Optimale Platzierung von Ladestationen für Elektrische Fahrzeuge

Sergio Brawand

Betreuer:
Prof. Kay W. Axhausen
Rashid A. Waraich

Bachelorarbeit
Studiengang Bauingenieurwissenschaften **Mai 2013**

IVT Institut für Verkehrsplanung und Transportsysteme
Institute for Transport Planning and Systems

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dank

Ich möchte mich bei folgenden Personen für die Unterstützung während der Durchführung meiner Bachelorarbeit bedanken:

- Professor Kay W. Axhausen (IVT, ETH Zürich) für die Möglichkeit, meine Bachelorarbeit am IVT durchzuführen sowie für das kritische Feedback und die nützlichen Tipps bei der Zwischenbesprechung,
- Rashid A. Waraich (IVT, ETH Zürich) für die engagierte Betreuung der Arbeit, die vielen Besprechungen, das wertvolle Feedback und die Hilfe beim Programmieren,
- Graziella Brawand für das kritische Durchlesen des Berichts.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Quelle Titelbild: Wilonsky (2011)

Bachelorarbeit Studiengang Bauingenieurwissenschaften

Optimale Platzierung von Ladestationen für Elektrische Fahrzeuge

Sergio Brawand
Hinterweidstrasse 19
8962 Bergdietikon
brawands@student.ethz.ch

Mai 2013

Kurzfassung

Elektrische Fahrzeuge (EF) sind zwar eine weitgehend saubere und emissionsfreie Alternative zu herkömmlichen Fahrzeugen mit einem Verbrennungsmotor, trotzdem ist der Marktanteil von EF heute immer noch sehr bescheiden. Ein Grund dafür ist auch die im Vergleich zu Benzin- und Dieseltankstellen deutlich weniger stark ausgebaute Ladeinfrastruktur. Durch eine optimale Platzierung von Ladestationen kann jedoch die zahlenmässige Unterlegenheit teilweise kompensiert werden.

In dieser Arbeit wurde ein Programm erarbeitet, welches für eine bestimmte Nachfrage die optimale Platzierung der Ladestationen berechnet. Das Problem wurde dabei als Constraint Satisfaction Problem (CSP) modelliert. Es wurden danach Experimente durchgeführt um zu zeigen, dass das Programm funktioniert. Zum Schluss wurden die Experimente ausgewertet und interpretiert. Die Auswertung ergab, dass aufgrund der langen Rechenzeit ein inkrementeller Ansatz für die Standortsuche vorteilhaft ist, und auf die globale Optimierung besser verzichtet wird.

Schlagworte

Optimale Platzierung; Elektrische Fahrzeuge; Ladestationen; Constraint Satisfaction Problem

Zitierungsvorschlag

Brawand, S. (2013) Optimale Platzierung von Ladestationen für Elektrische Fahrzeuge, *Bachelorarbeit*, IVT ETH Zürich, Zürich

1 Einleitung

1.1 Motivation

Mit zunehmendem Umweltbewusstsein werden seit längerer Zeit klimafreundlichere Alternativen zu herkömmlichen Fahrzeugen mit Verbrennungsmotoren gesucht. Elektrische Fahrzeuge (EF) sind eine weitgehend saubere und emissionsfreie Alternative. Gegen eine weite Verbreitung der EF sprechen jedoch die lange Ladezeit der Batterie sowie das Fehlen einer ausgereiften Ladeinfrastruktur. Das Problem ist schwierig zu lösen. Die Betreiber von Ladestationen scheuen einen Ausbau des Netzes, da die Nachfrage nur gering ist und neue Stationen unter Umständen nicht gewinnbringend betrieben werden können. Die möglichen Käufer von EF auf der anderen Seite entscheiden sich meistens gegen den Kauf eines EFs da unter anderem die vorhandene Infrastruktur für Diesel- und Benzinfahrzeuge deutlich besser ausgebaut ist. Weitere Gründe die gegen den Kauf eines EF sprechen sind der höhere Preis sowie der grosse Platzbedarf der Batterie. Durch eine optimierte Platzierung von Ladestationen für EF könnte man jedoch zumindest den Nachteil der viel geringeren Anzahl Ladestationen im Vergleich zu den Benzin- und Dieseltankstellen abmindern.

1.2 Zielsetzung und Vorgehen

Um Fragestellungen bezüglich des Einflusses von EF auf das elektrische Netz zu beantworten wird zurzeit am Institut für Verkehrsplanung und Transportsysteme (IVT) der ETH Zürich am Transportation Energy Simulation Framework (TESF) gearbeitet (Waraich et al., 2009 und Waraich et al., 2013). Im Rahmen dieser Arbeit sollen Konzepte für weitere Module für das TESF Framework erarbeitet und umgesetzt werden. Gemäss Aufgabenstellung liegt der Fokus dieser Arbeit auf der Platzierung von Ladestationen für EF. Um diese Ziele zu erreichen soll zuerst eine allgemeine Literaturrecherche zu Studien bezüglich Ladestationen und deren Platzierung durchgeführt werden. Danach sollen verschiedene mögliche Fragestellungen bezüglich der optimalen Platzierung der Ladestationen überlegt werden. Die am besten geeignete soll dann ausgewählt werden um dafür eine Lösung im TESF anzubieten. Die Fragestellung soll als sogenanntes Constraint Satisfaction Problem (CSP) erarbeitet und gelöst werden. Für die Optimierung solcher Probleme gibt es in Java mehrere Softwarepakete (Libraries). In einem nächsten Schritt sollen diese Libraries miteinander verglichen und eine geeignete für die Optimierung ausgewählt werden. Das ausgewählte Optimierungsproblem soll dann ins TESF Framework umgesetzt werden. Mithilfe von Experimenten soll gezeigt werden, dass die opti-

male Platzierung der Ladestationen und deren Integration ins Framework funktionieren. Zum Schluss sollen die Ergebnisse der Simulationen ausgewertet und diskutiert werden.

2 Literaturrecherche

Zum Thema „Platzierung von Ladestationen für Elektrische Fahrzeuge“ gibt es bereits zahlreiche Publikationen. In diesem Kapitel sollen die für diese Arbeit relevantesten kurz zusammengefasst und miteinander verglichen werden.

Die studierten Arbeiten betreffen unterschiedliche Gebiete. In einem ersten Schritt wurden verschiedene Arbeiten untersucht, die explizit die optimale Platzierung von Ladestationen für EF zum Ziel hatten. In einem zweiten Schritt wurden die technischen Herausforderungen, welche mit einer Zunahme der EF gemeistert werden müssten, genauer betrachtet. Zum Schluss wurde die Recherche noch auf weitere verwandte Gebiete ausgedehnt. Die Platzierung von Ladestationen ist auch für Erdgas- und Wasserstoff-Fahrzeuge essentiell. Die Situationen sind aufgrund der geringen Verbreitung der Fahrzeuge wie auch aufgrund der Zukunftsaussichten gut miteinander vergleichbar. Zudem wurden auch die Methoden für die Standortwahl von Supermärkten betrachtet.

2.1 Optimale Platzierung von Ladestationen

2.1.1 Vorgehen

Verschiedene Arbeiten hatten ihren Schwerpunkt auf der optimalen Platzierung von Ladestationen. Bei vielen Arbeiten mussten Annahmen getroffen und Anpassungen vorgenommen werden, um die Verhältnisse der Realität in ein Modell einzupassen. Der Hauptgrund dafür waren die sehr grossen Datenmengen, welche verkleinert werden mussten um die Optimierungen innerhalb einer sinnvollen Zeitspanne lösen zu können. Das Vorgehen dabei war sehr unterschiedlich. Ein wichtiger Schritt um die notwendige Rechenzeit zu verkürzen war eine sinnvolle Vorselektion der möglichen Standorte für Ladestationen. He et al. (2012) bezogen das geographische Informationssystem GIS in ihre Arbeit mit ein. So liessen sich unerwünschte oder unmögliche Standorte für Ladestationen bereits im ersten Schritt eliminieren. Einen anderen Ansatz wählten Chung und Kwon (2012). Wege, die kürzer waren als die halbe Reichweite eines EFs wurden in ihrer Arbeit nicht berücksichtigt. Häufig waren diese aber in grösseren Wegen enthalten und dementsprechend bereits abgedeckt. Zudem wurden nur Wege betrachtet, die eine bestimmte Mindestnachfrage erreichten (z.B. 20‘000 Fahrten pro Jahr). So konnte die unhandliche Anzahl von 105‘000 Wegen auf ca. 1‘600 – 3‘000 (je nach gewählter Mindestnachfrage) beschränkt werden. Für eine Mindestnachfrage von 30‘000 Fahrten pro Jahr und einer Reichweite von 120km wurden somit zwar nur ca. 1% der Wege direkt in die

Berechnungen miteinbezogen, diese deckten jedoch 73% aller Wege sowie 99% des Verkehrsvolumens ab.

2.1.2 Zielfunktion

Bei den meisten untersuchten Arbeiten war die Zielfunktion eine Kostenfunktion. Bei He et al. (2012) setzte sich diese aus den Bau- sowie den Betriebskosten zusammen. Bei Liu et al. (2012) enthielt die Zielfunktion zudem noch geographische Informationen. Im Unterschied dazu bestand diejenige von Jia et al. (2012) neben den Investitions- und Betriebskosten der Ladestationen auch aus den Wegkosten der Benutzer. Die Zeit, die die Benutzer benötigten um zur Ladestation zu gelangen, wurde so monetarisiert. Der Fokus wurde damit von der Angebotsseite gelöst und die Nachfrageseite somit gleichermassen betrachtet. Die Optimierung bei Jia et al. (2012) ergab, dass die Ladestationen fast ausschliesslich an zentralen Orten platziert sein sollten, da die Benutzer dort häufig auch noch andere Bedürfnisse erledigen können. Es zeigte sich, dass eine feinere Verteilung der Stationen zwar die Kosten der Nutzer verringerte, gleichzeitig jedoch die fixen Kosten für den Aufbau der Ladestationen erhöhte. Ein Problem waren die engen Platzverhältnisse welche meist an Knoten herrschen und eine Umsetzung in der Realität erschweren. Eine Reduktion der Anzahl Ladestationen führte zu einem längeren Weg und somit zu höheren Kosten der Nutzer.

2.1.3 Lösungsstrategien

Gelöst wurden die Optimierungen mithilfe von verschiedenen mathematischen Algorithmen. He et al. (2012) entwickelten aus dem „Standard Genetic Algorithm“ (SGA) und dem „Allocation Algorithm“ (ALA) einen hybriden genetischen Algorithmus und lösten ihr Problem damit. Liu et al. (2012) lösten ihre Optimierung mithilfe eines „Adaptive Particle Swarm Optimization“ (APSO) Algorithmus. Leider war meistens nicht ersichtlich, mit welchen Solvern diese Probleme gelöst oder ob sie selbst implementiert wurden. Einzig bei Jia et al. (2012) erfuhr man, dass um das Optimierungsproblem zu lösen Cplex, ein Solver von IBM, verwendet wurde.

2.2 Technische Herausforderungen

Die gegenwärtige Verbreitung von EF ist sehr gering. Dementsprechend sind die Einflüsse auf das elektrische Netz ebenfalls sehr klein. Trotzdem muss abgeklärt werden, ob bei einer stärkeren Marktpenetration von EF die Stabilität des Netzes gewährleistet ist. Dabei werden verschiedene Standpunkte vertreten. Da die Anzahl EF nur langsam zunimmt, sind Speidel et

al. (2012) der Meinung, dass Kapazitätsprobleme mit dem Stromnetz eher unwahrscheinlich sind. Laut Raviv (2012) ist das Stromnetz jedoch bereits heute zu Spitzenzeiten fast maximal ausgelastet. In den „Randzeiten“ (Nacht) ist jedoch noch viel Kapazität vorhanden. Johnson et al. (2012) betrachteten die Möglichkeiten, die sich ergäben, wenn die EF und die Ladeinfrastruktur mittels digitaler Kommunikation vernetzt wären. Dabei wies der zentrale Computer den Nutzern den bestmöglichen Weg um ihre Batterie aufzuladen. Es zeigte sich, dass sich sowohl die maximale Wartezeit bei den Ladestationen als auch die maximale Stromabgabe deutlich reduzierten. Dies hilft vor allem auch den Stromversorgungsbetrieben. Instabilitäten aufgrund eines erhöhten Bedarfs an einer Stelle können so einfacher vermieden werden.

2.3 Verwandte Probleme

Optimierungsprobleme für die Standortwahl gibt es in verschiedenen Bereichen. Da die grundsätzlichen Anforderungen sehr ähnlich sind, lassen sich auch Erkenntnisse aus anderen Bereichen gewinnen.

Nicholas et al. (2004) untersuchten die durchschnittlich notwendige Fahrzeit, um zu einer Wasserstoff-Tankstelle zu gelangen. Die Anzahl Wasserstoff-Tankstellen betrug im Versuch nur 30% der Anzahl der Benzin-und Dieseltankstellen. Sie konnten aber zeigen, dass sich bei einer optimierten Platzierung der Wasserstoff-Tankstellen die durchschnittliche Fahrzeit gegenüber den herkömmlichen Fahrzeugen nur um 16 Sekunden erhöhte. Dabei reduzierten sie den Rechenaufwand, indem immer nur 2 Stationen gleichzeitig platziert wurden. Nachdem für diese die besten Standorte gefunden wurden, galten sie als gesetzt und die Suche für die nächsten zwei Standorte konnte beginnen.

Auch Lin et al. (2008) betrachteten in ihrer Arbeit die Platzierung von Wasserstoff-Tankstellen. Dabei stellten sie fest, dass man in der Praxis eher daran interessiert ist, wie ein Tankstellennetzwerk wächst und eher weniger wie es im statischen Zustand aussieht. Es wurde zudem darauf hingewiesen, dass das Wachstum des Netzwerks logisch erfolgen muss. Das heisst, bereits bestehende Tankstellen werden nicht verschoben, sondern als gegeben betrachtet.

Frick et al. (2007) bauten das Tankstellen-Netz in ihrer Arbeit über die Verteilung von Erdgas-Tankstellen ebenfalls stufenweise aus. Als mögliche Standorte galten dabei die schon bestehenden Benzin- und Dieseltankstellen. Für den Ausbau des Netzes wurden jeweils 50 Stationen gleichzeitig hinzugefügt. Im nächsten Schritt wurde dann deren optimale Verteilung berechnet.

Auch die Standortwahl von Supermärkten besitzt eine grosse Ähnlichkeit mit der Standortwahl von Ladestationen für EF. Ciari und Axhausen (2011) zeigten auf, wie zwei Supermarktketten durch eine Optimierung der Standorte ihrer Läden die Anzahl ihrer Kunden maximieren könnten. Ihre Experimente führten sie mit einem 10% Szenario der Innenstadt von Zürich durch. Die Resultate wurden mithilfe von MATSim (Multi-Agent Transport Simulation) generiert. Dabei bestand jeder Simulationslauf aus drei Phasen. In der ersten Phase wurde so lange in MATSim iteriert, bis der durchschnittliche Nutzen für die Agenten konvergierte. Danach wurden die Läden (ausserhalb von MATSim) neu angeordnet. In der dritten Phase sollte dann ein zweites Gleichgewicht erreicht werden, diesmal auch in der Anzahl der Kunden in den Läden.

3 Fragestellung

Um eine spannende Fragestellung bezüglich der Platzierung von Ladestationen zu erhalten wurden verschiedene Formulierungen erarbeitet und miteinander verglichen. Die zur Auswahl stehenden Fragestellungen waren:

a) Wo muss man eine gegebene Anzahl von Ladestationen errichten, um die Gesamtlänge aller Umwege zu minimieren?

b) Wo muss man eine gegebene Anzahl von Ladestationen errichten, damit der Umweg maximal x % grösser wird als der kürzeste Weg?

c) Wo und in welcher Anzahl muss man die Ladestationen errichten, damit die Länge aller Umwege maximal x km beträgt?

d) Wo und in welcher Anzahl muss man die Ladestationen errichten, um die generalisierten Kosten zu minimieren?

e) Wo muss man eine gegebene Anzahl von Ladestationen errichten, um eine maximale Auslastung der Stationen zu erreichen?

f) Wo muss man eine gegebene Anzahl von Ladestationen errichten, um möglichst kurze Wartezeiten (Ladedauer beträgt ca. 20 bis 30 Minuten) zu erreichen (Hõimoja und Rufer, 2012)?

g) Wo und in welcher Anzahl muss man die Ladestationen errichten, um einen möglichst geringen Einfluss auf die Stabilität des Stromnetzes zu haben?

Aufgrund der Klarheit der Fragestellung und der Relevanz des Problems in der Praxis fiel die Wahl auf Fragestellung a). Diese ist im Gegensatz zu b) und c) für verschiedene Szenarien ohne Anpassung lösbar. Die Fragestellungen e) und f) sind für die optimale Platzierung von Ladestationen nicht zentral, sondern eher wünschenswert. Zudem könnte man diese eventuell zu einem späteren Zeitpunkt als Nebenbedingungen in a) einbauen. Die Aufgabenstellung d) ist eine Erweiterung von a) und könnte mit relativ wenig Aufwand adaptiert werden. Dafür müssten noch die Errichtungs- und die Betriebskosten einer Ladestation festgelegt sowie die Umrechnung von Fahrzeit in Kosten definiert werden. Aufgabenstellung g) ist sehr vielschichtig und aus diesem Grund für diese Arbeit nicht geeignet. Die Aufgabenstellung lautet also:

Wo muss man eine gegebene Anzahl von Ladestationen errichten, um die Gesamtlänge aller Umwege zu minimieren?

4 Constraint Satisfaction Problem

Viele der Fragestellungen bezüglich Platzierung von Ladestationen sind sogenannte Constraint Satisfaction Problems (CSP). Nach Laburthe und Jussien (2012) besteht ein CSP in der Regel aus den folgenden drei Teilen:

- Variablen
- Domain
- Constraints

Bei einem Optimierungsproblem benötigt man zudem noch eine von den Variablen abhängige Zielfunktion. Die Variablen sind die Unbekannten des Problems. Die Domain beschreibt diskrete Werte oder kontinuierliche Wertebereiche, welche eine Variable annehmen kann. Constraints sind logische Beziehungen, welche mit der Wahl der Variablen erfüllt werden müssen. Dies können zum Beispiel Gleichungen, Ungleichungen oder „Wenn …, dann …“ Anweisungen sein. Die Absicht der Optimierung ist es, durch bestmögliche Wertezuweisung für die Variablen die Zielfunktion zu minimieren oder zu maximieren.

4.1 Vergleich mehrerer Libraries

Es gibt mehrere Softwarepakete („Libraries“) für die Optimierung von CSPs in Java. In diesem Schritt wurden verschiedene Libraries kurz getestet und miteinander verglichen. In einem nächsten Schritt wurde dann die am besten geeignete Library ausgewählt. Hauptkriterien waren die Punkte Performance, Benutzerfreundlichkeit und Dokumentation. Zudem sollte die Library kostenlos erhältlich sein. Um Kompatibilitätsprobleme zu vermeiden sollte sie auch in reinem Java geschrieben sein. So lässt sie sich plattformunabhängig auf verschiedenen Betriebssystemen anwenden.

Tabelle 1 zeigt sämtliche untersuchten Libraries, inklusive deren Vor- und Nachteile. Es ist zu beachten, dass diese Bewertung subjektiv ist und in relativ kurzer Zeit erarbeitet wurde.

Tabelle 1 Vergleich von CSP-Libraries

Name	Link	Dokumentation	Besonderes	Eignung
Choco	http://www.emn.fr/z-info/choco-solver/	Sehr umfangreich, mehrere Beispiele	Gutes Verständnis	Sehr gut
Cream	http://bach.istc.kobe-u.ac.jp/cream/	Eher kurz, mit Beispielen	Gutes Verständnis	Gut
YACS	http://constraints.sourceforge.net/	Nur sehr knapp	-	Ungeeignet
CSP4J	http://sourceforge.net/projects/cspfj/files/	Keine gefunden	-	Ungeeignet
Easy CSP	http://easy-csp-lib.sourceforge.net/	Kurz, keine Beispiele	Sehr mühsam, Dateien müssen einzeln heruntergeladen werden	Ungeeignet
JaCoP	http://jacop.osolpro.com/	Sehr umfangreich, mit Beispielen	Verständnis schwierig	Gut

Mehrere der gefundenen Libraries (YACS, CSP4J und Easy CSP) waren entweder nur schlecht dokumentiert, hatten keine Beispielcodes zur Verfügung oder funktionierten aufgrund anderer Probleme nicht. Deshalb wurden diese dann nicht mehr weiter getestet. Die drei geeignetsten Libraries waren Choco, Cream und JaCoP. Alle drei erfüllten sämtliche der oben genannten Kriterien. Aufgrund der äusserst detaillierten Dokumentation und mehreren Beispielen wurde dann mit Choco weitergearbeitet.

4.2 Choco

Choco ist eine open-source Software, welche gratis verwendet werden kann. Vertrieben wird Choco auf sourceforge.net. Für diese Arbeit wurde die Version 2.1.5 verwendet, welche am 17. September 2012 veröffentlicht wurde. In der Zwischenzeit wurde bereits eine komplett überarbeitete Version (Choco 3) veröffentlicht. Diese ist von Grund auf neu konzipiert und nicht kompatibel mit Choco 2. Da dazu aber noch keine Dokumentation erhältlich war, wurde diese Arbeit mit der Version 2.1.5 durchgeführt.

4.2.1 Aufbau

Laburthe und Jussien (2012) beschreiben in ihrer Dokumentation den Aufbau von Choco. Die zentralen Elemente von Choco sind das Modell, die Variablen sowie die Constraints. Ein Modell ist dabei definiert durch ein Set von Variablen mit einer gegebenen Domain sowie einem Set von Constraints, welches die Variablen miteinander verknüpft. Die Variablen sind die Unbekannten des Problems. Werte für eine Variable werden aus der jeweiligen Domain gewonnen, welche meistens durch einen Wertebereich definiert ist. Der Wert einer Variablen ist erst dann bestimmt, wenn der Solver eine Lösung gefunden hat.

4.2.2 Lösungssuche

Der Solver liest das Modell, definiert die Suchstrategie und iteriert dann so lange bis die beste Lösung gefunden wird. Nach Laburthe und Jussien (2012) besteht das Standardvorgehen bei einem Branch-and-Bound Ansatz aus einer Tiefensuche (Depth-First-Search). Man kann sich das als Baumdiagramm vorstellen, in welchem das Problem in viele kleine Unterprobleme aufgeteilt wird. Bei einer Tiefensuche arbeitet sich das Verfahren möglichst schnell in die Tiefe, sodass normalerweise sehr schnell eine zulässige Lösung gefunden wird (Wikipedia, 2013). Es ist jedoch auch möglich, komplexere Suchstrategien zu definieren.

4.2.3 Stärken

Der grösste Vorteil von Choco ist die Einfachheit der Syntax. Das Erlernen der grundlegenden Funktionen und Befehle ist dank zahlreichen Beispielcodes relativ einfach und geschieht intuitiv. Auch die strikte Trennung von Modell und Solver vereinfacht die Anwendung von Choco. Ein weiterer Vorteil ist die gute Verfolgbarkeit der Lösungssuche. Mit Befehlen lassen sich zudem auch die Rechenzeit beschränken und die Suchstrategie ändern. Zudem beinhaltet die Library sehr viele Constraints, welche für unterschiedliche Zwecke verwendet werden können. Vor allem für die Optimierung von Terminplänen scheint Choco sehr geeignet zu sein. Dies könnte z.B. im TESF genutzt werden, um Ladezeiten von Autos gemäss einer vorgegebenen Nutzenfunktion zu optimieren.

Choco kann auch gut für „quick prototyping“ benutzt werden. Das Ziel dabei ist, möglichst schnell einen funktionierenden Code erstellt zu haben um erste Berechnungen durchzuführen. So kann die Entwicklung schnell vorangetrieben werden. In einem späteren Schritt kann dann das Programm noch verbessert werden.

Die Nachteile von Choco werden in Kap. 6.4 betrachtet.

5 Implementierung

5.1 Vorgehen

Mithilfe von Choco soll das Programm aus einer Menge von möglichen Standorten für Ladestationen die am besten geeigneten auswählen. Gegeben sind dabei die Anzahl Ladestationen, das Set von möglichen Standorten sowie die einzelnen Nachfragepunkte. Das finale Programm wurde schrittweise erarbeitet. Um die Funktionen der Library zu erlernen, wurden zuerst einfache Beispielprogramme geschrieben. Danach wurden diese zusammengefügt und erweitert. Es zeigte sich jedoch, dass das Programm nur für ein sehr kleines Szenario funktionierte. Die Rechenzeit nahm bereits mit einer kleinen Vergrösserung des Szenarios überproportional zu. Um auch grössere Szenarien durchführen zu können wurde das Programm zwei Mal komplett neu überarbeitet. Raimondo (2013) löste in seiner Arbeit über die optimale Platzierung von Windturbinen ein ähnliches Problem mithilfe von Binärvariablen. Dieser Ansatz wurde dann auch in dieser Arbeit verfolgt. Die Lösungssuche beschleunigte sich dadurch. Ein weiterer Fortschritt konnte erzielt werden, indem die Reihenfolge in welcher die Variablen angewählt und mit Werten belegt wurden, neu definiert wurde.

5.2 Formeln

In diesem Kapitel sollen die mathematischen Rahmenbedingungen des Programms genauer erklärt werden. Das zu lösende Problem ist verwandt mit dem p-Center Problem (Mladenović et al., 2003) und dem p-Median Problem (Resende und Werneck, 2004). Die Unterschiede dabei sind klein. Bei einem p-Center Problem wird der maximale Abstand eines Fahrzeuges zur nächsten Ladestation minimiert. In dieser Arbeit wird dagegen die totale Distanz aller Fahrzeuge minimiert. In einem p-Median Problem wird zusätzlich noch die Nachfrage gewichtet. Am besten wird das zu lösende Problem wohl durch eine Vereinfachung des Uncapacitated Facility Location Problems (Guha und Khuller, 1999) beschrieben. Dabei werden in dieser Arbeit die Errichtungskosten für die Ladestationen vernachlässigt.

5.2.1 Variablen

Die Binärvariable X gibt an, ob ein möglicher Standort einer Ladestation ausgewählt ist oder nicht.

$$X_i = \begin{cases} 1, & Station\ i\ ist\ ausgewählt \\ 0, & Station\ i\ ist\ nicht\ ausgewählt \end{cases}$$

Wege können nur in horizontaler und vertikaler Richtung zurückgelegt werden und die Distanz wird in Manhattan-Metrik (Poesio et al., 1998) berechnet. Das heisst, es werden die betragsmässigen Differenzen der Einzelkoordinaten addiert. Gemessen wird in der Einheit Wegeinheiten (WE).

In den Matrizen *a* und *Y* werden die Werte der verschiedenen Distanzen von einem Nachfragepunkt *j* zu einer möglichen Ladestation *i* erfasst. Der Index *m* steht für die Menge der möglichen Standorte für Ladestationen. Der Index *k* bezeichnet die Anzahl Fahrzeuge, die einen Bedarf für eine Aufladung ihrer Batterie haben.

$$a = \begin{pmatrix} a_{00} & \cdots & a_{0k} \\ \vdots & \vdots & \vdots \\ a_{m0} & \cdots & a_{mk} \end{pmatrix}$$

$$Y = \begin{pmatrix} Y_{00} & \cdots & Y_{0k} \\ \vdots & \vdots & \vdots \\ Y_{m0} & \cdots & Y_{mk} \end{pmatrix}$$

Damit nicht ausgewählte Ladestationen keinen Einfluss auf das Ergebnis ausüben, wird die tatsächliche Distanz a_{ij} noch mit einem Term erweitert.

$$Y_{ij}(X_i) = a_{ij} * X_i + \infty * (1 - X_i)$$

Die Variable *d* bezeichnet die minimale Distanz von einem Nachfragepunkt *j* zur nächsten ausgewählten Ladestation.

$$d_j = \min(Y_{0j}, \dots, Y_{mj})$$

5.2.2 Constraints

Die Summe aller *X* muss gleich der gewünschten Anzahl Ladestationen *n* sein.

$$\sum_{i=0}^{m} X_i = n$$

5.2.3 Zielfunktion

Die Zielfunktion *D* ist die Gesamtdistanz, welche zurückgelegt werden muss um von jedem Nachfrageort zur nächsten Ladestation zu kommen. Diese Funktion soll dabei minimiert werden.

$$minimiere\ D(X) = \sum_{j=0}^{k} dj$$

5.2.4 Beispiel

Um genauer zu sehen, wie das Programm funktioniert wird in diesem Kapitel ein kleines Beispiel vorgestellt. Es gelten dabei folgende Voraussetzungen aus Tabelle 2.

Tabelle 2 Eckpunkte Beispielszenario

Anzahl Nachfragepunkte *k*	Menge der möglichen Standorte m	Anzahl zu platzierende Ladestationen *n*
6	4	3

In Tabelle 3 werden die Koordinaten der Nachfragepunkte dargestellt.

Tabelle 3 Nachfrage

Nachfrage	X-Koordinate [WE]	Y-Koordinate [WE]
#0	8	8
#1	5	1
#2	7	3
#3	4	7
#4	2	0
#5	4	2

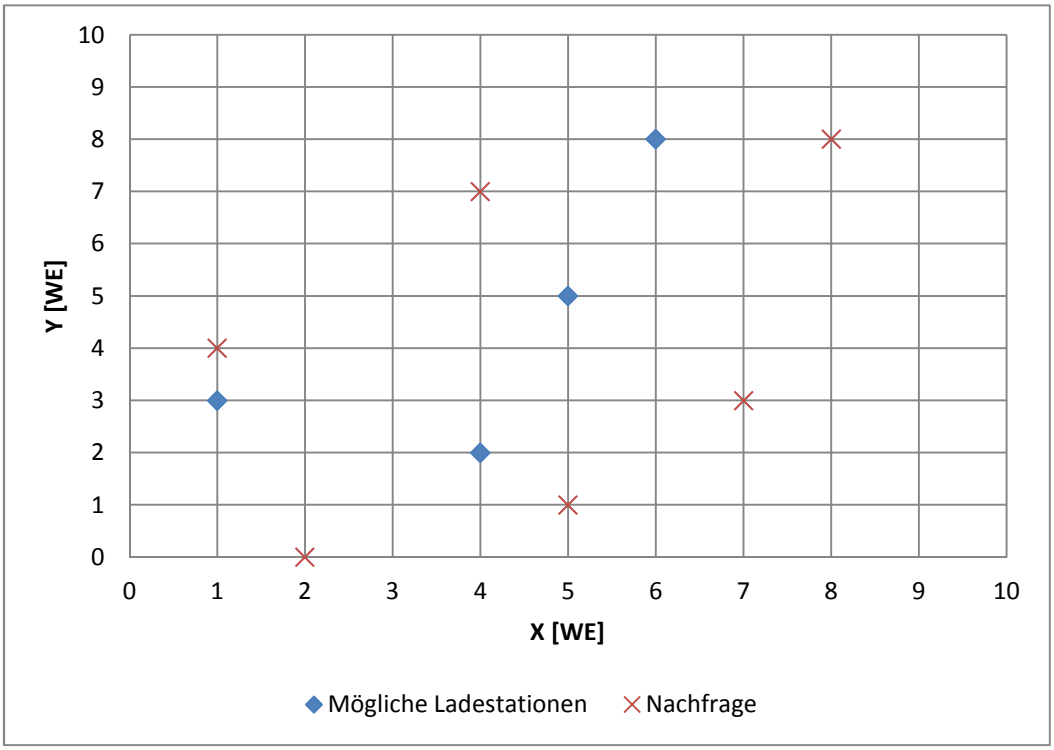
Die möglichen Standorte, an denen Ladestationen gebaut werden können werden in Tabelle 4 dargestellt.

Tabelle 4 Mögliche Standorte

Standort	X-Koordinate	Y-Koordinate
#0	5	5
#1	1	3
#2	6	8
#3	4	2

Die Verteilung der Nachfragepunkte sowie die möglichen Standorte für Ladestationen sieht man in Abbildung 1.

Abbildung 1 Beispielszenario



Die Anwendung des Programms generiert dann die verschiedenen Werte für die Variablen. Die Spalten in der Matrix *a* beziehen sich auf die jeweiligen Nachfragepunkte. Die Zeilen symbolisieren die möglichen Standorte für Ladestationen. Für die Distanz-Matrix *a* erhält man dabei folgende Werte:

$$\begin{array}{llllll} a_{00}=6 & a_{01}=4 & a_{02}=4 & a_{03}=3 & a_{04}=8 & a_{05}=5 \\ a_{10}=12 & a_{11}=6 & a_{12}=6 & a_{13}=7 & a_{14}=4 & a_{15}=1 \\ a_{20}=2 & a_{21}=8 & a_{22}=6 & a_{23}=3 & a_{24}=12 & a_{25}=9 \\ a_{30}=10 & a_{31}=2 & a_{32}=4 & a_{33}=5 & a_{34}=4 & a_{35}=5 \end{array}$$

Die Elemente des Vektors *X* zeigen an, ob der jeweilige Standort ausgewählt ist oder nicht.

$$\begin{array}{l} X_0=0 \\ X_1=1 \\ X_2=1 \\ X_3=1 \end{array}$$

In diesem Beispiel heisst das, dass Standort 0 nicht ausgewählt ist. Die Standorte 1-3 hingegen sind ausgewählt.

Die Matrix *Y*, welche von *X* abhängig ist, sieht folgendermassen aus:

$$\begin{array}{llllll} Y_{00}=\infty & Y_{01}=\infty & Y_{02}=\infty & Y_{03}=\infty & Y_{04}=\infty & Y_{05}=\infty \\ Y_{10}=12 & Y_{11}=6 & Y_{12}=6 & Y_{13}=7 & Y_{14}=4 & Y_{15}=1 \\ Y_{20}=2 & Y_{21}=8 & Y_{22}=6 & Y_{23}=3 & Y_{24}=12 & Y_{25}=9 \\ Y_{30}=10 & Y_{31}=2 & Y_{32}=4 & Y_{33}=5 & Y_{34}=4 & Y_{35}=5 \end{array}$$

Für jede Spalte *j* wird nun der minimale Wert gesucht. Dieser wird mit d_j bezeichnet.

$$d_0=2 \quad d_1=2 \quad d_2=4 \quad d_3=3 \quad d_4=4 \quad d_5=1$$

Durch eine Summation der minimalen Einzeldistanzen ergibt sich die minimale totale Distanz *D*.

$$D=16$$

Eine kurze Überprüfung von Hand zeigt, dass diese Lösung korrekt ist.

6 Experimentierplan und Analyse der Ergebnisse

Aufgrund der für ein grösseres Szenario ungenügenden Rechenleistung der Library wurden die Experimente nur mit einem sehr kleinen fiktiven Szenario durchgeführt. Das Ziel war es, zu zeigen, dass die optimale Platzierung der Ladestationen funktioniert. Folgende Fragestellungen sollten durch Experimente beantwortet werden:

1. Wie verändert sich die Lösung in Abhängigkeit der Rechenzeit?
2. Wie verändert sich die Lösung bei einer schrittweisen Zunahme der Ladestationen?
3. Wie unterscheidet sich die Lösung einer zufälligen Wahl der Ladestationen von der einer optimierten Platzierung?

Tabelle 5 zeigt das Szenario, mit dem die oben genannten Fragen beantwortet werden sollen.

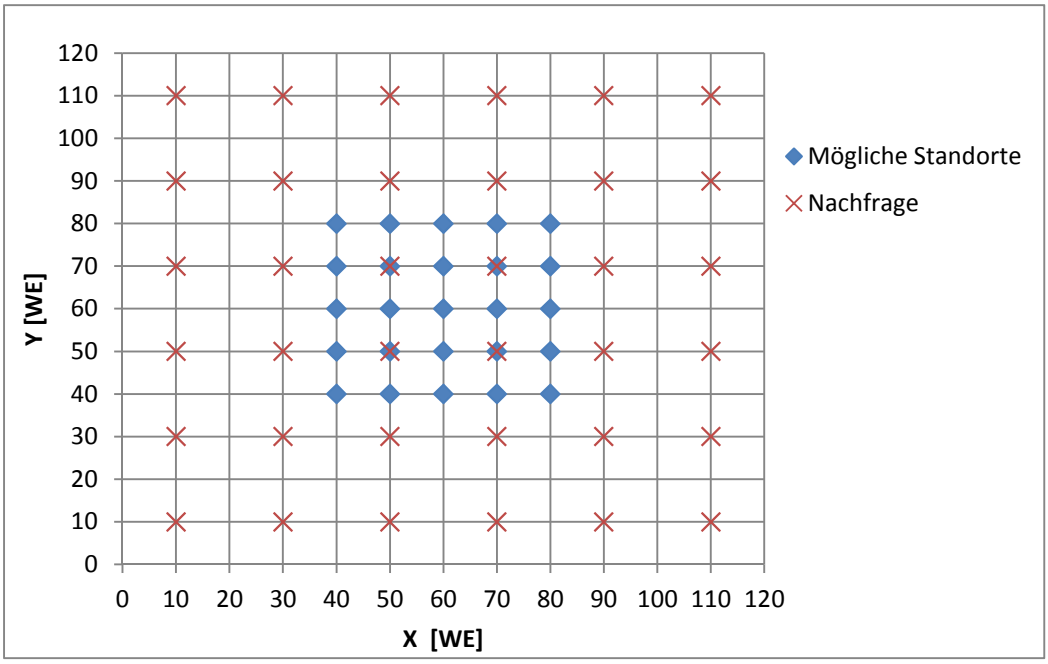
Tabelle 5 Eckpunkte Szenario

Anzahl Nachfragepunkte *k*	Menge der möglichen Standorte *m*	Anzahl zu platzierende Ladestationen *n*
36	25	Variabel

Wie bereits im Beispielszenario (Kap. 5.2.4) können Wege nur in horizontaler oder vertikaler Richtung zurückgelegt werden. Die Distanz wird ebenfalls in Manhattan-Metrik und Wegeinheiten (WE) berechnet.

Abbildung 2 zeigt die räumliche Verteilung der Objekte aus dem Szenario.

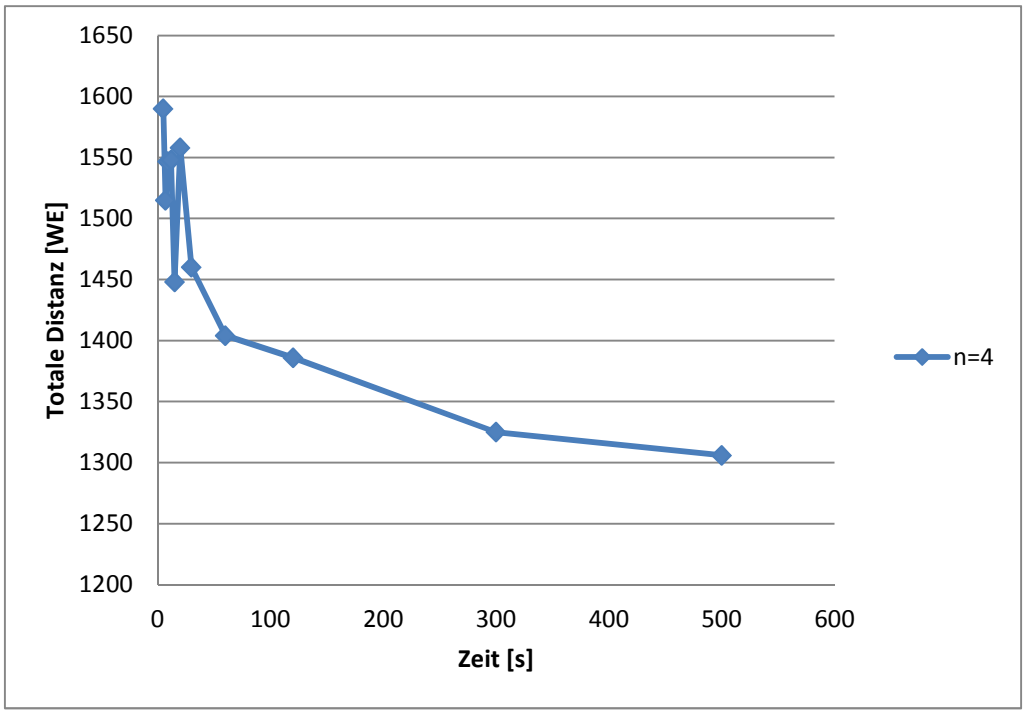
Abbildung 2 Szenario



6.1 Experiment 1: Lösungsqualität in Abhängigkeit der Rechenzeit

Um die erste Fragestellung zu beantworten wurde das Programm mit einer jeweiligen Rechenzeit von 5, 7, 9, 12, 15, 20, 30, 60, 120, 300 und 500 Sekunden gestartet. Um statistische Ausreisser abzumindern wurde dieser Vorgang für jedes Zeitintervall fünf Mal wiederholt. Danach wurde mit dem jeweiligen Mittelwert weitergearbeitet. Die Anzahl der Ladestationen n wurde für dieses Experiment auf n=4 festgesetzt.

Abbildung 3 Lösungsqualität in Abhängigkeit der Rechenzeit



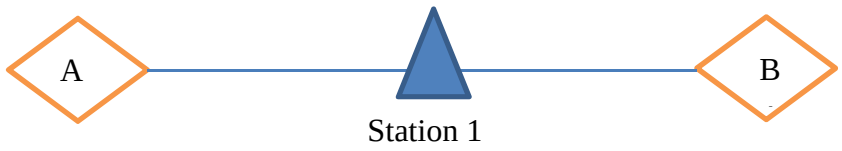
Die Resultate in Abbildung 3 zeigen, dass sich die totale Distanz sämtlicher Fahrzeuge zur nächsten Ladestation mit zunehmender Rechenzeit verkürzt. Für kurze Rechenzeiten springt die Lösung hin und her. Dies ist nachvollziehbar, da Choco den Lösungsbereich mit einem Branch-and-Bound Algorithmus absucht. Mit dem in Kap. 4.2.2 erklärten Verfahren der Tiefensuche wird zwar sehr schnell eine zulässige Lösung gefunden, die Qualität dieser ist jedoch eher zufällig und variiert stark. Bis zu einer Rechenzeit von 20 Sekunden wird meist erst eine Lösung gefunden, erst ab einer Rechenzeit von ca. 30 Sekunden wird die Lösung besser, je länger man dem Programm für die Suche Zeit gibt.

6.2 Experiment 2: Inkrementeller Ausbau und Globale Optimierung

Sowohl Nicholas et al. (2004) als auch Raimondo (2013) lösten ihr Problem mithilfe eines inkrementellen Ansatzes. Die Gefahr eines inkrementellen Ausbaus des Ladestationen-Netzes besteht darin, dass mit fortschreitendem Ausbau des Netzes bereits errichtete Stationen unter Umständen nicht mehr an den optimalen Standorten liegen.

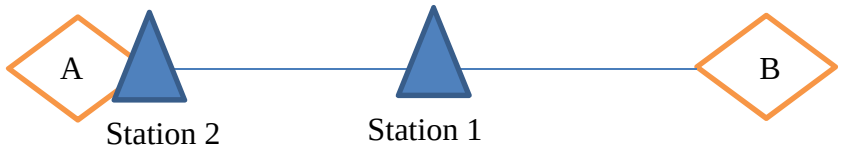
Dies lässt sich anhand eines einfachen Beispiels illustrieren: Innerhalb des untersuchten Gebietes gibt es zwei Nachfragepunkte, z. B. die zwei Dörfer A und B. Für die Errichtung der ersten Ladestation wird dann der dafür optimale Standort gesucht. Dieser befindet sich wie in Abbildung 4 zu sehen genau zwischen den beiden Dörfern.

Abbildung 4 Inkrementeller Ausbau I



Später möchte man noch eine zweite Ladestation errichten. Der dafür optimale Standort ist dann z. B. Dorf A. Die bereits errichtete Station 1 bleibt an ihrem Ort bestehen. In Abbildung 5 wird dieser Endzustand gezeigt.

Abbildung 5 Inkrementeller Ausbau II

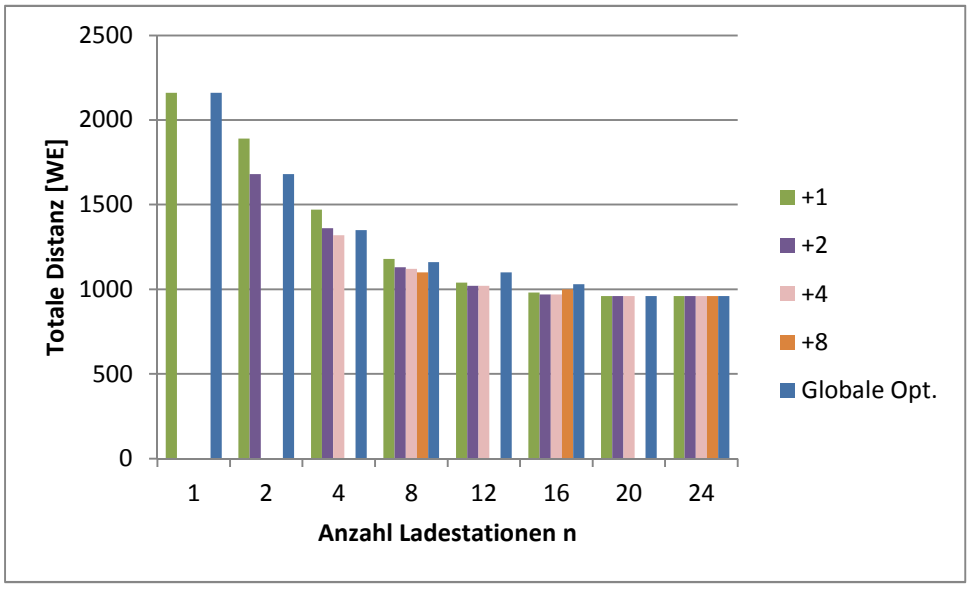


Im Vergleich dazu sieht man in Abbildung 6 den Vorteil einer globalen Optimierung. Werden bereits zu Beginn die für die zwei Stationen optimalen Standorte gleichzeitig gesucht, so erhält man eine Station in Dorf A und eine in Dorf B. Diese Lösung ist im Vergleich zum inkrementellen Ausbau natürlich die bessere.

Abbildung 6 Globale Optimierung

Die zweite Fragestellung lautete: Wie verändert sich die Lösung bei einer schrittweisen Zunahme der Ladestationen? Um dies zu beantworten wurde die Anzahl Ladestationen nach jedem Durchgang um die gewählte schrittweise Zunahme (+1, +2, +4, +8) erhöht. Die bereits ausgewählten Standorte wurden dabei fixiert. Ein Ausbauschritt von +2 bedeutet z.B., dass immer zwei Stationen miteinander errichtet werden. Dabei wird die gegenseitige Abhängigkeit bei der Lösungssuche beachtet. Die Rechenzeit wurde in einem ersten Schritt beschränkt auf 60 Sekunden pro Durchgang. Schnell wurde aber ersichtlich, dass sich die totalen Rechenzeiten der verschiedenen Ausbauschritte stark unterschieden und ein fairer Vergleich nicht möglich war. Während sich die totale Rechenzeit für den Ausbauschritt +8 auf 180 Sekunden belief summierte sich die totale Rechenzeit für die inkrementelle Zunahme um jeweils eine Station auf 1500 Sekunden. Das Experiment wurde daher noch einmal durchgeführt. Diesmal wurde die totale Rechenzeit für alle Ausbauschritte auf 1500 Sekunden festgesetzt. Dies bedeutete Rechenzeiten pro Durchgang zwischen 60 (bei +1) und 500 Sekunden (bei +8). Die maximale Rechenzeit wurde nur zu Beginn ausgenutzt. Nachdem ca. 12 Stationen fixiert waren, konnten die weiteren Standorte in kürzerer Zeit berechnet werden. Für die globale Optimierung wurde das Experiment mit einer Rechenzeit von 60 Sekunden pro Ladestation durchgeführt. Für n=5 belief sich die Rechenzeit so z.B. auf 300 Sekunden. Dadurch liessen sich die verschiedenen Ausbauschritte fair miteinander vergleichen. Abbildung 7 zeigt die totale Distanz in Abhängigkeit der Anzahl Ladestationen n sowie in Abhängigkeit der gewählten Ausbauschritte.

Abbildung 7 Lösungsqualität in Abhängigkeit der Anzahl Ladestationen und der Ausbauschritte des Ladestationen-Netzes



Grundsätzlich sieht man, dass sich die totale Distanz mit der Zunahme der Anzahl Ladestationen verringert hat. Dies erscheint logisch, verdichtet sich doch so das Netz der Ladestationen und die notwendige Fahrdistanz zur nächsten Station nimmt ab. Die Unterschiede zwischen den fünf verschiedenen Ausbauschritten des Netzes sind geringer als erwartet. Es muss beachtet werden, dass hier aufgrund der Versuchsanordnung keine gemittelten Werte berechnet werden konnten. Dies erhöht die Unsicherheit und schwächt die Aussagekraft der Lösung ab.

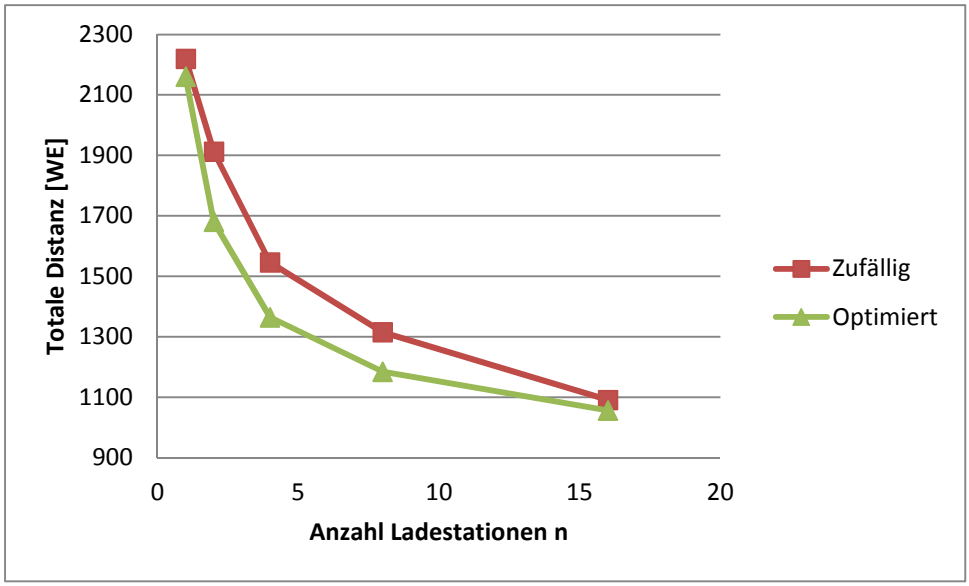
Grundsätzlich sollten sich mit grösseren Ausbauschritten bessere Lösungen erzielen lassen und die globale Optimierung müsste immer die beste Lösung ergeben. Bis zu einer Zunahme von + 4 stimmt diese Vermutung. Für Kurve +8 stimmt diese Aussage zwar nicht zu 100%, es muss jedoch festgehalten werden, dass diese aus sehr wenigen Werten (3) besteht und nur der Wert für n=16 nicht in die aufgestellte These passt. Da jedoch auch die Kurve für die globale Optimierung der oben geäusserten These nicht folgt, muss das Problem woanders liegen. Eine mögliche Erklärung könnte sein, dass in diesem Versuch nur einzelne Werte und keine Mittelwerte verwendet werden konnten und die Resultate sich deshalb nicht mit der These decken. Eine andere Erklärung könnte sein, dass sich die notwendige Rechenzeit für eine gute Lösung mit zunehmender Grösse der Ausbauschritte überproportional erhöht, und man deshalb für den Schritt +8 und die globale Optimierung in der gewählten Rechenzeit noch nicht das beste Ergebnis erhält.

Man sieht in Abbildung 7 jedoch auch, dass die globale Optimierung nur für sehr kleine n deutlich bessere Ergebnisse liefert. Mit einer Zunahme der Anzahl Ladestationen verringert sich der Vorteil der globalen Optimierung. Teilweise bekommt man mit dem inkrementellen Ausbau des Ladestationen-Netzes sogar bessere Resultate. Die Erkenntnis, dass sich durch eine inkrementelle Zunahme die Lösung nur wenig verschlechtert ist sehr wichtig. Durch die inkrementelle Zunahme lässt sich das Problem deutlich vereinfachen und auch auf grössere Szenarien anwenden.

6.3 Experiment 3: Optimierte und zufällige Anordnung der Ladestationen

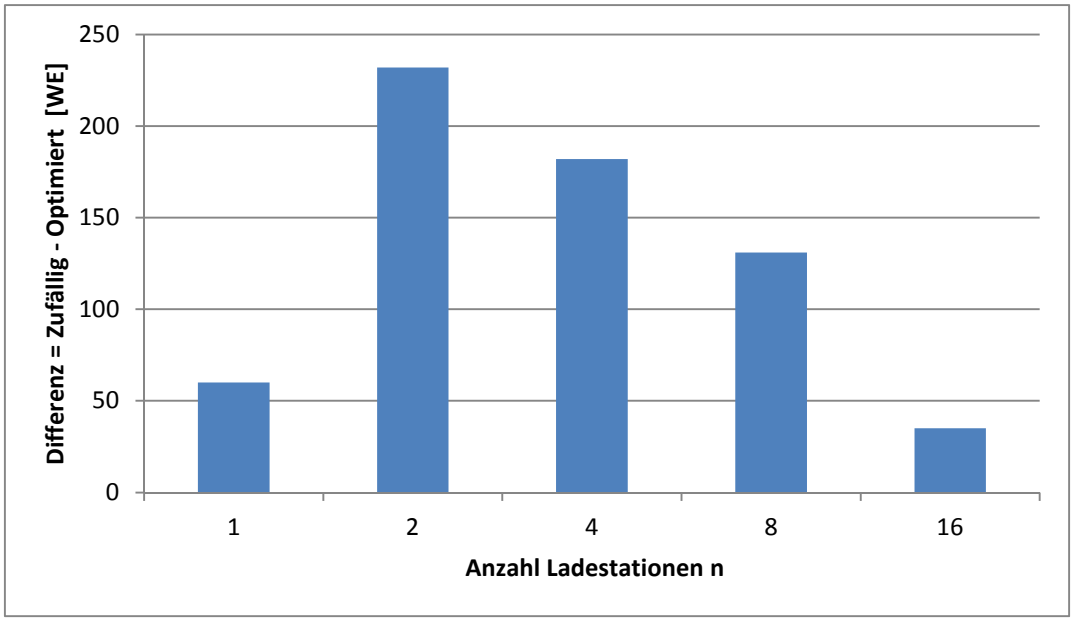
Der Vergleich der zufälligen und der optimierten Standortwahl wurde mit einer Rechenzeit von 120 Sekunden durchgeführt. Um einen stabileren Wert zu bekommen wurde bei der zufälligen Platzierung jeweils der Mittelwert von zehn Durchläufen, bei der homogeneren Lösung für die optimierte Platzierung jeweils der Mittelwert aus fünf Durchläufen berechnet. Dieses Prozedere wurde für die Anzahl Ladestationen von n=1, 2, 4, 8 und 16 wiederholt. Abbildung 8 zeigt die totale Distanz, die zurückgelegt werden muss bei einer zufälligen sowie einer optimierten Platzierung.

Abbildung 8 Zufällige und optimierte Platzierung von Ladestationen



Den Vorteil der optimierten Platzierung sieht man in Abbildung 9 noch einmal deutlicher.

Abbildung 9 Unterschied zwischen zufälliger und optimierter Platzierung



Im Vergleich zwischen einer optimierten und einer zufälligen Platzierung der Ladestationen sieht man in Abbildung 9, dass die optimierte Platzierung stets bessere Werte erreicht. Auffällig ist, dass für n=1 die Differenz nur 60 WE beträgt. Dies hängt mit der Wahl des Szenarios zusammen. Die Wahrscheinlichkeit, dass die zufällige Platzierung auch die ideale ist, beträgt 36%. 9 von 36 möglichen Standorten erreichen das Optimum der totalen Distanz von 2160 Wegeinheiten. Für n=2 und n=4 beträgt der Unterschied deutlich mehr und eine Optimierung der Standortwahl lohnt sich. Mit einer weiteren Zunahme von n nimmt dann der Nutzen der Optimierung wieder ab. Durch die erhöhte Anzahl Ladestationen nimmt die Wahrscheinlichkeit, dass die Stationen gut platziert sind wieder zu. Zudem wird das Netz generell dichter und die totale Distanz nimmt demzufolge ab. Die Rechenzeit für die optimierte Standortwahl wurde auf 120 Sekunden beschränkt. Mit einer Erhöhung dieser würde man für die optimierte Platzierung noch bessere Resultate erhalten und der Unterschied zur zufälligen Lösung würde sich noch weiter vergrössern.

6.4 Allgemeine Erkenntnisse

Der Einsatz von Choco ist in grösseren Szenarien nicht möglich. Bereits für relativ kleine Szenarien beträgt die Rechendauer bis zur korrekten Lösung mehrere Minuten. Durch eine Verbesserung des Programmcodes konnte die Rechenzeit reduziert werden, zufriedenstellend war der Fortschritt aber nicht. Zwar kann die Rechenzeit wie bereits erwähnt beschränkt werden, nur ist dann die Qualität der Lösung unter Umständen mangelhaft. Das grössere Problem ist jedoch der Arbeitsspeicher von Choco. Ab einer bestimmten Anzahl Inputs kann Choco das Problem nicht mehr lösen aufgrund des zu grossen Speicherbedarfs. Die maximale Anzahl Inputs, die gelesen werden können wurde durch ein weiteres Experiment näherungsweise abgeschätzt. Für n=4 und m=25 musste k kleiner als 196 sein, ansonsten reichte die Speicherkapazität nicht. Das Experiment wurde mit den Standard-Einstellungen durchgeführt. Auch eine Erhöhung des Java Virtual Machine Parameters Xmx auf z.B. 2 GB brachte keine Besserung. Eine weitere Schwierigkeit der Library ist das Ersetzen der Standard-Suchstrategie durch eine eigens definierte. Dies könnte das Problem lindern, ist jedoch kompliziert und erfordert einiges an Übung.

Trotzdem ist Choco für verschiedene Arbeiten sicher gut geeignet. Die grössten Pluspunkte sind die Übersichtlichkeit sowie die schnell erlernbare Sprache. Vor allem Constraints lassen sich sehr einfach hinzufügen oder entfernen.

Grundsätzlich könnte man das oben beschriebene Problem auch als Lineare Programmierung (LP) lösen. Der Vorteil von CSP gegenüber LP ist jedoch, dass auch nichtlineare Constraints

eingeführt werden können. Damit könnte man für das TESF ein hilfreiches Tool erarbeiten, um mit möglichst wenig Code eine grosse Bandbreite an Optimierungsproblemen anzugehen. Ein funktionierender Code liesse sich dann sehr schnell programmieren. In einem zweiten Schritt könnte man dann versuchen, das Programm mittels sinnvoller Algorithmen noch zu beschleunigen. Choco enthält beispielsweise sehr viele Constraints für die Termin- und Zeitplanung. Damit liessen sich rasch Algorithmen für die Zeitplanung des optimalen Ladens programmieren.

Hervorzuheben ist sicherlich die Wichtigkeit des inkrementellen Ansatzes. Für grössere Probleme mit vielen Kombinationsmöglichkeiten ist dies eine wirkungsvolle Methode um die Rechenzeit drastisch zu verkürzen.

7 Ausblick

Die Probleme mit der Memory und der Rechengeschwindigkeit von Choco schränkten diese Arbeit ein. Es fehlte auch die Zeit, um sich noch genauer mit der Semantik von Choco 2 zu befassen. Zudem wurde in der Zwischenzeit bereits eine komplett überarbeitete Version der Library, Choco 3, veröffentlicht. Diese müsste man sicher genauer anschauen und auf mögliche Verbesserungen prüfen. Auch die JaCoP-Library wäre sicher eine lohnenswerte Alternative, welche man untersuchen müsste.

Anstatt als CSP könnte man dieses Problem auch als LP lösen. Die Vorteile von CSPs sind jedoch nicht zu unterschätzen. Der grösste Vorteil ist die grundsätzliche Struktur von CSPs. Das Problem wird nicht einfach gelöst, sondern eher beschrieben. Innerhalb des Lösungsbereichs wird dann eine gute Lösung gefunden. Für die meisten Probleme genügt eine solche Lösung. Auch die sehr einfache Einführung von Constraints ist ein wichtiger Vorteil gegenüber einer Programmierung von Hand. Ein Nachteil ist dafür die grosse Abhängigkeit von der Leistung der Library.

Weiterentwicklungen könnte man auch noch bei der Fragestellung vornehmen. Anstatt der totalen Distanz könnte man beispielsweise die generalisierten Kosten minimieren. Dazu würden dann auch die Betriebs- sowie die Baukosten für die Ladestationen gehören. Zudem müsste man den zurückgelegten Weg in eine Geldeinheit umrechnen.

Man könnte auch die Auslastung der Ladestationen noch als Kriterium hinzufügen. Dies setzt die Einführung der Dimension Zeit voraus und eröffnet weitere Perspektiven. Die Lösung des Problems würde dadurch aber auch deutlich erschwert.

8 Literatur

Chung, S.H. und C. Kwon (2012) Multi-Period Planning for Electric-Car Charging Station Locations: a Case of Korean Expressways.

Ciari, F. und K.W. Axhausen (2011) Modeling Location decisions of retailers with an agent-based approach, IVT, ETH Zürich.

Frick, M., K.W. Axhausen, G. Carle und A. Wokaun (2007) Optimization of the distribution of compressed natural gas (CNG) refueling stations: Swiss case studies, *Transportation Research Part D: Transport and Environment*, **12**(1), 10-22.

He, J., B. Zhou, C. Feng, H. Jiao und J. Liu (2012) Electric Vehicle Charging Station Planning Based on Multiple-Population Hybrid Genetic Algorithm, *2012 International Conference on Control Engineering and Communication Technology*, Shenyang, Dezember 2012.

Guha, S. und S. Khuller (1999) Greedy strikes back: Improved facility location algorithms, *Journal of Algorithms*, **31**(1), 228-248.

Hõimoja, H. und A. Rufer (2012) Infrastructure Issues Regarding the Ultrafast Charging of Electric Vehicles, *International Advanced Mobility Forum.*

Jia, L., Z. Hu, Y. Song und Z. Luo (2012) Optimal siting and sizing of electric vehicle charging stations, *Electric Vehicle Conference (IEVC), 2012 IEEE International* (pp. 1-6), IEEE, März 2012.

Johnson, J.A., M.A. Chowdhury, Y. He, und J. Taiber (2012) Facilitating the Battery Charging Process in Electric Vehicles Through Connected Vehicle and Infrastructure, *Transportation Research Board 91st Annual Meeting* (No. 12-4488).

Laburthe F. und N. Jussien (2012) Choco solver Documentation, available at http://choco.sourceforge.net, 2004–2013.

Lin, Z., J. Ogden, Y. Fan und C.W. Chen (2008) The fuel-travel-back approach to hydrogen station siting, *international journal of hydrogen energy*, **33**(12), 3096-3101.

Liu Z.F., W. Zhang, X. Ji und K. Li (2012) Optimal Planning of Charging Station for Electric Vehicle Based on Particle Swarm Optimization, *2012 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, Tianjin, Mai 2012.

Mladenović, N., M. Labbé und P. Hansen (2003) Solving the p-Center problem with Tabu Search and Variable Neighborhood Search, *Networks*, **42**(1), 48-64.

Nicholas, M.A., S.L. Handy und D. Sperling (2004) Using geographic information systems to evaluate siting and networks of hydrogen stations, *Transportation Research Record: Journal of the Transportation Research Board*, **1880**(1), 126-134.

Poesio, M., S.S. im Walde, und C. Brew (1998) Lexical clustering and definite description interpretation, i*n Proc. of the AAAI Spring Symposium on Learning for Discourse* (pp. 82-89).

Raimondo, D.M. (2013) Optimal placement of wind turbines, *presented at ETH Zurich*, April 2013

Raviv, T. (2012) The battery switching station scheduling problem, *Operations Research Letters.*

Resende, M.G. und R.F. Werneck (2004) A hybrid heuristic for the p-median problem, *Journal of heuristics*, **10**(1), 59-88.

Speidel, S., F. Jabeen, D. Olaru, D. Harries und T. Bräunl (2012) Analysis of Western Australian electric vehicle and charging station trials, *Australasian Transport Research Forum (ATRF)*, 35th, Perth, Western Australia, 2012.

Waraich, R.A., M.D. Galus, C. Dobler, M. Balmer, G. Andersson und K.W. Axhausen (2009) Plug-in Hybrid Electric Vehicles and Smart Grid: Investigations Based on a Micro-Simulation, *12th International Conference on Travel Behaviour Research (IATBR)*, Jaipur, Dezember 2009.

Waraich, R.A, G. Georges, M.D. Galus und K.W. Axhausen (2013) Manuscript submitted for publication, "Adding Electric Vehicle Modelling Capability to an Agent-based Transport Simulation, *Arbeitsberichte Verkehrs- und Raumplanung*, **879**, IVT, ETH Zürich.

Wikipedia (2013) Branch-and-Bound, http://de.wikipedia.org/wiki/Branch-and-Bound, Mai 2013

Wilonsky, R. (2011) Electric Vehicle Charging Stations Coming to Fair Park As Part of Energy Dept.'s "EV Project", Dallas Observer Blogs, http://blogs.dallasobserver.com/unfairpark/2011/11/electric_vehicle_charging_stations_coming_to_fair_park_as_part_of_energy_depts_ev_project.php, Mai 2013.

9 Glossar

CSP	Constraint Satisfaction Problem
EF	Elektrische Fahrzeuge
LP	Lineare Programmierung
TESF	Transportation Energy Simulation Framework
WE	Wegeinheit
D	Zielfunktion totale Distanz
X	Binärvariable, zeigt ob Standort ausgewählt ist
Y	korrigierte Distanzmatrix
a	tatsächliche Distanzmatrix
d	minimale Distanz von einem Nachfragepunkt zu einer ausgewählten Ladestation
m	Menge der möglichen Standorte für Ladestationen
n	Anzahl zu platzierende Ladestationen
k	Anzahl Nachfragepunkte (Bedarf)

Anhänge

A 1 Programmcode

Mit diesem Code wurde das Experiment 1 durchgeführt. Da der Code für die Experimente 2 und 3 nur unwesentlich verändert werden musste, wird auf eine weitere Auflistung verzichtet.

```
public class Exp_1 {

	/**
	 * @param args
	 */
	public static void main(String[] args) {


		// Initialisierung des Models m und des Solvers s
		Model m = new CPModel();
		Solver s = new CPSolver();

		int numberOfChargingLocations = 16;
		int timelimit = 120 * 1000;

		LinkedList<Coord> demandCoordinates = new LinkedList<Coord>();
		LinkedList<Coord> possibleChargingLocationCoordinates = new
		LinkedList<Coord>();

		// Initialisierung der Demand-Koordinaten
		for (int i = 0; i < 6; i++) {
			for (int j = 0; j < 6; j++) {
				demandCoordinates.add(new Coord(10 + 20 * i, 10 + 20 * j));
			}
		}
		// Initialisierung der Koordinaten der möglichen Standorte für die
		// Ladestationen
		for (int i = 0; i < 5; i++) {
			for (int j = 0; j < 5; j++) {
				possibleChargingLocationCoordinates.add(new Coord(40 +
				10 * i,40 + 10 * j));
			}
		}

		// Binärvariable X[i] zeigt, ob der mögliche Standort i ausgewählt ist
		// oder nicht
		IntegerVariable X[] = Choco.makeBooLeanVarArray("X",
				possibleChargingLocationCoordinates.size(), Options.V_ENUM);

		// Initialisierunge des Arrays XInverted[]
		IntegerExpressionVariable XInverted[] = new
		IntegerExpressionVariable[possibleChargingLocationCoordinates.size()];
```

```
for (int i = 0; i < possibleChargingLocationCoordinates.size(); i++) {
     m.addVariables(X[i]);

     // XInverted[i]=1-X[i]
     XInverted[i] = Choco.minus(1, X[i]);
}
// Constraint wird definiert
// Summe des Arrays X muss gleich der Anzahl Ladestationen sein
m.addConstraint(Choco.eq(Choco.sum(X), numberOfChargingLocations));

// Initialisierung der Distanz-Matrix a
int a[][] = new int[possibleChargingLocationCoordinates.size()]
[demandCoordinates.size()];

// Berechnen des Abstands von Nachfrage j zu möglichem Standort i
for (int i = 0; i < possibleChargingLocationCoordinates.size(); i++) {
     for (int j = 0; j < demandCoordinates.size(); j++) {
          a[i][j] = (int)
          Math.round((Math.abs(demandCoordinates.get(j).x-
          possibleChargingLocationCoordinates.get(i).x) +
          Math.abs(demandCoordinates.get(j).y-
          possibleChargingLocationCoordinates.get(i).y)));
     }
}

// Initialisierung der Matrix Y
IntegerExpressionVariable Y[][] = new
IntegerExpressionVariable[possibleChargingLocationCoordinates
.size()][demandCoordinates.size()];

// Berechnung der Werte für Y
for (int i = 0; i < possibleChargingLocationCoordinates.size(); i++) {
     for (int j = 0; j < demandCoordinates.size(); j++) {
          Y[i][j] = Choco.plus(Choco.mult(a[i][j], X[i]),
                    Choco.mult(100000, XInverted[i]));
          // Anstatt 100000 auch eine andere grosse Zahl zulässig.
     }
}

// Initialisierung des Arrays dist, enthält für jede Nachfrage j die
// minimale Distanz zur nächsten Ladestation
IntegerExpressionVariable dist[] = new
IntegerExpressionVariable[demandCoordinates.size()];

// Berechnung der minimalen Distanz für jede Nachfrage j
for (int j = 0; j < demandCoordinates.size(); j++) {
     dist[j] = Choco.min(getColumn(Y, j));
}

// totalerUmweg = Summe des Arrays dist
IntegerExpressionVariable totalerUmweg = Choco.sum(dist);

// Initialisierung der Zielvariablen tot
IntegerVariable tot = Choco.makeIntVar("tot", Options.V_OBJECTIVE);
m.addVariable(tot);

// Constraint, tot = totalerUmweg
```

```
        m.addConstraint(Choco.eq(tot, totalerUmweg));

        // Model m wird gelesen von Solver s
        s.read(m);
        s.clearGoals();
        ChocoLogging.toSolution();

        Random random = new Random();

        // Definierung der Suchstrategie
        MyVariableSelector randomIntVarSelector = new MyVariableSelector(s);

        MyValueSelector randomIntValSelector = new MyValueSelector(s,
                  possibleChargingLocationCoordinates.size(),
                  numberOfChargingLocations);

        s.addGoal(new AssignVar(randomIntVarSelector, randomIntValSelector));

        // Maximale Rechenzeit wird bestimmt
        s.setTimeLimit(timelimit);
        s.getConfiguration().putInt(Configuration.SOLUTION_POOL_CAPACITY, 100);

        // Zielvariable soll minimiert werden
        s.minimize(true);

        // Ausgabe der totalen Distanz
        System.out.println("Totale Distanz  " + s.getVar(tot));

        // Ausgabe des Arrays X
        for (int i = 0; i < possibleChargingLocationCoordinates.size(); i++) {
             System.out.println(s.getVar(X[i]));
        }

    }

    private static class MyVariableSelector extends AbstractIntVarSelector {

        public MyVariableSelector(Solver solver) {
             super(solver);
        }

        @Override
        public IntDomainVar selectVar() {
        ArrayList<IntDomainVar> candidates = new ArrayList<IntDomainVar>();

             for (int i = 0; i < vars.length; i++) {
                  if (!vars[i].isInstantiated()) {
                       if (vars[i].getName().toString().contains("X")) {
                            candidates.add(vars[i]);
                       }
                  }
             }

             if (candidates.size() > 0) {
                  Random random = new Random();
                  int nextIndex = random.nextInt(candidates.size());
                  return candidates.get(nextIndex);
```

```
            }

            for (int i = 0; i < vars.length; i++) {
                if (!vars[i].isInstantiated()) {
                    return vars[i];
                }
            }
            return null;
        }

    }

    private static class MyValueSelector implements ValSelector<IntDomainVar> {

        private Solver solver;
        private int totalNumberOfStations;
        private int numberOfStrations;

        public MyValueSelector(Solver solver, int totalNumberOfStations,
                int numberOfStrations) {
            this.solver = solver;
            this.totalNumberOfStations = totalNumberOfStations;
            this.numberOfStrations = numberOfStrations;
        }

        /**
         * try to select 'totalNumberOfStations'
         *
         * @param x
         *            the variable under consideration
         * @return what seems the most interesting value for branching
         */
        public int getBestVal(IntDomainVar var) {
            if (var.getName().toString().contains("X")) {
                Random r = new Random();

                if (r.nextInt(totalNumberOfStations) > totalNumberOfStations
                        - numberOfStrations - 1) {
                    return 1;
                } else {
                    return 0;
                }
            }

            return var.getInf();
        }
    }

    public static IntegerExpressionVariable[] getRow(
            IntegerExpressionVariable[][] matrix, int rowIndex) {
        int numColumn = matrix[0].length;
        IntegerExpressionVariable[] result = new
        IntegerExpressionVariable[numColumn];

        for (int i = 0; i < numColumn; i++) {
            result[i] = matrix[rowIndex][i];
        }
```

```
            return result;
      }

      public static IntegerExpressionVariable[] getColumn(
                  IntegerExpressionVariable[][] matrix, int colIndex) {
            int numRows = matrix.length;
            IntegerExpressionVariable[] result = new
            IntegerExpressionVariable[numRows];

            for (int i = 0; i < numRows; i++) {
                  result[i] = matrix[i][colIndex];
            }

            return result;
      }


}
```