Universität Duisburg-Essen

Virtueller Weiterbildungsstudiengang Wirtschaftsinformatik (VAWi)

Projektarbeit im Modul „Modelle & Methoden der Entscheidungsunterstützung“

Implementierung des Simplex-Algorithmus mit didaktisch aufbereiteter Ausgabe

Implementation of the Simplex Algorithm with Educational Structured Output

Vorgelegt der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen

Verfasser: **Wolfgang Bongartz**
Von-Stauffenberg-Str. 8
53757 Sankt Augustin
Matrikelnummer: 2222059

Erstgutachter: Prof. Dr. Natalia Kliewer, Freie Universität Berlin
Zweitgutachter:

Abgabe : 23.05.2016 / Sommersemester 2016

INHALTSVERZEICHNIS

Verzeichnis der eingesetzten Hilfsmittel

- **JavaCC**
  Java Compiler Compiler, SF JavaCC Eclipse Plug-in, Version 1.5.33
  https://javacc.java.net/, http://eclipse-javacc.sourceforge.net/,
  https://de.wikipedia.org/wiki/JavaCC

- **JavaDoc**
  https://de.wikipedia.org/wiki/Javadoc

- **JUnit**
  http://junit.org/, https://de.wikipedia.org/wiki/JUnit

- **Eclipse**
  Eclipse IDE for Java Developers, Version 4.4.2 („Luna“)
  https://eclipse.org/, https://de.wikipedia.org/wiki/Eclipse_%28IDE%29

- **commons-math**
  The Apache Commons Mathematics Library
  http://commons.apache.org/proper/commons-math/

1 Überblick

Diese Projektarbeit besteht im Wesentlichen aus einem in Java erstellten Programm, das bestimmte Optimierungsprobleme durch Anwendung des Simplex-Algorithmus lösen kann. Die Benutzer des Programms können den Algorithmus schrittweise durchlaufen. Das Programm zeigt dabei jedes Zwischenergebnis an. Auf diese Weise soll das Erlernen der Anwendung des Simplex-Algorithmus erleichtert werden.

Das Programm basiert auf dem Vorlesungsskript [OR-Skript] und den Büchern [Suhl] und [Ellinger]. Die im Programm und in diesem Dokument verwendete Terminologie orientiert sich am Vorlesungsskript. Die dem Programm im Verzeichnis ‚examples' beigefügten Optimierungsprobleme entstammen größtenteils ebenfalls [OR-Skript] und [Suhl][1].

1.1 Die Bestandteile dieser Projektarbeit

- Die Quelltexte des Programms (im Unterverzeichnis /src)[2]
- Dateien mit diversen Optimierungsproblemen (im Hauptverzeichnis; Endung: *.lpp)
- Dokumentation der Quelltexte (im Unterverzeichnis /doc; Startpunkt: ‚index.html')[3]
- Ein CSS-Stylesheet, mit dem sich die Anzeige der Lösungsschritte konfigurieren lässt (‚/ressources/SimplexSolver.css')
- Skript-Dateien zum Starten des Programms (im Hauptverzeichnis)
- Dieses Dokument

1.2 Starten und Benutzen des Programms

Ein Doppelklick auf eins der folgenden Skripte startet das Programm:

- SimplexSolver.bat (MS-Windows)
- SimplexSolver.command (Mac OS X)
- SimplexSolver.sh (UNIX, Linux)

Nach dem Start lädt das Programm das zuletzt verwendet Optimierungsproblem. Mit dem Dateimenü lassen sich Optimierungsprobleme laden, speichern und neu anlegen. Hier findet sich auch der Menüpunkt zum Verlassen des Programms.

Im linken Bereich des Fensters kann das gerade geladene Optimierungsproblem editiert werden. Unterhalb dieses Bereichs befinden sich die Buttons, mit denen sich der Ablauf des Simplex-Algorithmus steuern lässt:

- 'Lösen': Alle zur Lösung des Optimierungsproblems notwendigen Schritte werden durchgeführt und im rechten Bereich des Fensters angezeigt
- 'Nächster Schritt': Nur der nächste zur Lösung des Optimierungsproblems notwendige Schritt wird durchgeführt und im rechten Bereich des Fensters angezeigt

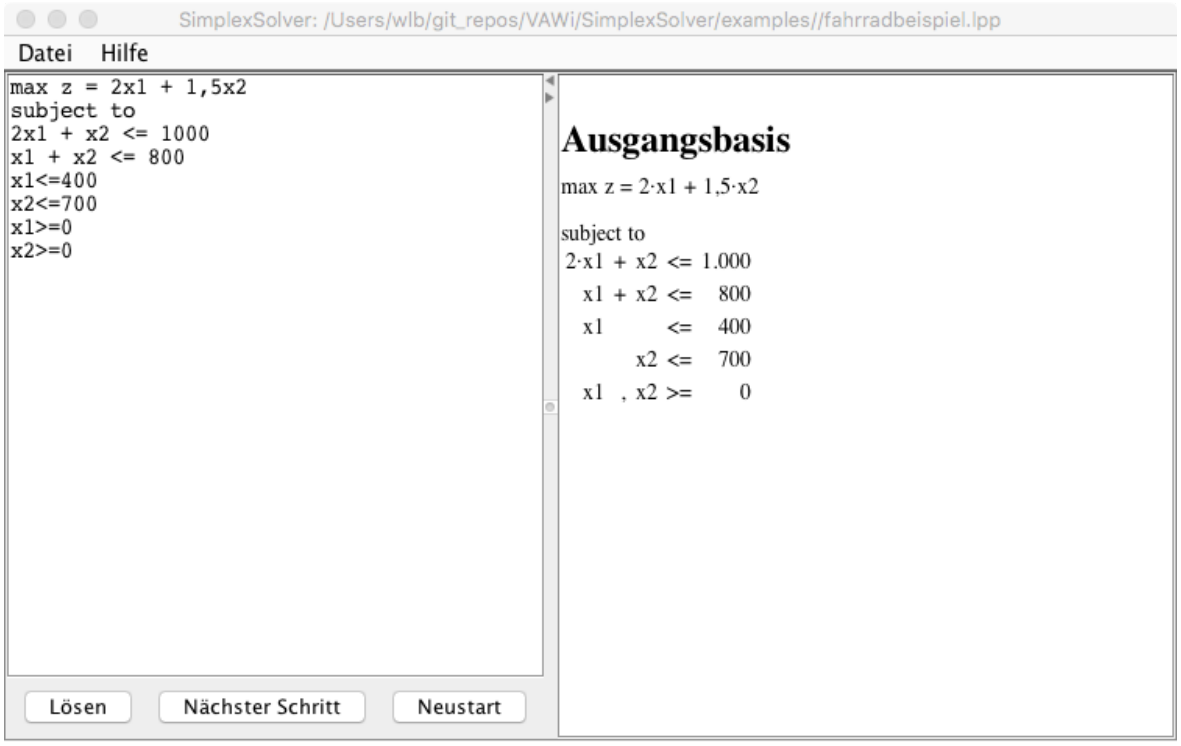
---

[1] Einige Beispiele wurden auch den der während der VAWi-Veranstaltung ‚Operations Research' im Wintersemester 2015/2016 ausgegebenen Teilleistungs-Aufgaben entnommen.

[2] Dazu zählen alle Java-Quelltexte (*.java) mit Ausnahme der Dateien im Unterverzeichnis /srv/parser. Letztere wurden mit dem Tool ‚JavaCC' aus dem Quelltext ‚Grammar.jj' generiert.

[3] Die HTML-Dokumentation wurde mit dem Tool JavaDoc aus den in den Java-Quelltexten hinterlegten Kommentaren generiert.

- 'Neustart': Der rechte Fensterbereich wird gelöscht. Der Simplex-Algorithmus kann erneut durchlaufen werden



1.3 Darstellungsform

In der Fachliteratur und im Internet existieren verschiedene Darstellungsformen des Simplex-Algorithmus. Die Darstellung im Programm orientiert sich an der im Vorlesungsskript [OR-Skript] und im Buch [Suhl] verwendeten Form. Das Programm ist jedoch so aufgebaut, dass auch andere Darstellungsformen bei Bedarf mit relativ geringem Aufwand implementiert werden können.

2 Die Arbeitsweise des Programms

2.1 Schritt 1: In interne Darstellung übersetzen

Sobald der Benutzer den Programmablauf startet (durch Klick auf einen der Buttons ‚lösen' oder ‚nächster Schritt'), wird versucht, die in Textform vorliegende Problembeschreibung aus dem linken Fensterbereich in eine für das Aufstellen der Ausgangsbasis geeignete interne Darstellung zu überführen.

Dazu wird der Text zunächst vom Parser bearbeitet. Der Parser prüft, ob der Text den Regeln für die Formulierung eines Optimierungsproblems genügt[4]. Wenn das der Fall ist, liefert er ein Objekt der Klasse `LPProblem` zurück. Diese Klasse stellt die interne Darstellung eines LP-Problems dar.

[4] Diese Regeln sind in der Datei ‚Grammar.jj' formuliert.

2.2 Schritt 2: Umwandeln in Standardgleichungsform

Es folgt dann das schrittweise Umwandeln in die Standardgleichungsform. Also das umformen der Zielfunktion von einer Minimierungs- in eine Maximierungsfunktion. Das Umwandeln von Gleichung-Restriktionen. Das Umwandeln von „Größer-als-Restriktionen“ und das Einführen von Schlupfvariablen. All diese Umwandlungsschritte werden letztlich von Methoden der Klasse `LPProblem` erledigt.

2.3 Schritt 3: Aufstellen der Startbasis

Programmintern entspricht dem Aufstellen der Startbasis die Überführung des `LPProblem`-Objekts in ein Objekt der Klasse `SimplexProblem`. Während die Klasse `LPProblem` darauf spezialisiert ist, die oben beschriebenen Umwandlungsschritte durchzuführen, ist die Klasse `SimplexProblem` darauf spezialisiert, die Iterationen des Simplexverfahrens durchzuführen.

2.4 Schritt 4: Phase I durchführen

Falls in Schritt 3 keine zulässige Startbasis gebildet werden konnte, wird Phase I durchgeführt, indem das Objekt der Klasse `SimplexProblem` in ein Objekt der Klasse `SimplexProblemPhase1` überführt wird. Dabei wird implizit das nötige Hilfsproblem aufgestellt. Es folgen dann die zur Lösung des Hilfsproblems nötigen Iterationsschritte, die alle als Methoden der Klasse `SimplexProblemPhase1` implementiert sind[5]. Die so gefundene Startbasis wird dann wieder in ein Objekt der Klasse `SimplexProblem` überführt.

2.5 Schritt 5: Phase II durchführen

Nun werden die Schritte ‚Pivot-Spalte suchen’, ‚Pivot-Zeile suchen’ und ‚Basistausch durchführen’ solange wiederholt, bis der Simplex-Algorithmus terminiert. Auch diese drei Schritte sind als Methoden der Klasse `SimplexProblem` implementiert.

3 Der innere Aufbau des Programms

Wie in der objektorientierten Programmierung üblich ist das Programm so gestaltet, dass seine einzelnen Komponenten möglichst unabhängig voneinander verwendbar sind. Zum einen vereinfacht dies die Fehlersuche, zum anderen können die einzelnen Komponenten auf diese Weise auch in anderen Programmen benutzt werden. Die folgenden Abschnitte beschreiben die Komponenten und ihr Zusammenspiel, wobei nur die für das Verständnis unbedingt nötigen Aspekte beschrieben werden. Die einzelnen Klassen und ihre Methoden sind in Form von Quelltext-Kommentaren dokumentiert[6].

---

[5] Das ist eigentlich nicht ganz korrekt, denn die Klasse `SimplexProblemPhase1` ist von der Klasse `SimplexProblem` abgeleitet und erbt diese Methoden daher von `SimplexProblem`. Für die Beschreibung des Programmablaufs ist dies jedoch unerheblich.

[6] Diese Dokumentation wurde mit dem Tool JavaDoc im HTML-Format aufbereitet und steht im Unterordner /doc auch separat zur Verfügung.

3.1 Das Paket ‚process'[7]

Die meisten mit einem Computer simulierbaren Prozesse bestehen aus einzelnen Prozessschritten, die in einer definierten Reihenfolge durchlaufen werden müssen. Die Grundidee hinter den Klassen in diesem Paket ist es, diese Prozessschritte jeweils als separate Klasse zu implementieren und die Ablaufsteuerung in eine zentrale Klasse auszulagern. So lassen sich Prozesse flexibel aus einzelnen Komponenten zusammensetzen und schrittweise durchführen. Um auch in anderen Konstellationen einsetzbar zu sein haben die Klassen in diesem Paket keinen konkreten Bezug zum Simplex-Algorithmus.

Die Prozess-Steuerung übernimmt eine Instanz der Klasse `ProcessMaster`[8]. Sie stellt Methoden bereit, mit denen sich der Prozessablauf definieren lässt und sie kontrolliert die Durchführung des Ablaufs. Dabei kann der Ablauf entweder schrittweise oder im Ganzen erfolgen. Außerdem verwaltet die Klasse auch den aktuellen Zustand des Prozesses.

Nach der Durchführung jedes einzelnen Prozessschrittes und bei jeder Änderung des Prozesszustandes wird ein Event versendet. Diese Events werden im Programm verwendet, um den Aktivierungszustand der Buttons in der GUI zu steuern[9] und um das Ergebnis des letzten durchgeführten Simplex-Schrittes auszugeben.

Alle Klassen, die einen konkreten Prozessschritt implementieren, müssen das Interface `ProcessStep` erfüllen.

3.2 Das Paket ‚process.statements'

Dieses Paket beinhaltet Klassen, die bestimmte Kontrollstrukturen implementieren, welche dann in konkreten Prozessen verwendet werden können. Beispielsweise Schleifen und If-Then-Else-Anweisungen.

3.3 Das Paket ‚parser'

Ein Parser ist eine Softwarekomponente, die einen Quelltext in ein Format übersetzt, welches von einem Computer verarbeitet werden kann. Der Quelltext muss dazu einer definierten Syntax folgen. Im vorliegenden Programm gibt der Benutzer das zu lösende LP-Problem in Textform ein. Die Klassen im Paket 'parser' überführen diesen Text in ein Objekt der Klasse `LPProblem`.

Ein Parser, der in der Lage ist ein als Text formuliertes LP-Problem zu übersetzen ist relativ umfangreich und besteht aus mehreren Komponenten. Ein solcher Parser wird in einem recht schematisch ablaufenden Verfahren erstellt. Deshalb ist es möglich, den Parser automatisiert erzeugen zu lassen. Hier wurde dazu das Tool JavaCC verwendet, mit dem alle Java-Quelltexte im Paket ‚parser' erzeugt wurden[10]. Grundlage für die automatische Erzeugung ist die Syntax-Definition, die in der Datei ‚Grammar.jj' abgelegt ist.

---

[7] Der Quellcode ist im Unterverzeichnis ‚/src/process' abgelegt.

[8] Tatsächlich ist diese Klasse als 'abstract' gekennzeichnet. Für die Abbildung eines konkreten Prozesses muss also eine von dieser Klasse abgeleitete Klasse implementiert werden. Im hier vorliegenden Programm ist dies die Klasse `ProzessManager` im Paket ‚solver'.

[9] Also beispielsweise um den Button 'Lösen' auf 'inaktiv' zu schalten, sobald das Problem vollständig gelöst ist.

[10] Die Java-Quelltexte in diesem Paket wurden also wie bereits erwähnt ‚generiert'.

3.4 Das Paket ‚lp_problem'

Die Klassen in diesem Paket dienen dazu, ein lineares Optimierungsproblem abzubilden. Der Schwerpunkt liegt dabei auf der Umformung in die Standarddarstellung.

Ein lineares Optimierungsproblem besteht aus Sicht des Programms aus zwei Komponenten:

- Eine Zielfunktion (Klasse `TargetFunction`)
- Eine Menge von Restriktionen (Klasse `Restriction`)

Eine Zielfunktion wiederum besteht aus einer Linearkombination (Klasse `LinearCombination`). Sie ist entweder eine Minimierung- oder eine Maximierungsfunktion (Klasse `TargetFunctionType`) und sie beinhaltet einen Bezeichner für den Zielfunktionswert[11].

Eine Restriktion besteht ebenfalls aus einer Linearkombination. Außerdem beinhaltet sie ein Operator (Klasse `Operator`), einen konstanten Wert (auf der rechten Seite des Operators) und gegebenenfalls eine Schlupfvariable[12].

Eine Linearkombination setzt sich aus einer Menge von variablen Komponenten (Klasse `VarComponent`) und gegebenenfalls einem konstanten Wert zusammen[13].

Eine variable Komponente besteht aus einem Koeffizienten und einer Entscheidungsvariablen.

3.5 Das Paket ‚simplex_problem'

Wie die Klassen im Paket lp_problem bilden auch die Klassen dieses Pakets ein lineares Optimierungsproblem ab. Hier bildet jedoch die schrittweise Lösung des Problems mithilfe des Simplex-Algorithmus den Schwerpunkt. Dazu muss das Problem bereits in die Standardgleichungsform überführt worden sein.

Die Klasse `SimplexProblem` beinhaltet alles, was für das durchlaufen der Phase II des Simplex-Algorithmus nötig ist. Dazu wird das Problem in Form einer Matrix dargestellt. Die Zielfunktion befindet sich in Zeile 0 dieser Matrix. Die Restriktionen folgen in den Zeilen 1 bis n. Jede Zeile bildet die betreffende Linearkombination ab. In der Spalte 0 befinden sich die jeweiligen konstanten Werte. In den Spalten 1 bis n folgen die Koeffizienten der Entscheidung- und Schlupfvariablen. Auf dieser Basis kann das Problem durch einfache Matrizenoperationen gelöst werden.

Aus Sicht des Programms unterscheidet sich die Phase I des Simplex-Algorithmus nicht wesentlich von Phase II. Denn letztlich muss nur das Hilfsproblem aufgestellt und die ursprüngliche Zielfunktion in der Matrix untergebracht werden. Daher wird die Phase I in der Klasse `SimplexProblemPhase1` implementiert. Diese Klasse ist von der Klasse `SimplexProblem` abgeleitet und überschreibt lediglich einige wenige Methoden, die in den beiden Phasen unterschiedlich arbeiten. Die ursprüngliche Zielfunktion wird dabei in Zeile 1 der Matrix abgelegt und bei allen Operationen wie eine Restriktion behandelt.

---

[11] Im Beispiel `max z = 2x1 + 3x2 – 3` ist `2x1 + 3x2 –3` die Linearkombination, `max` ist der Zielfunktionstyp und `z` ist der Bezeichner des Zielfunktionswerts.

[12] Im Beispiel `2x1 + x2 <= 1000` ist `2x1 + x2` die Linearkombination, `<=` der Operator und 1000 ist der konstante Wert.

[13] Im Beispiel `2x1 + 3x2 – 3` sind `2x1` und `3x2` variable Komponenten und `3` ist der konstante Wert.

3.6 Das Paket ‚solver'

Die Klassen in diesem Paket bringen die Funktionalität der Klassen aus dem Paket ‚process' mit der Funktionalität der Klassen aus den Paketen ‚lp_problem' und ‚simplex_problem' zusammen. Denn sie enthalten die eigentliche Implementation des Simplex-Algorithmus. Dabei nimmt die Klasse `ProcessManager` eine zentrale Rolle ein. Sie ist von der Klasse `ProcessMaster` abgeleitet und beinhaltet die Abfolge der Prozessschritte des Simplex-Algorithmus. Die anderen Klassen in diesem Paket implementieren wiederum die einzelnen Prozessschritte.

3.7 Die Bibliothek commons-math

Aus der Bibliothek commons-math[14] wird die Klasse `BigFraction` verwendet. Diese Klasse bildet rationale Zahlen ab. Sie wird im Programm verwendet, um die bei Berechnungen mit Fließkommazahlen der Datentypen ‚float' und ‚double' unweigerlich auftretenden Rundungsfehler zu vermeiden. In der GUI wurde auf die Darstellung von Brüchen allerdings verzichtet.

[14] Siehe Verzeichnis der verwendeten Hilfsmittel. Die Bibliothek ist abgelegt im Verzeichnis /lib.

4 Literaturverzeichnis

[OR-Skript]
Prof. Dr. Natalia Kliewer, Operations Research, Vorlesungsskript zur Veranstaltung “Operations Research” im Rahmen des Studiengangs VAWi, Wintersemester 2015/2016, Universität Duisburg-Essen

[Suhl]
Leena Suhl und Taieb Mellouli (2013), „Optimierungssysteme – Modelle, Verfahren, Software, Anwendungen“, 3. Auflage, ISBN 0937-7433, Springer Gabler, Berlin

[Ellinger]
Ellinger, Bauermann, Leisten (2003), „Operations Research – Eine Einführung“, 6. Auflage, ISBN 3-540-00477-7, Springer, Berlin

5 Eidesstattliche Versicherung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorliegende Projektarbeit „Implementierung des Simplex-Algorithmus mit didaktisch aufbereiteter Ausgabe" selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift