HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Anwendung der linearen Optimierung WIN 6

Inan Ödünc, 288944, Wirtschaftsinformatik 7. Semester

Koray Aras, 288949, Wirtschaftsinformatik 6. Semester

Sommersemester 16

„Paket Transport Simulation System“

Inhalt

1. Einführung

Im Rahmen der Veranstaltung „Anwendung der linearen Optimierung“ im Sommersemester 2016 das durch Herrn Prof. Dr. Grütz angeboten wird, hatten wir das Tool „Paket Transport Simulation System“ das analysiert und auch verbessert werden müsste.

Unsere Ausarbeitung haben wir aufgeteilt auf vier Kapiteln. Das Kapitel „Ist-Zustand“ soll das Tool näher erläutern und beschreiben für welchen Zweck, von wem, und wann es Entwickelt wurde. Im Kapitel darauf „Programmierung/Umsetzung“ werden die Ziele beschreiben die erreicht werden sollten und auch wie wir vorgegangen sind, welche Programmierungstechnischen Änderungen vorgenommen werden müssten. Zum Schluss der Arbeit gibt es noch ein Fazit und auch Verbesserungsvorschläge für das Tool.

2. Ist-Zustand der Methode (31.03.2016)

Die Methode „Paket Transport Simulation System“ ist ein Werkzeug für die Modellierung und Simulation von paketorientierten Netzwerken mit integriertem optimierendem Routenplaner. Es können z.B. Routen und Kosten einer beliebigen Stecke errechnet werden.
Als Ergebnis werden jene Strecken gezeigt, welche für eine optimale Route in Frage kommen.
Als Beispiel: User könnten sich z.B. einfach die Route (mehrere Routen) zwischen Konstanz und Hamburg errechnen lassen.

Die Methode wurde von zwei ehemaligen Studenten (Mathias Jehle und Thomas Geldner) im Wintersemester 2002/2003 entwickelt und ist lauffähig unter den Betriebssystemen Windows 7 und 10. Die Applikation wurde komplett in Java erstellt.

In der folgenden Abbildung können wir anhand eines Beispiels sehen, wie wir eine Route berechnen lassen können.

2.1 Beispiel

Der gelbe Knoten ist als Startknoten gekennzeichnet. Diese bietet zwei verschiedene Möglichkeiten, an das Ziel (blauer Knoten) Knoten zu gelangen. Somit wird die kostengünstigere Strecke ermittelt.

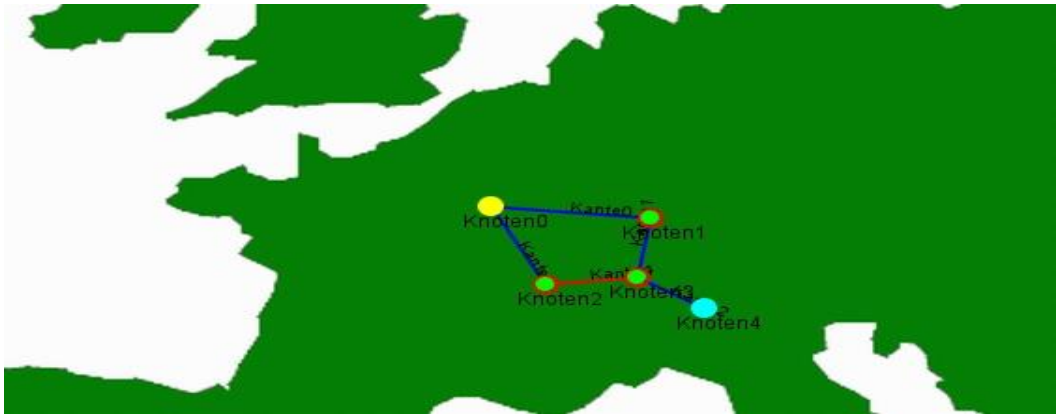

Abbildung 1

2.2 In der Methode gibt es folgende Buttons:

Abbildung 2

Hier können wir zwischen **Datei**, **Ansicht** und **Hilfe** auswählen.

Abbildung 3

Unter dem Button „**Datei**“ hat der User die Auswahl zwischen *Neu* und *Modell laden*.

Unter Neu kann man sich eine Route bzw. Karte anzeigen lassen

Abbildung 4

Der Button „**Modell laden**“ ermöglicht dem User eine bereits vorhandene Route/Karte zu laden.

Abbildung 5

Zusätzlich kann der User das Modell unter „**Modell speichern**“ abspeichern.

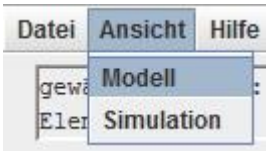

Abbildung 6

Der Nutzer hat die Möglichkeit, sich die **Ansicht** als *Modell* oder auch *Simulation* anzuzeigen.

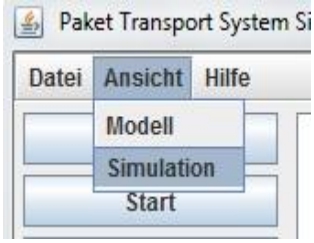

Abbildung 7

Die Funktion *Simulation* unter dem Button **Ansicht** ist in der Methode nicht voll funktionsfähig

2.3 LP-Ansatz

ZF: 1x0_1 + 2x0_2 + 1x1_2 + 1x2_1 + 1x2_3 + 3x1_3 → min!

KN0s: x0_1 + x0_2 >=1
KN1: x0_1 – x1_2 + x2_1 – x1_3 >=0
KN2: x0_2 + x1_2 – x2_1 – x2_3 >=0
KN3z: x2_3 + x1_3 >=1

Solverausgabe:

Value of objective function: 3

X0_1=0;
X0_2=1;
X1_2=0;
X1_3=0;
X2_1=0;
X2_3=1;

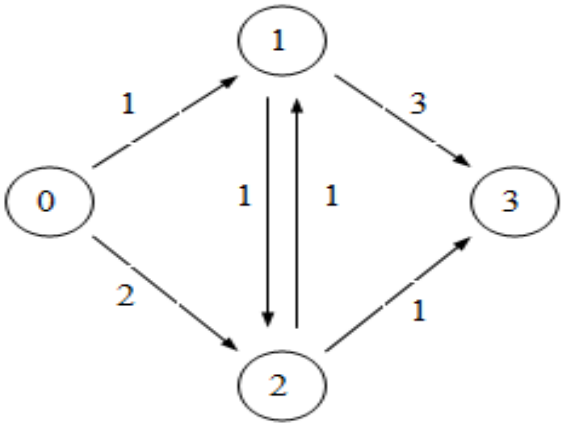

Abbildung 8

In unserem Projekt hatten wir die Aufgabe, folgende Punkte zu verbessern:

1. Hilfefunktion ist nicht aufrufbar
2. Gespeicherte Modelle können nicht geladen werden
3. Der Button „Alle Routen berechnen“ funktioniert nicht

Die obigen Punkte mussten analysiert und programmiert werden, sodass der User auch diese Funktionalitäten nutzen kann.

3. Programmierung/ Umsetzung

3.1 Hilfe Funktion Implementieren

In der Klasse „Hauptframe“ wurde die Methode „jMenuItem6_actionPerformed“ erweitert um den folgenden code-teil:

```
try {
    Runtime.getRuntime().exec(
            "rundll32 url.dll,FileProtocolHandler "
                    + new java.io.File(".\\Programmhilfe\\Programmhilfe_Paket Transport Simulation System.html")
                            .getAbsolutePath());
} catch (Exception ex) {
        JOptionPane.showMessageDialog(null, "Onlinee Hilfe fehler"
                +"\n ");
}
}
```

Abbildung 9

Die exec Methode ist eine Funktion die Shell Kommandos ausführt in diesem fall folgendes Kommando:

```
".\\Programmhilfe\\Programmhilfe_Paket Transport Simulation System.html"
```

Das ermöglicht jetzt beim Drücken auf den Hilfe Button das eine HTML Datei im Standard Browser als Hilfsdatei geöffnet wird.

3.2 Alle Routen berechnen

Die Methode „doAlleRouten()“ in der Klasse „RouterManager“ wurde so umgeändert, das jetzt beim Aufruf dieser Methode (Beim Klicken auf das Button „Berechne alle Routen“), die Methode „FindAlleRoute()“ aufgerufen wird, diese Methode ist dafür zuständig das alle routen die im Modell eingezeichnet wurden gefunden werden.

```
public synchronized void doAlleRouten(){

    routenAlleListe = new Vector<Route>();
    routenAlleListe.clear();

    mView.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

    Route tmpRoute = new Route();
    FindAlleRouten(KnotenListe.getStartKnotenId(), tmpRoute);

    mView.setCursor(Cursor.getDefaultCursor());

    mView.repaint();
}
```

Abbildung 10

```
public synchronized boolean FindAlleRouten(int nVonKnotenId, Route aRoute)
{
    Knoten tmpNachKnoten;
    for (Iterator iterNach = KnotenListe.iterator(); iterNach.hasNext(); )
    {
        tmpNachKnoten = (Knoten) iterNach.next();

        int nKantenId = KantenListe.getKanteIdByKnotenId(nVonKnotenId, tmpNachKnoten.getId());
        if( nKantenId != -1 )
        {
            if( aRoute.existTeilAsKnote(tmpNachKnoten.getId()) )
                continue;

            Route tmpRoute = aRoute.copy();

            tmpRoute.addTeilRoute(nKantenId, nVonKnotenId, tmpNachKnoten.getId());

            if( tmpNachKnoten.isZiel() )
            {
                tmpRoute.setStart(KnotenListe.getStartKnotenId());
                tmpRoute.setZiel(KnotenListe.getZielKnotenId());
                tmpRoute.sort();
                this.routenAlleListe.add(tmpRoute);
            }
            else
            {
                FindAlleRouten(tmpNachKnoten.getId(), tmpRoute);
            }
        }
    }
    return true;
}
```

Abbildung 11

3.3 Das Öffnen von vorher gespeicherten Modellen

Zum Öffnen eines Modells gab es schon eine Methode „OpenModell()“ in der Klasse „ModellierView“, doch beim Ausführen der Funktion Modell laden kam folgende Fehlermeldung:

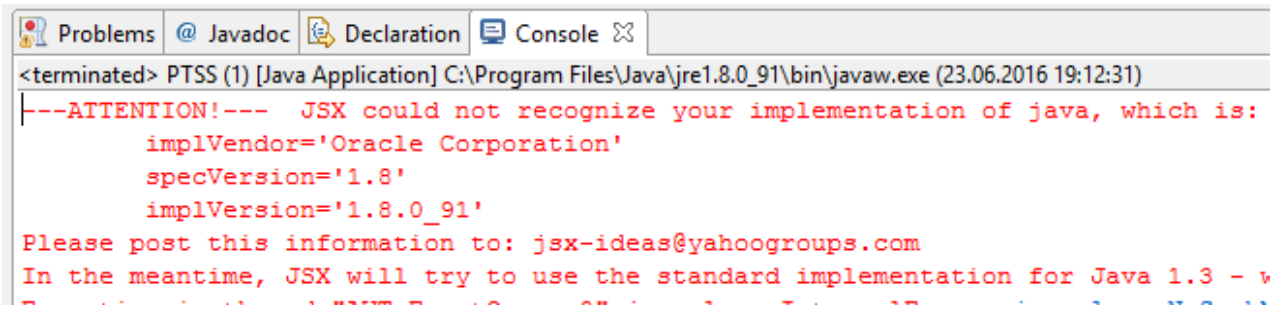

Abbildung 12

Nach einigen Recherchen und versuchen den Fehler zu beheben, kamen wir auf die Lösung wieso diese Fehlermeldung angezeigt wird.

Es lag an der Datei „JSX 1.0.6.0.jar“ im „lib“ ordner, diese Datei ist eine Bibliothek die in dem Code eingebunden wurde. Die Datei „JSX 1.0.6.0.jar“ war wurde durch

eine neuere Version ersetzt. Die neue Datei „JSX2.3.0.5.jar“ wurde auch im Ordner „lib“ abgelegt. In der Methode „OpenModell()“ wurde folgender Code-teil:

```
public void openModell(){

  String fileToOpen = (openDia(1));
  if (fileToOpen != null){

    KnotenListe knotenListe = new KnotenListe();
    KantenListe kantenListe = new KantenListe(knotenListe);
    RoutenListe routenListe = new RoutenListe();
    RouterManager rManager = new RouterManager();
    try{
      ObjIn in = new ObjIn( new FileReader(fileToOpen));
      Object o = in.readObject( );
```

Abbildung 13

Umprogrammiert in die folgende Version:

```
public void openModell(){

  String fileToOpen = (openDia(1));
  if (fileToOpen != null){

    KnotenListe knotenListe = new KnotenListe();
    KantenListe kantenListe = new KantenListe(knotenListe);
    RoutenListe routenListe = new RoutenListe();
    RouterManager rManager = new RouterManager();
    try{
      JSX.ObjectReader in = new JSX.ObjectReader( new FileReader(fileToOpen));
      Object o = in.readObject( );
```

Abbildung 14

4. Abbildungsverzeichnis

5. Fazit/Verbesserungsvorschläge

Die Anforderungen konnten komplett umgesetzt werden. Das Projekt erwies sich dabei als durchaus fordernd, komplex und Zeitaufwendig. Es bot aber die Möglichkeit ein konkretes Problem im Team umzusetzen.

Am Ende des Projektes konnten wir im gesamten ein positives Endergebnis erzielen.