

Dokumentation - Standortplanung SS13

Manuel Thoma (284987) und Markus Klemens (284983)

23.06.2013

HTWG Konstanz

Fakultät Informatik

Studiengang Wirtschaftsinformatik

Vertiefung Geschäftsprozessmanagement

Anwendung der Linearen Optimierung (ANLO) , Prof. Dr. Grütz

Inhaltsverzeichnis

Abbildungsverzeichnis.....	3
1. Einführung	4
2. Ist-Analyse	5
2.1 Überblick.....	5
2.2 Fehleranalyse.....	6
3. Commitment (Zielsetzung).....	10
4. Umsetzung.....	11
4.1. GUI	11
4.1.1 Einstellungen.....	12
4.1.2 Menu-Buttons	13
4.1.3 Asymmetrische Matrix.....	14
4.1.4 Registerkarte	14
4.1.5 Restriktionen.....	14
4.1.6 Solverausgabe	15
4.1.7 HTWG-Logo	16
4.1.8 Sonstige Umsetzungen.....	17
5. Ausblick.....	18
6. Fazit	20

Abbildungsverzeichnis

Abbildung 1: Übersicht Ist-Analyse GUI und fehlerhafte Solveransteuerung „MOPS“	5
Abbildung 2: Übersicht Ist-Analyse symmetrische Standortplanung.....	5
Abbildung 3: Fehleranalyse symmetrische Standortplanung	6
Abbildung 4: Alte JAVA-Klasse "IniLaden.java" mit Solveraufrufen	7
Abbildung 5: Alte "path.ini-Datei"	7
Abbildung 6: Alte JAVA-Klasse "IniLaden.java" mit Solver.pif-Aufruf	8
Abbildung 7: Alte JAVA-Klasse "LP_Solve.java"	8
Abbildung 8: Alte "solverCopy.bat-Datei"	9
Abbildung 9: Alte JAVA-Klasse "LP_Solve.java" mit Solveraufruf	9
Abbildung 10: Umsetzung neue GUI	11
Abbildung 11: Umsetzung Einstellungen ¹¹	12
Abbildung 12: Umsetzung neuer, partitionsübergreifender Solveraufruf	13
Abbildung 13: Umsetzung Menu-Buttons ¹³	13
Abbildung 14: Umsetzung Asymmetrische Matrix ¹⁴	14
Abbildung 15: Umsetzung Registerkarte "Restriktionen" ¹⁵	14
Abbildung 16: Umsetzung Registerkarte "Restriktionen" Quellcode.....	15
Abbildung 17: Umsetzung "Solverausgabe"	15
Abbildung 18: Umsetzung Solverausgabe "MOPS" Quellcode ¹⁸	16
Abbildung 19: Umsetzung HTWG-Logo Quellcode ²⁰	16
Abbildung 20: Umsetzung HTWG-Logo ¹⁹	16
Abbildung 21: Umsetzung Validierung Matriceingabe ²¹	17
Abbildung 22: Ausblick Tool "Eisenstadt"	18
Abbildung 23: Ausblick Mockup	19

1. Einführung

Im Rahmen der Veranstaltung „Anwendung der Linearen Optimierung“ im Sommersemester 2013, unter Anleitung von Herrn Professor Dr. Grütz, dokumentieren wir im Folgenden unsere Arbeit und Umsetzung an dem Tool „Standortplanung“.

Das ausgewählte Tool „Standortplanung“ soll durch eine Projektarbeit analysiert und anhand eines eigens verfassten Commitments umgesetzt werden.

Unsere Dokumentation setzt sich aus den Kapiteln „Ist-Analyse“, „Commitment“, „Umsetzung“, „Ausblick“ und „Fazit“ zusammen, auf die wir im Folgenden näher eingehen werden.

2. Ist-Analyse

Zunächst geben wir Ihnen einen kurzen Überblick über den aktuellen Ist-Zustand, um im Anschluss daran eine genauere Fehleranalyse durchzuführen.

2.1 Überblick

Zum gegenwärtigen Zeitpunkt befindet sich die Anwendung „Standortplanung“ in einem Zustand, in dem sie sich unter den Windows 7-Systemen (32-Bit / 64-Bit) lediglich starten lässt. Die Berechnung des bzw. der optimalen Standorte lässt sich zu keinem Zeitpunkt durchführen. Sie erfolgt im Erfolgsfall durch das Aufstellen eines Zielsystems, welches an einen externen Solver weitergegeben wird. Es erscheint bei allen, in der Anwendung implementierten Schnittstellen zu den Solvern (LP-Solve, XA, MOPS) eine Fehlermeldung, wodurch keine Berechnung möglich ist. Außerdem lässt sich die Anzahl der Standorte nur in einer symmetrischen Matrix abbilden. Desweiteren lässt die Usability des Tools durch den Umstand eines sehr schlicht gehaltenen Front-Ends zu wünschen übrig, so dass hier großes Optimierungspotenzial vorhanden ist.

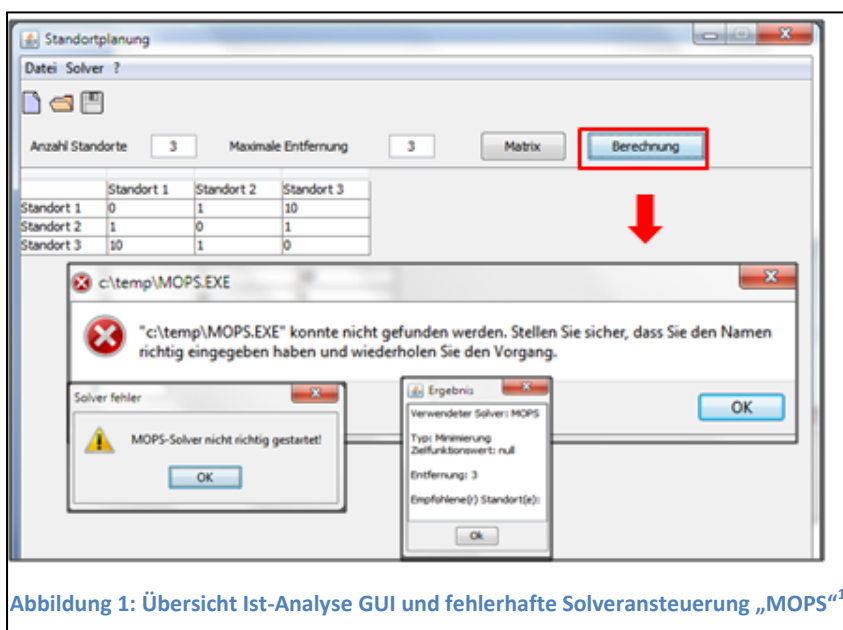


Abbildung 1: Übersicht Ist-Analyse GUI und fehlerhafte Solveransteuerung „MOPS“¹

	Standort 1	Standort 2	Standort 3
Standort 1	0	1	10
Standort 2	1	0	1
Standort 3	10	1	0

Abbildung 2: Übersicht Ist-Analyse symmetrische Standortplanung²

^{1, 2} Screenshots aus Tool „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013

2.2 Fehleranalyse

In diesem Abschnitt widmen wir uns der Fehleranalyse, um die gravierendsten Fehler der alten Anwendung ausfindig zu machen und zu analysieren, weshalb die Anwendung nicht optimal arbeitet.

Zunächst können in der alten Anwendung die Standorte nur in einer symmetrischen Matrix angegeben werden, ein Umstand, der es erschwert, Problemstellungen mit unterschiedlicher Anzahl von Standorten zu lösen (siehe Screenshot).

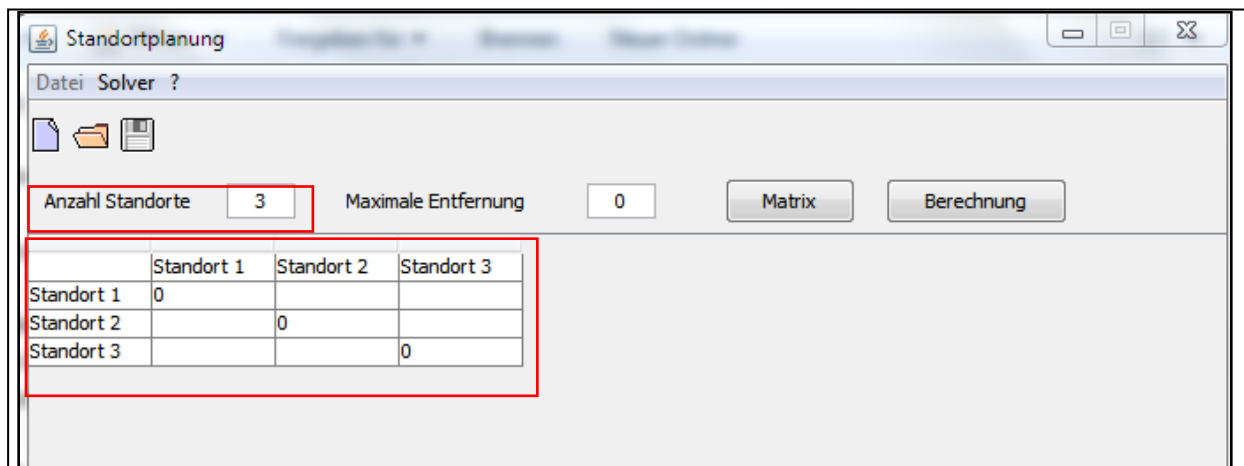


Abbildung 3: Fehleranalyse symmetrische Standortplanung³

Desweiteren werden die in der Anwendung verwendeten Solver nicht richtig angesteuert, so dass keine Lösung geliefert wird. Die Logik der jeweiligen Solveraufrufe sieht vor, dass die einzelnen Solverpfade über eine „path.ini.Datei“ auslesen werden. Wenn der Dateiaufruf fehlschlagen sollte bzw. der Pfad nicht ausgelesen kann, wird auf einen Default-Pfad zurückgegriffen, der jedoch nicht funktionieren kann, da die Solver sich im Ordner „Solver“ befinden, wohingegen der Aufruf über den relativen Pfad mit „solver/HierStehtDerJeweiligeSolver“ implementiert ist. Außerdem wird beim Solver „MOPS“ der Default-Pfad auf das Laufwerk „//L:“ festgesetzt, das jedoch nur bei Aufruf über die Virtual Machine/Remote Desktop existent ist. Das heißt, dass die Solverpfade umständlich manuell in der „path.ini-Datei“ gepflegt werden müssen und die Default-Pfad im Source-Code angepasst werden müssen, damit die Solver richtig aufgerufen werden können (siehe nachfolgenden Code-Ausschnitt).

³ Screenshot aus Tool „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013

```

public static String getLpSolvePath(){
    //Beispiel für das Ini-File: lpsolve=solver/lp_solve.exe
    String ausgabe = "solver/lp_solve.exe";
    String iniPath = "path.ini";
    String input = laden(iniPath);
    String output = tokenize(input,"lpsolve");

    if(output != null){
        ausgabe = output;
    }
    return ausgabe;
}

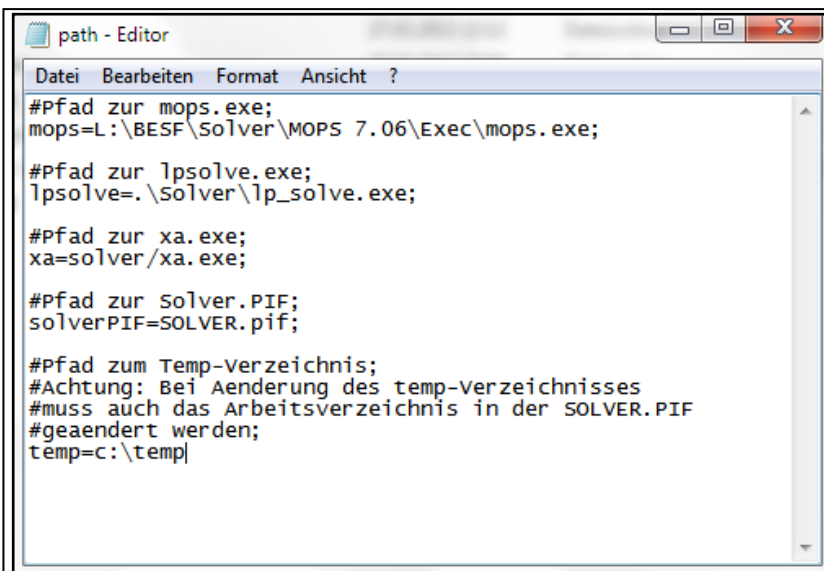
public static String getXA(){
    //Beispiel für das Ini-File: xa=solver/xa.exe
    String ausgabe = "solver/xa.exe";
    String iniPath = "path.ini";
    String input = laden(iniPath);
    String output = tokenize(input,"xa");

    if(output != null){
        ausgabe = output;
    }
    return ausgabe;
}

public static String getMOPS(){
    //Beispiel für das Ini-File: mops=L:\\BESF\\Solver\\MOPS 7.06\\Exec\\mops.exe
    String ausgabe = "L:\\BESF\\Solver\\MOPS 7.06\\Exec\\mops.exe";
    String iniPath = "path.ini";
    String input = laden(iniPath);
    String output = tokenize(input,"mops");

    if(output != null){
        ausgabe = output;
    }
    return ausgabe;
}

```

Abbildung 4: Alte JAVA-Klasse "IniLaden.java" mit Solveraufrufen⁴Abbildung 5: Alte "path.ini-Datei"⁵⁴ Screenshot aus Quellcode „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013⁵ Screenshot aus path.ini-Datei „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013

Zudem wird versucht, auf eine „Solver.pif-Datei“ zuzugreifen, die aber nicht mehr vorhanden ist und außerdem aus „alten“ MS-DOS-Zeiten stammt. Diese lässt sich somit nicht anpassen und wird heutzutage zu 99% von Viren verwendet (siehe Screenshot).

```
public static String getSolverPIF() {
    //Beispiel für das Ini-File: solverPIF=SOLVER.PIF
    String ausgabe = "SOLVER.pif";
    String iniPath = "path.ini";
    String input = laden(iniPath);
    String output = tokenize(input, "solverPIF");

    if(output != null){
        ausgabe = output;
    }
    return ausgabe;
}
```

Abbildung 6: Alte JAVA-Klasse "IniLaden.java" mit Solver.pif-Aufruf⁶

Bei der konkreten Ansteuerung des Solvers „LP-Solve“ sieht die Ablauflogik vor, dass zunächst eine „Solver.bat-Datei“ erzeugt wird, deren Inhalt den Solveraufruf realisieren soll. Dieser Inhalt ist jedoch fehlerhaft, da anstelle des „LP-Solve“ versucht, wird auf den Solver „XA“ und die bereits erwähnte, nicht existente „Solver.pif-Datei“ zuzugreifen (siehe Code-Ausschnitt).

```
try
{
    //BatchDatei und InputFile für Solver erstellen
    PrintWriter myLpSolveBatch = new PrintWriter(new BufferedWriter(new FileWriter(solverDirectory + "/Solver.bat")));
    PrintWriter myLPSolveIn = new PrintWriter(new BufferedWriter(new FileWriter(solverDirectory + "/lp_solve.in")));
    PrintWriter myLPSolveCopyBatch = new PrintWriter(new BufferedWriter(new FileWriter(solverDirectory + "/solverCopy.bat")));
    //Batchdateien schreiben
    File SolverLP = new File (IniLaden.getLpSolvePath()+" "+solverDirectory);
    myLPSolveCopyBatch.println(Variablen.copy + SolverLP);
    File SolverPif = new File (IniLaden.getSolverPIF()+" " + solverDirectory);
    myLPSolveCopyBatch.println(Variablen.copy + SolverPif);
    myLPSolveCopyBatch.close();
    //Batch-Datei für Solver-Start
    myLpSolveBatch.println("@echo off");
    myLpSolveBatch.println("REM Solver: XA");
    myLpSolveBatch.println("set Oldpath=%path%");
    File Solver = new File(solverDirectory + "/xa");
    myLpSolveBatch.println("path " + Solver + "%path%");
    myLpSolveBatch.println("lp_solve -p < lp_solve.in > lp_solve.out");
    myLpSolveBatch.println("path %Oldpath%");
    myLpSolveBatch.println("set Oldpath=");
    myLpSolveBatch.close();
}
```

Abbildung 7: Alte JAVA-Klasse "LP_Solve.java"⁷

^{6,7} Screenshots aus Quellcode „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013

Außerdem wird der Solver durch eine „solverCopy.bat-Datei“ in das angegebene Arbeitsverzeichnis kopiert, bei dessen Pfad jedoch nicht abgefangen wird, wenn in dem Pfad ein Leerzeichen angegeben sein sollte. Dies lässt sich umsetzen, in dem man den übergebenen Pfad zum Arbeitsverzeichnis in „“ setzt, welches Feature wir in unser nachfolgenden Umsetzung konkret um realisiert haben.

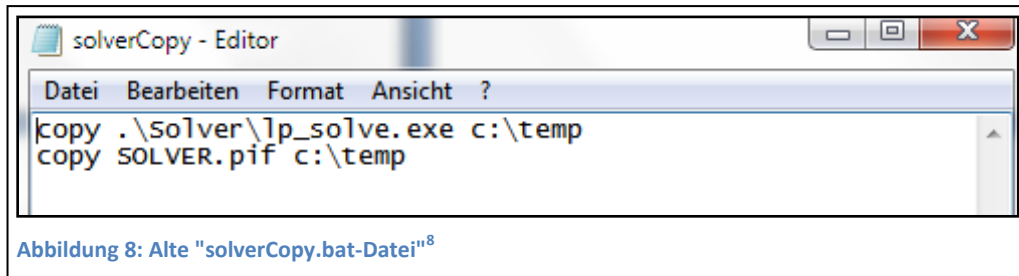


Abbildung 8: Alte "solverCopy.bat-Datei"⁸

Ein weiterer Fehler stellt der Aufruf des Solvers über die „bat.Datei“ dar. Dieser erfolgt über die „Solver.pif-Datei“, welche jedoch nicht existent ist. Der eigentliche Aufruf sollte über die „Solver.bat-Datei“ erfolgen, die jedoch auch noch angepasst werden müsste, da sie ja, wie bereits erwähnt, fehlerhaft ist.

```
//***** LP-Solver starten *****
solver = Runtime.getRuntime().exec(solverDirectory + "/solverCopy.bat");
try{
    solver.waitFor();
}
catch(InterruptedException eth)
{
    JOptionPane.showMessageDialog(null,"Solver nicht richtig gestartet!","Solver fehler", JOptionPane.WARNING_MESSAGE);
    return;
}

solver = Runtime.getRuntime().exec(solverDirectory + "/Solver.pif");

try{
    solver.waitFor();
}
catch(InterruptedException eth)
{
    JOptionPane.showMessageDialog(null,"Solver nicht richtig gestartet!","Solver fehler", JOptionPane.WARNING_MESSAGE);
    return;
}
```

Abbildung 9: Alte JAVA-Klasse "LP_Solve.java" mit Solveraufruf⁹

Analog dazu kann der Solver „MOPS“ nicht angesteuert werden, da sowohl der hinterlegte Pfad in der „path.ini-Datei“ als auch der Default-Pfad falsch sind.

^{8,9} Screenshots aus Quellcode „Standortplanung“ (gegenwärtige, alte Version), Abruf am 23.06.2013

3. Commitment (Zielsetzung)

An dieser Stelle werden wir einen Überblick über unser Commitment für das ausgewählte Tool „Standortplanung“ geben, das gleichzeitig auch unsere Zielsetzung sowie den Project Scope markiert.

Ausgehend von der Ist-Analyse haben wir es uns zum Ziel gesetzt, das Tool für die Windows 7 –Systeme (32 Bit / 64 Bit) kompatibel zu gestalten. Desweiteren dokumentieren wir unsere Vorgehensweise sowie das Testing in Form dieser Ausarbeitung. Zudem machen wir es uns zur Aufgabe, einen eingesetzten Solver richtig anzusteuern, um die optimalen Standorte in Abhängigkeit der eingegebenen Daten berechnen zu können. Zusätzlich sollen die Solverpfade in der laufenden Anwendung dynamisch durch den Nutzer angepasst werden können. Außerdem soll die Möglichkeit gegeben sein, die Anzahl der Standorte in einer asymmetrischen Matrix angeben zu können. Abschließend verpflichten wir uns dazu, die GUI sowie das „Look and Feel“ für eine bessere Usability anzupassen.

Das eingereichte Commitment kann jederzeit in dem Dokument „Klemens_Thoma-Commitment_StandortplanungSS13.docx“ eingesehen werden.

4. Umsetzung

4.1. GUI

Nach erfolgreicher Fehleranalyse haben wir uns zuerst überlegt, ob wir weiter auf der vorhandenen GUI aufbauen oder doch lieber eine neue GUI gestalten. Wir haben uns für die neue GUI entschieden, da wir erstens Elemente aus der alten GUI hätten löschen müssten und danach fast komplett die GUI leer war und zweitens das „Look and Feel“ aus einer früheren Zeit entsprach und nicht Windows 7 gerecht wurde. Desweiteren war es Teil unseres Commitments für eine bessere Usability zu sorgen, was bedeutet, neue GUI-Elemente, neue Funktionen und Möglichkeiten, asymmetrische Aufgaben der Standortplanung leichter zu bewältigen.

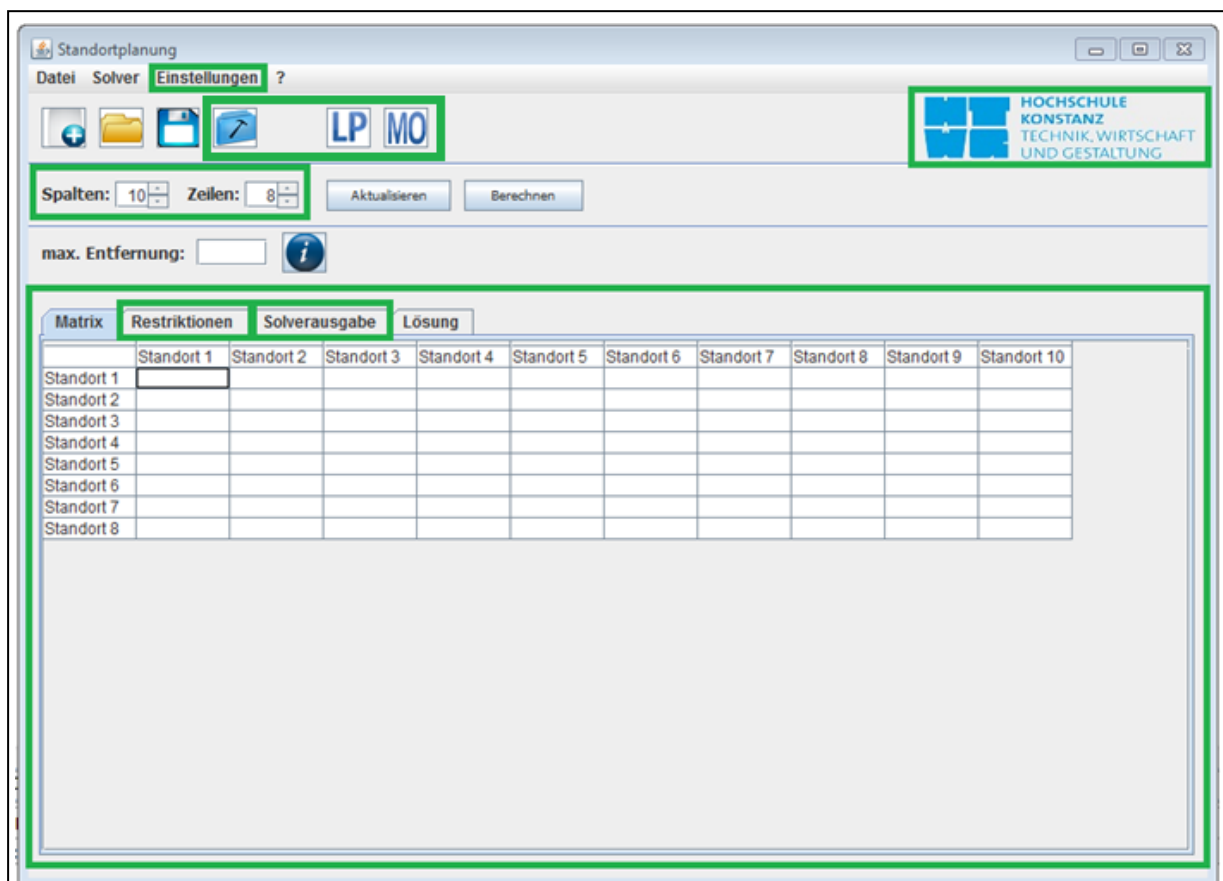


Abbildung 10: Umsetzung neue GUI¹⁰

Im Folgenden möchten wir zuerst anhand der neuen GUI zeigen, welche Erweiterungen hinzukamen und Ihnen kurz erläutern, wozu die Funktionen dienen. Danach werden wir tiefer einsteigen und alle Änderungen aufzeigen inkl. kleinerer Ausschnitte aus dem Quellcode.

¹⁰ Screenshot aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

- **Menu-Einstellungen:** Hier ist es nun möglich, die Solverpfade dynamisch anzugeben. Dies ermöglicht für den User mehr Möglichkeiten, sowie lokal zu arbeiten, durch die Angabe des Arbeitspfades und das ganze sogar partitionsübergreifend zu gestalten.
- **Menu-Buttons:** Es wurden Buttons (Solvereinsetzung, Solverauswahl (LP, Mops)) in die GUI hinzugefügt, um schneller arbeiten zu können.
- **Asymmetrische Matrix:** Der User hat nun die Möglichkeit, asymmetrische Matrizen zu erstellen. Bei vielen Standortplanungen ist dies relevant (siehe Übungsaufgaben in der Veranstaltung „Operations Research“ bei Prof. Dr. Herr Grütz).
- **Registerkarte:** Das Hauptgeschehen ist nun in einer Registerkarte verteilt. Hier befinden sich die Punkte *Matrix*, welches zur Eingabe dient, *Restriktionen*, *Solverausgabe* und *Lösung*, in dem sich das Ergebnis der Berechnung befindet.
- **Restriktionen:** Da es sich um ein Operations Research-Tool handelt, gehören natürlich auch Restriktionen dazu. Hier findet der User die Restriktionen, die anhand der Matrix bzw. der Eingabe gebildet werden.
- **Solverausgabe:** Das Programm ist momentan mit 2 fertigen Solver (MOPS und LP) gekoppelt, die die Berechnung durchführen. In diesem Register ist die komplette Ausgabe des ausgewählten Solvers hinterlegt.
- **HTWG-Logo:** Da dieses Programm evtl. auch außerhalb der HTWG eingesetzt wird, dient diese Funktion als Marketingzweck. Mit einem Klick auf das Logo öffnet sich die HTWG-Webseite.

Dies sind die hauptsächlichen Punkte, die umgesetzt wurden. Natürlich wurden noch einige Dinge umgesetzt, die so in der GUI nicht zu sehen sind, wie Fehler abfangen, Bugs beheben und den Code auf den Stil und Ordnung verbessern. Darauf möchten wir unter dem Punkt 4.1.8 Sonstige Umsetzungen etwas näher eingehen. Was wir nun anschauen sind die Hauptpunkte, mit den genauen Funktionen/Umsetzung und ein paar wichtige Code-Ausschnitte.

4.1.1 Einstellungen

In der „Standortplanung v2.0“ war es nicht möglich, die Solverpfade sowie den Arbeitsbereich manuell einzugeben bzw. selbst zu bestimmen. Dies wollten wir in der

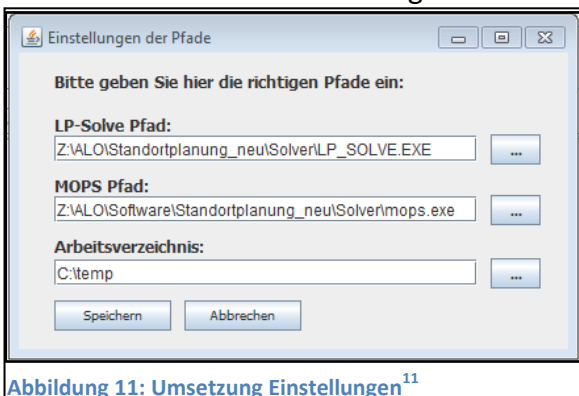


Abbildung 11: Umsetzung Einstellungen¹¹

aktuellen Version vermeiden und haben es dementsprechend umgesetzt. Denn wenn mal das Verzeichnis der Solver geändert wird oder ein Umzug der Arbeitsumgebung stattfindet, kann es ohne Probleme unter dem Menüpunkt *Einstellungen* wieder auf die Software angepasst werden.

¹¹ Screenshot aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

Die abgespeicherten Pfade werden im Installationsverzeichnis in der Textdatei „solverpath.txt“ festgehalten. Bei Beendigung des Programmes werden so die Pfade nicht gelöscht und können bei Neustart wieder ohne Probleme aus der Textdatei gelesen werden. Desweiteren war es nicht möglich, partitionsübergreifend zu arbeiten. Eine Umsetzung einer solchen Funktion, damit eine dynamische Arbeitsumgebung geschaffen wird, war daher erforderlich. Wir werfen daher einen ersten Blick in den Quellcode der neuen Version.

```
FrameInfo laufwerksbuchstabe
BufferedReader brLaufwerksbuchstabe = new BufferedReader(new FileReader("solverpath.txt"));
for (int o = 0; o < 4; o++) {
    if (o==3) {
        tempZeile = brLaufwerksbuchstabe.readLine();
        continue;
    }
    brLaufwerksbuchstabe.readLine();
}
brLaufwerksbuchstabe.close();

for (int o = 0; o < tempZeile.length(); o++) {
    if(o==5){
        laufwerksbuchstabe = String.valueOf(tempZeile.charAt(o));
    }
}
// Console Ausführung
Process pr = rt.exec("cmd /C start cmd.exe /C \"" + laufwerksbuchstabe + ":" +
    "&& cd " + solverDirectory + " && solver.bat\"");
```

Abbildung 12: Umsetzung neuer, partitionsübergreifender Solveraufruf¹²

Der Quellcode zeigt, dass wir aus der „solverpath.txt“. den Pfad des Arbeitsverzeichnisses auslesen. Der Solver selbst benötigt aber nur den Laufwerksbuchstaben, den wir in der 2. for-Schleife rausfiltern. Als letztes erfolgt die Console-Ausführung für den Solver, indem der Laufwerksbuchstabe, Solververzeichnis und die „solver.bat“ übergeben wird.

4.1.2 Menu-Buttons

Es wurden Menu-Buttons in die GUI integriert, sodass es nun dem User ermöglicht wird,

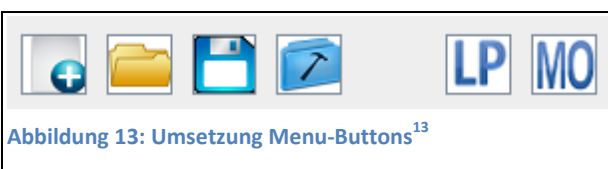


Abbildung 13: Umsetzung Menu-Buttons¹³

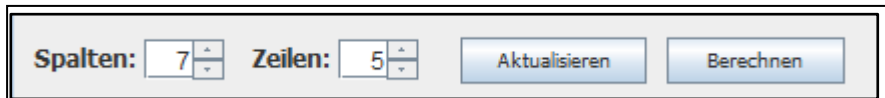
schneller und effizienter zu arbeiten. Ein Blick in den Code ist hier nicht nötig, die Buttons bestehen lediglich aus

gewöhnlichen Swing Buttons, die ein Event auslösen und die passende Methode aufrufen, die ja bereits schon durch die Menu Punkte (oben an der GUI) realisiert wurden.

^{12, 13} Screenshots aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

4.1.3 Asymmetrische Matrix

Die asymmetrische Matrix war ebenfalls eine wichtige Umsetzung, die es dem User nun ermöglicht, noch einfacher seine



Problemstellung zu einer

Abbildung 14: Umsetzung Asymmetrische Matrix¹⁴

Standortplanung zu lösen. Früher mussten zusätzliche Werte in eine Spalte/Zeile eingefügt werden, sobald die Problemstellung asymmetrisch war. Dies wurde für einen Laien meist eine Herausforderung. Dennoch soll der User daraus lernen und er hat nun bei Ausführung der Problemstellung die Möglichkeit, die Bildung der Restriktionen unter der Registerkarte "Restriktionen" in einer symmetrischen Matrix nachzuvollziehen, dazu aber später mehr.

4.1.4 Registerkarte

Nachdem wir geplant haben, was wir alles dem User an Ausgaben und Lösungen bieten wollen, war uns klar, dass eine Registerkarte für eine bessere Usability sorgen kann. Alles andere wäre zu unübersichtlich geworden oder es hätte zu viel Platz verschwendet.

4.1.5 Restriktionen

Nicht jeder User hat das logische Fachwissen und weiß, wie die Restriktionen zu bilden sind. Daher entschieden wir uns, die eingegeben Werte, die der User macht, in Restriktionen umzuwandeln, sodass er sieht, wie diese gebildet werden und evtl. sogar auch anderweitig weiter nutzen kann.

Matrix	Restriktionen								Solverausgabe	Lösung
	Standort 1	Standort 2	Standort 3	Standort 4	Standort 5	Standort 6	Standort 7	Standort 8		b
Standort 1	0	0	0	1	1	1	0	0	>=	1
Standort 2	0	1	1	0	1	1	0	0	>=	1
Standort 3	1	1	0	0	0	0	0	0	>=	1
Standort 4	1	1	1	1	0	0	1	1	>=	1
Standort 5	1	1	1	1	0	0	0	0	>=	1
Standort 6	1	0	0	0	0	0	1	0	>=	1
Standort 7	0	0	0	0	0	0	0	0	>=	1
Standort 8	0	0	0	0	0	0	0	0	>=	1
Zielfunktion	1	1	1	1	1	1	1	1	->	min!

Abbildung 15: Umsetzung Registerkarte "Restriktionen"¹⁵

In diesem Beispiel hat der User eine Matrix mit 8 Spalten und 6 Zeilen (=asymmetrische Matrix) generiert. Hier ist nun ersichtlich, dass dennoch für den Solver eine symmetrische Matrix notwendig ist, daher wurde aus der asymmetrischen Matrix eine symmetrische Matrix mit 8 Restriktion und 8 Variablen pro Restriktion gebildet. Weiterhin ist zu erkennen, dass die Restriktion bei Standort 7 und 8 die 2 Restriktionen sind, die neu dazu kamen, da diese den Wert 0 bei jeder Variable haben.

^{14, 15} Screenshots aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

Nun werfen wir noch einen kleinen Blick in den Code.

```
//Umrechnung in 0 oder 1
for (int i = 1; i < tabelle.getRowCount(); i++) {
    for (int j = 1; j < tabelle.getColumnCount(); j++) {
        if (Integer.parseInt(tabelle.getValueAt(i, j).
            toString()) > Integer.valueOf(tEntfernung.getText()).intValue()) {
            tabelle2.setValue2At(0, i, j);
        } else {
            tabelle2.setValue2At(1, i, j);
        }
    }
}
```

Abbildung 16: Umsetzung Registerkarte "Restriktionen" Quellcode¹⁶

Hier ist ein wichtiger Teil der Berechnung der Restriktionen abgebildet. Wir lesen die Tabelle der Matrix ein und überprüfen jeden Wert mit der Eingabe der Entfernung (z.B. max. 15 Minuten), die der User getroffen hat. Alle Werte, die nun über 15 Minuten sind, werden 0 und alle die kleiner gleich 15 sind, werden 1, da nur diese Werte relevant sind. Um den Rest des Quellcodes für die Darstellung zu sehen, verweisen wir sie auf die JAVA-Klasse "HauptFrame_new" (Quellcodezeile: 557-635), die im Programmordner „Standortplanung v3.0/src" zu finden ist.

4.1.6 Solverausgabe

Für die Studenten, die in einer Operations Research Vorlesung teilnehmen, ist dies sicherlich ein interessanter Punkt. Die Solverausgabe haben wir derart umgesetzt, dass weitere Werte, also nicht nur rein das Ergebnis ausgelesen werden kann. Unter anderem sind die Werte wie Solverrestriktionen, Zielfunktion, Iterationsanzahl usw. auslesbar, sofern Grundkenntnisse im Bereich Operations Research vorhanden sind. Wir wollen ein Beispiel anhand des MOPS-Solver aufzeigen:

Matrix	Restriktionen	Solverausgabe	Lösung					
iteration number = 4								
...name...		...activity...	defined as					
functional		2.00000	ZF					
restraints			MYRHS					
bounds....			B1234					
SECTION 1 - ROWS								
NUMBER	...ROW..	AT	...ACTIVITY...	SLACK	ACTIVITY	..LOWER LIMIT.	..UPPER LIMIT.	.DUAL ACTIVITY
7	R1	LL	1.00000	0.00000	0.00000	1.00000	none	-1.00000
8	R2	BS	1.00000	0.00000	0.00000	1.00000	none	0.00000
9	R3	LL	1.00000	0.00000	0.00000	1.00000	none	-1.00000
10	R4	BS	1.00000	0.00000	0.00000	1.00000	none	0.00000
11	R5	BS	1.00000	0.00000	0.00000	1.00000	none	0.00000
12	R6	BS	1.00000	0.00000	0.00000	1.00000	none	0.00000
SECTION 2 - COLUMNS								
NUMBER	.COLUMNS	AT	...ACTIVITY...	..INPUT COST..	..LOWER LIMIT.	..UPPER LIMIT.	.REDUCED COST.	
1	C1	LL	0.00000	1.00000	0.00000	1.00000	0.00000	
2	C2	BS	1.00000	1.00000	0.00000	1.00000	0.00000	
3	C3	LL	0.00000	1.00000	0.00000	1.00000	0.00000	
4	C4	BS	1.00000	1.00000	0.00000	1.00000	0.00000	
5	C5	LL	0.00000	1.00000	0.00000	1.00000	1.00000	
6	C6	LL	0.00000	1.00000	0.00000	1.00000	1.00000	

Abbildung 17: Umsetzung "Solverausgabe"

^{16, 17} Screenshots aus „Standortplanung v3.0" (neue Version), Abruf am 23.06.2013

Wenn die Ausgabe des MOPS-Solver betrachtet wird, ist zu erkennen, dass der MOPS 4 Iterationsschritte (iterationnumber = 4) benötigt hat, um das Problem zu lösen. Desweiteren sind 2 Zielwerte (functional 2.00000) das Optimum, dazu betrachten wir parallel in der Section 2, Number 2 und 4. Hier stellen wir fest, dass der Wert 1 bei der Spalte ...ACTIVITY... , dies bedeutet, dass der Standort 2 und 4 das Optimum wäre.

Die Umsetzung selbst sieht im Quellcode folgendermaßen aus (Beispiel MOPS):

```
// MOPS
if(solverTyp == 3){
    try{
        BufferedReader br = new BufferedReader(new FileReader(new FrameSolverpfade().getTfTEMP().getText()
            + "\\mops.lps"));

        String zeile = null;
        String zeichenkette = "";
        while ((zeile = br.readLine()) != null) {
            zeichenkette += zeile + "\n";
        }
        taSolverAusgabe.setText(zeichenkette);
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Abbildung 18: Umsetzung Solverausgabe "MOPS" Quellcode¹⁸

Wir lesen die Datei "mops.lps" ein, die der Mops im angegebenen Arbeitsverzeichnis generiert. Danach wird Zeile für Zeile durch den BufferedReader eingelesen und dem TextArea "taSolverAusgabe" weiter gegeben. Hinzu kommen noch Formatierungsaufgaben, speziell beim LP-Solve, damit der Text richtig angeordnet wird und gut leserlich ist. Dazu verweisen wir wieder in den Quellcode auf die JAVA-Klasse "HauptFrame_new" (Quellcodezeile:783-856), da es sonst hier den Rahmen sprengen würde.

4.1.7 HTWG-Logo

Ein kleines "nice-to-have" ist das implementierte HTWG-Logo. Sobald der User auf dieses Logo klickt, öffnet sich der Standardbrowser des Systems und wird direkt auf die Webseite der HTWG weiter geleitet. Wir fanden dies ein sehr nettes Gimmick. Denn es besteht die Wahrscheinlichkeit, dass externe Studenten oder externe User, sprich User außerhalb der HTWG, die Software nutzen und das soll dementsprechend als Marketingzeck dienen.



Abbildung 20: Umsetzung HTWG-Logo¹⁹

```
public void actionPerformed(ActionEvent arg0) {
    // Link zur HTWG- Website im Standardbrowser öffnen
    try{
        String url = "http://www.htwg-konstanz.de";
        URI uri = new URI(url);
        Desktop dt = Desktop.getDesktop();
        dt.browse(uri.resolve(uri));
    } catch (Exception e){
    }
}
```

Abbildung 19: Umsetzung HTWG-Logo Quellcode²⁰

^{18, 19, 20} Screenshots aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

4.1.8 Sonstige Umsetzungen

Es wurden noch viele Dinge umgesetzt, die so anhand der GUI nicht zu sehen sind. Dazu

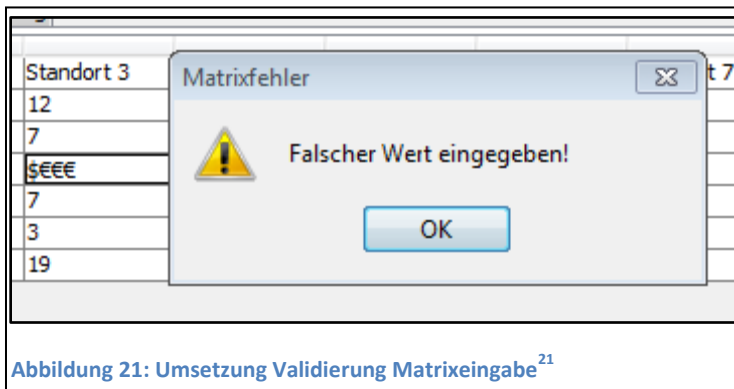


Abbildung 21: Umsetzung Validierung Matriceingabe²¹

gehören, z.B. dass der User keinen ungültigen Wert in die Matrix eingeben kann. Ebenfalls muss ein Solver ausgewählt sein, sonst wird der Fehler auch abgefangen und eine Meldung erscheint, dass ein Solver auszuwählen ist.

Desweiteren wurden viele Formatierungsarbeiten für die

Lösungsausgabe geleistet, weil jeder Solver von der Syntax andere Lösungen liefert und dafür mussten dementsprechend die Werte herausgefiltert werden, damit der Laie die Lösung sofort erkennen kann (siehe Register Lösung). Der Quellcode zu den Formatierungsausgaben ist in der JAVA-Klasse "Controller" (Quellcodezeile: 60-88 + die dazugehörigen Methoden).

²¹ Screenshot aus „Standortplanung v3.0“ (neue Version), Abruf am 23.06.2013

5. Ausblick

In diesem Kapitel widmen wir uns in einem Ausblick den möglichen Weiterentwicklungsmöglichkeiten der Anwendung, die, beispielsweise, durch ein Folgeprojekt realisiert werden können.

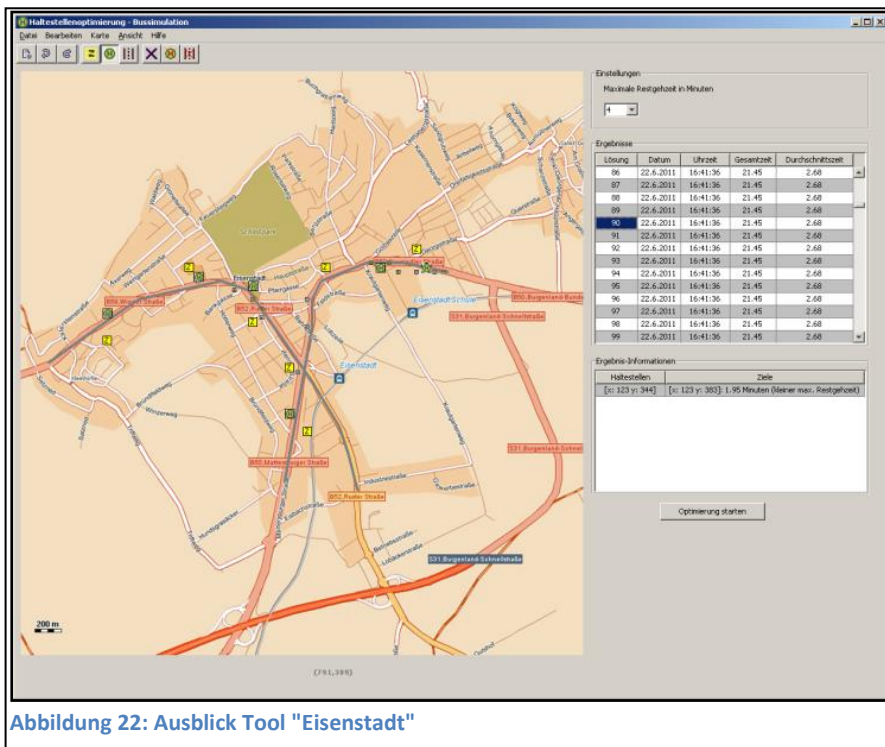


Abbildung 22: Ausblick Tool "Eisenstadt"

Auf Basis der realisierten Anwendung ist es denkbar, unser Projekt als Vorstufe bzw. Vorprojekt zu dem Tool „Eisenstadt“ (siehe Abbildung XY) zu betrachten. Hierbei könnten die Parameter bzw. die Ergebnisse graphisch dargestellt werden. Zum einen wird dem Anwender die Möglichkeit gegeben, die Standorte auf einem Kartenausschnitt zu platzieren, und zum anderen werden die weiteren, relevanten Parameter übersichtlich in einer Tabelle dargestellt.

²² Screenshot aus Tool „Eisenstadt“ mit graphischer Darstellung, Abruf am 23.06.2013

Ein erster Ansatz für die Umsetzung der graphischen Kartendarstellung könnte so aussehen (siehe Abbildung XY), dass man zunächst den für die jeweilige Problemstellung passenden Kartenausschnitt in die Anwendung lädt. Nachdem die infrage kommenden Standorte vom Anwender auf der Karte platziert wurden, können mithilfe eines darüber gelegten Koordinatensystems und dem Anwenden mathematischer Grundsätze (z.B. „Satz des Pythagoras“) Dreiecke zwischen den jeweiligen Standorte gebildet werden und auf diese Weise die kürzeste Entfernung bzw. Entfernungen berechnet werden.

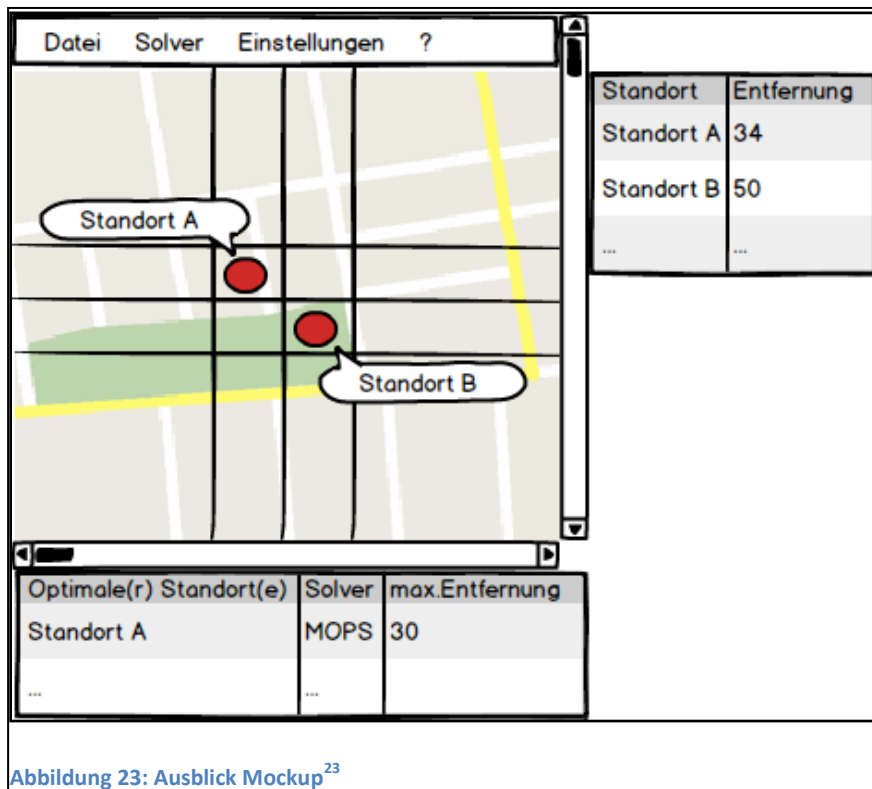


Abbildung 23: Ausblick Mockup²³

Desweiteren bestehe die Möglichkeit, durch die Ansteuerung der zukünftigen Schnittstelle „MightyLP“, ohne viel Mehraufwand weitere Solver in die Anwendung einzubinden. Dadurch ergibt sich der Mehrwert, dass für die Einbindung von weiteren Solver nicht zusätzlich jeweils eine eigene Schnittstelle entwickelt werden muss, sondern über eine einzige Schnittstelle generisch beliebige Solver angesteuert werden können.

²³ Eigenes erstelltes Mockup mit dem Tool „Balsamiq Mockups“, Abruf am 23.06.2013

6. Fazit

Zum Abschluss möchten wir Ihnen noch ein abschließendes Fazit über unsere Projektanalyse und Umsetzung geben.

Ausgehend von der Fehleranalyse ist uns aufgefallen, dass der Code z.T. nicht richtig fertiggestellt und sehr statisch programmiert worden ist. Konkret haben wir das an dem Problem festgemacht, dass die Solverpfade und die Arbeitsumgebung fest in den Code bzw. in die path.ini-Datei implementiert wurden. Hier erfolgte keine dynamische Anpassung durch den Anwender. Auch wirkte die Oberfläche der alten Anwendung etwas in die Jahre gekommen und altbacken, so dass hier einiges angepasst werden musste, um das „Look and Feel“ auf die heutige Zeit zu hieven. Bzgl. der Umsetzung hatten wir den größten Aufwand bei den Solvern, da diese bereits wie erwähnt sehr fehleranfällig programmiert wurden. Die GUI selbst und deren Button-Funktionen waren weniger ein Problem. Zur besseren Darstellung der Restriktionen und der Ausgabe der Solver wurden zusätzliche Registerkarten implementiert, um die Usability des Programms weiter zu verbessern, wovon wir überzeugt sind, dass dies die beste Lösung war.