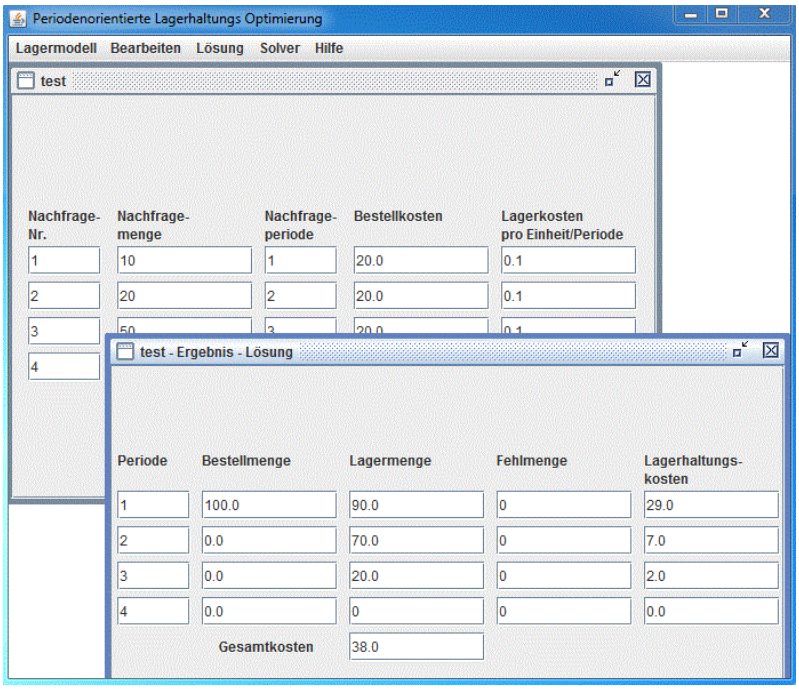
Analysebericht

Wagner Whitin

LP 1.1



Teammitglieder

Eugen Gering - 287426
Melisa Gündüz - 289100

Projektname

Wagner Whitin LP 1.0

Projektnummer

ALO Projekt Nr.3
WS 2015/16

Abgabetermin

31.01.2016

Abbildungsverzeichnis

Inhaltsverzeichnis

1 Fehlerstatus

Das in OR-Alpha bestehende Wagner-Whitin-LP hat einige Mängel vorzuweisen. Anbei eine Auflistung dieser:

- Der Pfad ist fix und eine dynamische Pfadanpassung wird nicht gewährleistet.
- Unvollständige Dokumentationen hinterlegt.
- Unzureichende Hilfestellung in der Methode.
- Ergebnisausgabe über Editor muss ausgewählt werden.
- Keine automatische Nummerierung.
- Rechenfehler bei Bedarfsmengen größer 1000.

Einige dieser Fehler werden im Rahmen dieses Projektes analysiert, angegangen und behoben.

2 Fehlerbehebung

2.1 Ergebnisausgabe

Der Wagner Whitin benötigt für die Lösungsausgabe den Editor, um die Daten des LP-Solves im MPS-Format anzeigen zu können. Die Losgrößen sind nicht leicht zu erkennen und nähere Informationen zu den Kosten werden nicht angezeigt. In Abbildung 1 wird die ursprüngliche Lösungsausgabe angezeigt:

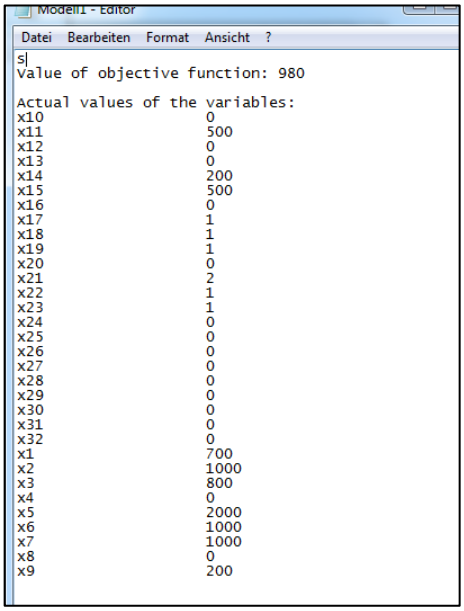


Abbildung 1: Ausgabe der Lösung über den Editor

Um eine Lösungsausgabe in der Methode selbst zu ermöglichen, muss einerseits die Ausgabe im MPS-Format für die Methode übersetzt werden und der Editoraufruf unterbunden werden. Die in der Klasse LPSolver enthaltene try-catch-Methode, die einen LP-Solver-Aufruf startet und gleichzeitig versucht den Editor aufzurufen wurde entsprechend angepasst.

```
try
{ // Stapeldatei erzeugen fuer Solver Aufruf, da direkter Aufruf nicht
  // funktioniert..!
  FileOutputStream fileOutStream = new FileOutputStream(executeBatch);
  PrintWriter out = new PrintWriter(fileOutStream);
  out.println("@echo of");
  out.println("echo Automatisch erzeugte Stapeldatei startet den LP-Solver");
  out.println(executeSolver);

  //Auskommentiert damit die Lösung nicht erst über ein Texteditor geöffnet wird
  //out.println(parameter2);
  out.println("exit");
  out.close();
}
catch (Exception e)
{
  _errMsg = "Konnte Start.bat zum starten des Sovers nicht schreiben";
  return -1;
}
```

Abbildung 2: Editoraufruf auskommentiert

Des Weiteren wurde eine Schleife eingebaut, um die Datei in MPS-Format auszulesen und die Auswertung starten zu können.

```
    //Iteration über den "neuen" Ausgabestring: Actual values of the variables:
    for (int i = 0; i < 35; i++) {
        _token = getNextElement();
    }
    return macheAuswertung();
}
```

Abbildung 3: Aufruf der macheAuswertung() Methode

Diese Methode `private int macheAuswertung()` befindet sich in der Klasse `LPSolveParser` und ist für das Auslesen der LP-Solver Ausgabe zuständig.

2.2 Rechenfehler

In der Klasse `plo_Eingabemaske` gibt es eine Methode `public void produceMatrix()`, welche den LP-Ansatz des Modells erstellen soll. Die Methode arbeitet mit einer M-Variable mit einer ursprünglich Höhe von 1'000. Diese Variable wurde von uns auf 100'000 erhöht. In den Abbildungen 1 und 2 wird der Unterschied nochmals aufgezeigt:

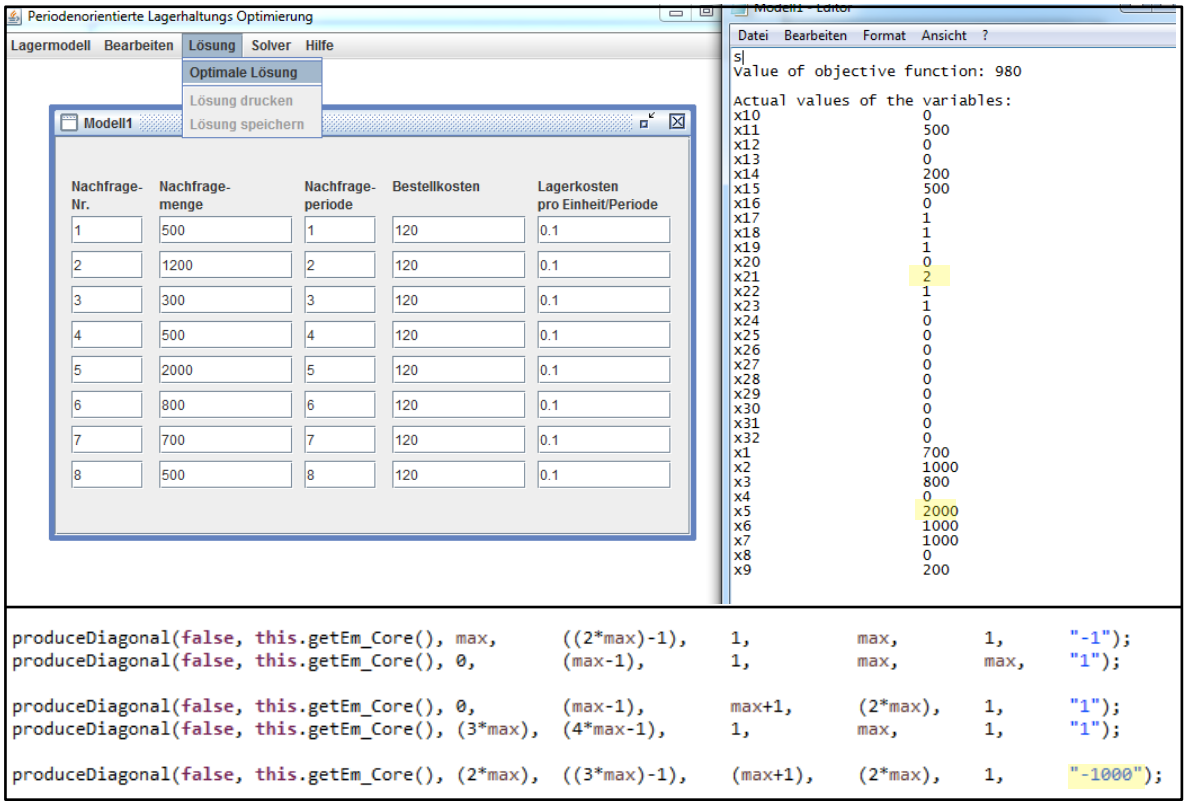


Abbildung 4: Falsche Bestellmenge

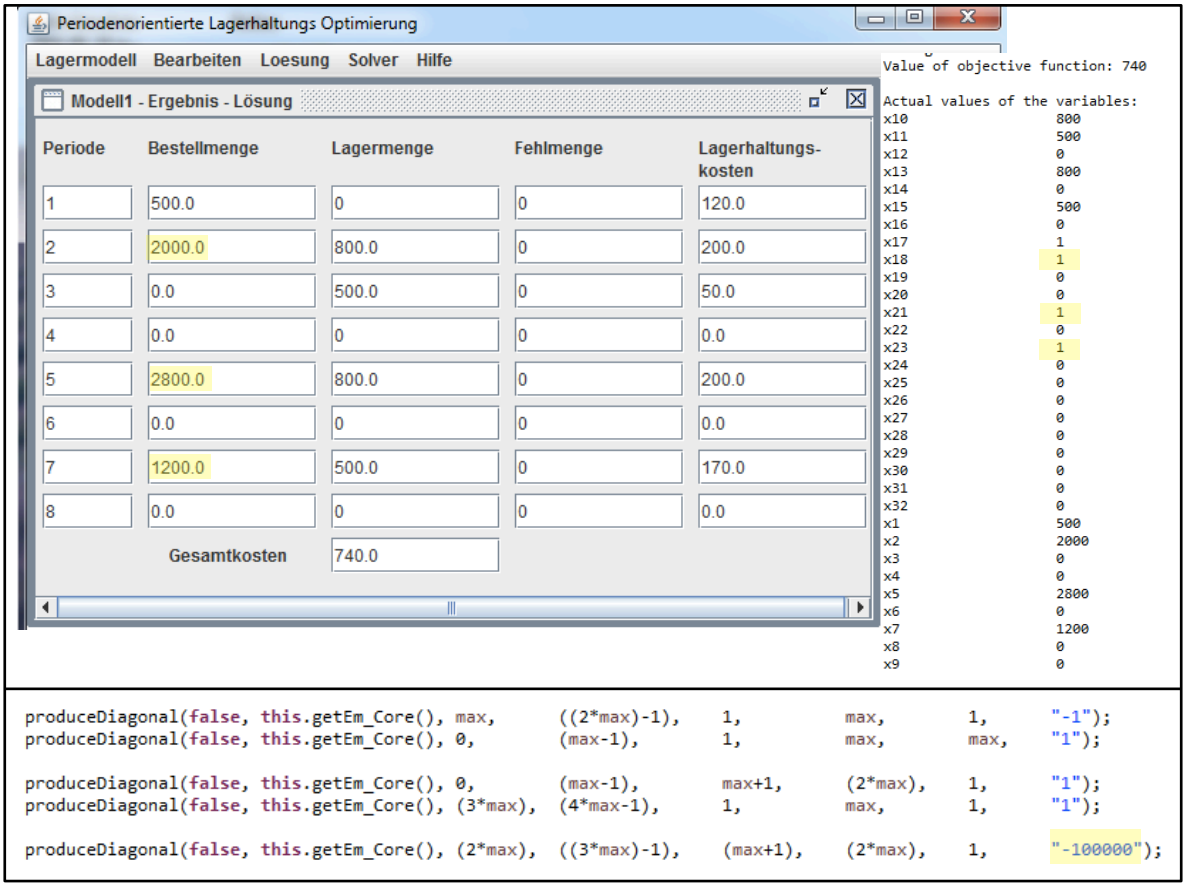


Abbildung 5: Verbesserte Bestellmengenangaben

Wie man in Abbildung erkennen kann, ist der Fehler nun behoben. Bei Bestellmengen größer 1000 wird einmal und nicht doppelt bestellt.

2.3 Hilfefunktion

In Abbildung 1 ist die ursprüngliche Hilfefunktion für Wagner Whitin dargestellt.

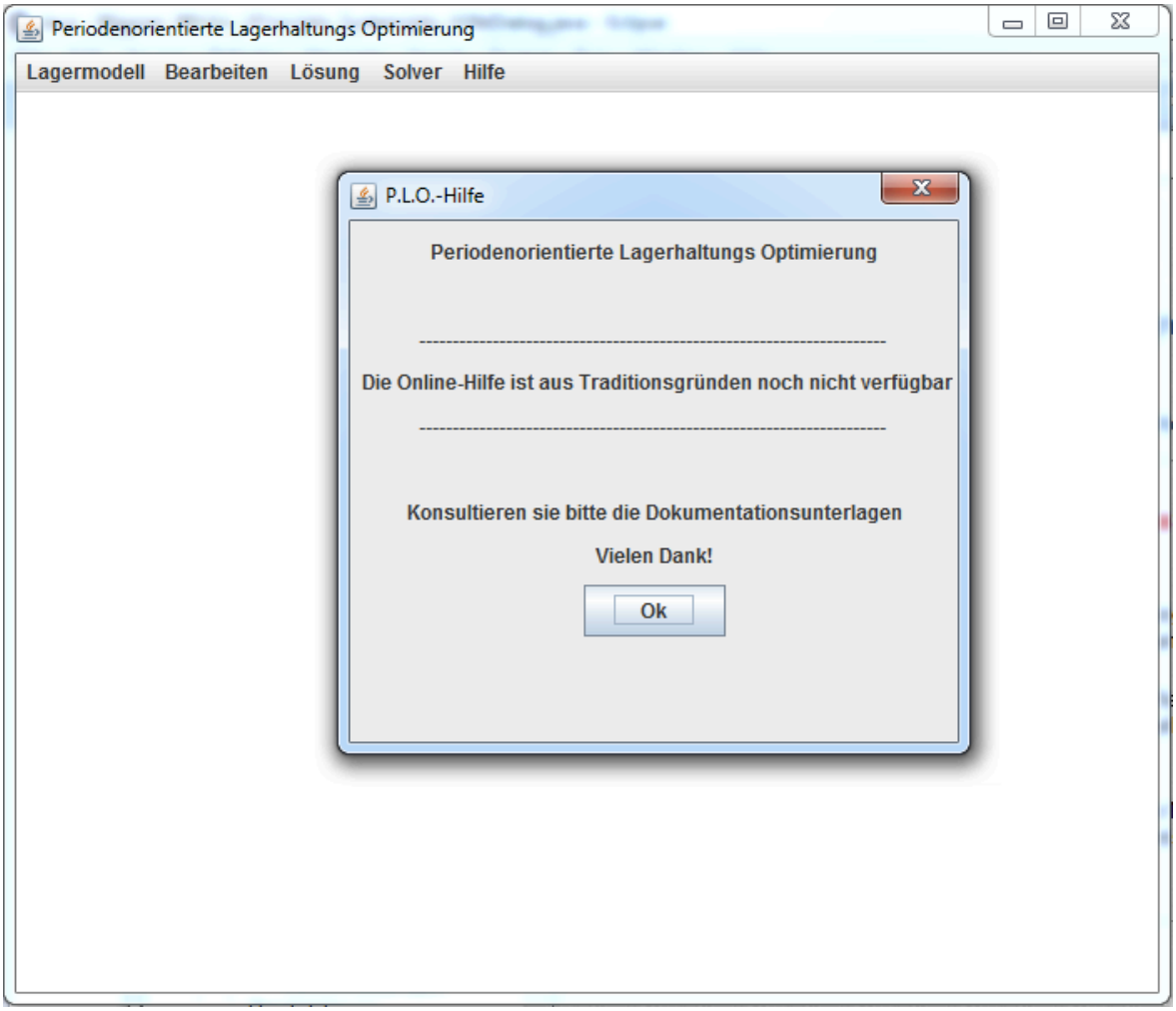


Abbildung 6: P.L.O.-Hilfe – ursprünglich

Da diese keine Hilfestellung leistet, war es unsere Aufgabe die Hilfe-Funktion zu editieren. Nach einigen Anpassungen im Quellcode (siehe Abbildung 4), ist eine Vorgehensweise beschrieben, die ein reibungsloses Benutzen der Software gewährleisten kann.

```
// Konstruktor
public plo_HilfeDialog() {
    // Initialisieren des Referenzobjekts fuer dispose()-Aufrufe
    final plo_HilfeDialog ref = this;

    hilfeFrame = new JInternalFrame();
    this.setTitle("P.L.O.-Hilfe");
    hilfeFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    hilfePanel      = new JPanel();
    hilfeLabel1     = new JLabel("   Periodenorientierte Lagerhaltungs Optimierung");
    hilfeLabel2     = new JLabel(" ");
    hilfeLabel3     = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel4     = new JLabel("   Neue Modelle erstellen");
    hilfeLabel5     = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel7     = new JLabel("   1.   Klicken Sie auf Lagermodell --> Neues Modell.");
    hilfeLabel8     = new JLabel("    2.   Nun geben Sie die Anzahl der Perioden ein.");
    hilfeLabel9     = new JLabel("   3.   Im nächsten Schritt tippen Sie alle Daten ein.");
    hilfeLabel10    = new JLabel("   4.   Jetzt klicken Sie auf Loesung --> Optimale Loesung.");
    hilfeLabel11    = new JLabel("   5.   Die Lösung mit den optimalen Lossgroessen erscheint.");
    hilfeLabel12    = new JLabel(" ");
    hilfeLabel13    = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel14    = new JLabel("   Modelle speichern");
    hilfeLabel15    = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel16    = new JLabel("   1.   Klicken Sie auf Lagermodell --> Modell speichern.");
    hilfeLabel17    = new JLabel("   2.   Nun waehlen Sie einen Speicherort und bestaetigen mit OK.");
    hilfeLabel18    = new JLabel(" ");
    hilfeLabel19    = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel20    = new JLabel("   Modelle laden");
    hilfeLabel21    = new JLabel("-------------------------------------------------------------------------");
    hilfeLabel22    = new JLabel("   1.   Gespeicherte Modelle können geladen werden, indem Sie auf ");
    hilfeLabel23    = new JLabel("          Lagermodell --> Modell laden klicken.");
    hilfeLabel24    = new JLabel("   2.   Waehlen Sie das zu oeffnende Modell und bestaetigen mit OK.");
    hilfeLabel25    = new JLabel(" ");
    hilfeLabel26    = new JLabel("   Vielen Dank!   ");

    jb_Ok           = new JButton("     Ok     ");
    jb_Ok.setSize(150, 50);

    hilfeGridBagLayout = new GridBagLayout();
    hilfeGridBagConstraints = new GridBagConstraints();

    hilfePanel.setLayout(hilfeGridBagLayout);
```

Abbildung 7: Anpassung der Hilfe-Funktion im Programcode

In Abbildung 8 ist dargestellt, wie die Hilfe-Funktion nach der Anpassung aussieht.

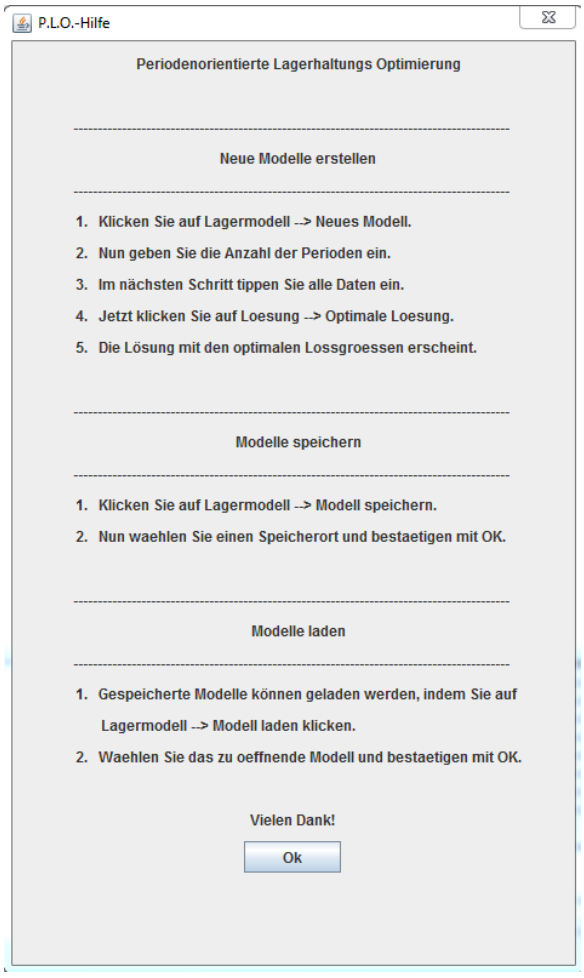


Abbildung 8: Neue Hilfe-Funktion

2.4 Automatische Nummerierung der Perioden

Beim Erstellen neuer Modelle, ist ursprünglich die Spalte der Nachfrage-Nr. und der Nachfrageperiode auf „0“ gesetzt. Es war notwendig, die Nummerierung manuell durchzuführen. Nach einer Anpassung in der Klasse `plo_Eingabemaske` sind nun default-Werte eingebettet. Die automatische Belegung der Nachfragenummer und Nachfrageperiode ist nun möglich. In den folgenden Abbildungen ist zu sehen, an welcher Stelle im Code eine Anpassung gemacht wurde und wie die Eingabemaske nach Erstellen eines Modells mit 8 Perioden aussieht.

```
int i;
for(i = 2; i < (anzahlNachfragen+2); i++)                    //Hinzufuegen der leeren Nachfragen in der
{                                                            //Anzahl, die vom Benutzer eingegeben worden ist.
    em_nach = new nachfrage();                               //
    dTemp = new Double(this.getRoot().getDefaultBestellkosten());
    em_nach.setTf_BestellkostenText(dTemp.toString());
    dTemp = new Double(this.getRoot().getDefaultLagerkosten());
    em_nach.setTf_LagerkostenText(dTemp.toString());
    //Setzen der Nachfragennummer und Perioode aufsteigend
    em_nach.setTf_NummerText(String.valueOf(i-1));
    em_nach.setTf_PeriodeText(String.valueOf(i-1));

    this.setEm_NachfragenListe((i-2), em_nach);
```

Abbildung 9: Anpassung im Code zur automatischen Nummerierung

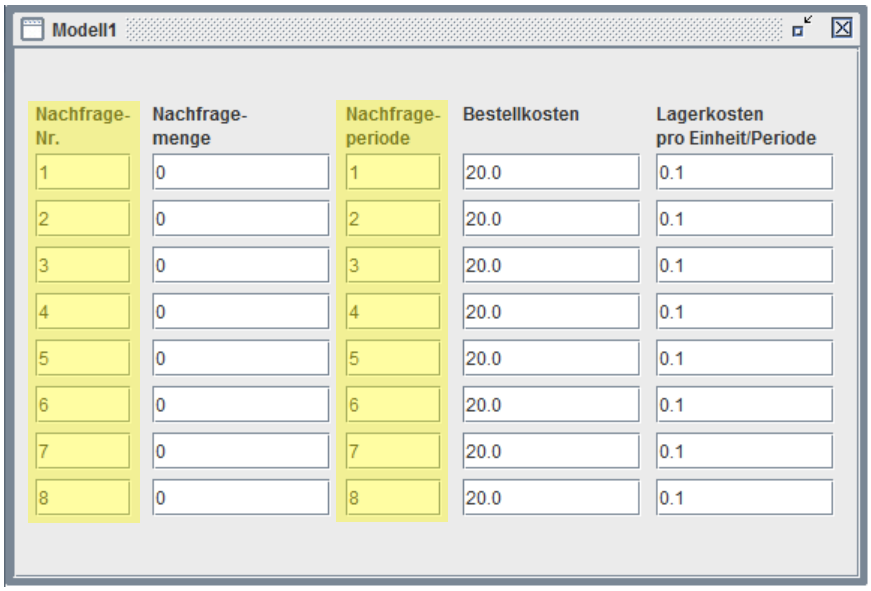


Abbildung 10: Automatische Nummerierung im Modell

2.5 Automatische Pfadanpassung

Bei erstem Programmstart müssen folgenden Angaben zu den Verzeichnissen gemacht werden:

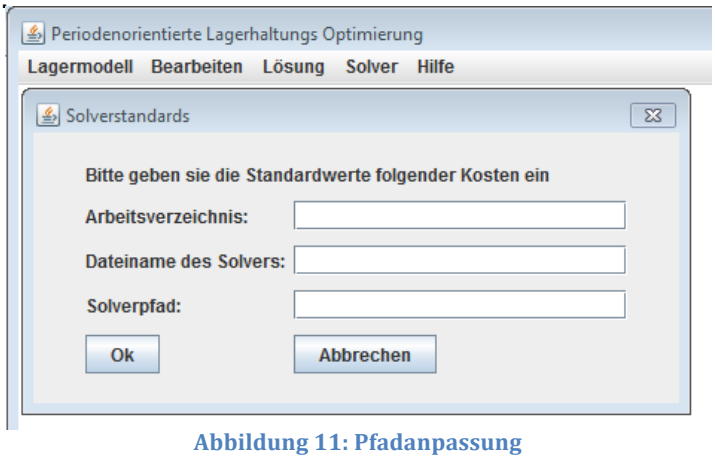


Abbildung 11: Pfadanpassung

Diese Daten werden dann in einer *solverini.txt* Datei (siehe Abbildung 13) gespeichert und bei weiteren Programmstarts ausgelesen.

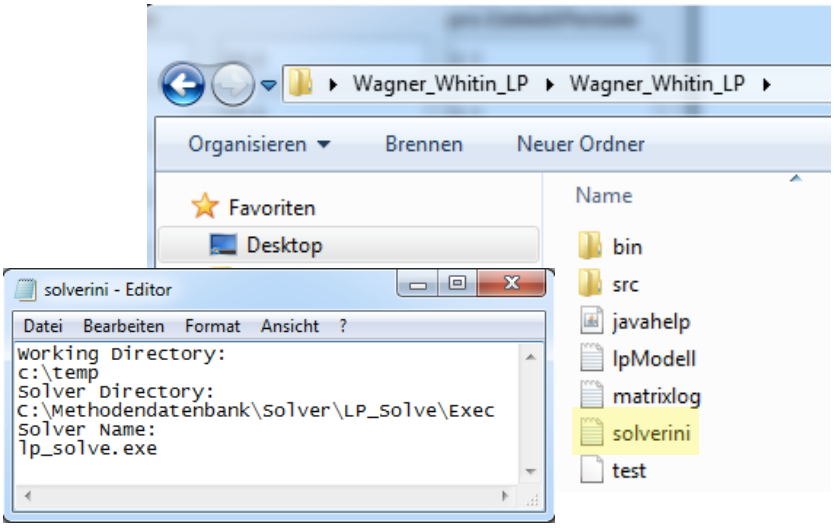


Abbildung 12: solverini.txt - Datei

Um eine automatisierte Einbettung dieser Pfaddaten zu generieren, wurden Änderungen in der Klasse `plo_Eingabemaske` vorgenommen. Siehe folgende Abbildung:

```
try                                                         //Anlegen eines Stromobjekts fuer
{                                                           //die Datei "solverini.txt"
    BufferedInputStream bis = new BufferedInputStream (new FileInputStream("solverini.txt"));
}
catch (FileNotFoundException fnfe){

    //Falls keine solverini.txt exestiert, wird eine mit vordefenierten Werten erzeugt
    //und die Methode zum einlesen wird erneut durchlaufen
    try {

        File file = new File("solverini.txt");
        FileWriter writer = new FileWriter(file);
        // Text wird in den Stream geschrieben
        writer.write("Working Directory:");
        // Platformunabhängiger Zeilenumbruch wird in den Stream geschrieben
        writer.write(System.getProperty("line.separator"));
        // Text wird in den Stream geschrieben
        writer.write("c:\\temp");
        // Platformunabhängiger Zeilenumbruch wird in den Stream geschrieben
        writer.write(System.getProperty("line.separator"));
        // Text wird in den Stream geschrieben
        writer.write("Solver Directory:");
        // Platformunabhängiger Zeilenumbruch wird in den Stream geschrieben
        writer.write(System.getProperty("line.separator"));
        // Text wird in den Stream geschrieben
        writer.write("C:\\Methodendatenbank\\Solver\\LP_Solve\\Exec");
        // Platformunabhängiger Zeilenumbruch wird in den Stream geschrieben
        writer.write(System.getProperty("line.separator"));
        // Text wird in den Stream geschrieben
        writer.write("Solver Name:");
        // Platformunabhängiger Zeilenumbruch wird in den Stream geschrieben
        writer.write(System.getProperty("line.separator"));
        // Text wird in den Stream geschrieben
        writer.write("lp_solve.exe");
        writer.flush();

        solverIniEinlesen();

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Abbildung 13: Anpassung im Programmcode zur Pfadanpassung

Der try-catch-Block ist für das Erstellen der txt-Datei und das auslesen dieser verantwortlich. Es wird erst überprüft, ob eine *solverini.txt* Datei besteht. Falls nicht, wird die Datei im Hintergrund erstellt und abgelegt. Der Vorteil ist, dass keine manuelle Eingabe der Pfade nötig ist.

2.6 Beheben der Schreibfehler

In der Menü-Leiste Lagermodell sind Rechtschreibfehler (siehe nächste Abbildung), die wir verbessert haben:

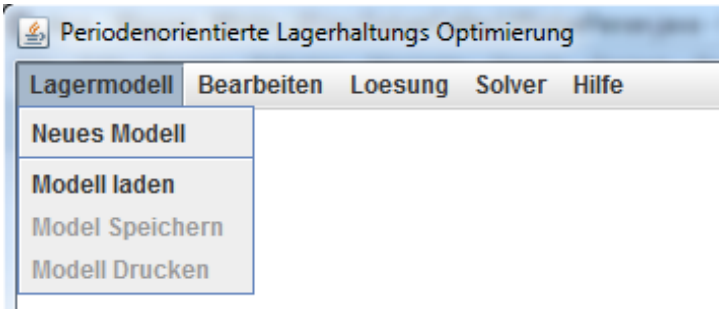


Abbildung 14: Rechtschreibfehler in der Navigationsleiste

In der Klasse `plo_MenuBar` wurden entsprechende Anpassungen gemacht.

```
m_Lagermodell = new JMenu("Lagermodell");                    //Initialisieren der Menues
m_Bearbeiten = new JMenu("Bearbeiten");
m_Loesung = new JMenu("Lösung");
m_Solver = new JMenu("Solver");
m_Hilfe = new JMenu("Hilfe");

mi_NeuesModell = new JMenuItem("Neues Modell");              //Initialisieren der Menuepunkte
mi_ModellLaden = new JMenuItem("Modell laden");
mi_ModellSpeichern = new JMenuItem("Model speichern");
mi_ModellSpeichern.setEnabled(false);
mi_ModellDrucken = new JMenuItem("Modell drucken");
mi_ModellDrucken.setEnabled(false);

mi_NachfrageEinfuegen = new JMenuItem("Nachfrage einfügen");
mi_NachfrageEinfuegen.setEnabled(false);
mi_NachfrageEntfernen = new JMenuItem("Nachfrage entfernen");
mi_NachfrageEntfernen.setEnabled(false);
mi_AllesAendern = new JMenuItem("Alles ändern");
mi_AllesAendern.setEnabled(false);
mi_Defaultkosten = new JMenuItem("Defaultkosten");

mi_OptimaleLoesung = new JMenuItem("Optimale Lösung berechnen");
mi_OptimaleLoesung.setEnabled(false);
mi_LoesungDrucken = new JMenuItem("Lösung drucken");
mi_LoesungDrucken.setEnabled(false);
mi_LoesungSpeichern = new JMenuItem("Lösung speichern");
mi_LoesungSpeichern.setEnabled(false);

mi_SolverConfigAendern = new JMenuItem("Solver Konfiguration ändern");
mi_SolverConfigAendern.setEnabled(true);

mi_PloHilfe = new JMenuItem("PLO-Hilfe");
mi_ueber = new JMenuItem("Über");
```

Abbildung 15: Behobene Rechtschreibfehler im Programmcode

2.7 Anpassung der „Über“-Daten

Die über die Leiste Hilfe → Über zu findenden Informationen wurden entsprechend der neuen Version angepasst. In der folgenden Abbildung ist die aktuelle Klasse `plo_ueberDialog` zu sehen.

```
//*** Konstruktor ***-----------------------------------------------
    public plo_ueberDialog()
    {
        final plo_ueberDialog ref = this;                          //Erstellen eines Referen

        überFrame = new JInternalFrame();
        this.setTitle("Über P.L.O.");
        überFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        überPanel = new JPanel();
        überLabel1 = new JLabel("Periodenorientierte Lagerhaltungs Optimierung");
        überLabel2 = new JLabel("(Version 1.1)");
        überLabel3 = new JLabel("(c) 2015/20016 ");
        überLabel4 = new JLabel("-----------------------------------------------------------
        überLabel5 = new JLabel("Programm von:");
        überLabel6 = new JLabel("Eugen Gering");
        überLabel7 = new JLabel("Melisa Gündüz");
        überLabel8 = new JLabel("Francis Göltner");
        überLabel9 = new JLabel("Helmut Lindinger");
        überLabel10 = new JLabel("Bernd Saile");
        jb_Ok = new JButton("     Ok     ");
        jb_Ok.setSize(150,25);
        überGridBagLayout = new GridBagLayout();
        überGridBagConstraints = new GridBagConstraints();
        überPanel.setLayout(überGridBagLayout);

        this.buildConstraints(überGridBagConstraints, überLabel1, 0, 0, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel2, 0, 1, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel3, 0, 2, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel4, 0, 3, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel5, 0, 4, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel6, 0, 5, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel7, 0, 6, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel8, 0, 7, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel9, 0, 8, 1, 1, GridBagConstrai
        this.buildConstraints(überGridBagConstraints, überLabel10, 0, 9, 1, 1, GridBagConstra
        this.buildConstraints(überGridBagConstraints, jb_Ok, 0, 12, 1, 1, GridBagConstraints.
```

Abbildung 16: Angepasstes Über-Dialog im Programmcode

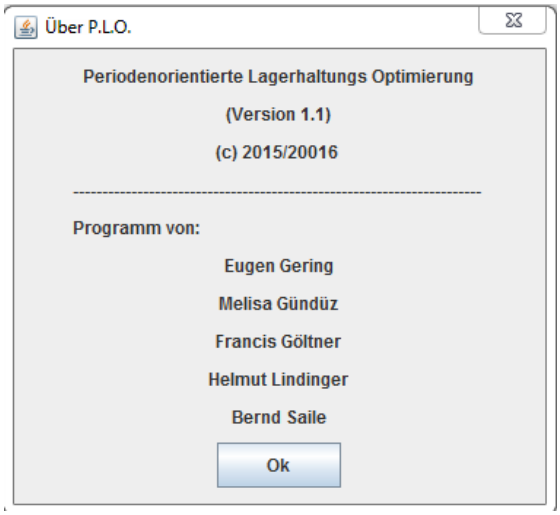


Abbildung 17: Neues Über-Dialog

3 Gegenüberstellung des Commitment-Inhalts und Umgesetzten

Aufgabenbereich	Commitment-Inhalt	Umsetzung
Hilfefunktion	Die Hilfe-Funktion soll editiert werden.	✓
Lösungsausgabe	Das Ergebnis soll nicht im Editor, sondern im Programm selbst ausgegeben werden. (Layout)	✓
Rechenfehler	Bei Bestellmengen höher als 1000 werden doppelte Bestellkosten bestimmt. Herausfinden wieso das so ist und anpassen.	✓
⊕ **LOGO**		
⊕ **Automatisierte Pfadanpassung**		
⊕ **Vollständige Dokumentation**		
⊕ **Automatische Nummerierung der Perioden**		
⊕ **Beheben der Rechtschreibfehler**		

Tabelle 1: Gegenüberstellung