„HTWG Konstanz – Fakultät Informatik

2015-06-17 Softwaredokumentation Umladeproblem 4.1

Tuba Demirci, 288725, Wirtschaftsinformatik 6.Semester
Raissa Frolov, 286457, Wirtschaftsinformatik 6.Semester

Inhaltsverzeichnis

1 Abbildungsverzeichnis 2
2 Einführung 3
3 Ist-Analyse bei Übernahme (08.04.2015) 3
3.1 Übersicht 3
3.2 Fehleranalyse 5
4 Umsetzung 7
4.1 Einstellungsfunktion für Solver-Pfad 7
4.2 Ergebnisausgabe 10
4.3 Restriktionen 11
5 Weitere Verbesserungsmöglichkeiten 13

1 Abbildungsverzeichnis

Abbildung 1	Zahleneingabe
Abbildung 2	Modell Anlegen
Abbildung 3	Speichern des Modells
Abbildung 4	Restriktionenbildung
Abbildung 5	Code von Nachfrager-Restriktionen
Abbildung 6	Fenster der Ergebnisausgabe
Abbildung 7	Datei für die Änderung des Solver-Pfads
Abbildung 8	Code mit dem Dateipfad
Abbildung 9	neue Benutzeroberfläche
Abbildung 10	Deklaration und Erzeugung des neuen Buttons
Abbildung 11	Beschriftung des Buttons
Abbildung 12	Methode „function_Pfadeingeben()“
Abbildung 13	Textfeld
Abbildung 14	Fenster für den Solver-pfad
Abbildung 15	Deklaration der Fenster
Abbildung 16	Weiterentwciklung der Funktionen für „OK“ und Speichern des Solver-Pfads
Abbildung 17	Fuktion für Abbrechen
Abbildung 18	paths-Datei
Abbildung 19	Erstellung des INI-Dateis
Abbildung 20	Solver-Pfad
Abbildung 21	neues Fenster der Ergebnisausgabe
Abbildung 22	Zeichnung der Nachfrager-Restriktionen
Abbildung 23	Code der neuen Nachfrager-Restriktionen
Abbildung 24	neue Restriktionenausgabe

2 Einführung

Im Rahmen der Veranstaltung „Anwendung der Linearen Optimierung“ bei Herrn Prof. Dr. Grütz im Sommersemester 2015 bearbeiten und dokumentieren wir die Methode „Umladeproblem 4.0“.

Die ausgewählte Methode „Umladeproblem 4.0“ soll analysiert und mittels des genehmigten Pflichtenheftes umgesetzt werden.

Unsere Arbeit besteht aus vier Kapiteln. In dem Kapitel „Ist-Analyse“ werden wir die Methode beschreiben und analysieren. Bei dem nächsten Kapitel „Pflichtenheft“ werden wir unsere Ziele definieren und danach in „Umsetzung“ unsere Vorgehensweise erläutern. Schließlich folgt ein „Fazit“.

3 Ist-Analyse bei Übernahme (08.04.2015)

Vorerst geben wir einen Überblick über den Ist-Zustand, um danach eine genauere Fehleranalyse durchzuführen.

3.1 Übersicht

Bei der Methode „Umladeproblem“ geht es um ein Programm, das angewendet wird, wenn die Güter auf dem Weg zu ihrem Bedarfsort vom Anbieter an Umladeorten umgeschlagen werden müssen. Die Aufgabe des Programms ist das Umladeproblem in ein lineares Modell zu beschreiben, welches mit dem Simplexalgorithmus berechnet wird.

Das Programm lässt sich unter Windows 7 Systemen(64-Bit) starten. Zur Berechnung wird das Solver LP-Solver verwendet. Bei der Solverauswahl steht der MightyMightyLp zur Verfügung, der aber nicht funktioniert und deswegen nicht aktiviert ist. Unter Option sind die „Help-“ und „Info-“ Funktionen vorhanden, die keinen Inhalt haben.

Die drei Buttons auf der Benutzeroberfläche sind ausführbar. Wobei zu beachten ist, dass beim „Modell anlegen“ die Dezimalzahlen mit Punkt eingetragen werden müssen. Anderenfalls erscheint eine Fehlermeldung:

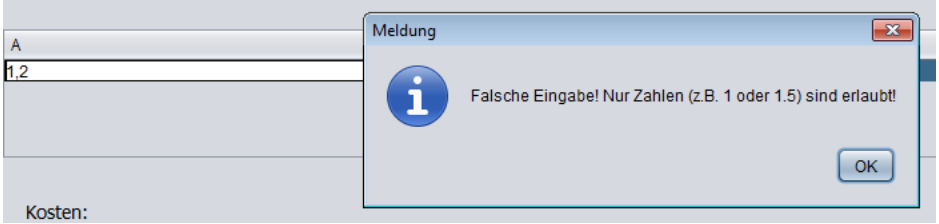

Abbildung 1: Zahleneingabe

Eine Bedingung ist, dass nach der Zahleneingabe auf die Spaltennamen geklickt werden muss, da sonst die Werte bei der Restriktionenbildung nicht übernommen werden.

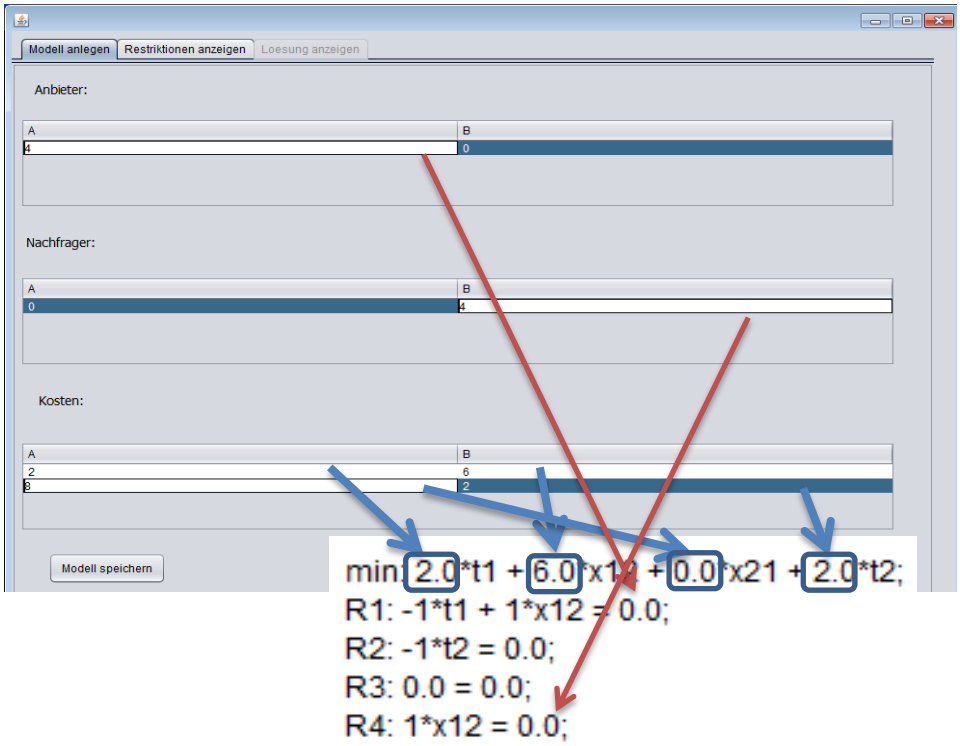

Abbildung 2: Modell Anlegen

Nach dem Anlegen des Modells ist das Speichern des Modells möglich. Dabei muss berücksichtigt werden, dass das Speichern mit „ .xml “ erfolgt.

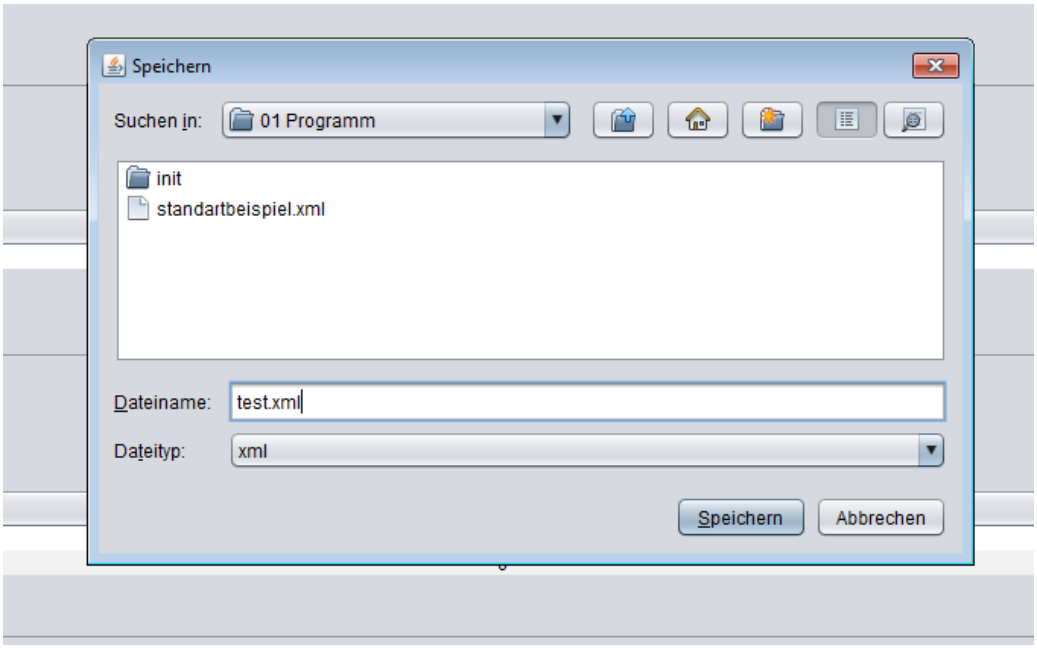

Abbildung 3: Speichern des Modells

3.2 Fehleranalyse

In diesem Abschnitt beschäftigen wir uns mit der Fehleranalyse, indem wir die Fehler in der Methode und in dem Source-Code ausfindig machen und sie analysieren.

Wenn die oben genannte Bedingung bei der Restriktionenbildung eingehalten wird, können die Restriktionen vollständig angezeigt werden. Wir konnten feststellen, dass die Restriktionen der Nachfrager nicht fehlerfrei sind. Bei der Bildung werden dieWerte der Kostenmatrix nicht richtig übernommen (siehe Screenshot).

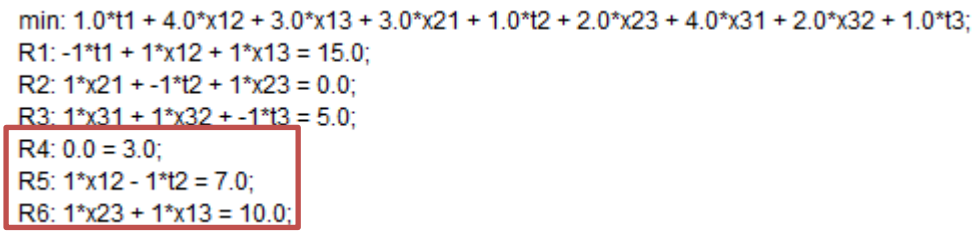

Abbildung 4: Restriktionenbildung

Bei der ersten Nachfragerestriktion werden die Werte der ersten Spalte der Kostenmatrix nicht durchlaufen. Bei den nächsten Restriktionen gibt der untere Code die Variablen z.B. wie 1x32 nicht aus. Das heißt das Programm gibt die eingehenden Kanten nicht aus.

```
//-------------------------------------------------------------------------------------
    NachfragerRestriktionen :
//-------------------------------------------------------------------------------------

if (row == column) {  // -> Knoten
    for (int n = (row - 1); n >= 0; n--) {
        if (kostenmatrix[n][column] > 0.0) {
            if(setColumnNachfrager == true){
                nachfragerString += " + ";
            }
            nachfragerString += "1*" + "x" + (n + 1) + (column + 1);
            setColumnNachfrager = true;
        }
    }
```

```
    if (row != kostenmatrix.length - 1) {
        if (setColumnNachfrager == true) {

            // Restriktion Nachfrager groesser Anbieter
            if(difference < 0.0){
                nachfragerString += " + 1*x" + (kostenmatrix.length + 1) + (column + 1);
            }//Restriktion Nachfrager groesser Anbieter ende

            nachfragerString += " - 1*" + "t" + (row + 1);

        } else {
            nachfragerString += "0.0";
        }
    }
}
```

Abbildung 5: Code von Nachfrager-Restriktionen

Ein wichtiges Problem ist, dass die Methode kein Ergebnis liefert.

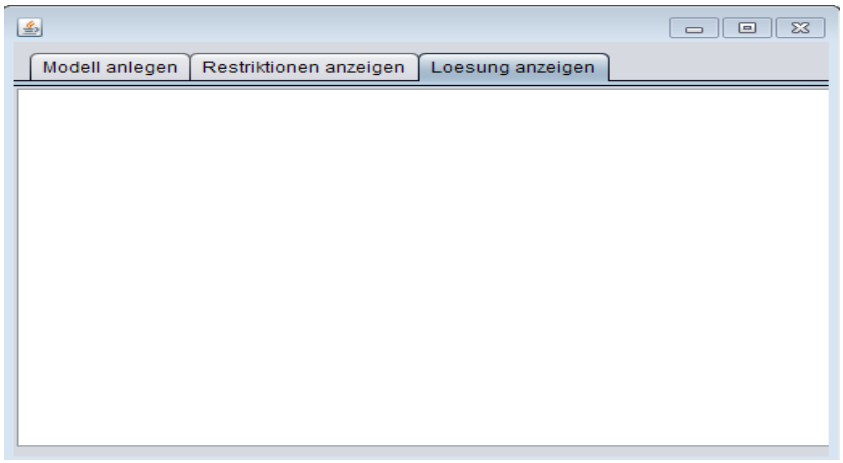

Abbildung 6: Fenster der Ergebnisausgabe

Die Methode kann kein Ergebnis ausgeben, da sie die Datei mit dem Solverpfad nicht findet. Ein weiterer Faktor, weshalb die Modelle nicht berechnet werden, ist der falsche Solverpfad im Datei „up.properties“.

```
#Beim Verschieben dieser Datei (up.properties) entsprechende Codestelle (solve.SolverPath.java) anpassen!

#Angabe des Solver Pfads:

lpSolvePath=C:\\or\\LocalOR\\Besf\\Solver\\LP_Solve\\Exec\\LP_SOLVE.EXE
```

Abbildung 7: Datei für die Änderung des Solver-Pfads

```
    prop.load(new FileReader("..\\Umladeplanung_v0.3.8\\up.properties"));
    solverPath = (String) prop.get("lpSolvePath");
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
```

Abbildung 8: Code mit dem Dateipfad

4 Umsetzung

4.1 Einstellungsfunktion für Solver-Pfad

Nach der erfolgreichen Fehleranalyse haben wir uns überlegt, ob wir die Einstellung für den Solverpfad bei der Option einfügen oder einen neuen Button bei der Benutzeroberfläche erstellen.

Wir haben uns für den Button entschieden, da es uns optisch schöner erschien (siehe Abbildung 9).

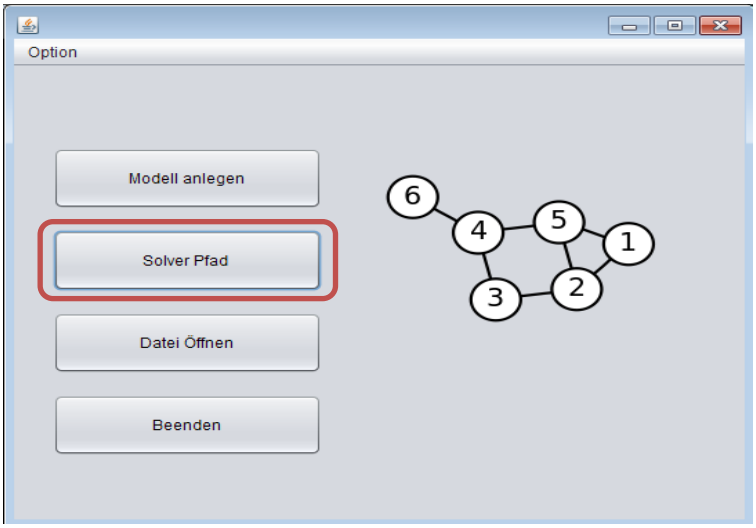

Abbildung 9: neue Benutzeroberfläche

Dafür haben wir im Source-Code in dem Package „gui“ die Klasse „Hauptmenue“ ergänzt. Zuerst haben wir einen neuen Button deklariert und erzeugt:

```
private javax.swing.JButton solverPfad;

solverPfad = new javax.swing.JButton();
```

Abbildung 10: Deklaration und Erzeugung des neuen Buttons

Später haben wir eine Methode erstellt, die die Methode „solverPfadActionPerformed()“ aufruft.

```
solverPfad.setText("Solver Pfad ");
solverPfad.addActionListener(new java.awt.event.ActionListener(){
   public void actionPerformed(java.awt.event.ActionEvent evt) {
        solverPfadActionPerformed(evt);
    }
});
```

Abbildung 11: Beschriftung des Buttons

Die Methode ruft dann die Methode „function_Pfadeingeben()“ auf,

```
protected void solverPfadActionPerformed(java.awt.event.ActionEvent evt) {
    function_Pfadeingeben();
}
```

Abbildung 12: Methode „function_Pfadeingeben()“

bei dem wir ein Objekt von der Klasse „SolverPfad()“ erstellen und mit „setVisible(true)“ sichtbar machen.

```
public void function_Pfadeingeben() {
    SolverPfad pfadEingabe = new SolverPfad();
    pfadEingabe.setVisible(true);
}
```

Abbildung 13: Textfeld

Unsere selbst geschriebene Klasse „SolverPfad“ im Package „lpSolver“ öffnet das Fenster für die Solver Pfad Einstellung. Hier haben wir ein Textfeld erstellt, indem ein Solver-Pfad eingegeben werden kann. Der eingegebene Pfad kann mit dem Button „OK“ gespeichert werden oder mit dem Button „Abbrechen“ kann der Vorgang abgebrochen werden.

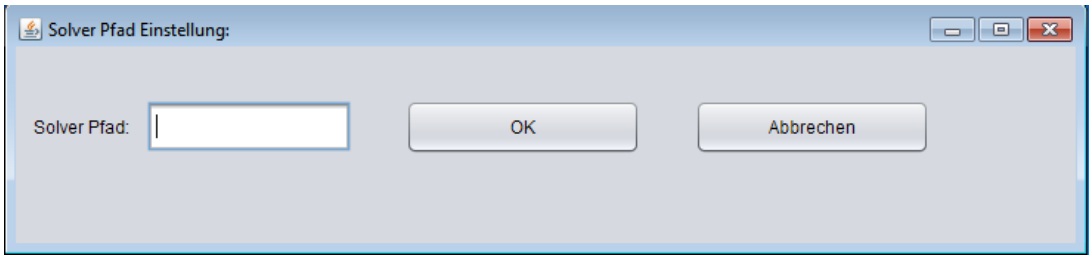

Abbildung 14: Fenster für den Solver-pfad

Für die Umsetzung des Fensters werden wir mit dem Konstruktor die Methode „initComponents()“ aufrufen, die die Funktionen für den Layout und den SolverPfad enthält.

```
            public SolverPfad() {
                super.setTitle("Solver Pfad Einstellung:");
                initComponents();
            }
```

Abbildung 15: Deklaration der Fenster

```
private void initComponents() {


    jPanel1 = new javax.swing.JPanel();
    jLabel1 = new javax.swing.JLabel();


    pfadEingabeTextField = new javax.swing.JTextField();
    OK = new javax.swing.JButton();
    Abbrechen = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

    jLabel1.setText("Solver Pfad:");

    pfadEingabeTextField.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            pfadEingabeTextFieldActionPerformed(evt);
        }

        private void pfadEingabeTextFieldActionPerformed(java.awt.event.ActionEvent evt) {


        }
    });
   pfadEingabeTextField.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyPressed(java.awt.event.KeyEvent evt) {
            pfadEingabeTextFieldKeyPressed(evt);
        }

        private void pfadEingabeTextFieldKeyPressed(java.awt.event.KeyEvent evt) {


        }
   });

   OK.setText("OK");
   OK.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            OKActionPerformed(evt);
        }

        protected void setLpSolvePath(String newLpSolvePath) {

            if (newLpSolvePath != null) {
                paths.setzeString("LPSolve", "Path", newLpSolvePath);
                paths.schreibeINIDatei(iniPath, true);
            }
            dispose();
        }

        private void OKActionPerformed(java.awt.event.ActionEvent evt) {

            String txt = pfadEingabeTextField.getText();
            setLpSolvePath(txt);
            dispose();
        }
   });
```

Abbildung 16: Weiterentwciklung der Funktionen für „OK“ und Speichern des Solver-Pfads

Wenn auf den OK.Button geklickt wird, wird die Methode „OKActionPerformed“ aufgerufen die den eingegegeben Pfad in der INI-Datei speichert.

```
Abbrechen.setText("Abbrechen");
Abbrechen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        AbbrechenActionPerformed(evt);
    }

    private void AbbrechenActionPerformed(java.awt.event.ActionEvent evt) {
        //neue Funktion die zum beenden führt
         dispose();         }
});
```

Abbildung 17: Fuktion für Abbrechen

Für die Änderung des SolverPfads haben wir eine IniDatei erstellt, die das eingegebene Solverpfad einliest und mit Setter-Methode speichert.

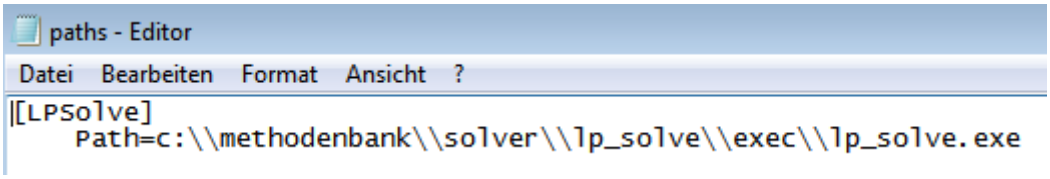

Abbildung 18: paths-Datei

4.2 Ergebnisausgabe

Nach dem Solver-Pfad Funktion haben wir uns an die Arbeit gemacht, die Ergebnisausgabe zu ermöglichen. Bei der Fehleranalyse hatten wir festgestellt, dass die Ursache an dem Laden der Filereader lag. Der Filereader konnte die Datei mit dem Solver-Pfad nicht finden, da der Pfad des Ordners falsch war (siehe Abbildung 19)

Bei der Umsetzung haben wir eine INIDatei erstellt, die bei der Pfadeingabe den Pfad in die Datei einliest.

```
public class SolverPath {
    private static final String iniPath = "paths.ini";
    private static final INIDatei paths = new INIDatei(iniPath);

    public String getLPSolverPath() throws IOException {

        String result = "c:\\methodenbank\\solver\\lp_solve\\exec\\lp_solve.exe";
        String lpSolverPath = paths.leseString("LPSolve", "Path");
        if (lpSolverPath != null) {
            result = lpSolverPath;
        }
        return result;
    }

    public static void setLpSolvePath(String newLpSolvePath) {
        if (newLpSolvePath != null) {
            paths.setzeString("LPSolve", "Path", newLpSolvePath);
            paths.schreibeINIDatei(iniPath, true);
        }
    }
```

Abbildung 19: Erstellung des INI-Dateis

Wie bereits im Abschnitt Fehleranalyse erwähnt, war der Solver-Pfad auch fehlerhaft, den wir danach angepasst haben.

```
lpSolvePath=C:\\Methodenbank\\Solver\\LP_Solve\\Exec\\lp_solve.exe
```

Abbildung 20: Solver-Pfad

Nach den ganzen Änderungen sieht nun die Ausgabe wie folgt aus:

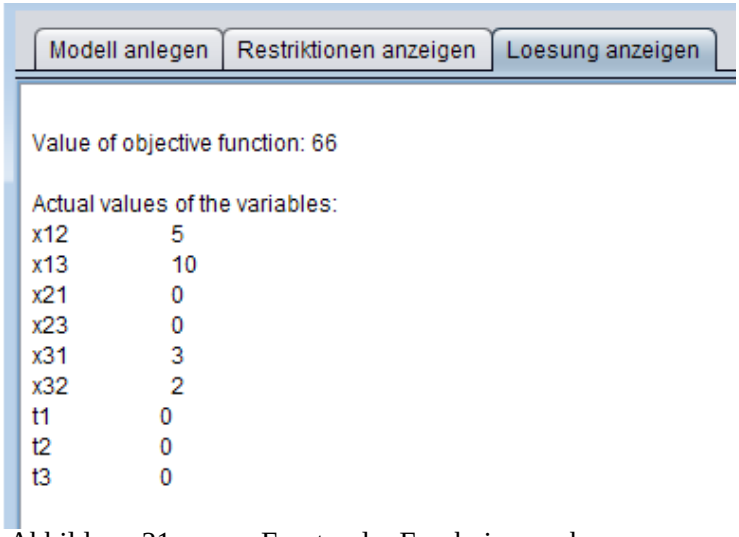

Abbildung 21: neues Fenster der Ergebnisausgabe

4.3 Restriktionen

Nach der Feststellung der Fehler bezüglich der Restriktionen hatten wir die Wahl zwischen Behebung der Fehler im Code oder Neuprogrammierung der Nachfrager Restriktionen. Wir haben uns für die Neuprogrammierung entschieden, weil wir die Ursache im Code nicht ausfindig machen konnten.

Da die Restriktionen der Anbieter richtig gebildet werden, haben wir bei der Bildung der Nachfrager-Restriktionen den Code der Anbieter-Restriktionen berücksichtigt. Die richtige Restriktionen-Bildung erfolgt, indem jede Zelle spaltenweise durchgegangen wird (siehe Abb. 22).

	R4	R5	R6	
	A	B	C	Anbieter
A	1.0	4.0	3.0	15.0
B	3.0	1.0	2.0	0.0
C	4.0	2.0	1.0	5.0
Nachfrager	3.0	7.0	10.0	

Abbildung 22: Zeichnung der Nachfrager-Restriktionen

Der neue Code ist ähnlich wie der für die Anbieter und sieht wie folgt aus:

```
if (kostenmatrix[column][row]>0.0){
    if(column==row){
        nachfragerValueTemp =-1;
        }else{
        nachfragerValueTemp =1;
        }
    }else {
    nachfragerValueTemp =0;
    }

    if(nachfragerValueTemp  !=0){
    if (setNachfragerFirstRow == false) {
        setNachfragerFirstRow = true;
        if( row == column){
        nachfragerString += nachfragerValueTemp + "*" + "t" + (column + 1);
        setRowNacfrager = true;
        } else {
        nachfragerString += nachfragerValueTemp + "*" + "x" + (column + 1) + (row + 1);
        setRowNacfrager = true;
        }

        } else {
        if (column== row) {
            nachfragerString += " + " + nachfragerValueTemp + "*" + "t" + (column + 1);
            setRowNacfrager = true;
        } else {
        nachfragerString += " + " + nachfragerValueTemp + "*" + "x" + (column + 1) + (row + 1);
        setRowNacfrager = true;
        }
    }
    }
if ((kostenmatrix.length)== row) {
    if (setRowNacfrager == true) {

        if(difference > 0.0){
            nachfragerString += " + 1*x" + (column + 1) + (kostenmatrix.length) ;
        }
        nachfragerString += " = " + nachfragerArray[column] + ";"   + "\n";
    } else {
        nachfragerString += "0.0 " + "= 0.0 ;"+" \n";

    }
}
```

Abbildung 23: Code der neuen Nachfrager-Restriktionen

Nun sind die Nachfrager-Restriktionen richtig (siehe Abb.24).

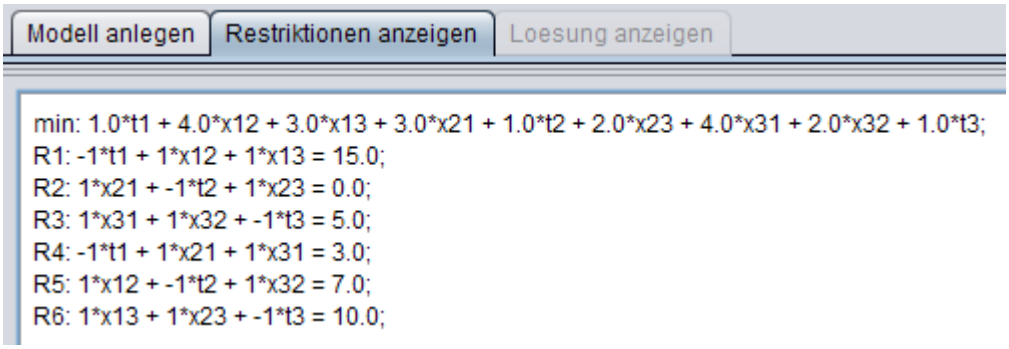

Abbildung 24: neue Restriktionenausgabe

5 Weitere Verbesserungsmöglichkeiten

Das Umladeproblem 4.1 kann weiterentwickelt werden, indem die im Kapitel 3.1 „Übersicht“ genannten Kleinigkeiten beseitigen.

Aufgehoben muss der Fehler bei der „Modell Anlegen“ (siehe Abb. 2). Sowie die Funktionen für „Help“ und „Info“ müssen auch mit nützlichen Informationen aufgefüllt werden. Ein weiterer Vorschlag ist, dass der MightyMightyLp- Solver zum Laufen gebracht wird.

Als ein Verschönerungspunkt könnte das Layout des Fensters der Restriktionen und der Ergebnisausgabe vergrößert werden.