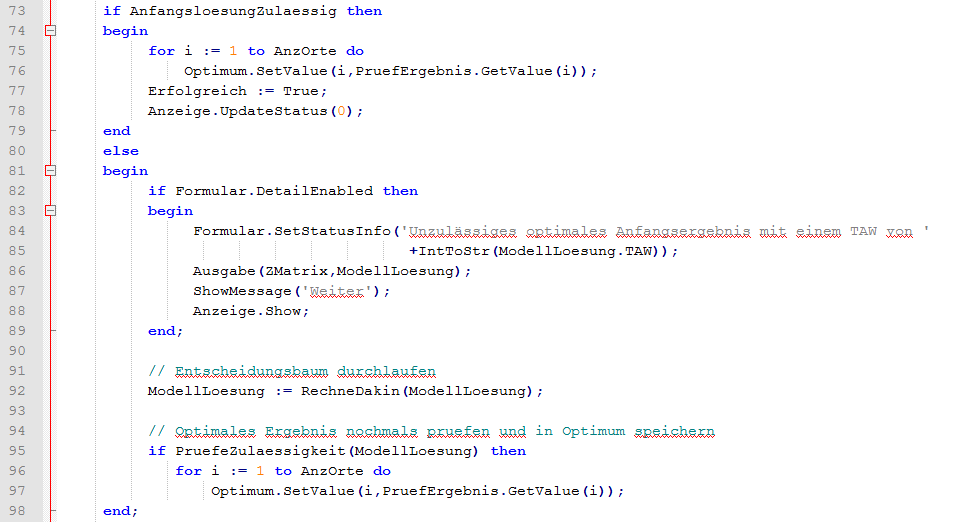


Abbildung 15: Teil 2 Methode Execute() in Klasse Entscheidungsbaum.pas

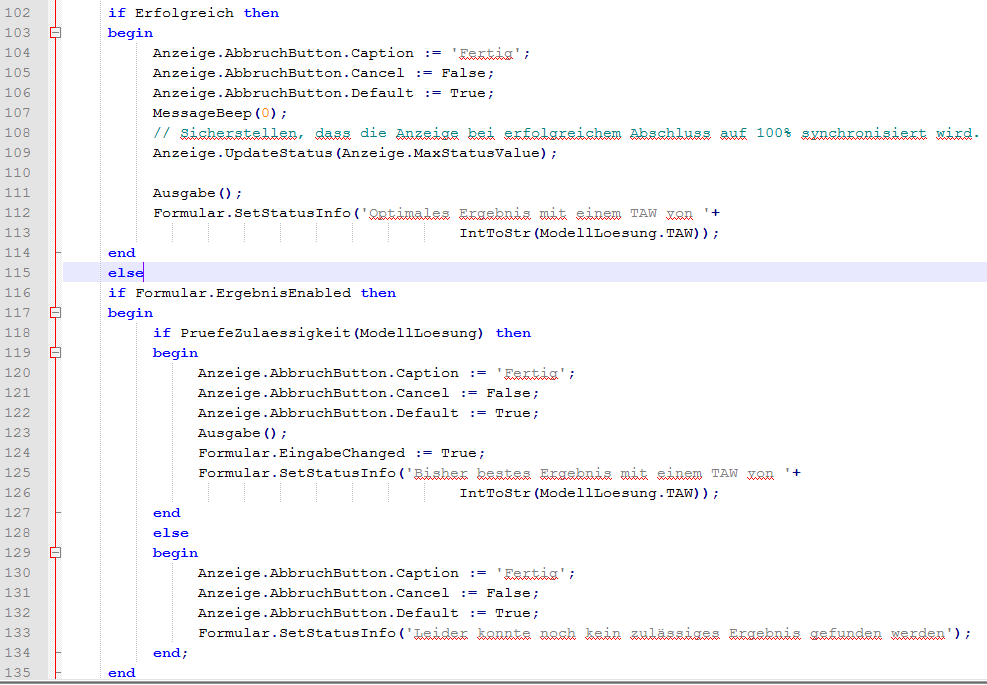


Abbildung 16: Optimum durch Entscheidungsbaumverfahren ausgeben

In dieser Methode RechneDakin() wird solange durch eine Schleife iteriert bis das optimale Ergebnis gefunden wurde. Die Methode besitzt mehr als 250 Zeilen Code deshalb werden im Folgenden nur Teile davon gezeigt.

Zunächst werden die optimale Lösung und die aktuelle Lösung initialisiert (siehe Abbildung 17).

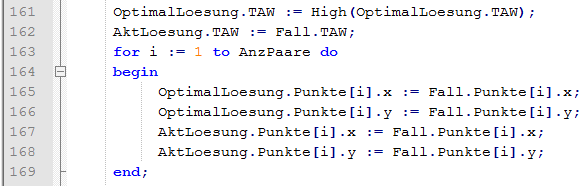


Abbildung 17: Initialisierung optimale Lösung in Methode RechneDakin()

Anschließend wird in Zeile 177 bis 191 (Abbildung 18) der Entscheidungsindex ermittelt.

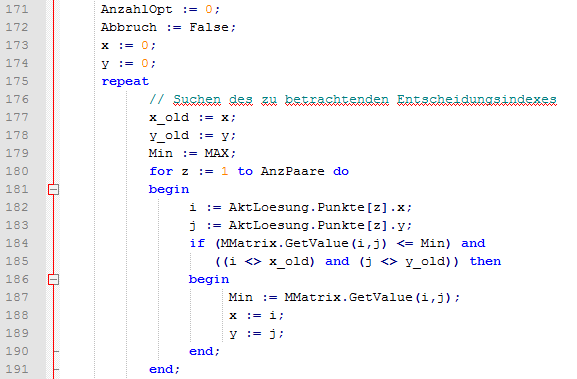


Abbildung 18: Entscheidungsindex suchen

Es gibt innerhalb der Methode RechneDakin() zwei Varianten das Ergebnis zu berechnen. Zum einen ohne Entscheidungsindex (siehe Abbildung 19) und zum anderen mit Entscheidungsindex (siehe Abbildung 20).

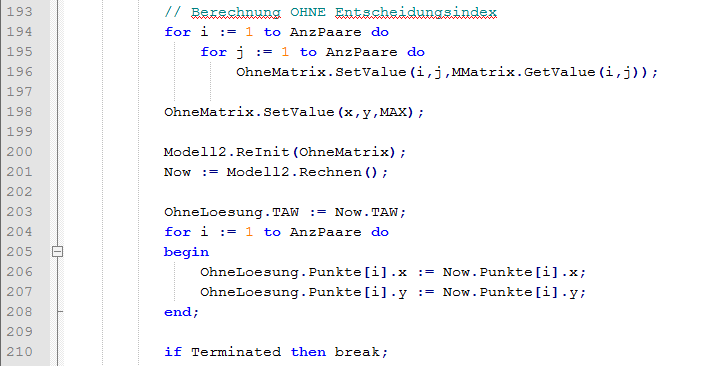


Abbildung 19: Entscheidungsbaum ohne Index

Bei der Berechnung mit einem Entscheidungsindex wird in Zeile 218 (Abbildung 20) die Methode RechneMitModell() aufgerufen.

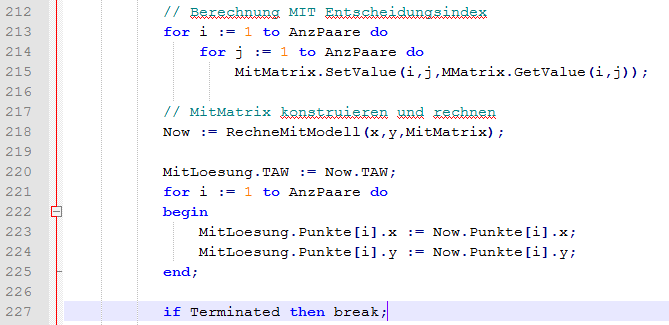


Abbildung 20: Entscheidungsbaum mit Index

In dieser Methode werden folgende Schritte in dieser Reihenfolge abgearbeitet:

1. Beide initialisierten Matrizen mit übergebenen Werten füllen (Abbildung 21)
2. Die Spalten und Zeilen des Indexes maximieren (Abbildung 22)
3. Den Index und den gespiegelten Index auf aktuellen Wert setzen (Abbildung 23)
4. Maschinen und Standorte identifizieren (Abbildung 24)
5. Fall 1 – Maschinenkombination: unmögliche Werte maximieren (Abbildung 25)
6. Fall 2 – Maschinenkombination: unmögliche Werte maximieren (Abbildung 26)
7. Fall 1 – Transportmodell: Rechnen und Speichern (Abbildung 27)
8. Fall 2 – Transportmodell: Rechnen und Speichern (Abbildung 28)
9. Den Fall, der beiden Fälle mit Index, mit geringerem TAW wählen (Abbildung 29)
10. Rückgabe der Lösung mit dem geringeren TAW (Zeile 556, Abbildung 29)

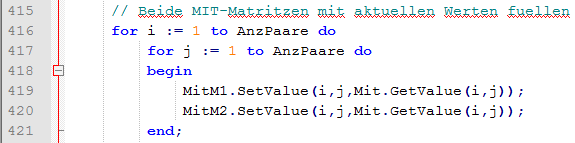


Abbildung 21: 1. Matrizen mit aktuellen Werten füllen

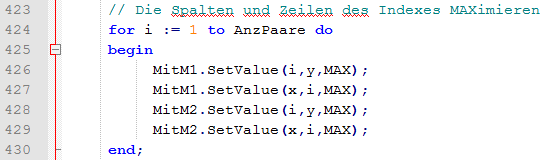


Abbildung 22: 2. Spalten und Zeilen maximieren



Abbildung 23: 3. Index und gespiegelter Index auf aktuellen Wert setzen

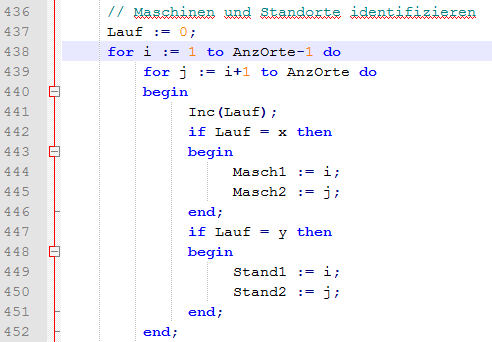


Abbildung 24: 4. Maschinen und Standorte identifizieren

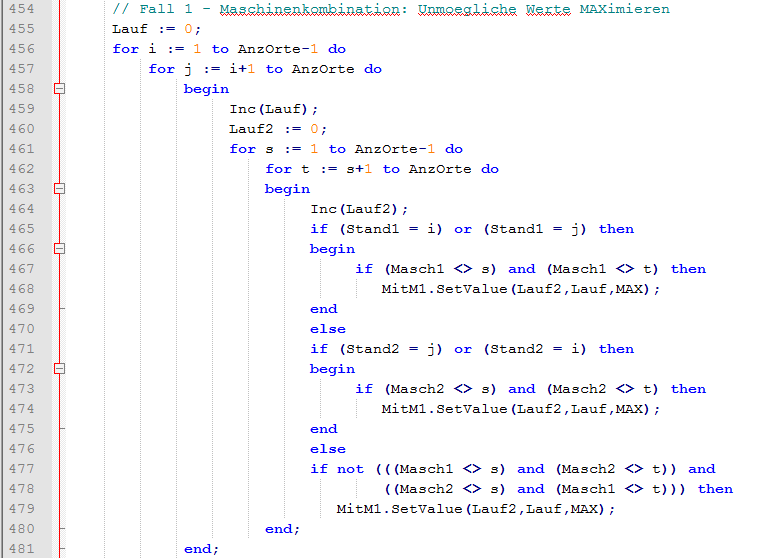


Abbildung 25: 5. Fall 1: Maschinenkombination

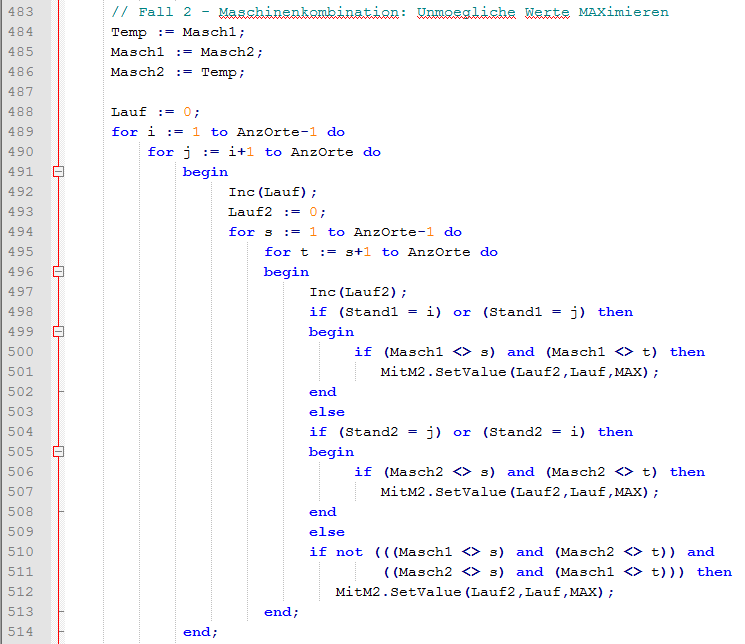


Abbildung 26: 6. Fall 2: Maschinenkombination

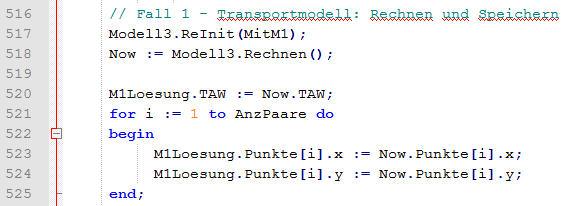


Abbildung : 7. Fall 1: Transportmodell

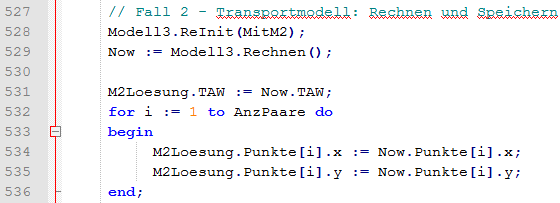


Abbildung 28: 8. Fall 2: Transportmodell

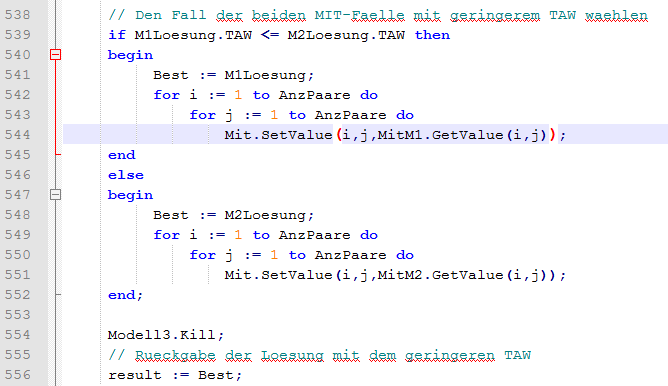


Abbildung 29: 9. & 10.: Wahl des Falles mit geringerem TAW

Nachdem die Berechnungen durchgeführt wurden, werden die beiden Lösungen auf Zulässigkeit geprüft (siehe Zeile 230 – 231, Abbildung 30).

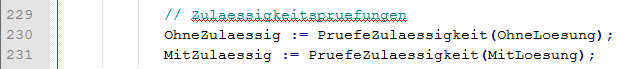


Abbildung 30: Zulässigkeitsprüfung in Methode RechneDakin()

Anschließend wird überprüft ob der TAW der Rechnung mit Index größer ist als der TAW der Berechnung ohne Index (siehe Zeile 237, Abbildung 31). Ist dies der Fall wird die neue optimale Lösung auf den TAW der Berechnung ohne Index gesetzt (siehe Zeile 243 – 246, Abbildung 31) und die neue Matrix wird gebildet (siehe Zeile 250, Abbildung 31).

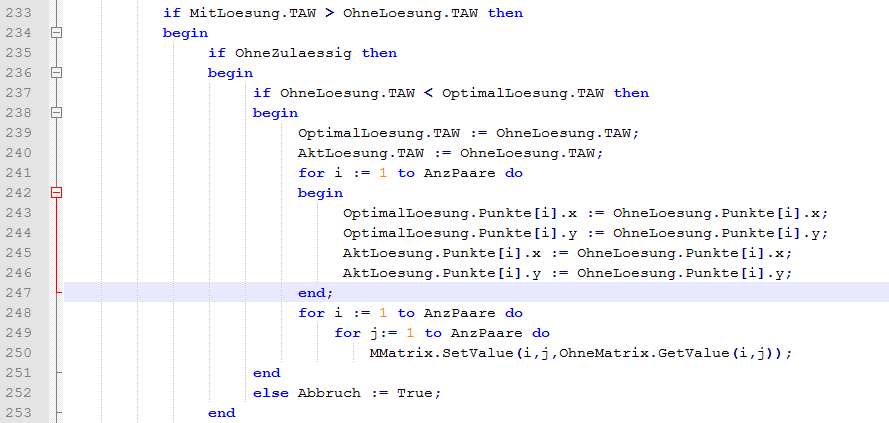


Abbildung 31: MitLösung.TAW > OhneLösung.TAW

Ist die Lösung der Berechnung ohne Index jedoch nicht zulässig wird das Optimum mit dem TAW der Berechnung mit Index gesetzt (siehe Zeile 265 – 266, Abbildung 32).

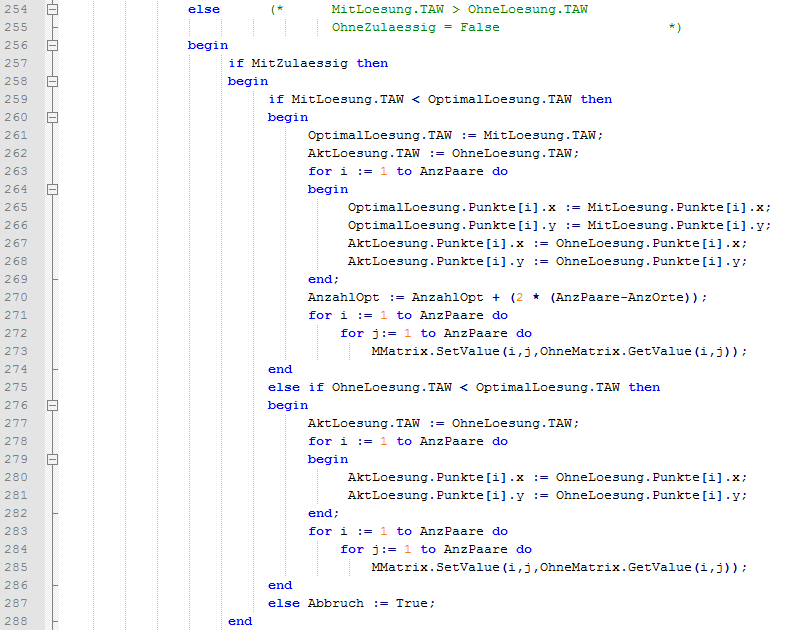


Abbildung 32: Ohne Index: TAW unzulässig

Sind beide Lösungen unzulässig wird kein Optimum gesetzt, sondern nur die aktuelle Lösung auf den TAW der Berechnung ohne Index. (siehe Abbildung 33)

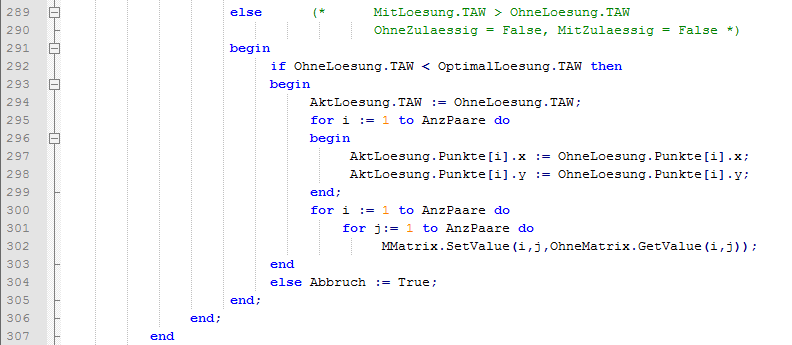


Abbildung 33: beide Lösungen unzulässig

Das gleiche wird für den umgekehrten Fall gemacht. Das heißt wenn der TAW der Berechnung mit Index kleiner ist als der TAW der Berechnung ohne Index, werden die Schritte von Abbildung 31 bis 33 in umgekehrter Form durchgeführt.

Ist der TAW der Berechnung mit Index kleiner oder gleich dem TAW der Berechnung ohne Index wird dieser TAW als Optimum gesetzt (Siehe Abbildung 34).

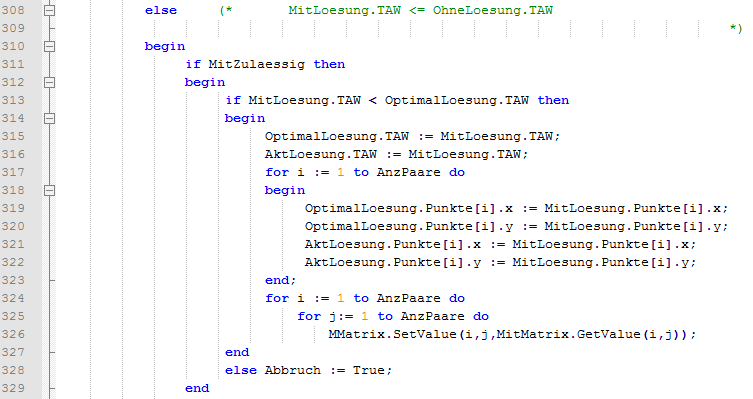


Abbildung 34: MitLösung.TAW < OhneLösung.TAW

Ist jedoch diese Lösung unzulässig wird der TAW der Berechnung ohne Index als Optimum gesetzt. (Siehe Abbildung 35)

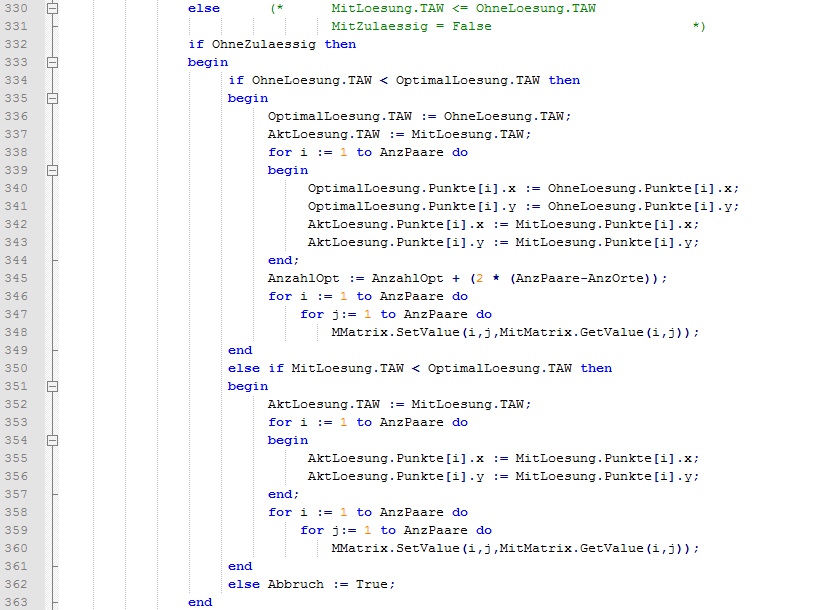


Abbildung 35: Mit Index: Lösung unzulässig

Sind beide Lösungen unzulässig wird der TAW der Lösung mit Index als aktuelle Lösung gesetzt. (siehe Abbildung 36)

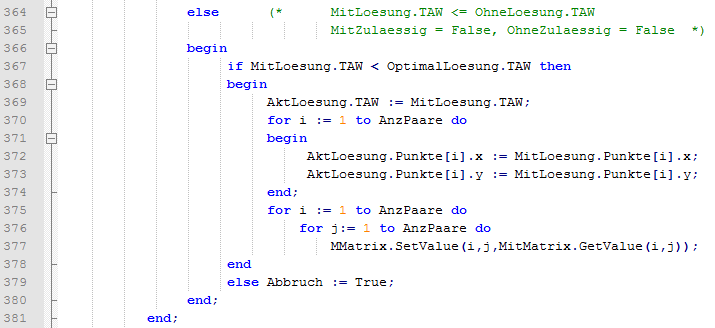


Abbildung 36: beide Lösungen unzulässig

Abschließend wird die Anzeige entsprechend der Lösung angepasst. Der ganze Vorgang wird jetzt solange wiederholt (durch repeat-Anweisung in Zeile 175 in Abbildung 18) bis alle Äste des Entscheidungsbaumes durchlaufen wurden (siehe Zeile 391, Abbildung 37) oder ein Fehler aufgetreten ist und die Variable Abbruch auf true gesetzt wurde.

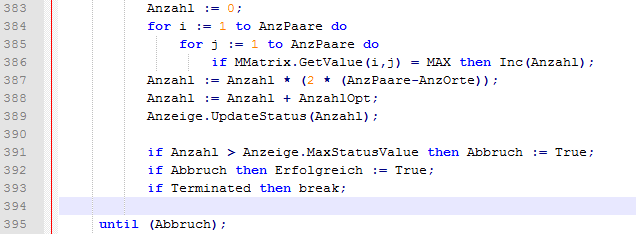


Abbildung 37: Abbruchbedingung

# Abbildungsverzeichnis

[Abbildung 1: Berechnung starten in Startklasse Standort.pas 5](#_Toc423439514)

[Abbildung 2: Initialisierung der Berechnungsklasse 6](#_Toc423439515)

[Abbildung 3: Eingabedaten in interne Felder einlesen 7](#_Toc423439516)

[Abbildung 4: Berechnung Zuordnungsmatrix 9](#_Toc423439517)

[Abbildung 5: Berechnung durch Enumeration 10](#_Toc423439518)

[Abbildung 6: Klasse Enumeration.pas mit Methode Create() 11](#_Toc423439519)

[Abbildung 7: Methode Execute() in Klasse Enumeration.pas 12](#_Toc423439520)

[Abbildung 8: Schleife "Permutation der Zuordnungsfälle" 13](#_Toc423439521)

[Abbildung 9: Methode Rechne() in Klasse Enumeration.pas 14](#_Toc423439522)

[Abbildung 10: Ausgabe in Methode Execute() 15](#_Toc423439523)

[Abbildung 11: Aufruf des Entscheidungsbaumverfahrens 16](#_Toc423439524)

[Abbildung 12: Berechnung Entscheidungsbaumverfahren 17](#_Toc423439525)

[Abbildung 13: Execute()-Methode in Klasse Entscheidungsbaum.pas 18](#_Toc423439526)

[Abbildung 14: Methode Rechne() in Klasse Transportmodell.pas 18](#_Toc423439527)

[Abbildung 15: Teil 2 Methode Execute() in Klasse Entscheidungsbaum.pas 19](#_Toc423439528)

[Abbildung 16: Optimum durch Entscheidungsbaumverfahren ausgeben 20](#_Toc423439529)

[Abbildung 17: Initialisierung optimale Lösung in Methode RechneDakin() 21](#_Toc423439530)

[Abbildung 18: Entscheidungsindex suchen 21](#_Toc423439531)

[Abbildung 19: Entscheidungsbaum ohne Index 22](#_Toc423439532)

[Abbildung 20: Entscheidungsbaum mit Index 23](#_Toc423439533)

[Abbildung 21: 1. Matrizen mit aktuellen Werten füllen 24](#_Toc423439534)

[Abbildung 22: 2. Spalten und Zeilen maximieren 24](#_Toc423439535)

[Abbildung 23: 3. Index und gespiegelter Index auf aktuellen Wert setzen 24](#_Toc423439536)

[Abbildung 24: 4. Maschinen und Standorte identifizieren 25](#_Toc423439537)

[Abbildung 25: 5. Fall 1: Maschinenkombination 25](#_Toc423439538)

[Abbildung 26: 6. Fall 2: Maschinenkombination 26](#_Toc423439539)

[Abbildung 27: 7. Fall 1: Transportmodell 26](#_Toc423439540)

[Abbildung 28: 8. Fall 2: Transportmodell 27](#_Toc423439541)

[Abbildung 29: 9. & 10.: Wahl des Falles mit geringerem TAW 27](#_Toc423439542)

[Abbildung 30: Zulässigkeitsprüfung in Methode RechneDakin() 28](#_Toc423439543)

[Abbildung 31: MitLösung.TAW > OhneLösung.TAW 28](#_Toc423439544)

[Abbildung 32: Ohne Index: TAW unzulässig 29](#_Toc423439545)

[Abbildung 33: beide Lösungen unzulässig 30](#_Toc423439546)

[Abbildung 34: MitLösung.TAW < OhneLösung.TAW 31](#_Toc423439547)

[Abbildung 35: Mit Index: Lösung unzulässig 32](#_Toc423439548)

[Abbildung 36: beide Lösungen unzulässig 33](#_Toc423439549)

[Abbildung 37: Abbruchbedingung 33](#_Toc423439550)