



Projektdokumentation

JobShop 2.2

Projekt Nr. 6

Name, MatrNr. : - Irina Murug, 290912
- Priskilla Natasha, 291068

Studiengang : WIN6

Fach : Anwendungen der linearen Optimierung

Semester : SS 2016

Betreuer : **Prof. Dr. Michael Grütz**

INHALT

1.	Einleitung.....	3
1.1	Motivation	3
1.2	Zielsetzung	3
2.	Projektorganisation	4
2.1	Änderungsanforderungen.....	5
2.2	Probleme und Fehler	8
2.2.1	Technische Probleme	8
2.2.2	Lösungsansatz.....	9
3.	Umsetzung.....	10
3.1	Hilfe-Funktion	10
3.2	Load/Save Funktion	13
3.3	LP-Ansatz und Matrixanalyse	16
4.	Fazit	21

1. EINLEITUNG

Im Rahmen der betrieblichen Systemforschung pflegt die Hochschule für Technik, Wirtschaft und Gestaltung (HTWG) seit vielen Jahren eine Methodenbank. In der Methodenbank sind verschiedene Programme für die Lösung unterschiedlicher Problemstellungen der linearen Programmierung (LP) enthalten. Die verschiedenen Software-Anwendungen greifen dabei auf unterschiedliche Solver zur Lösung der erstellten LP-Modelle zurück. Die Methodenbank umfasst sowohl Open-Source-Solver (LpSolve, MOPS) als auch kommerzielle Solver-Produkte (IBM ILOG CPLEX).

1.1 MOTIVATION

Entwickelt und auch eingesetzt wird diese Methodenbank primär in der Vorlesung „Operation Research“. Das Ziel der Weiterentwicklung und Sicherstellung der Lauf- und Funktionsfähigkeit der einzelnen Methoden und Programmen und der Methodenbank selbst ist Gegenstand der Veranstaltungen „ALO“ in den höheren Wirtschaftsinformatik-Semestern.

1.2 ZIELSETZUNG

Das Primärziel des Projekts Nr.6 „Erweiterung des Tools JobShop 2.1 „ ist es, die Hilfefunktion auf einen ausreichenden Stand zu bringen. Des Weiteren soll ein neuer Solver (GLPK) in die Methode integriert werden. Es besteht die Möglichkeit Beispieldateien zu öffnen, jedoch wird das falsche Verzeichnis aufgerufen. Es soll ermöglicht werden, die Beispieldateien aus dem richtigen Verzeichnis z.B (C://.../Jop_Shop_64/**example**/beispiel.job) zu laden. Parallel soll versucht werden, das LP-Ansatz auf Richtigkeit zu überprüfen und gegebenenfalls überarbeiten.

Mit dieser Dokumentation soll der Ist-Zustand des Programms JobShop 2.1 erfasst, sowie die aktuellen Probleme analysiert und festgehalten werden. Auf dieser Basis sollen dann die Weiterentwicklungsmöglichkeiten getroffen werden.

Bei JobShop 2.1¹ handelt es sich um ein Programm dessen Aufgabe darin besteht, eine vorgegebene Zahl von Produkten hinsichtlich einer optimalen, kürzesten Gesamtdurchlaufzeit auf verschiedene Maschinen einzuplanen.

Derzeit befindet sich JobShop 2.1 in der Methodenbank OR_ALPHA und ist unter Windows 7,8 und 10 lauffähig. Die Methode wird in 32- und 64Bit Version zur Verfügung gestellt und wird über die entsprechende Exe-Datei gestartet.

Für das Programm JobShop 2.1 gibt es folgende Prämissen:

- maximal 16 Produkte/Maschinen
- maximale Bearbeitungsdauer von 99 Zeiteinheiten eines Produkts auf einer Maschine
- unterstützt nur Natürliche Zahlen
- Fertigstellungszeiträume werden nicht berücksichtigt

Das Interface des Programms soll dem Benutzer die Möglichkeit bieten, neue Maschinen und Produkte anzulegen und die entsprechenden Durchlaufzeiten zu hinterlegen. Es soll die Möglichkeit bestehen, die LP-Modelle abzuspeichern bzw. zu laden (z.B. Lpi- oder txt-Datei). Über eine Schnittstelle zu einem Solver (*lpsolve55j.jar*) soll ein generiertes LP-Modell gelöst werden.

Das Ergebnis wird dem Benutzer in einem neuen Fenster textuell oder graphisch angezeigt. Durch die Angabe eines Dateiverzeichnisses soll das Ergebnis als XML-, TXT-, CSV- oder ähnlichem Dateiformat dauerhaft gesichert werden können. Außerdem wird eine Eingabeüberprüfung insb. bei der Eingabe der Durchlaufzeiten angestrebt. (siehe Prämisse) Dadurch sollen Falscheingaben vorgebeugt und die Software benutzerfreundlicher gestaltet werden.

2. PROJEKTORGANISATION

Im Folgendem werden die technische und organisatorische Details erläutert, sowie die im Pflichtenheft festgehaltene Änderungsanforderungen vorgeführt. Die in diesem Abschnitt erwähnten Funktionalitäten dienen der langfristigen Orientierung hinsichtlich der Weiterentwicklung von JobShop 2.1

Nachdem die Aufgabenstellung klar definiert war, stellte sich die Frage welche organisatorische und technische Einflussfaktoren das neue Projekt beeinflussen oder einschränken werden.

Der erste Aspekt, welcher zu beachten war, die Einhaltung aller Abgabetermine. Der Projektzeitraum belief sich von 17.03.16 bis 8.06.16

Die Weiterentwicklung sollte in Programmiersprache *Java* mit Entwicklungsumgebung *Eclipse* erfolgen. Das Projekt endete mit einer Präsentation des Programms, inklusive der gesamten Dokumentation durch die Abnahme von Herrn Prof. Dr. Grütz und Serkan Önnisan.

Im Folgenden werden die Änderungsaufforderungen anhand der Bilder der Programminterface dargestellt.

Das Tool JobShop 2.1 bei Übernahme:

	Maschine1	Maschine2
Produkt1	0	0
Produkt2	0	0

Abbildung 1: Programminterface

Getestet und geändert wurde die Software nur unter Windows. Die Lauffähigkeit unter MAC konnte nicht gewährleistet werden.

Die Software besteht aus insgesamt 10 Klassen. Um die Aufbauarchitektur der Software besser zu verstehen, wird im Folgendem die Beziehung der Klassen zueinander näher beschrieben.

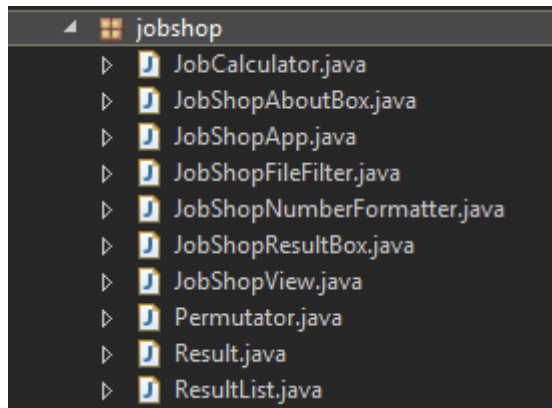


Abbildung 2: Java-Ausschnitt (Klassen)

Das Hauptfenster des JobShops ist die Klasse **JobShopView**. Hier können die Bearbeitungsdauern der Produkte für jede Maschine eingetragen und geändert werden. Der **JobShopNumberFormatter** kontrolliert dabei, dass es sich bei den Einträgen um Zahlen zwischen 0 und 99 handelt.

Beim Klick auf den Berechnen-Button wird der **JobCalculator** aufgerufen. Hier wird das LP-Model erstellt und ein optimales Ergebnis mit dem IpSolve berechnet. Die Klasse ResultList speichert die Ergebnisse, Das Ergebnis, welches Produkt wird zu welcher Zeit an welcher Maschine hergestellt(**Result**) wird in der Klasse **ResultList** abgespeichert. Die ResultList wird dann an die **JobShopResultbox** übergeben, welche dann das Ergebnis anzeigt.

Der **JobShopFileFilter** hat die Aufgabe zu kontrollieren, dass nur Text-Dateien gespeichert und geladen werden, deren Bezeichnung mit „.job“ endet. Die **JobShopAboutBox** gibt die grundlegende Metainformationen über Version, Autor und Erstellungsdatum.(Impressum)

Die **Permutator**-Klasse ist ein erster Entwurf, um das Job-Shop-Problem so zu lösen, wie in der alten JobShop-Version².

² vgl. Dokumentation – JobShopNeu.docx (21.01.2015)

Das neue JobShop 2.2:

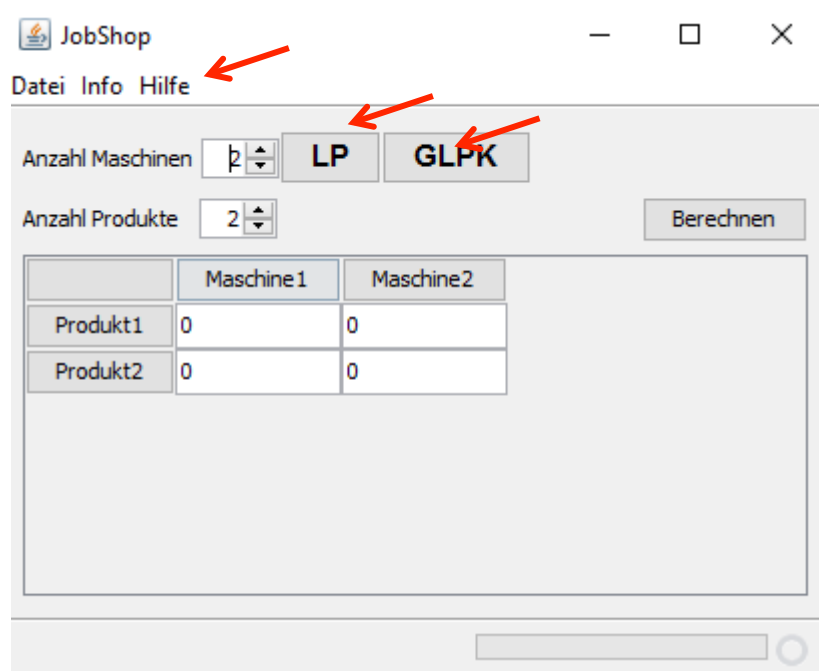


Abbildung 3: JobShop2.2-Interface (neu)

Wie in der oberen Abbildung zu sehen ist, sind ein paar neue Funktionen dazugekommen. Die Hilfefunktion erscheint in der oberen Leiste und in der Benutzeroberfläche kommt ein neuer Solver zum Einsatz (GLPK Solver). Der Benutzer soll dadurch die Möglichkeit bekommen zwischen beiden Solvern *LP* und *GLPK* zu wechseln. GLPK bietet eine graphische Benutzeroberfläche zum Bearbeiten und Lösen eines beliebigen LP-Problems. Das Lösen des Problems erfolgt durch das GNU Linear Programming Kit:

- **GlpkGui.exe**
- **GlpkSharp.dll** (Anbindung an das Code)

Das komplette GLPK Paket sowie Dokumentation und „Downloadings“ sind unter diesem Link zu finden: <http://www.gnu.org/software/glpk/>

Es sind einige technische Probleme bei der Umsetzung der Anforderungen aufgetreten, welche den Implementierungsprozess zum großen Teil beeinflussen. Im Folgenden werden diese Probleme mit dem dazugehörigen Lösungsansatz genauer beschrieben.

2.2.1 TECHNISCHE PROBLEME

1) Fehlermeldung bei der Ausführung in Eclipse

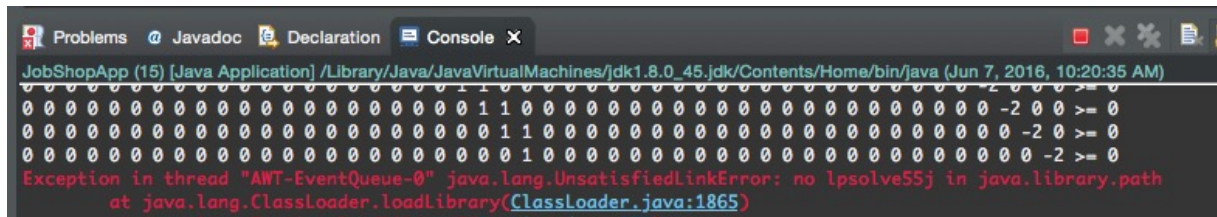


Abbildung 4: Error-Meldung bei der Ausführung des Programmes

Bei der Ausführung des Programmes in eclipse ist ein Error-Meldung von **Java.lang.UnsatisfiedLinkError: no lpsolve55j in java.library.path aufgetreten**. Laut der vom Vorgänger erstellten Dokumentation gab dieses Problem im vorherigen Semester. Nach einigen Versuchen sowohl mit einem der vorherigen Entwickler als auch mit externen Programmierern wurde dieses Problem nicht erfolgreich gelöst.

2) Direkte Änderungen bei Hilfe-Funktion sind in Eclipse nicht möglich

Die Umsetzung der aktuellen Hilfe-Funktion wird in dem weiteren Kapitel genauer erklärt. Es geht teilweise darum, Inhalte der alten Hilfe-Funktion vollständig zu machen und zu verbessern. Um das umzusetzen müssen die bestehende Datei innerhalb der JobShopGuide.jar Datei aktualisiert werden und einige neue Datei müssen dazu addiert werden.

Eine direkte Aktualisierung der JobShopGuide.jar im Eclipse führt zu folgender Error- Meldung.

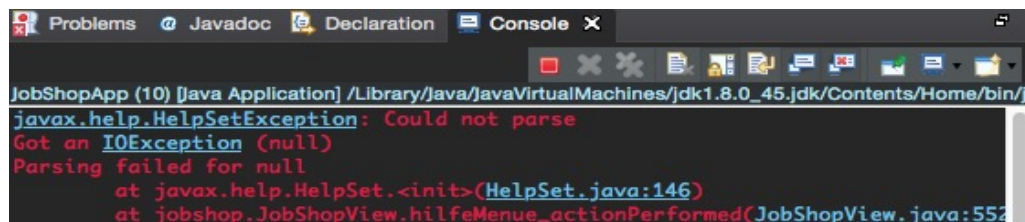


Abbildung 5: Error-Meldung bei Änderung von Hilfe-Funktion

Im Folgenden sind die Lösungsansätze der oben genannten Probleme:

1) Lösungsansatz zu Problem 1

Um das Programm lauffähig zu machen müssen folgende Schritte gemacht werden:

1.Schritt : eine neue .jar Datei vom Eclipse erstellen,

2.Schritt: die .jar Datei im Ordner 01 Programm mit der neugestellten Datei vom Eclipse einsetzen, und

3.Schritt: Die neue .jar Datei von diesem Ordner ausführen.

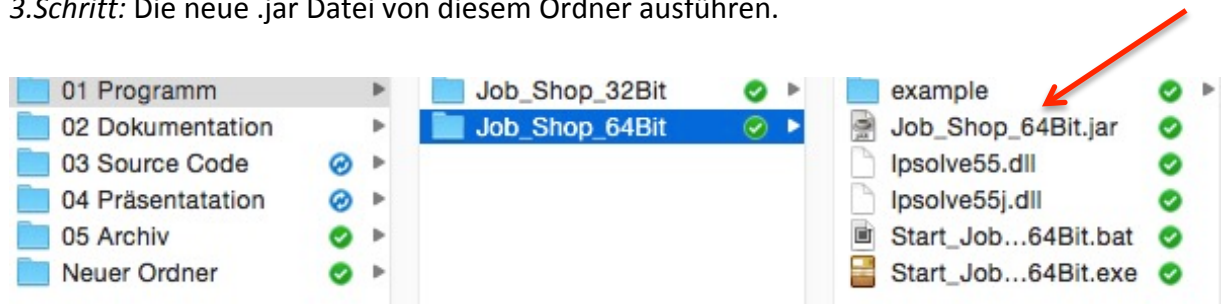


Abbildung 6: Programm mit .jar Datei ausführen

Das heißt, um jede Änderung testen zu können müssen die oben genannten Schritte gemacht werden.

2) Lösungsansatz zu Problem 2

Der Lösungsansatz zu diesem Problem wird in Kapitel 3.1 Hilfe-Funktion genauer beschrieben.

3. UMSETZUNG

In diesem Kapitel wird detaillierte Umsetzung jeder Änderungsanforderung genauer erklärt. Wie vorhin im Kapitel 2.2 erwähnt, sind es insgesamt vier zu implementierenden Anforderungen: Hilfe Funktion, Load/Save Funktion, LP-Ansatz und Matrixanalyse und GLPK Solver. Bei Hilfe Funktion und Load/Save Funktion handelt es sich im großen Teil um technische Änderungen am Source Code. Als nächstes wird eine theorie-basierte Analyse von dem im Code bestehenden Matrixkonzept –welches das LP-Ansatz ausliefert- durchgeführt. Zu guter Letzt wird das allgemeine Konzept dargestellt, wie mehrere Solver im JobShop 2.2 aufgebaut und implementiert wurden. Aufgrund technischer Problemen wurde die Umsetzung von GLPK-Solver nicht erfolgreich implementiert. In diesem Abschnitt werden die aufgetretenen Probleme bei der Umsetzung von GLPK Solver präziser detailliert.

3.1 HILFE-FUNKTION

Die erste Aufgabe ist eine gültige Hilfe-Funktion zu implementieren. In der Version 2.1 befindet sich die Hilfe-funktion unter dem Menu-Bar Info.

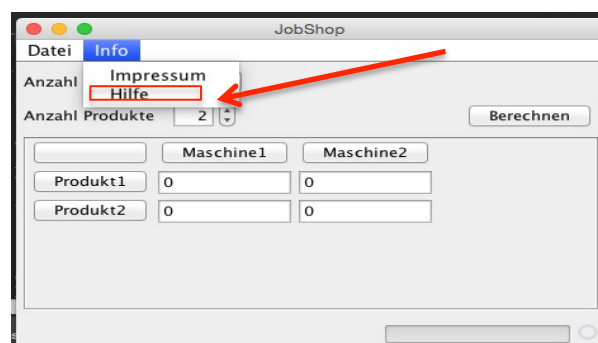


Abbildung 7: Darstellung von Hilfe-Funktion im JobShop 2.1

Die Inhalte der bestehenden Hilfe-Funktion sind mehr als technische Dokumentation anstatt als Hilfe und Informationen zur Bedienung der Methode angesehen.

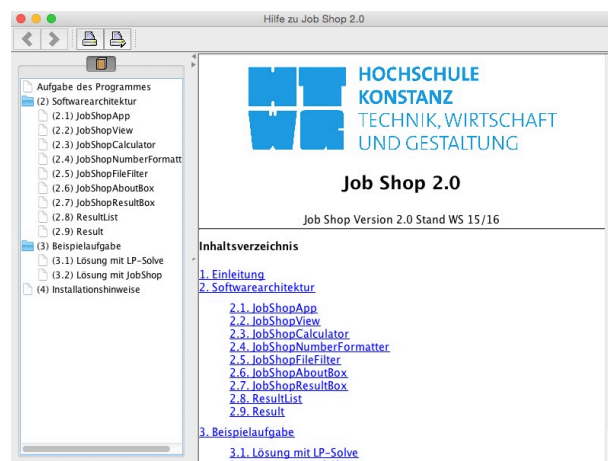


Abbildung 8: Inhaltsverzeichnis von Hilfe-Funktion im JobShop 2.1

Aus Sicht der Benutzerfreundlichkeit wurde die Hilfefunktion in der neuen Version als eigenes Menu-Bar anstatt als Teil von Info dargestellt. Für die Umsetzung dieses neuen Konzepts wurden folgende Änderungen erfordert:

1) Erstellung von einem neuen Text im JobShopView.properties

Hilfefunktion.text definiert den Name des zu implementierenden Menu-Bar, welches als Hilfe genannt wurde.

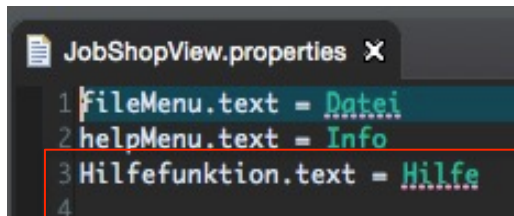


Abbildung 9: Erstellen vom Text für die Hilfe-Funktion

2) Erzeugen eines neuen JMenu im JobShopView.java

Zum erzeugen eines neuen Menu-Bar im GUI Fenster wurde ein neues JMenu in der JobShopView.Java erstellt.

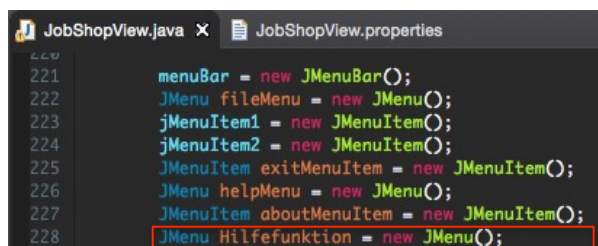


Abbildung 10: Erstellen von einem JMenu für die Hilfe-Funktion

3) Aufgerufene Methode in JobShopView.java zuordnen

Hier wird die zugeordnete Methode aufgerufen, die die Hilfe-Funktion als neues Fenster im GUI eröffnet.

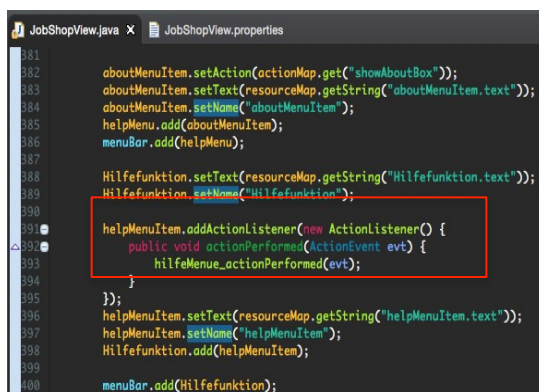


Abbildung 11: Zuordnen von Methode der Hilfe-Funktion

Wir vorhin erwähnt müssen die Inhalte der Hilfe-Funktion in der alten Version geändert werden.

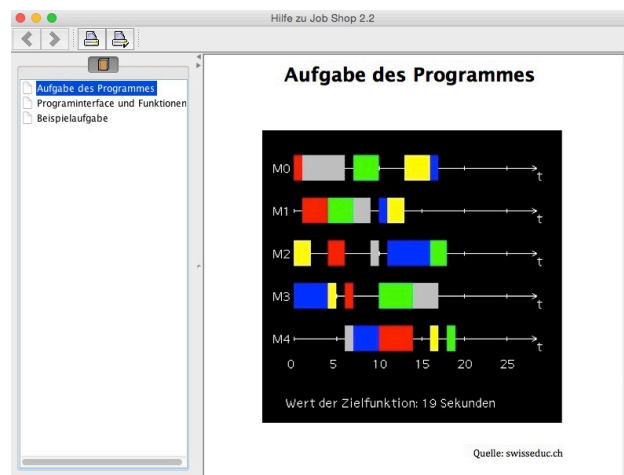


Abbildung 12: Darstellung von Hilfe-Funktion im JobShop 2.2

Die Inhalte der neuen Hilfe-Funktion bestehen aus drei Teilen:

1) Aufgabe des Programmes

Damit Benutzer ein gewisses Überblick über das Programm erhalten kann, wird die allgemeine Funktion des Programmes in diesem Teil erklärt.

2) Programminterface und Funktionen

Um die einzelnen Funktionen des Programmes zu verstehen und das Programm leicht zu bedienen werden in diesem Teil das Interface und die Funktionen des Programmes genauer erklärt.

3) Beispielaufgabe

Eine Beispielaufgabe wird den Benutzern helfen, das Programm besser zu verstehen. Die in diesem Teil dargestellte Beispielaufgabe ist eine Aufgabe im Thema JobShop, die aus dem Skript Operation Management von Herrn Prof. Grütz genommen wurde.

Bei den Änderungen von Inhalten waren einige technische Probleme aufgetreten, die einen sehr zeitaufwendigen Aufwand verursacht. Alle Inhalte von Hilfe-Funktion werden in einer .jar Datei gespeichert. Die JobShopGuide.jar Datei besteht aus verschiedenen Dateien, die die Inhalte der Hilfe-Funktion beinhalten.

Das größte Problem war, dass eine direkte Änderung aus unbenannten Gründen im Eclipse nicht machbar ist und somit mussten alle Dateiänderungen manuell im Terminal eingetragen werden. Nachdem jede benötigte Datei geändert bzw. erstellt wurde, mussten die JobShopGuide.jar neu aktualisiert und im Programm getestet werden. Dieser Schritt würde viel schneller sein, wenn Änderungen direkt im Eclipse gemacht werden könnten.

Im Folgenden sind dargestellte Ausschnitte von Änderungsschritten für die Inhalte der neuen Hilfe-Funktion

1) JobShopGuide.jar im Referenced Libraries

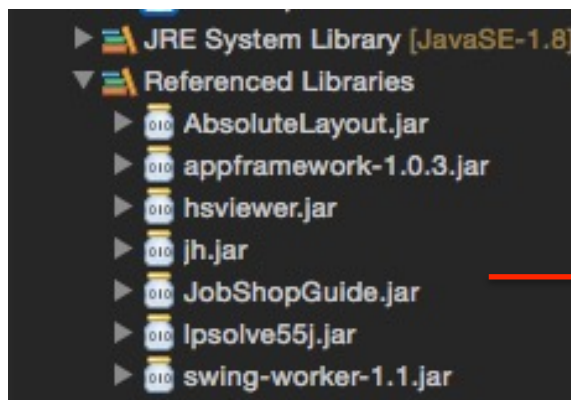


Abbildung 13: Referenced Libraries vom JobShop 2.1

2) zu geänderte Dateien

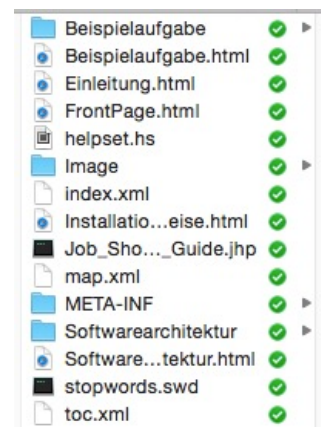


Abbildung 14: Inhalte der JobShopGuide.jar vom JobShop 2.1

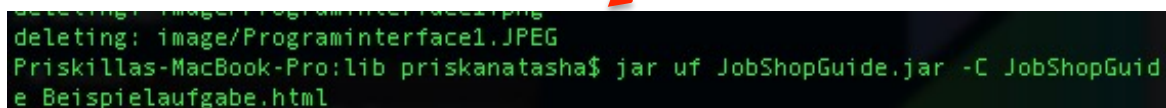


Abbildung 15: Aktualisierung von Beispielaufgabe.html durch Terminal

3.2 LOAD/SAVE FUNKTION

Wie bereits im Kapitel 1.2 beschrieben, musste das Problem mit dem richtigen Verzeichnis beim Laden und Speichen gelöst werden. Die Dateien sollten nicht in „Dokumente“ sondern im Ordner „example“ gespeichert und von dort geladen werden.

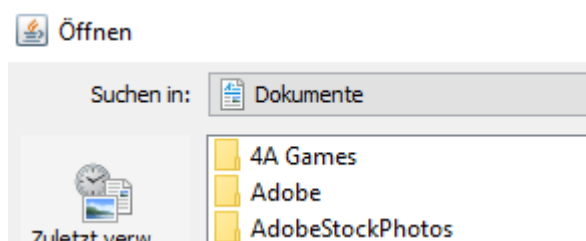
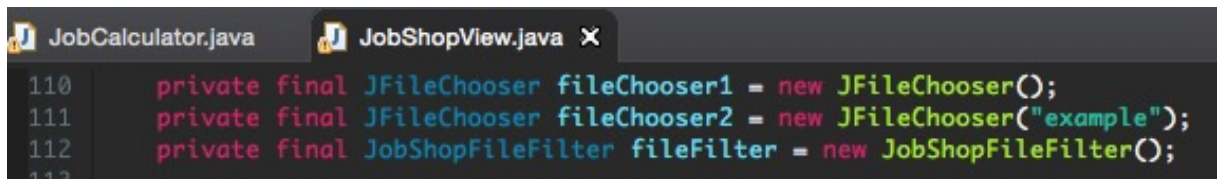


Abbildung 16: Datei öffnen (früher)

Die Klasse, in welcher dies umgesetzt wurde ist „JobShopView.java“ und „JobCalculator.java“ (s.Abb.unten).

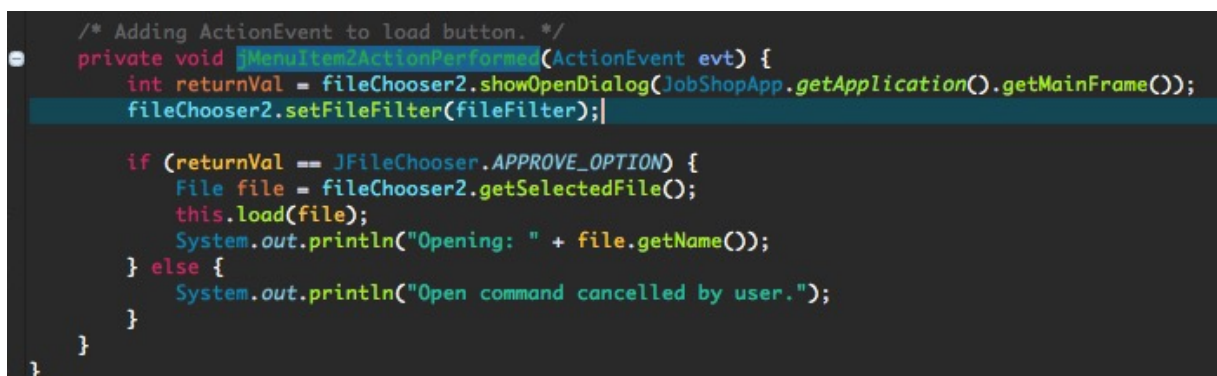
1. Laden



```
JobCalculator.java  JobShopView.java X
110     private final JFileChooser fileChooser1 = new JFileChooser();
111     private final JFileChooser fileChooser2 = new JFileChooser("example");
112     private final JobShopFileFilter fileFilter = new JobShopFileFilter();
113
```

Abbildung 17: Code-Ausschnitt vom Objekt fileChooser2

Das Objekt fileChooser2 mit dem Wert example wird dann der Methode JMenuItem2ActionPerformed übergeben.



```
/* Adding ActionEvent to load button. */
private void JMenuItem2ActionPerformed(ActionEvent evt) {
    int returnVal = fileChooser2.showOpenDialog(JobShopApp.getApplication().getMainFrame());
    fileChooser2.setFileFilter(fileFilter);

    if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser2.getSelectedFile();
        this.load(file);
        System.out.println("Opening: " + file.getName());
    } else {
        System.out.println("Open command cancelled by user.");
    }
}
}
```

Abbildung 18: Code-Ausschnitt von Methode JMenuItem2ActionPerformed

Nach dieser Änderung wird das Ergebnis so angezeigt wie erwartet:

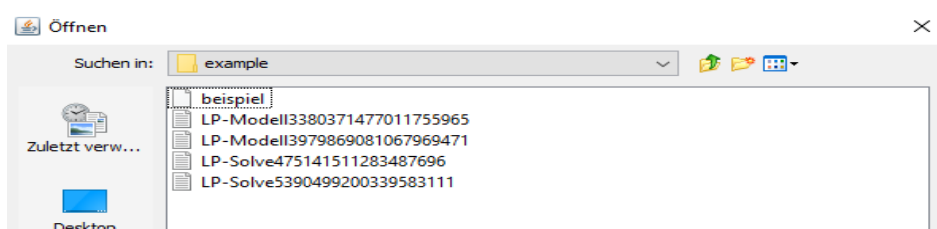


Abbildung 19: Datei öffnen (Nacher)

2. Speichern (in der Klasse JobCalculator)

Bei jeder Speicherung der Dateien wurde die Meldung aufgepoppt, dass die Datei im Ordner Job_Shop_64Bit temporär gespeichert wurde.(siehe Abb.)

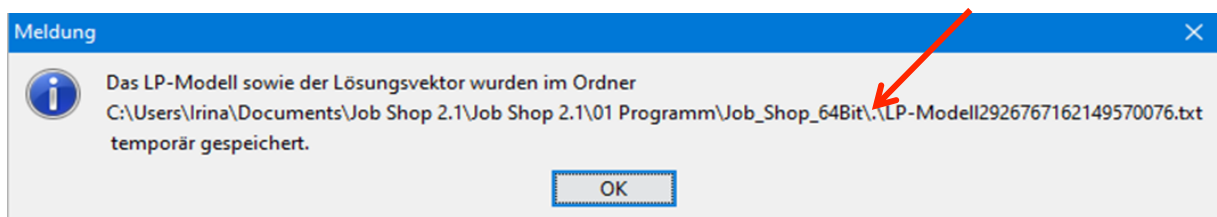


Abbildung 20: temporäres Speichern

Laut funktionalen Anforderungen soll die Meldung nicht mehr erscheinen. Das Problem mit der Meldung wird behoben, indem das `showMessageDialog` ausgeschaltet wurde.

```
//OptionPane.showMessageDialog(null, "Das LP-Modell sowie der Lösungsvektor  
wurden im Ordner \n"+ file.getAbsolutePath() + "\n temporär gespeichert.");  
//file.deleteOnExit();  
//file2.deleteOnExit();
```

Beim Speichern werden 2 .txt-Dateien (LP-Modell und Lp-solve Ergebnis) gespeichert und in den richtigen Ordner abgelegt (bei uns „example“).

Die dafür vorgenommene Änderungen im Code, sehen Sie anhand der untenstehenden Codeausschnitten:

```
public JobCalculator() {  
    this.sumDauer = 0;  
    this.anzM = 0;  
    this.anzP = 0;  
    this.separator = ";";  
    this.dirName = "example";  
}
```

Abbildung 21: richtiges Verzeichnis definieren

Lp-Modell:

```
try {  
    file = File.createTempFile("LP-Modell", ".txt", new File(dirName));  
    writer = new FileWriter(file);  
    for (Map<String, String> map : matrix) {  
        writer.write(map.get("function"));  
        writer.write(map.get("operator") + this.separator);  
        writer.write(map.get("result"));  
        writer.write("\r\n");  
    }  
}
```

Abbildung 22: Speichern vom Lp-Modell

Lp-Solve:

```
try {  
    file2 = File.createTempFile("LP-Solve", ".txt", new File(dirName));  
    solver.setOutputfile("example/"+file2.getName());  
    solver.printObjective(); // print lpsolve solution (output)  
    solver.printSolution(1); // print lpsolve solution (output)  
    solver.printConstraints(1); // print lpsolve solution (output)  
} catch (IOException e) {  
    e.printStackTrace();  
    System.out.println("File konnte nicht erstellt werden!");  
}
```

Abbildung 23: Speichern vom Lp-Solve


Ergebnis:

```
LP-Modell3979869081067969471  
LP-Solve475141511283487696
```

Abbildung 24: Gespeicherte Dateien im Verzeichnis example

Für das Berechnen von Aufgabe mit LP-Solve wird die Eingabe sowohl von Anzahl der Maschinen als auch Anzahl der Produkten durch ein LP-Ansatz gelöst. Die Anforderung besteht darin, das vom Programm erstellte LP-Ansatz zu überprüfen, ob es alle LP-Ansatz Regelungen für JobShop Probleme erfüllen. Um eine korrekte Analyse durchzuführen, wurde das LP Ansatz Matrix nach dem Skript von Herrn Prof. Grütz im Fach Operation Management analysiert. Diese Analyse wurde mit Unterstützung von Herrn Prof. Grütz durchgeführt. Als nächstes werden die einzelnen Schritte der Analyse detailliert.

- 1) Um die Analyse besser darzustellen wird ein einfaches Beispiel genommen

 **JobShop**

Datei Info

Anzahl Maschinen

Anzahl Produkte

	Maschine 1	Maschine 2
Produkt1	1	1
Produkt2	1	1

Abbildung 25: Eingabe vom Beispielaufgabe

- 2) Das LP-Ansatz muss nach dem Skript vom Herrn Prof. Grütz folgende Regel erfüllen
 - a. Es muss Restriktionen geben, die Bearbeitungsdauer der Produkte auf den Maschinen von den Zeitpunkten t_1 bis t_n darstellen
 - b. Eine Maschine kann nur ein Produkt pro Zeitpunkt bearbeiten
 - c. Ein Produkt kann nur auf einer Maschine pro Zeitpunkt bearbeitet werden
 - d. Maschinen sollen ohne Unterbrechung genutzt werden (ein Produkt wird auf einer Maschine möglichst am Stück bearbeitet – Vermeidung von Rüstzeiten und -kosten)
 - e. Anzahl Variable:
$$\text{Anz. Maschine} * \text{Anz. Produkte} * t(\text{Sum Bearbeitungsdauern}) * 2$$
- 3) die im Punkt 1) gegebene Aufgabe wird durch ein LP-Ansatz gelöst. Im Folgenden ist das zu analysierende Matrix des LP-Ansatzes.

[illegible]

Abbildung 26: Matrixergebnis von Beispielaufgabe

Teil 2

[illegible]

Abbildung 28: Matrixergebnis für R24-R46

Analyse von Teil 2:

R24-R46: Hier geht es darum, dass ein Produkt auf einer Maschine möglichst am Stück ohne Unterbrechung bearbeitet wird (Vermeidung von Rüstzeiten und –kosten).

➔ Erfüllen von Regel d)

Als letztes wird die Anzahl der Variablen überprüft

$$\begin{aligned} & \text{Anz. Maschine} * \text{Anz. Produkte} * t(\text{Sum Bearbeitungsauern}) * 2 \\ &= 2 * 2 * 4 * 2 \\ &= 32 \end{aligned}$$

➔ Erfüllen von Regel e)

Nach dieser Analyse wurde zusammengefasst, dass die betroffene Matrix alle Regelungen erfüllt und somit als ein richtiges Matrix bezeichnet wird.

Die Idee von dieser Aufgabe ist es, JobShop Probleme mit verschiedenen Solvern zu lösen und die Ergebnisse von unterschiedlichen Solvern miteinander vergleichen zu können. Aus diesem Grund wird ein allgemeines Konzept abgebildet, dass mehrere Solver in das Programm implementiert werden kann. Mit Switch-Case wird eine Auswahl von Solver benötigt und das Ergebnis von dem ausgewählten Solver ausgeliefert. Der Grund, warum das Switch-Case Konzept genommen wurde, ist damit es flexible ist, mehrere Solver in der Zukunft flexible zu implementieren, da nur ein neues Case erstellt werden muss.

1) Switch-Case Funktion im JobShopView.java

```
//LP-Solverwahl
private void jButton3ActionPerformed(ActionEvent evt) {
    solverWahl = 1;
}

//GLPK-Solverwahl
private void jButton4ActionPerformed(ActionEvent evt) {
    solverWahl = 2;
}

/* Adding ActionEvent to Berechnen-Button. */
private void jButton1ActionPerformed(ActionEvent evt) {

    switch (solverWahl){

        case 1:
            JobCalculator calc = new JobCalculator();
            ResultList result = calc.calculate(jFTFieldList);
            this.showResultBox(result);
            break;

        case 2:
            JOptionPane.showMessageDialog(null,"Das GLPK Solver
            break;

        default:
            JOptionPane.showMessageDialog(null,"Sie haben kein
```

Abbildung 29: Code Ausschnitt vom jButton1ActionPerformed

2) neue LP und GLPK Buttons wurden erstellt und zu dem betroffenen Case zugewiesen

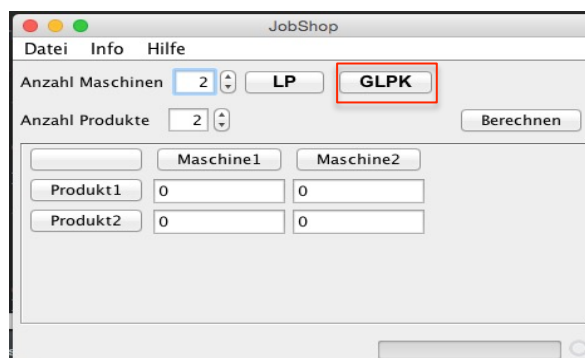


Abbildung 30: Neue Buttons LP und GLPK

Aus folgenden Gründen wurde GLPK Solver nicht erfolgreich implementiert:

1. .dll file ist in eclipse nicht ausführbar

Da das Programm selber aufgrund nicht lauffähiger LPSolve55j.dll Datei in eclipse nicht ausführbar ist war eine Implementierung von einem neuen Solver schwer zu realisieren. Das Problem liegt daran, dass kontinuierliches Testen nach jeder Änderung in Eclipse nicht möglich ist.

2. MPS-Format nicht vorhanden

Das *Mathematical Programming System (MPS)* ist ein standardisierte spaltenorientierte Datei Format, das Probleme bei Lineare Programmierung definiert und von fast allen kommerziellen LP Solver gelesen werden kann. Im JobShop Programm ist dieses Format nicht vorhanden und somit ist es sehr aufwendig, neuen Solver zu implementieren.

3. Code Layout-Vorlage für JobShop Probleme wurde nicht gefunden,

Es wurde keine ausführliche Code Layout-Vorlage gefunden, die JopShop Probleme mit GLPK Solver in Java Programmierungssprache lösen. Soche Vorlage würde als alternativ sehr hilfreich sein, GLPK Solver im JobShop zu implementieren.

4. FAZIT

Die Lehrveranstaltung Anwendung der Linearen Optimierung ermöglicht uns erste Einblicke in die Planung, Modellierung und Implementierung von anwenderfreundlichen Problemlösungen auf der Grundlage von Modellen der Linearen Programmierung.

Durch die Bearbeitung verschiedener Problemstellungen in Form von Teamprojekten, konnten wir bereits gelerntes direkt anwenden und so weitere praktische Erfahrungen sammeln.

Nach genauer Auseinandersetzung mit dem JobShop Tool, sind drei Entwicklungsmöglichkeiten von uns durchaus denkbar:

- Einen neuen Solver zu integrieren
- Felderwechsel durch Pfeiltasten
- Eine schönere Darstellung des Ergebnisses

Es lässt sich feststellen, dass die Projektarbeit sehr interessant war. Da es sich um ein Java-Projekt handelte, konnten wir zusätzlich unsere Programmier- und Softwaremodellierungskenntnisse erweitern. Im Team, lernten wir für auftauchende Problemstellungen neue Alternativen auszuarbeiten und umzusetzen.

Als einziger Kritikpunkt, der unsere Arbeit erschwert hat, sehen wir die Fehler und Exceptions in Eclipse.