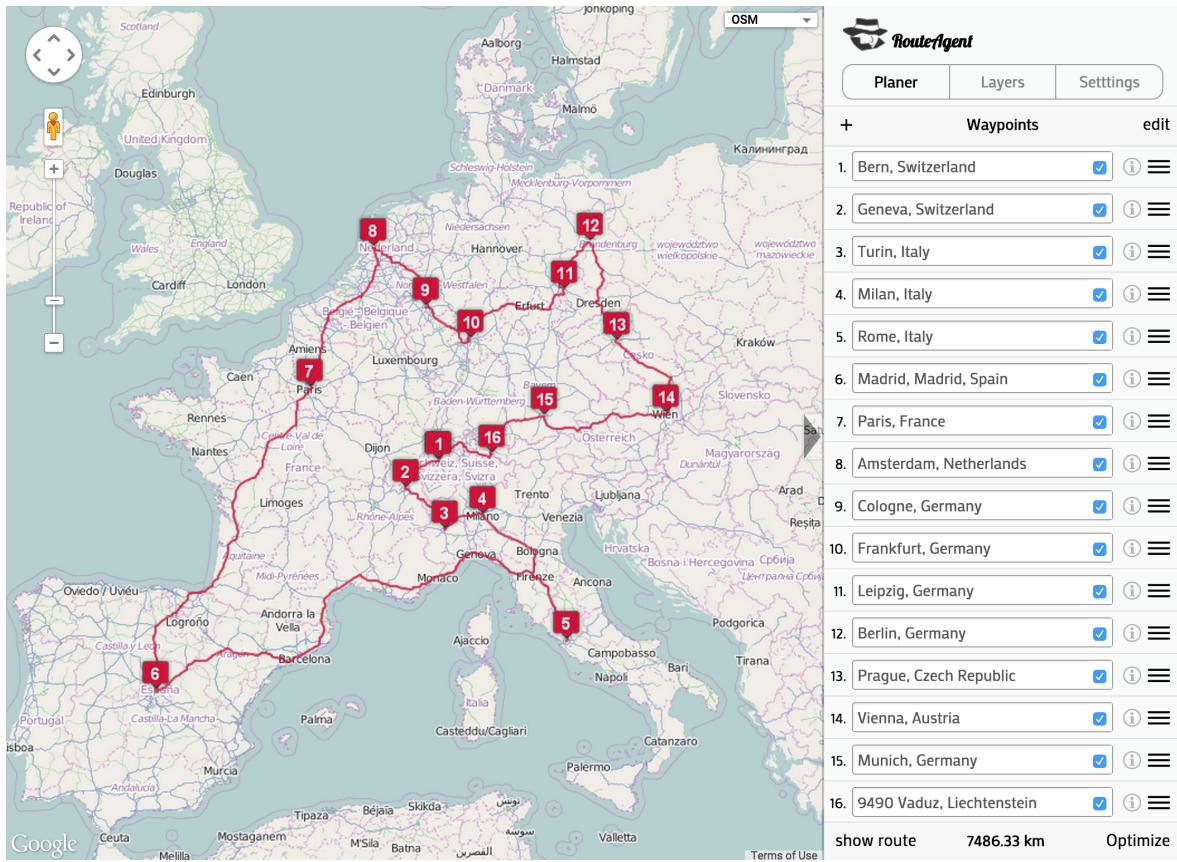


BACHELORTHESES FS 2015

EVALUATION VON LÖSUNGSKONZEPTEN VON NP-VOLLSTÄNDIGEN PROBLEMEN IN WEBAPPLIKATIONEN

AM BEISPIEL DES TRAVELLING SALESMAN PROBLEMS



DIPLOMAND

Roman Lickel, S07906670
lickerom@students.zhaw.ch

BETREUUNGSPERSON

Alain Lafon
xlaf@zhaw.ch

Zusammenfassung

Ein Endkunde von der Firma Bachmann Support GmbH führt eine Reiseunternehmung und verwaltet Reiseziele und Hotels im Programm FlexBüro. Um die Route bezogen auf die Fahrzeit zu minimieren, ist eine Systemerweiterung des Programms gefordert, welche mittels Webtechnologien umgesetzt werden soll.

Die Optimierung von Wegrouten von einem Startpunkt über mehrere Wegpunkte (*WP*), zurück zum Startpunkt wird als *Travelling Salesman Problem (TSP)* bezeichnet. Der Aufwand für die exakte Berechnung der Lösung für eine steigende Anzahl *WP* nimmt mehr als exponentiell zu, weshalb mittels geeigneter Architekturen und Näherungsalgorithmen versucht wird, das Problem effizient zu lösen.

In der vorliegenden Bachelorarbeit werden mittels einer Markt- und Anforderungsanalyse die genauen Anforderungen an das System erstellt. Es wird versucht, die aus der Literatur zusammengetragenen und aus anderen Systemen bekannten Architekturen auf das Webumfeld anzuwenden. Um die Architekturvarianten in der anschliessenden Nutzwertanalyse vergleichen zu können, werden Konzepte und Proof of Concepts einiger davon erstellt.

Aus der Arbeit geht hervor, dass der Einfluss auf die Berechnungsgeschwindigkeit von der richtigen Algorithmenwahl viel grösser ist als der Einfluss der Wahl der Architektur. Dennoch wird gezeigt, dass auch im Web mittels geeigneter Architekturwahl eine Effizienzsteigerung der Berechnung erreicht werden kann.

Vorwort

Roman Lickel, School of Engineering, ZHAW Zürich:

Während der gesamten Projektdauer zwischen Oktober 2014 und Mai 2015 durfte ich auf die Unterstützung folgender Personen zählen, welchen ich meinen Dank aussprechen will:

Meinem Betreuer Alain Lafon danke ich für seine hilfreichen Anregungen und seine konstruktive Kritik bei der Erstellung dieser Arbeit.

Ein gebührender Dank gilt Bruno Bachmann und Ramon Selinger von der Firma Bachmann Support GmbH, welche immer offene Ohren bei technischen Fragen hatten.

Armin Stucki, gab hilfreiche Tipps zur Gestaltung der Bachelorarbeit und übernahm das Korrekturlesen.

Meiner Freundin Seline Jossi spreche ich einen innigen Dank für die alltägliche Unterstützung und Hilfe während der gesamten Studienzeit aus. Besonders wichtig war dies in der intensiven Zeit während der Bachelorarbeit.

Mein ganz besonderer Dank gilt abschliessend meiner Mutter Gerda Lickel, meinem Vater Bruno Lickel, meinem Bruder Simon Lickel und meinen beiden Schwestern Aline und Janah Lickel. Sie sind mir stets helfend zur Seite gestanden.

Inhaltsverzeichnis

I Einleitung	1
1 Problemabgrenzung	2
1.1 Ausgangslage	2
1.2 Ziel der Arbeit	2
1.3 Aufgabenstellung	2
1.4 Erwartete Resultate	3
1.5 These	3
1.5.1 Methodik	4
2 Projektplanung	5
2.1 Termine	5
II Theoretische Grundlagen	7
3 NP-vollständige Probleme	8
3.1 Komplexitätsklassen	8
3.2 Das <i>Travelling Salesman Problem (TSP)</i>	8
3.3 Algorithmen zur Lösungsfindung des <i>TSP</i>	10
3.3.1 Exakte Algorithmen	10
3.3.2 Heuristik	12
4 Geodäsie	13
5 Webtechnologien	15
5.1 Grundlagen	15
5.1.1 Relevante Internetprotokolle	16
5.2 Web 2.0	18
6 Parallelle und verteilte Berechnung	19
6.1 Parallelrechnung	19
6.2 Verteilte Systeme	19
6.3 Im Webumfeld	23
6.3.1 Multithreading im Browser	23
6.3.2 Mehrprozessor-Berechnung auf dem Server	24
6.3.3 Verteilte Systeme auf mehreren Clients	24

6.3.4 Verteilte Systeme auf mehreren Servern	24
7 Kartendienste und Frameworks	25
7.1 Kartenserver	25
7.2 Routingserver	26
7.3 Map Clients	26
III Lösungserarbeitung	27
8 Anforderungskonflikt	28
9 Requirements Engineering (R1)	29
9.1 Problembeschreibung	29
9.2 IST-Analyse	29
9.3 Zieldefinition	30
9.3.1 Endprodukt	30
9.3.2 Bachelorarbeit	30
9.4 Stakeholderliste	31
9.5 Annahmen und Einschränkungen des Systems	31
9.6 Systemkontext und Randbedingungen	32
9.7 Anforderungsanalyse	33
9.7.1 Funktionale Anforderungen	34
9.7.2 Nicht-funktionale Anforderungen	34
9.7.3 Anwendungsfälle (Use Cases)	36
9.8 Entscheidungen	37
9.8.1 Technologien	37
9.8.2 Karten- und Routingserver	37
9.8.3 Map Clients	39
9.9 Risikoanalyse	39
10 Existierende Applikationen (R0)	42
10.1 Kennzahlen	42
11 Lösungskonzepte	46
12 Anwendung auf das TSP	50
12.1 Vorversuche	50
12.1.1 Parallelere Berechnung im Web	50
12.1.2 Verteilte Berechnung unter Clients im Web	53

12.2 Algorithmenwahl (R3)	56
12.3 Lösungskonzepte für das TSP im Web	59
12.3.1 Technologie-/Architekturkonzepte (R2)	59
12.3.2 Konzept der Algorithmenauswahl	68
13 POCs der Architekturvarianten (R6)	69
13.1 Implementation	69
13.2 Debugging	72
13.3 Dokumentation	72
13.4 Testing	73
13.5 Benchmark	73
14 Nutzwertanalyse (R4)	75
IV Ergebnisse	80
15 Resultate (R5,R6)	81
15.1 Kritische Betrachtung der Resultate	83
16 Produkt (R6)	84
17 Reflexion und Fazit	85
18 Weiterführende Untersuchungsgebiete	86
19 Schlusswort	87
V Anhang	88
A Proof of Concepts (POCs)	89
B Quellcode	90
C Verwendete Soft- und Hardware	91
D Verzeichnisse	92
D.1 Glossar	92
D.2 Abbildungsverzeichnis	95
D.3 Tabellenverzeichnis	96
D.4 Quellenverzeichnis	97
E Eigenständigkeitserklärung	101

Teil I

Einleitung

Der erste Teil dient der Problemabgrenzung und Projektplanung. Daneben wird die zugrundeliegende These vorgestellt. Ebenfalls wird in diesem Teil die Methode erwähnt, wie die These bewiesen oder widerlegt werden soll.

1 Problemabgrenzung

Dieses Kapitel gibt einen Überblick über die Bachelorarbeit, die Aufgabenstellung, das Ziel der Arbeit und die darin behandelten Themen. Ebenfalls sind die erwarteten Resultate für die Bachelorarbeit und die Problemabgrenzung hier festgehalten.

1.1 Ausgangslage

Täglich müssen in vielen Geschäften Kundenbesuche oder andere Rundreisen zusammengestellt werden. Gute Routenplanung und deren Optimierung sind wichtige Grundlagen, um Kosten sparen zu können. Wenn eine Route optimal zusammengestellt wird, können sowohl Zeit als auch weitere Ressourcen eingespart werden. Das zugrunde liegende Problem wird *Travelling Salesman Problem (TSP, Problem des Handelsreisenden)* genannt. Der Aufwand zur Berechnung nimmt mit steigender Anzahl Wegpunkten (*WP*) überexponentiell zu. Es stellt auch in heutigen Programmen eine Schwierigkeit dar, eine Route in nützlicher Zeit berechnen zu können. Mittels *Heuristik* können Näherungslösungen berechnet werden, welche jedoch gegebenenfalls nicht die optimalste Lösung darstellen. Diese Näherungslösungen benötigen weniger Zeit und Ressourcen, lösen aber das Problem dennoch für die jeweilige Anwendung genügend genau. Aus diesem Grund bietet sich die Heuristik für den Einsatz im Webumfeld an. Die Firma Bachmann Support GmbH, mein derzeitiger Arbeitgeber, entwickelt ein *CRM* System für kleine und mittelgrosse Unternehmen und möchte einen solchen Routenplaner in die Applikation integrieren, damit die *CRM* Benutzer ihre Kundenbesuche besser planen können.

1.2 Ziel der Arbeit

Der Fokus der Arbeit liegt darin, mögliche Implementationsvarianten von existierenden, geeigneten Algorithmen in einer Webapplikation zu eruieren, zu vergleichen und anhand einiger Teileimplementationen zu realisieren. In der Arbeit sollen verschiedene Lösungskonzepte für das *TSP* in Webapplikationen miteinander verglichen und das Geeignetste weiter ausgearbeitet werden. Es geht in der Arbeit nicht darum, neue Algorithmen zur Lösung des *TSP* zu erarbeiten, sondern vielmehr um das Zusammenspiel und mögliche Kombinationsvarianten der Algorithmen für die Lösung des Problems im Webumfeld.

1.3 Aufgabenstellung

In der Bachelorarbeit werden die folgenden Aufgabestellungen verfolgt:

- A0 Analyse von bereits bestehenden, öffentlich verfügbaren Applikationen.

- A1 Anforderungsanalyse (Requirements Engineering).
- A2 Technische Möglichkeiten zur Einbindung von Algorithmen zur Lösung des *TSP* im Webumfeld erarbeiten.
- A3 Vergleich geeigneter Algorithmen zur Lösung des *TSP* erstellen.
- A4 Vergleich der technischen Möglichkeiten aus Punkt A2.
- A5 Erarbeitung eines technischen Konzepts zur Erstellung einer Webapplikation, welches das *TSP* löst.
- A6 Proof of Concept (im Folgenden abgekürzt als *POC* bezeichnet) einzelner Teilkomponenten.

1.4 Erwartete Resultate

Aus der Aufgabenstellung ergeben sich folgende Resultate, welche in drei Teile gegliedert sind:

Ein technisch-wissenschaftlicher Bericht:

- R0 Marktübersicht von bereits existierenden, öffentlich zugänglichen Applikationen.
- R1 Artefakte der Anforderungsanalyse: Problembeschrieb, Zieldefinition, Stakeholderanalyse, Systemabgrenzung, User Stories.
- R2 Kapitel über technische Grundlagen zur Einbindung von Algorithmen zur Lösung des *TSP* im Webumfeld.
- R3 Gegenüberstellung und Eignungsprüfung von geeigneten Algorithmen zur Lösung des *TSP* im Webumfeld.
- R4 Vergleich der ausgearbeiteten Möglichkeiten in einer Nutzwertanalyse.
- R5 Gesamtkonzept für die Kombination der Resultate in R3, R4 und R5.

Implementation:

- R6 *POCs* einzelner Teilkomponenten aus Punkt R3, R4 und R5.

Zusammenfassung, Präsentation, Poster:

- R7 Zusammenfassung des obig beschriebenen technisch-wissenschaftlichen Berichts.
- R8 Präsentation.
- R9 Poster.

1.5 These

Die These der Arbeit besteht darin, dass bekannte Architekturen und Implementationsvarianten aus anderen Bereichen der Informationstechnologie zur Lösung von *NP-vollständigen* Problemen (Kapitel 3) auch auf den Webbereich anwendbar sind. Für

die Validierung werden *POCs* unter Verwendung derzeit verfügbarer Webtechnologien erstellt.

1.5.1 Methodik

Die Strukturierung der Bachelorarbeit widerspiegelt die Vorgehensweise der Erarbeitung, welche in folgende Schritte unterteilt werden kann:

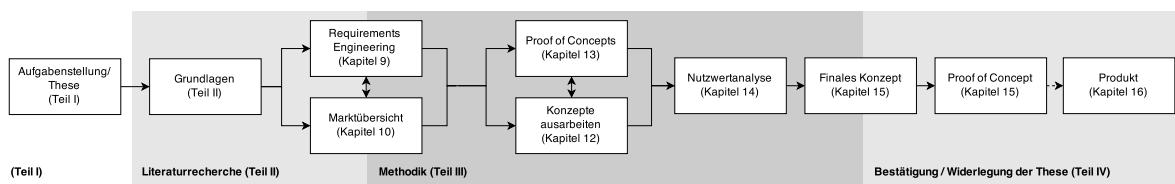


Abb. 1.1: Lösungsvorgehen (*Quelle: Eigene Darstellung*)

Abb. 1.1 zeigt, wie ausgehend von der Aufgabenstellung und These, mittels Literaturrecherchen die Grundlagen der notwendigen Themenbereiche erarbeitet und im Teil II "Grundlagen" zusammengefasst werden. Mit dem Auftraggeber Bachmann Support GmbH und dem Endkunden *EK* werden, unter Einfluss der Marktanalyse, die Anforderungen an das Produkt definiert. Die Nutzwertanalyse wird mit Hilfe von den auf der Literatur und der Anforderungsanalyse basierenden Konzepten und *POCs* erstellt. Das aus der Nutzwertanalyse resultierende finale Konzept und dessen *POC* wird die These bestätigen oder widerlegen. Bei erfolgreicher Bestätigung kann daraus das Produkt für die Bachmann Support GmbH und den Endkunden *EK* abgeleitet und implementiert werden.

2 Projektplanung

2.1 Termine

Tab. 2.1 zeigt die festgesetzten Termine für die Bachelorarbeit, von der Erfassung der Aufgabenstellung im System der ZHAW (*EBS*) bis zur Schlusspräsentation.

Datum	Beschreibung
12.11.14	Freigabe des Themas im <i>EBS</i>
3.12.14	Kick-Off Termin
18.03.15	Design Review
14.05.15	Abgabe der Bachelorarbeit
24.06.15	Präsentation Bachelorarbeit

Tab. 2.1: Administrative Termine der Bachelorarbeit (*Quelle: Eigene Darstellung*)

Tab. 2.2 listet die ausgearbeiteten Projektphasen und Meilensteine für den Arbeitsverlauf auf und gibt deren Aufwandsschätzung an. Sie stellt den Zusammenzug der als Gantt-Diagramm in der nachfolgenden Abb. 2.1 dargestellten Zeitplanung dar.

Schätzung [h]	Projektphasen	Meilenstein
3	Aufgabenstellung ausarbeiten und im <i>EBS</i> erfassen	01.10.14
12	Recherche und Analyse bestehender Applikationen (R0)	28.12.14
50	Anforderungsanalyse (R1)	15.02.15
60	Technische Grundlagen im Webumfeld (R2)	15.01.15
80	Grundlagen <i>TSP</i> erarbeiten und Algorithmen evaluieren (R3)	31.01.15
80	Nutzwertanalyse und Gesamtkonzept für die Implementation des <i>TSP</i> im Webumfeld (R4,R5)	25.03.15
70 (250)	Proof of Concept von Teilkomponenten (R6) Implementation des Konzepts im Auftrag von Bachmann Support GmbH (nicht im Rahmen der Arbeit)	15.04.15 -
6	Zusammenfassung der Bachelorarbeit (R7)	10.05.15
6	Präsentationsvorbereitungen (R8)	14.05.15
10	Poster zur Bachelorarbeit (R9)	18.05.15
5	Administratives (Email, Koordination)	-
382	Total	

Tab. 2.2: Grobe Aufwandsschätzung (*Quelle: Eigene Darstellung*)

Auswertung:

Aufwand geplant:	382 Stunden	Differenz:	-23.3 Stunden	Verhältnis Konzept / Umsetzung	Konzept geplant	Konzept effektiv	Umsetzung geplant	Umsetzung effektiv	Verhältnis geplant (%)	Verhältnis effektiv (%)
Aufwand effektiv:	405 Stunden				312	309	70	99	22	32

Task	Aufwand [Stunden]	November	Dezember	Januar	Februar	März	April	Mai
		2014				2015		

Vorstudie: Vorabklärungen / Vorbereitungen / Administratives

Administratives	geplant	KickOff 26. Nov 14 8 :<> Thema im EBS erlässen	Besprechung Betreuer	Besprechung Betreuer	Besprechung Betreuer	Design Review	Besprechung Betreuer	Abgabe : 14.05.15
	ist	KickOff 3. Dez 14 9 :2 2 Terminplanungsarbeiten	Besprechung Betreuer					

Zusammenfassung, Poster und Präsentation

Zusammenfassung / Poster und Präsentation										Milestein																																																																																												
Zusammenfassung / Kurzfassung	geplant	6	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
	ist	6	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
Präsentation, Poster	geplant	16	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
	ist	13	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

Projektphasen

Geschätzter Aufwand pro Woche: 7 10 17 20 20 18 16 30.5 26 18 21.5 16 16 14 14 16 16 16.5 9 5 16 16 0.5 13 3 4 3

Abb. 2.1: Zeitplanung als Gantt-Diagramm (Quelle: Eigene Darstellung)

Teil II

Theoretische Grundlagen

In diesem Teil werden die Grundlagen und Technologien erläutert, welche für die nachfolgenden Teile der Bachelorthesis benötigt werden. Basiskompetenzen in Informationstechnologien werden für das Verständnis vorausgesetzt.

3 NP-vollständige Probleme

Das Kapitel "NP-vollständige Probleme" erläutert die Grundlagen von Problemen, welche sich nichtdeterministisch in Polynomialzeit lösen lassen und abgekürzt als NP-vollständige Probleme bezeichnet werden. Es handelt sich um ein Thema der Komplexitätstheorie aus der Theoretischen Informatik. Neben der Kategorisierung solcher Probleme und deren Komplexitätsstufen wird der Fokus auf das *Travelling Salesman Problem* und deren Algorithmen zur Lösung gelegt [22], [23].

Definition 1. Ein Problem π gehört zur Klasse der NP-vollständigen Probleme, genau dann wenn folgende Bedingungen erfüllt sind [22], [23]:

$$\pi \in NP$$

$$\forall \pi' \in NP : \pi' \preceq \pi$$

Die erste Bedingung beschreibt, dass π in der Klasse NP liegt und die zweite Bedingung, dass π NP-schwer sein muss und somit jedes Problem π' in NP durch eine Polynomialzeitreduktion auf π reduziert werden kann.

Ein Problem heisst NP-schwer, wenn sich jedes in NP liegende Problem π' , in deterministisch polynomieller Zeit auf π reduzieren lässt. Genau genommen wird aber nur von NP-Vollständigkeit gesprochen, falls es sich um Entscheidungsprobleme handelt. Diese können mit Ja oder Nein beantwortet werden. Bei Optimierungsproblemen respektive Suchproblemen wie dem *TSP* wird von NP-äquivalenten Problemen gesprochen [22], [23].

3.1 Komplexitätsklassen

Um die Probleme der Komplexitätstheorie besser gliedern zu können werden Komplexitätsklassen gebildet. Mittels den Landau-Symbolen wird die "Schwierigkeit" zur Lösung der Probleme angegeben. Es wird der schlechteste Fall (Worst Case) in Abhängigkeit von der Problemgrösse n angegeben und als Funktion dargestellt [45]:

$$\mathcal{O}(n)$$

3.2 Das *Travelling Salesman Problem (TSP)*

Das *Travelling Salesman Problem*, abgekürzt auch *TSP* genannt, wird auf Deutsch als Problem des Handelsreisenden bezeichnet. Gegeben sind n WP c_i , mit i als Index der WP. Es gilt $i \in \{1, 2, \dots, n\}$. Für diese WP (im Folgenden mit WP abgekürzt)

soll die kürzeste Rundreise gefunden werden. Dabei soll jeder *WP* einmal besucht und schliesslich zum Ursprungsort zurückgekehrt werden. Es gilt folgende Definition [9]:

Definition 2. Um das *TSP* zu lösen muss eine Permutation π gefunden werden, welche folgende Formel minimiert:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

c_i : *WP i*

n : Anzahl *WP*

$d(c_i, c_j)$: Distanz zwischen den *WP i und j*

Es existieren zwei verschiedene Varianten des *TSP*. Das asymmetrische *TSP*, abgekürzt *ATSP*, ist die verallgemeinerte Form des symmetrischen *TSP*, welches auch als *STSP* abgekürzt wird. Beim *STSP* werden die Distanzen von $d(c_i, c_j)$ und $d(c_j, c_i)$ als gleich lang betrachtet, was die Lösungsmenge um die Hälfte einschränkt [30].

Das *TSP* wird für die Berechnungen oft als ein gerichteter (*ATSP*) oder ungerichteter (*STSP*) gewichteter Graph dargestellt. Die Knoten stellen die *WP* dar und die Kanten die Distanzen zwischen den *WP*.

Um das Problem allgemeiner zu formulieren, werden nicht Distanzen zwischen den *WP* angegeben, sondern ein Gewicht, welches sich aus einer Funktion zusammensetzt. Dadurch können die Kosten der Kanten beliebig zusammengesetzt sein und zusätzlich oder anstelle der Distanz andere Parameter einbezogen werden.

Um die Kosten zwischen den einzelnen *WP* anzugeben und für die Berechnungen zu verwenden bietet sich die Bildung einer Distanzmatrix an. Die Matrix hat die Dimension $n * n$, wobei die Elemente mit den Indizes $i = j$ unendlich sind (je nach Literatur auch 0, oder NULL [44]). Die Distanz $d_{i,j}$ beschreibt die Distanz vom *WP i* zum *WP j* und kann auch als Funktion $d(c_i, c_j)$ angegeben werden. Für drei *WP* sieht die Distanzmatrix wie folgt aus:

$$D_{3,3} = \begin{bmatrix} \infty & d_{1,2} & d_{1,3} \\ d_{2,1} & \infty & d_{2,3} \\ d_{3,1} & d_{3,2} & \infty \end{bmatrix}$$

Für das *STSP* vereinfacht sich diese Matrix zu:

$$D_{3,3} = \begin{bmatrix} \infty & d_{1,2} & d_{1,3} \\ d_{1,2} & \infty & d_{2,3} \\ d_{1,3} & d_{2,3} & \infty \end{bmatrix}$$

Aus Gutin and Punnen [31], Michiels et al. [44] und Gross et al. [29].

3.3 Algorithmen zur Lösungsfindung des *TSP*

Zur Lösung des *TSP* existieren verschiedene Ansätze. Dieses Kapitel behandelt lediglich eine mehr oder weniger zufällige Auswahl an Algorithmen. Es kann generell zwischen exakten Algorithmen und Heuristik unterschieden werden. Heuristik löst das gegebene Problem durch Annäherung und liefert somit nicht garantiert die korrekte Lösung. Im Gegenzug findet eine Heuristik die Lösung mit deutlich weniger Ressourcen.

Je nach verfügbaren Ressourcen kann bei bereits relativ kleinen Problemgrößen nur noch Heuristik die Lösung finden. Dies ist auf die vielen Kombinationsmöglichkeiten zurückzuführen. Beim *ATSP* existieren für n WP P Permutationen:

$$P(n) = (n - 1)!$$

Für 3 WP sind dies 2 mögliche Rundreisen, für 4 WP 6, für 5 WP 24, und für 10 WP bereits 362880 Möglichkeiten und so weiter. Beim *STSP* reduzieren sich die Möglichkeiten um die Hälfte:

$$P(n) = (n - 1)!/2$$

3.3.1 Exakte Algorithmen

Unter der Kategorie der exakten Algorithmen finden sich Algorithmen, welche immer eine korrekte Lösung zum Problem ergeben. Diese Art von Algorithmen eignen sich nicht immer, da bei grösseren Problemgrößen nicht genügend Ressourcen verfügbar sind.

3.3.1.1 Brute-Force-Algorithmus

Beim Brute-Force-Algorithmus, was auf Deutsch übersetzt "rohe Gewalt" bedeutet, werden alle Lösungen durchprobiert. Es werden alle möglichen Kombinationen von Rundreisen generiert, deren Gewicht berechnet und dadurch die optimale Rundreise gesucht. Schnell wird klar, dass dies für eine grosse Anzahl an WP nicht mehr effizient ist, da zu viele Kombinationen entstehen.

3.3.1.2 Branch-and-Bound

Branch-and-Bound-Algorithmen gibt es in verschiedenen Varianten. Alle Variationen des Algorithmus folgen denselben Grundregeln:

- Branching: Problem in Teilprobleme unterteilen
- Bounding: Untere Schranke (Lower Bound) berechnen
- Selection: Für die nächste Unterteilungsstufe (Branching) die Ausgangslage wählen
- Elimination: Ausschliessen von Teilproblemen

Es wird das Ziel verfolgt, Teirläufe möglichst früh auszusortieren, welche keine Erfolgschancen auf die kürzeste Rundreise haben. Um dies zu erreichen, wird ein Baum erstellt, der alle Rundreiserouten in den Blättern des Baumes und die gesamte Route im Wurzelknoten enthält. Jeder Knoten enthält eine Teirlaufführung und wird bei jeder nächsttieferen Stufe im Baum um einen in der Rundreise noch nicht enthaltenen WP ergänzt. In Abb. 3.1 wird der Entscheidungsbaum für $n = 3$, also 3 WP, angegeben und in Abb. 3.2 ist angedeutet wie dieser Baum für $n = 5$ aussehen könnte [81], [82], [83], [84], [10].

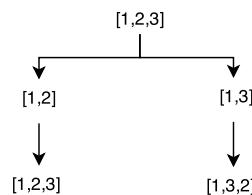


Abb. 3.1: Branch-and-Bound Tree Space mit $n = 3$ (Quelle: Eigene Darstellung)

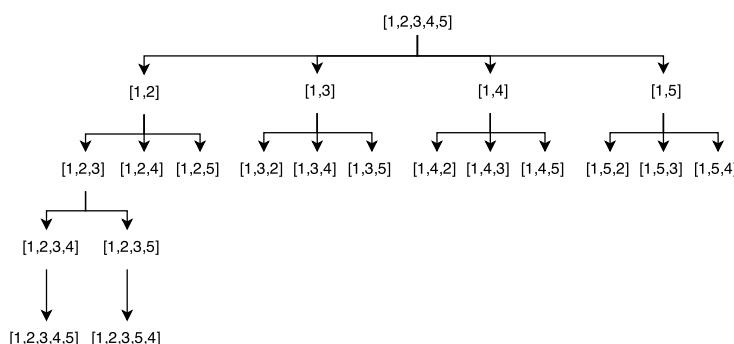


Abb. 3.2: Ansatz des Branch-and-Bound Tree Space mit $n = 5$ (Quelle: Eigene Darstellung)

Ein mit n WP erstellter Entscheidungsbaum weist folgende Merkmale auf:

- Die Höhe des Baumes beträgt n .
- Die maximale Anzahl Nodes pro Stufe ist immer auf der Stufe 2 und beträgt: $N_{maxProStufe}(n) = n - 1$.
- Die Anzahl Blätter und somit die Anzahl Lösungsmöglichkeiten beträgt $B(n) = (n - 1)!$.

3.3.1.3 Held–Karp–Algorithmus

Wie der Branch-and-Bound-Algorithmus gehört der Held-Karp-Algorithmus der Dynamischen Programmierung an. Der Held-Karp-Algorithmus errechnet die Lösung mittels rekursiver Anwendung des Grundsatzes "Jede Teilstrecke von einer Strecke mit minimaler Länge ist selbst auch von kürzester Distanz". Der Algorithmus berechnet alle Teilprobleme und beginnt mit der kürzesten Teilstrecke [59], [32], [87].

3.3.2 Heuristik

Neben den exakten Algorithmen gibt es heuristische Strategien, um zu einer Lösung zu finden. Diese Algorithmen können in konstruktive, iterative und zufallsgesteuerte (randomisierte) Lösungsverfahren gegliedert werden. Während die konstruktiven Algorithmen versuchen die optimale Lösung zu konstruieren, versuchen die iterativen Algorithmen durch logische Veränderung einer berechneten Zwischenlösung eine bessere Lösung zu finden. Die randomisierten Algorithmen versuchen durch zufällige Mutation der Lösungsvarianten auf die optimale Lösung zu kommen [35]. Hier eine Auswahl von Algorithmen der drei Kategorien:

- Konstruktive Algorithmen
 - Nearest Neighbour (NN), auch Greedy Algorithmus genannt
- Iterative Algorithmen
 - 2-Opt
 - K-Opt
 - V-Opt
- Zufallsgesteuerte Algorithmen
 - Genetische Algorithmen
 - Simulated Annealing
 - Tabu Search
 - Ant-Colony-Optimization

4 Geodäsie

Um das *TSP* auf einer realen Karte und das Routing auf der Ebene der Strassen vorzunehmen, werden in diesem Kapitel die Grundlagen der Geodäsie kurz zusammengefasst. Bei der Kartendarstellung sind wichtige Punkte zu beachten. Obwohl die Map Clients viele Details und Probleme abstrahieren, helfen die Grundlagen der Kartendarstellung beim Verständnis und der Implementation. Die Verzerrung stellt das Hauptproblem bei jeder Abbildung, auch Projektion genannt, von einer Kugel auf eine flache Ebene dar. Das dreidimensionale Geoid muss auf eine zweidimensionale Fläche abgebildet werden. Das Geoid beschreibt die Oberfläche der effektiven Erdform und wird auf ein sogenanntes Referenzellipsoid vereinfacht, worauf die Daten dann projiziert werden.

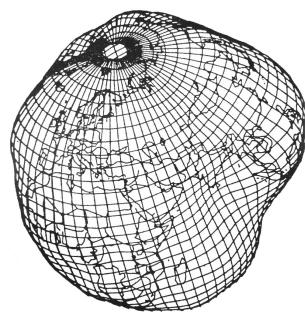


Abb. 4.1: Das Geoid (*Quelle: Kale [39]*)

Die meist verwendete Projektion bei den *Map Clients* ist die Mercator Projektion, welche eine Zylinderprojektion darstellt. Die Verzerrung ist bei dieser Projektionsart grösser, je weiter man sich vom Äquator entfernt. Das bedeutet beispielweise, dass die Arktis und Antarktis überproportional gross dargestellt werden. Dieses Phänomen wird auch als nicht flächentreu bezeichnet. Ebenfalls ist die Projektion nicht richtungstreu und somit werden Grosskreise, welche die kürzeste Verbindung zwischen zwei Punkten auf einer runden Oberfläche darstellen, nicht als Gerade gezeichnet. Vorteil der Mercator-Projektion ist hingegen die Winkeltreue der Abbildung. Das heisst, dass geometrische Formen im Kleinen unverzerrt bleiben [39] [46].

Für Webanwendungen mit Kartenmaterial werden vorwiegend die beiden geodätischen Referenzsysteme EPSG:3857 und EPSG:4326 (auch WGS 1984 oder WGS84 genannt) verwendet. Die *Map Client* der Kartenanbieter von Google [4], Leaflet, Openlayers und der Webclient von Bing arbeiten alle mit der EPSG:3857. Google nennt dieses Referenzsystem auch EPSG:900913. 900913 steht dabei für Google in Zahlenform dargestellt ("gOOgLE"). Mit EPSG:4326 arbeitet sowohl Google Earth [27] als auch *Openstreetmap (OSM)* [52]. Auch GPS [3] verwendet für die Positionsangabe dieses Referenzsystems. Für die Definition eines Punktes auf der Erde werden vom zugrunde liegenden Referenzsystem abhängige Koordinaten verwendet.

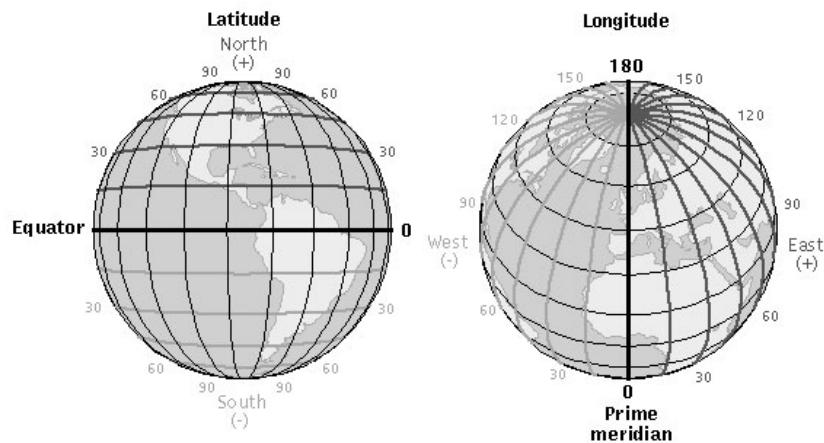


Abb. 4.2: Längen- und Breitengrade (Quelle: Illinois State University [37])

Die Einteilung erfolgt, wie in Abb. 4.2 dargestellt, in Längen- und Breitengrade, welche die Erde umspannen. Die auf einer nach Norden ausgerichteten Karte horizontal verlaufenden Kreise sind die Breitengrade (englisch: latitude), die Vertikalen werden Längengrade oder Meridiane (englisch: longitude) genannt. Der Koordinatenursprung befindet sich bei 0° Nord und 0° Ost. 0° Nord befindet sich auf dem Äquator und 0° Ost befindet sich auf dem 0° -Meridian, welcher durch die Sternwarte in Greenwich bei London geht.

5 Webtechnologien

Im Kapitel "Webtechnologien" sind die Grundlagen der Webtechnologien im Allgemeinen, die im Web eingesetzten Konzepte zur verteilten und parallelen Berechnung und die Darstellung von Kartenmaterial, *Markern* und das *Routing* auf Webseiten beschrieben.

5.1 Grundlagen

Dieses Kapitel erhebt nicht den Anspruch die kompletten Grundlagen des Internets abzudecken. Vielmehr geht es darum, die Basis der in der Arbeit verwendeten Technologien zu umschreiben und ein gemeinsames Basisvokabular zu definieren, damit die Abläufe mit einem Computer-Basiswissen verstanden werden können.

Das Internet basiert auf einer Client/Server Architektur (mehr dazu im Abschnitt 6.2). Der Webserver stellt die Webseiten für die Clients, beispielsweise die Browser der einzelnen Benutzer, bereit. Damit ein Webserver erreicht werden kann, hat er eine weltweit eindeutige *IP* Adresse. Da aber für den alltäglichen Gebrauch die *IP* Adresse nicht sehr einprägsam ist, wurde eine weltweite Datenbank eingeführt, worin jeder *IP* Adresse einem oder mehreren Namen (Domainnamen) zugeordnet werden können. Die Clients rufen den Domainnamen auf, werden über den DNS (Domain Name Server) über die IP des gesuchten Server informiert, und können daraufhin die *Ressourcen* auf dem angesprochenen Webserver benutzen. Um eine solche *Ressource* anzusprechen reicht der Domainname nicht aus, da dies lediglich die Adresse des Servers darstellt. Um aber auf dem Webserver den korrekten Empfänger zu finden, muss ein Port angegeben werden, da auf einem Webserver verschiedene "Services" angeboten werden können. Für Webseiten, welche über das *HTTP*-Protokoll kommunizieren, gilt der Port 80, wobei für *HTTPS* verwendende Webseiten der Port 443 verwendet wird (genauere Erläuterungen zu den *HTTP*- und *HTTPS*-Protokollen finden sich im Abschnitt 5.1.1). Ebenfalls können unter einer Domainadresse und einem Port wiederum verschiedene "Services" angeboten werden. Der genaue Service wird durch die URL, Abkürzung für "Uniform Resource Locator", angesprochen [72].

Der Begriff URI, der für "Uniform Resource Identifier" steht, muss klar vom Begriff URL abgegrenzt werden. Die URI definiert die eindeutige Ressourcenidentität als Gesamtheit und muss nicht zwingend den Ort, wo sich die *Ressource* befindet und mit welchem Protokoll diese anzusprechen ist angeben. Die URL hingegen gibt immer die genaue Lokalität, das Protokoll und den Port der Ressource an. Der letzte wichtige Begriff für den Ressourenzugriff lautet "URN" ("Uniform Resource Name") und beschreibt, wie die Ressource nach Name in einem Namensraum identifiziert werden kann. Das heißt die URN beschreibt immer die URI, ohne den Ort derer anzugeben oder wie sie anzusprechen ist [5] [77].

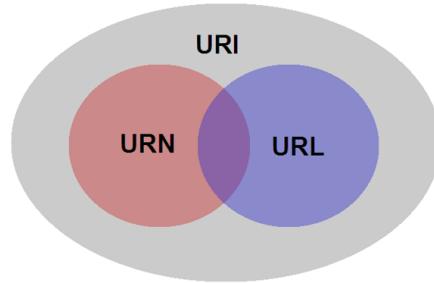


Abb. 5.1: Mengendiagramm der Begriffe URL, URI, URN (Quelle: David P. Heitmeyer [12])

Abb. 5.1 verdeutlicht, dass die URI sowohl nur aus der URL, als auch nur aus der URN oder aus beiden zusammen bestehen kann. Eine URI im Internet ist grundsätzlich nach dem folgenden Schema aufgebaut [12]:

```
scheme://subdomain.domain:port/path?query_string#fragment_id
```

Das "scheme" gibt das verwendete Protokoll an und wird in den meisten Browsern nicht mehr dargestellt. Es handelt sich im Internet in der Regel um http oder https. Die "subdomain" gibt eine Unterseite der Domain an. Dadurch können beispielsweise einzelne Unterseiten einer Webseite besser von der Hauptseite abgegrenzt werden. Ein Beispiel hierfür ist Google, deren Hauptadresse http://google.com ist. Der Kartendienst von Google ist unter http://maps.google.com zu erreichen. Die "domain" ist, wie bereits oben beschrieben, die Hauptadresse. Mit einem Doppelpunkt abgetrennt folgt in der URL der Port des "Services" und nachfolgend der "path", welcher die genaue Lokalität der Ressource innerhalb des "Services" angibt. Es folgen die zusätzlichen Parameter "query_string", welcher mit einem "?" abgetrennt wird, und die "fragment_id", welche wiederum mit einem "#" vom Rest der URL separiert wird.

5.1.1 Relevante Internetprotokolle

Da Protokolle und die im Internet verwendeten Protokoll-Verben von grosser Wichtigkeit sind, werden diese hier, in einem separaten Unterabschnitt betrachtet. Das *OSI-Modell* wird für das Verständnis dieses Kapitels vorausgesetzt und wird nur am Rande erwähnt.

5.1.1.1 HTTP

Für das Internet wird das Internet Protokoll (IP) als Vermittlungsschicht verwendet, worin die einzelnen Segmente über die Transportschicht TCP transportiert werden. Darauf aufbauend ist das statuslose *HTTP- / HTTPS*-Protokoll für die Anwendungen

definiert und dient dem Sitzungshandling, der Darstellung und der Anwendung. Die übermittelte Einheit bei *HTTP / HTTPS* sind Daten, welche über die verschiedenen *HTTP*-Verben ausgetauscht werden. Die *HTTP* Verben sind GET, PUT, POST, DELETE, TRACE, CONNECT, HEAD und OPTIONS [61]. Die *HTTP*-Statuscodes können R. Fielding [60] entnommen werden.

Eine typische Anfrage an einen Webserver erfolgt über die *HTTP*-GET (siehe auch 6.2) Anfrage vom Client:

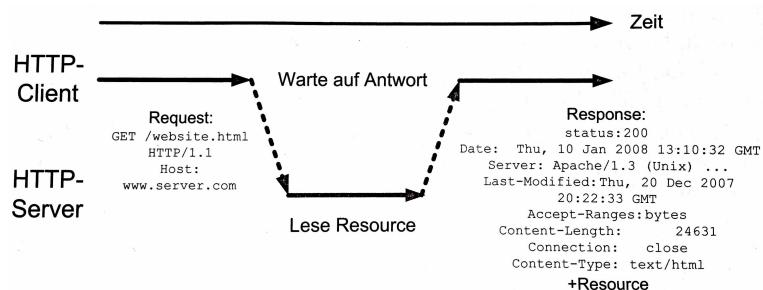


Abb. 5.2: Erfolgreiche HTTP-GET Anfrage (Quelle: Dunkel et al. [15])

Für eine persistente *HTTP*-Verbindung (keep-alive) wurde in *HTTP* 1.0 der zusätzliche Header "Connection Keep-Alive" eingeführt, welcher in *HTTP* 1.1 als Standard definiert wurde. In Abb. 5.3 sind folgende Vorteile der persistenten Verbindung ersichtlich [66]:

- Weniger Handshakes nötig und somit reduzierte Latenz bei wiederholten Requests (nur noch eine Round-Trip-Time).
- Weniger TCP Verbindungen.
- Die zentrale Recheneinheit (*CPU*) wird weniger beansprucht und Speichergebrauch ist geringer, da weniger gleichzeitige Verbindungen geöffnet sind.
- HTTP-Pipelining von Anfragen und Antworten ist möglich.

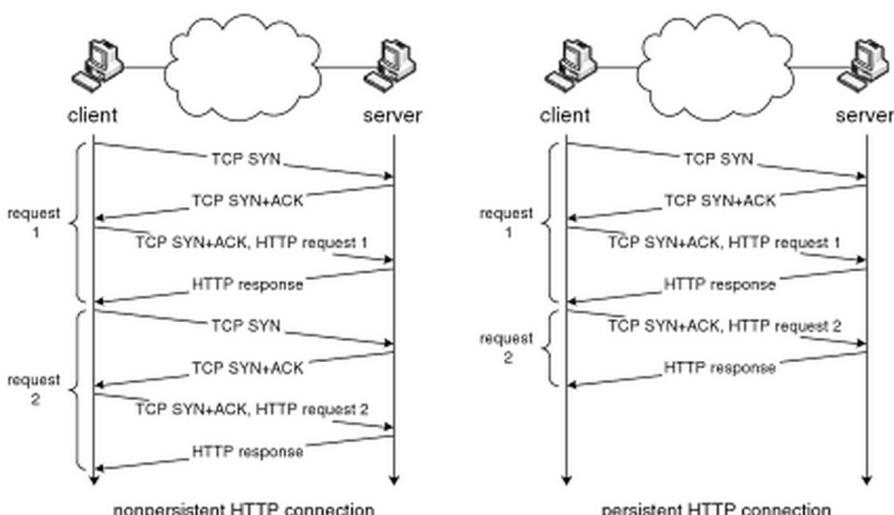


Abb. 5.3: Vergleich nicht-persistente und persistente *HTTP* Verbindung (Quelle: Serpanos and Wolf [66])

Damit Webanwendungen nicht bei jedem Aufruf die gesamte Webseite neu vom Webserver laden müssen, können sogenannte *AJAX* Anfragen gemacht werden. *AJAX* steht für "Asynchronous JavaScript and XML". Die *asynchronen* Anfragen erfolgen aus der Webanwendung heraus und laden zusätzliche Informationen und Applikationsteile vom Webserver, welche dann in die Webanwendung eingefügt werden können. Da die Anfragen *asynchron* sind, können mehrere Anfragen gleichzeitig erfolgen und die Verarbeitung erfolgt bei Ankunft der Daten in einer sogenannten *Callback Funktion* in JavaScript.

5.1.1.2 Websocket Protokoll

Das Websocket Protokoll ist klar vom *HTTP*-Protokoll unterscheidbar. Zwar können über persistente Verbindungen auch mehrere *HTTP* Anfragen innerhalb der gleichen TCP Verbindung erfolgen, jedoch folgen diese Anfragen immer dem *HTTP*-Protokoll und dem darin verankerten Request-Response-Prinzip. Beim Websocket Protokoll kann der Server auch Daten auf den Client schicken (push) [28]. Für detailliertere Informationen über das Websocket-Protokoll wird an dieser Stelle auf die Spezifikation des Websocket-Protokolls (I. Fette and A. Melnikov [36]) hingewiesen.

5.2 Web 2.0

Web 2.0 beschreibt keine neue Version des Internets im eigentlichen Sinne, da kein Update mit technischen Spezifikationen erfolgt ist. Es hat sich vielmehr eine Anhäufung von Änderungen durchgesetzt, die ein völlig neues Internet bilden. Die grösste Änderung hierbei ist die Interaktion der Benutzer (User) mit dem Inhalt des Internets. Im Web 1.0 wurden statische Inhalte von einigen wenigen Entwicklern den Usern zur Verfügung gestellt. Im Web 2.0 hingegen kann mit den Inhalten interagiert werden. User können eigene Blogs erstellen, kommentieren, bewerten und sich untereinander austauschen. Dies erfordert dynamische Webseiten und somit "scriptfähige" Programmiersprachen. Die HyperText Markup Language (HTML), das den Grundbaustein des Internets bildet, ist lediglich eine Markupsprache und kann keinen dynamischen Inhalt anzeigen. Hierfür werden sogenannte Scriptsprachen benötigt, welche dann wiederum HTML Code generieren um den Webinhalt im Browser anzuzeigen. Mit dem Web 2.0 sind auch Webapplikationen (Webapps) und Mashups aufgekommen. Darin werden Application Program Interface (*API*) implementiert, welche die Daten der Applikation über eine vorgeschriebene Schnittstelle zur Nutzung bereitstellt [56].

6 Parallele und verteilte Berechnung

Dieses Kapitel soll eine Einführung in parallele Verarbeitung (englisch: parallel computing) und verteilte Systeme (englisch: distributed computing), fokussiert auf den Webbereich, geben. Bevor auf die beiden Berechnungsarten näher eingegangen wird, soll der Unterschied zwischen "Parallel Computing" und "Distributed Computing" erläutert werden. Bei Lösungsansätzen des "Parallel Computing" wird das primäre Ziel der parallelen Ausführung von Tasks oder Programmen verfolgt, wobei beim "Distributed Computing" das Ziel der verteilten Ausführung von Tasks oder Programmen auf unterschiedlichen Rechnern verfolgt wird [40].

Das Internet ist bereits in seiner Grundstruktur ein verteiltes System (siehe Abschnitt 6.2). Hinsichtlich komplexer Berechnungen ist für das Internet aber auch die Kombination von paralleler und verteilter Berechnung interessant. Viele einzelne Rechner, welche jeder für sich zumeist eine relativ geringe Rechnerleistung (im Vergleich zu beispielsweise einem Mainframerechner) hat, haben zusammengeschlossen ein enormes Rechenpotential. Natürlich muss bei einem solchen Zusammenschluss die Verbindungs geschwindigkeit und Übertragungsrate berücksichtigt werden und das Rechenpotential kommt erst bei anspruchsvollen Berechnungen zum tragen.

6.1 Parallele Berechnung

Die Parallelisierung der Berechnung ermöglicht es, mehrere *CPUs* gleichzeitig für die Berechnung eines Problems einzusetzen. Bis vor zehn Jahren verdoppelte sich die CPU Leistung regelmäßig. In den letzten zehn Jahren hingegen stagnierte diese Entwicklung, weshalb für eine bessere Rechenleistung mehrere Kerne (Cores) eingesetzt werden. Diese Cores können aber nur genutzt werden, falls das Programm auch entsprechend parallelisiert werden kann.

Neben den Vorteilen bringt die Parallelisierung auch eine Reihe von Nachteilen mit sich. Beispiele sind Fehler bei gleichzeitigem Ressourcenzugriff, Deadlocks oder eine erschwerete Fehlersuche. Zudem entsteht sowohl bei der Implementation als auch bei der Berechnung ein Mehraufwand, wie beispielsweise für die Kommunikation, die Synchronisation, die Speicherverwaltung oder das Locking [70].

6.2 Verteilte Systeme

Das zugrundeliegende Konzept von verteilten Systemen sind dezentralisierte Systemlösungen, welche die Einzelkomponenten entkoppeln und somit Lastenverteilung, bessere Verfügbarkeit und Skalierbarkeit ermöglichen [71]. Verteilte Systeme können verschiedenen klassifiziert werden. In der Literatur wird zumeist eine Kategorisierung nach der

Systemarchitektur gemacht. Die folgende Auflistung von Systemarchitektur-Kategorien stellt ein Zusammenzug der Quellen [64], [15], [73], [55], [47], [20] und [17] dar:

Client-Server-Modell

Im Zentrum des Client-Server-Modells steht der Server, welcher die Daten und Dienste den Clients zur Verfügung stellt. Das Internet stellt mit den Webservern und den verbindenden Clients einen typischen Repräsentanten des Client-Server-Modells dar. Die einzelnen Clients verbinden sich dabei mit dem Server und warten auf das Resultat des Servers. Das heisst, die *Anfrage* (englisch: *Request*) findet synchron statt. Asynchrone *Requests* verlaufen ähnlich, jedoch wartet der Client nicht unmittelbar auf das Resultat, sondern erhält dieses zum späteren Zeitpunkt, sobald der Webserver das Resultat bereitgestellt und gesendet hat.

Neben den Vorteilen des direkten und einfachen Zugriffs auf den Server weist das Client-Server-Modell einige Schwachstellen auf: Die Möglichkeit der Skalierung des Systems, beispielsweise bei einer zunehmenden Anzahl an Clients, wird zwar durch das statuslose *HTTP*-Protokoll vereinfacht, jedoch bleiben Schwierigkeiten wie zum Beispiel bei der Skalierung von Datenbanken bestehen. Zudem wird die Rechenleistung der Clients oft nicht sinnvoll genutzt und der Server stellt einen Single-Point-Of-Failure dar. Dies bedeutet, dass sobald der Server ausfällt, das gesamte System nicht mehr genutzt werden kann, da die Clients ohne den Server nicht funktionstüchtig sind.

Peer-to-Peer-Modell

Das Peer-to-Peer-Modell versucht die Einschränkungen des Client-Server-Modells zu eliminieren. Es verteilt die Rechenleistung, Daten und *Services* auf verschiedene, gleichwertige Rechner, auf die sogenannten *Peers*. Für die Verwaltung der einzelnen *Peers* gibt es zwei Architekturmodelle. Bei der zentralen Architektur, die für Napster [14] entwickelt wurde, verwaltet ein Server die einzelnen Clients. Auf diesem Server sind die kompletten Metainformationen des Systems abgelegt. Das heisst, bei der Anmeldung eines neuen Clients setzt sich dieser mit dem Server in Verbindung und wird im System registriert. Wenn ein Client eine Ressource benötigt, fragt er den Server an, auf welchem Client sich diese befindet. Erst danach findet die effektive Peer-to-Peer Aktivität statt, indem der anfragende Client den Client mit den Ressourcen direkt kontaktiert und die Ressource bezieht. Es ist ersichtlich, dass diese Architektur die Probleme des Client-Server-Modells nur teilweise lösen kann, weshalb die reinere Ausprägung, nämlich die verteilte Architektur des Peer-to-Peer-Modells entwickelt wurde. Hier existiert kein Server mehr, der das System verwaltet, und jegliche Informationen der *Peers* werden auf die einzelnen *Peers* verteilt. Um ein Peer-to-Peer-Netzwerk dieser Art aufzubauen, wird

eine Liste von hoch verfügbaren *Peers* erstellt, mit welchen sich ein anzumeldender Client verbinden kann. Der Verbindungsaufbau funktioniert bei der verteilten Architektur über 'Ping' und 'Pong' Nachrichten. 'Ping' ist dabei die Anmeldeanfrage, welche mittels flooding an jeweils den nächsten *Peer* weitergeleitet wird und 'Pong' die Bestätigung, dass der *Peer* erfolgreich in das System eingebunden wurde.

3- und N-Tier Architektur

Mit *Tiers* (englisch für Schichten) sind physikalische Schichten gemeint, welche Softwareabstraktionsstufen der Architektur darstellen. Jede Schicht funktioniert als einzelnes Teilsystem und darf jeweils nur mit direkt angrenzenden Schichten kommunizieren und deren Dienste nutzen.

Betrachten wir das Beispiel der 3-Tier Architektur im Webumfeld, so repräsentiert der Browser und die darauf dargestellten HTML-, CSS- und JavaScript-Elemente die Präsentationsschicht. Diese kommuniziert über das *HTTP*-Protokoll mit dem darunterliegenden Webserver, holt mit einem GET Request die genannten Elemente und verarbeitet Daten, wie beispielsweise Formulardaten, welche über einen POST Befehl an den Webserver geschickt wurden. Diese Schicht wird auch Anwendungsschicht genannt. Die zuunterst liegende Schicht wird Persistenzschicht genannt und ermöglicht den Datenbankzugriff, um Daten von der Datenbank zu holen und wieder darin persistent abzuspeichern.

Service Oriented Architecture (SOA)

Mit der SOA wird versucht, den meist heterogenen Systemen innerhalb eines Gesamtsystems gerecht zu werden. Damit ist gemeint, dass ein Gesamtsystem einer Firma beispielsweise aus den unterschiedlichsten Teilsystemen zusammengesetzt ist. Diese gewährleisten gemeinsam die korrekte Ausführung der Abläufe im Betrieb und müssen somit Daten untereinander austauschen. Jedes Teilsystem stellt hierzu Services bereit, welche an einen Service Bus angeschlossen werden können. Dieser Service Bus legt die Middleware fest und standardisiert den Nachrichtenaustausch der einzelnen Services. Somit kommunizieren alle Teilsysteme über diesen Service Bus miteinander und es finden keine proprietäre Querverbindungen mehr untereinander statt.

Zur Umsetzung der SOA Architektur im Webumfeld werden die Web Services genutzt. SOAP (Simple Object Access Protocol, Service Oriented Architecture Protocol) und WSDL (Web Service Description Language) sind die gängigsten Web Service Sprachen. Es wird hier nicht weiter auf Web Services eingegangen, da sich mit den derzeitigen Entwicklungen des Web 2.0 zunehmend eine Abwendung von Web Services hin zu Web Oriented Architecture (WOA) und RESTful Architekturen erkennen lässt (siehe Abschnitt "Web 2.0 und Web-orientierte Architekturen (WOA)").

Event Driven Architecture (EDA)

Aus Studien von Schulte [65] geht hervor, dass ereignisgesteuerte Geschäftsprozesse weit verbreitet sind. Daher wurde EDA als Erweiterung der SOA entwickelt. Der gesamte Kontrollfluss wird bei dieser Architektur über Ereignisse gesteuert.

Cluster Architektur

Bei der Cluster Architektur werden viele verfügbare *Ressourcen*, wie zum Beispiel Rechenleistung, zu einem System zusammengeschlossen. Der Cluster verwaltet diese Ressourcen, nimmt Anfragen an das System entgegen und verteilt die Rechenlast möglichst gleichmäßig auf die verfügbaren Ressourcen, um eine optimale Antwortzeit zu erzielen. Grosse Vorteile bringt ein Cluster bei der Ausfallsicherheit und dem Verarbeiten vieler Anfragen.

Grid Architektur

Die Grid Architektur ist eine Middleware, die *Ressourcen* zentral verwaltet und bereitstellt. Zentral bedeutet hier, dass das gesamte Grid diese Aktivitäten koordiniert und nicht ein zentraler Rechner. Das heisst, die einzelnen Prozesse, welche sich am Grid beteiligen, koordinieren sich selbstständig. Im Unterschied zur Cluster Architektur führt jede *Node* bei der Grid Architektur unterschiedliche Berechnungsschritte oder Programme aus. Grids lassen sich in die Arten Rechen-Grid, Datenbank-Grid und das Ressourcen-Grid gliedern. Das Rechen-Grid bündelt Rechenleistung einzelner Rechner und stellt dieses zur Verfügung. Das Datenbank-Grid verwaltet grosse Datenbestände und das Ressourcen-Grid verwaltet bestimmte Ressourcen wie beispielsweise den Speicherplatz und bietet diesen den anderen Systembeteiligten an.

Web 2.0 und Web-orientierte Architekturen (WOA)

WOA sind eigentlich eine Mischform von den bereits beschriebenen Architekturmodellen. Da der Fokus dieser Arbeit auf dem Webbereich liegt, sollen diese hier noch etwas genauer beleuchtet werden. Der Begriff Web-orientierte Architekturen entstand aus der Tatsache, dass Technologien des Web 2.0 Einzug in Geschäftsapplikationen hielten und für Anwendungen im Geschäftsumfeld eingesetzt werden. WOA basieren vollständig auf Internetstandards wie *HTTP*, *HTTPS*, SSL, Sockets, FTP, Proxies, telnet, IMAP, etc. und erweitern SOA zu webbasierte Applikationen. Nach der Definition von Gartner [20] folgen WOA den folgenden Interface Richtlinien:

- Identifikation von Ressourcen
- Bearbeitung von Ressourcen durch Zustände
- Selbstbeschreibende Nachrichtenformate

- Hypermedien als Antreiber der Applikationszustände
- Zustandsneutral

Um eine WOA umzusetzen kann der folgende *Stack* verwendet werden:

- Verteilung (HTTP(s), feeds)
- Zusammenstellung (Mashups)
- Sicherheit (OpenID, SSL)
- Portabilität der Daten (XML)
- Daten Representation (JSON)
- Übertragungsmethoden (REST, HTTP)

WOA folgen dem von Fielding [17] eingeführten *REST* Paradigma. REST steht für Representational State Transfer und weist die folgenden Datenelemente auf:

Datenelement	Beispiel im Web
resource	Beabsichtigte konzeptionelle Lokalität von einer Hypertext Referenz
resource identifier	URL, URN
representation	HTML Dokument, JPEG Bild
representation metadata	Medien Typ, Modifikationszeit
resource metadata	Quelllink, Alternativen, Varianten
control data	Cache Handling und Kontrolle

Tab. 6.1: Datenelemente von REST, übersetzt in Deutsch aus Fielding [18]

REST besagt somit, dass eine Ressource über eine genau definierte Syntax (dem "resource identifier") angesprochen werden kann. Dies geschieht über ein zustandsloses Client-Server-Protokoll. Um die Dienste über die REST Schnittstelle anzubieten werden die *HTTP* Operationen, wie bereits in Abschnitt 5.1 beschrieben, verwendet.

6.3 Im Webumfeld

6.3.1 Multithreading im Browser

Dank der Einführung von HTML5 im Web 2.0 und dessen Umsetzung in den gängigsten Browsern kann JavaScript in Browserapplikationen in mehreren Threads (multithreaded) betrieben werden. Die Implementation erfolgt mit sogenannten *Webworkers*. Es existieren "Shared *Webworker*" und "Dedicated *Webworker*". Die "Shared *Webworker*" funktionieren übergreifend über mehrere Browsetabs, werden aber im Rahmen der Arbeit nicht weiter betrachtet. Nachfolgend sind mit *Webworker* immer die "Dedicated *Webworker*" gemeint, deren Browserunterstützung Alexis Deveria [1] entnommen werden kann.

Wichtige Elemente und Besonderheiten bei *Webworker*n sind (aus W3Schools [80]):

- Die Kommunikation zwischen Workern (Threads) ist nur über die Main Applikation (Main-Thread) möglich.
- Innerhalb von *Webworker*n existieren keine "document", "window" oder "parent" Objekte.
- "self" referenziert in *Webworker*n auf das "WorkerGlobalScope" Objekt.
- Eine verschachtelte Struktur von *Webworker*n wird nur von *Firefox* unterstützt.
- Innerhalb von *Webworker*n können weitere Scripts importiert werden, um auch Code-Bibliotheken in den Threads verwenden zu können:

```
1 self.importScripts("PFAD_ZU_LIBRARY");
```

6.3.2 Mehrprozessor-Berechnung auf dem Server

Node.js unterstützt kein Multithreading, es können aber mehrere Prozesse erzeugt werden. Die Funktionsweise ist dabei sehr ähnlich wie diejenige der *Webworker* und die Kommunikation erfolgt wie auch bei den *Webworker*n über Nachrichten (Messages) [38].

6.3.3 Verteilte Systeme auf mehreren Clients

Das Ziel bei einer Verteilung unter Webclients ist es, möglichst viele Browser für die Berechnung einzubeziehen, um deren Ressourcen nutzen zu können. Während dem Besuch einer Webseite werden nur kleine Teile der verfügbaren Ressourcen des Rechners genutzt. Das Ziel besteht darin, keinen zentralen Server zu verwenden, welcher die gemeinsamen Daten verwaltet. Der Webclient, welcher die Berechnung startet, ist der Master und verteilt die Aufgaben an die anderen Clients (Slaves). Das grosse Problem stellen dabei gemeinsam zu nutzende Daten für die Berechnung und deren Verteilung auf die Slaves dar. Obwohl verteilte Systeme und die Nutzung der verfügbaren Ressourcen als sehr sinnvoll erscheinen, nutzen bis heute wenige Applikationen diese Möglichkeiten [6] [67] [43].

6.3.4 Verteilte Systeme auf mehreren Servern

Beim verteilten System unter Servern wird die zu lösende Aufgabe an den Webserver geschickt, welcher mittels eines verteilten Algorithmus auf verschiedenen Servern die Lösung sucht und dann dem Webclient zurückschickt [43]. Da dafür viele Serverressourcen erforderlich sind, wird dieses Konzept in der Arbeit vernachlässigt.

7 Kartendienste und Frameworks

Dieses Kapitel soll die Komponenten für die Routenplanung im Browser veranschaulichen. Die drei Komponenten sind der Kartenserver, der Routingserver und der *Map Client*. Es gilt also zwischen dem *Map Client*, der die Karten lediglich darstellt, dem Kartenserver, welcher das Kartenmaterial liefert und dem Dienst für die Routenberechnung zu unterscheiden.

7.1 Kartenserver

Kartenserver oder auch WMS (Web Map Server) genannt, sind Server, welche Geodaten aus einem GIS beziehen und über das Internet zur Verfügung stellen. Ein GIS ist ein Geoinformationssystem, also ein System zur Haltung, Verwaltung, Analyse und Manipulation von Geodaten. Der etwas unscharfe Begriff Kartenserver kann aber auch einen Tileserver (TMS, Tile Map Server) bezeichnen, von welchem lediglich Geodaten bezogen, nicht aber bearbeitet werden können [57]. Das TMS Protokoll folgt URI Konventionen und orientiert sich an REST.

Geodatenanbieter gibt es nicht sehr viele, da die Aktualisierung solcher Daten sehr aufwendig ist. Eingeschränkt nach freier Verfügbarkeit und weltweiter Abdeckung bleibt nur *OSM* übrig, welche jedoch beispielsweise keine Satellitendaten (Aerial) bereitstellen. Mehr Informationen zu den Anbietern finden sich im Abschnitt 9.8.2.

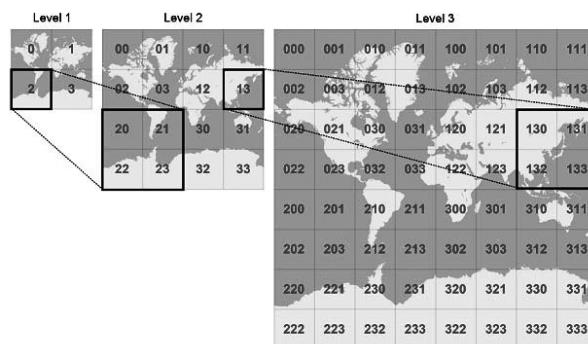


Abb. 7.1: Karten Unterteilung bei Kartenservern (Quelle: Bing [7])

Die Kartenteile werden von den Kartenanbietern in sogenannten Tiles aufbereitet, damit die Karte für den jeweils gewünschten Ausschnitt geladen und mit einem sogenannten Map Client im Browser dargestellt werden kann. Das Anfordern solcher Tiles gestaltet sich äusserst einfach mittels einem GET Request und den nötigen Parametern. Die Parameter der Map Server sind zumeist nach dem folgenden Schema aufgebaut [7]:

`http://mapserver.com/z/x/y`

z : Zoomstufe (Level)

x : Tilenummer auf der X-Achse

y : Tilenummer auf der Y-Achse

Der Koordinatensystemursprung befindet sich, wie in Abb. 7.1 gezeigt, am nordöstlichsten Punkt des Gesamtbilds. Die X-Achse verläuft horizontal, die Y-Achse vertikal. Die totale Tiles-Anzahl pro Zoomstufe z folgt der in Abhängigkeit von z stehenden Formel T:

$$T(z) = 2^{2*z}$$

Die Darstellung der Karte mittels Tiles ermöglicht effizientes Caching. Die Tiles können zudem progressiv nachgeladen werden, was den Datenverkehr (Traffic) um ein Vielfaches reduziert.

7.2 Routingserver

Der Service für die Routenberchnung zwischen einzelnen *WP* erfolgt vom Routingserver. Oft handelt es sich beim Kartenserver und dem Routingserver um denselben, da auch Karteninformationen für das Routing benötigt werden. Das Routing wird auf dem Server gemacht, und die berechneten Routen werden danach zurück an den *Map Clients* übermittelt. Das Format der Routen ist abhängig vom Service, wobei sich alle am Standard der Keyhole Markup Language (KML) orientieren [11], welcher ursprünglich von Google eingeführt wurde. Neben den bekannten, kostenpflichtigen Diensten wie Google Maps und Bing Maps existieren weitere kostenpflichtige Dienste, aber auch die Opensource-Lösung von Openstreetmap. Damit kann auf einem eigenen Server ein Routingservice installiert werden. Für die Bachelorarbeit werden ohne weitere Begründung die beiden Dienste Google Maps, wegen dessen Einfachheit und Openstreetmap, wegen der freien Verfügbarkeit, eingesetzt. Ein Vergleich der Dienste kann auf der Webseite von Openstreetmap [53] gefunden werden.

7.3 Map Clients

Die Visualisierung im Browser erfolgt mittels einem sogenannten *Map Client*. Es handelt sich bei den aktuell verfügbaren oft um JavaScript Frameworks, welche die Kartendarstellung, *Markerdarstellung* und Routenabbildung auf der Karte ermöglichen. Die Unterschiede der einzelnen Frameworks sind gering. Die bekanntesten sind Google Maps JavaScript API, Bing Maps API, OpenLayers, LeafletJS, MapBox. Die Auswahl des *Map Clients* erfolgt in Abschnitt 9.8.3.

Teil III

Lösungserarbeitung

Mittels einer Anforderungs- und Marktanalyse werden die an das System gestellten Anforderungen zusammengetragen. Anschliessend werden Konzepte dafür ausgearbeitet, welche mittels Proof of Concepts in einer Nutzwertanalyse verglichen werden.

8 Anforderungskonflikt

Die grosse Schwierigkeit bei der Lösung des *TSP* im Allgemeinen und insbesondere im Webumfeld, stellt der zugrunde liegende, in Abb. 8.1 aufgezeigte, Dreieckskonflikt dar.

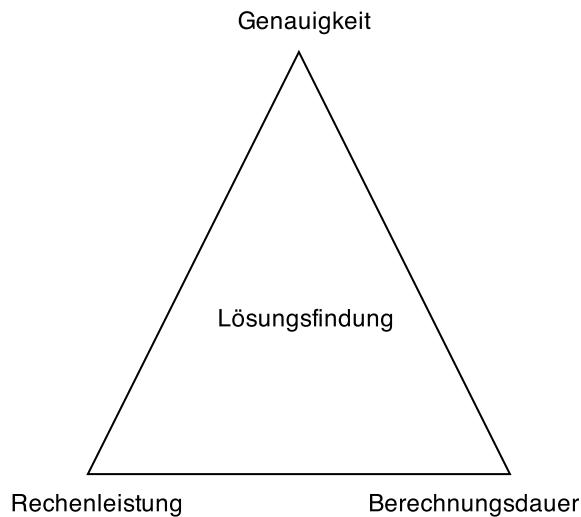


Abb. 8.1: Anforderungskonflikt Berechnung des *TSP* im Webumfeld (*Quelle: Eigene Darstellung*)

Die optimale Lösung wäre 100% genau, würde minimale Rechenleistung benötigen und wäre in kürzester Zeit berechenbar. Die bestehenden Anforderungen schliessen sich aber gegenseitig aus, was bedeutet, dass sobald die Genauigkeit erhöht werden soll, die Rechenleistung ebenfalls erhöht oder die Zeit der Ausführung ausgedehnt werden muss. Im Umkehrschluss muss die Genauigkeit der erwarteten Lösung reduziert werden, falls die Lösungsfindungszeit oder die Rechenleistung verringert werden soll. Auch bei der Reduzierung der Rechenleistung müssen Genauigkeitsverluste oder eine verlängerte Berechnungsdauer akzeptiert werden. Daraus resultiert, dass Kompromisse gefunden werden müssen, da nicht alle Anforderungen gleichzeitig erfüllbar sind.

9 Requirements Engineering (R1)

Das Kapitel "Requirements Engineering (R1)" dient zur Ermittlung der Requirements, also der Anforderungsanalyse des Systems und dessen Eingrenzung. Es enthält neben der Problembeschreibung und den Randbedingungen auch die Zieldefinition, welche die Ziele des Konzepts und dessen Umsetzung beschreibt. Weiter enthält das Kapitel die Anforderungsanalyse mit den funktionalen sowie den nicht-funktionalen Anforderungen. In der Anforderungsanalyse soll das Problem genauestens erfasst und abgegrenzt werden, damit das Konzept und die spätere Umsetzung davon den genauen Vorstellungen des Kunden entspricht. Um gewonnene Erkenntnisse aus bestehenden Applikationen einfließen lassen zu können, erfolgt die Anforderungsanalyse zusammen mit der sich in Kapitel 10 befindenden Marktanalyse.

9.1 Problembeschreibung

Ein Endkunde, im Folgenden als *EK* bezeichnet, hat eine Reiseunternehmung und arbeitet mit dem *CRM*-Programm FlexBüro von Bachmann Support GmbH. Er veranstaltet Reisen in die Schweiz und angrenzende Gebiete, wie das Südtirol und den Schwarzwald. Die Reisen stellt *EK* jeweils ein Jahr im Voraus zusammen und verschickt Werbematerial an seine potentiellen Kunden.

Um die Reisen verwälten und effizient planen zu können, erfasst *EK* die Reiseziele und Hotels im Programm FlexBüro in Adressdatensätzen. Bis zum jetzigen Zeitpunkt muss *EK* alle Daten in *Google Maps* manuell eingeben, damit er die Route planen und für seine Kunden veranschaulichen kann. Ebenfalls optimiert *EK* die Route manuell, um eine möglichst optimale Reisestrecke zu erhalten. Optimal bedeutet für *EK*, dass die Route zeitlich gesehen so kurz wie möglich ist und die Reiseteilnehmer keine unnötigen Umwege in Kauf nehmen müssen.

Aus technischer Perspektive betrachtet kann das Problem wie folgt umschrieben werden: *FlexBüro* ist in *FileMaker* programmiert und bietet aktuell keinerlei Möglichkeiten effiziente Routenberechnungen vorzunehmen. Da *FlexBüro* keine weitere Programmiersprachen oder andere Installationen bei den einzelnen Kunden erfordert, soll auch die Routenberechnung und Optimierung ohne Zusatzinstallationen oder Plugins auskommen.

9.2 IST-Analyse

Im FileMaker 13.0v4 basierten Programm FlexBüro befinden sich alle Adressdatensätze mit zugehöriger Strasse, PLZ und Ort. Geografische Koordinaten der Adressen sind nicht hinterlegt. Zur Zeit kann keine Routenplanung oder Routenoptimierung aus dem

Programm heraus gemacht werden.

Die Infrastruktur, welche für die Applikation zur Verfügung steht:

- Der Clientrechner der Endkunden ist unbekannt. Als Voraussetzung können durchschnittliche Rechner mit mehreren Kernen und genügend Arbeitsspeicher angenommen werden, die aktuelle Browser ohne Plugins installiert haben.
- 2 Server: je Vmware, 4 CPU, Ubuntu-14.04.1-LTS-64bit, Standort Kanada (Anhang C)
- 1 Webserver: Hosting bei Hostpoint, Konfiguration unbekannt, Standort Schweiz

9.3 Zieldefinition

Das Projekt "Routenplaner in FlexBüro" stellt die Erweiterung des *CRM*-Programm FlexBüro um einen Routenplaner dar. Der Routenplaner bezieht die Adressdaten aus der Adresskartei von FlexBüro, stellt die geocodierten Adressen auf der Karte als *Marker* dar und berechnet die Ausgangsroute dazu. Die Ausgangsroute ist die Rundreise, welche vom Startpunkt ausgehend über alle Zwischenpunkte zurück zum Startpunkt führt. Die Route kann optimiert werden, um die auf die Fahrzeit bezogen kürzeste Rundreise anzuseigen.

9.3.1 Endprodukt

Für die Gesamtlösung und Einbindung des Routenplaners in FlexBüro sind weit mehr als in dieser Arbeit abgehandelten Themen relevant. Sie sind bewusst ausgeklammert und nicht Bestandteil der Bachelorarbeit. Dennoch ist die Anforderungsanalyse auf das Endprodukt ausgerichtet, damit keine essentiellen Teile vernachlässigt werden. Das Grobziel besteht darin, die Routenplanung aus den im Programm FlexBüro erfassten Adressen und Orten realisieren zu können.

9.3.2 Bachelorarbeit

Die Zieldefinition der Bachelorarbeit besteht darin, ein Konzept für die Berechnung von NP-vollständigen Problemen mit Webtechnologien zu konzipieren. Diese Konzept soll dann anhand dem Beispiel der Routenberechnung, nämlich dem *TSP*, verifiziert und diskutiert werden. Einzelne *POCs* sollen helfen, das Konzept aufzustellen und am Beispiel des *TSP* zu verifizieren. Die *POCs* werden als evolutionärer Prototyp entwickelt, damit diese Komponenten im Endprodukt eingesetzt werden können.

9.4 Stakeholderliste

Die *Stakeholderliste* für dieses Projekt umfasst die folgenden Personen, Firmen und Einrichtungen:

Name	Funktion (Rolle)
ZHAW	Auftraggeber
Alain Lafon	Betreuer
Endanwender (Reiseunternehmung <i>EK</i>) des Gesamtsystems	Auftraggeber
Roman Lickel (ZHAW)	Konzept und Umsetzung des Systems
Bruno Bachmann (Bachmann Support GmbH)	Entscheidungsträger bei anschliessender Integration in FlexBüro Applikation
Roman Lickel (Bachmann Support GmbH)	Schnittstellen-Entwickler und Umsetzung der finalen Applikation
Ramon Selinger (Bachmann Support GmbH), Gaby Schwendener (Bachmann Support GmbH)	Entwicklerfirma der CRM Applikation FlexBüro (weitere Projektbeteiligte)
Roman Lickel (ZHAW)	Projektleitung

Tab. 9.1: Stakeholderliste (*Quelle: Eigene Darstellung*)

Die *Stakeholder* für das Projekt stammen alle aus der Firma Bachmann Support GmbH, der ZHAW oder aus der Reiseunternehmung *EK*. Dabei stellt die Reiseunternehmung *EK*, bezogen auf das Endprodukt, der einflussreichste *Stakeholder* dar. *EK* definiert die Anforderungen an das System, welche später mit Roman Lickel (Bachmann Support GmbH) ausgearbeitet und in den Anforderungen in den Abschnitten 9.7.1 und 9.7.2 erfasst werden. Nach Absprache mit den weiteren Auftraggebern und der Abnahme durch den Entscheidungsträger Bruno Bachmann wird das Projekt durch Roman Lickel (ZHAW) ausgeführt und von Alain Lafon betreut.

9.5 Annahmen und Einschränkungen des Systems

Bachmann Support GmbH ist bestrebt eine generische Lösung zu implementieren, welche auch für weitere Kunden verwendet werden kann. Natürlich sollen aber dennoch alle Anforderungen von *EK* abgedeckt sein.

Für die Routenberechnung im FlexBüro sollen keinerlei Plugins oder andere Softwarekomponenten installiert werden müssen.

Aufgrund der im vorangehenden Abschnitt beschriebenen Problemdarstellung, dem vorhandenem Know-How und Ressourcen hat Bachmann Support GmbH aufgrund derzeitigen Entwicklungen der Informationstechnologien beschlossen, die Lösung mit

Webtechnologien zu lösen. Diese Entscheidung soll ohne weitere Begründung hingenommen werden können.

Weitere Einschränkungen des Systems sind, dass nur ein Handelsreisender unterstützt wird und dass die Gesamtlänge der Route ohne Zeitlimitierung erfolgt. Das bedeutet, dass die Route nicht in pro Tag machbare Etappen gegliedert wird, sondern die Gesamtroute berechnet und angezeigt wird.

Ebenfalls werden lediglich Routen berücksichtigt, welche über bestehende Straßenverbindungen erreicht werden können und vom Routingserver bereitgestellt werden. Das bedeutet, dass keine Hindernisse wie Meere und Gebirge überquert werden können, falls keine Brücken oder Tunnels existieren.

9.6 Systemkontext und Randbedingungen

Die Randbedingungen und der Systemkontext sind in Abb. 9.1 dargestellt:

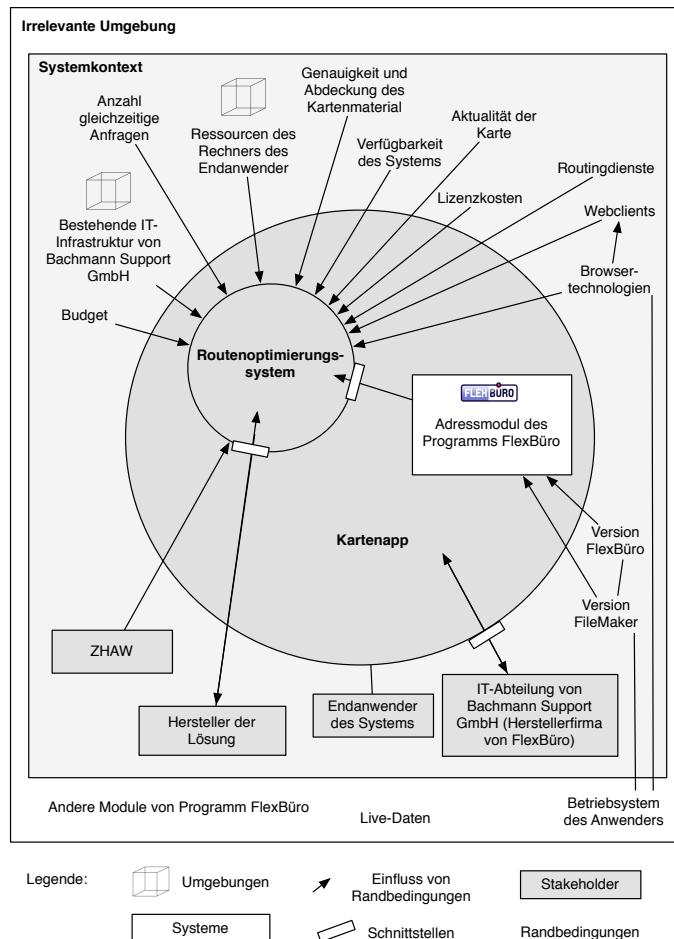


Abb. 9.1: Kontextabgrenzung des Systems (Quelle: Eigene Darstellung)

Die gesamte Erweiterung kann in einen Kern, genannt "Routenoptimierungssystem", und eine erweiterte Systemumgebung "Kartenapp" unterteilt werden. Diese Abgrenzung

zeigt auf, welche Teile für die Bachelorthesis relevant sind, nämlich der Teil "Routenoptimierungssystem", und welche zusätzlichen Teile für die gesamte Systemumsetzung nötig sind. Ebenfalls wird zwischen dem Systemkontext und der irrelevanten Umgebung unterschieden. Alle Module vom Programm FlexBüro, mit Ausnahme des Adressmoduls, sind für die Systementwicklung irrelevant. Da die Randbedingung "Version FileMaker" der limitierende Faktor neben den Browsetechnologien für die Systementwicklung darstellt, kann das Betriebssystem ebenfalls in der irrelevanten Umgebung angesiedelt werden. Live-Daten wie Verkehrsaufkommen, Baustellen, Unfälle und Umleitungen werden bei der Umsetzung des Systems ebenfalls nicht berücksichtigt.

Das Adressmodul hingegen, welches sowohl von der FileMaker Version als auch der Version von FlexBüro abhängig ist, ist für die Systemimplementation von Relevanz. Bachmann Support GmbH und die Endanwender des Systems nehmen grossen Einfluss auf das Routenoptimierungssystem. Für den Kern des Systems existieren weitaus mehr einflussnehmende Randbedingungen. Als grösste, kaum beeinflussbare Randbedingung kann das System und die vorhandenen Ressourcen des Rechners des Endanwenders angenommen werden. Daneben stellt die IT-Infrastruktur von Bachmann Support GmbH eine wichtige Randbedingung dar, da von einer ressourcenintensiven Berechnung ausgangen werden muss, um das Problem des *TSP* zu lösen. Auch eng damit verbunden sind die Lizenzkosten. Die Einflüsse "Anzahl gleichzeitige Anfragen" und "Verfügbarkeit des Systems" dürfen ebenfalls nicht vernachlässigt werden. Weitere Randbedingungen sind auf die Karten und Routingdienste zurückzuführen und beeinflussen das System wesentlich, da ohne deren Daten das System nicht funktionsfähig ist. Ebenfalls essentiell ist der Web Client, welcher für die Darstellung der Kartenapplikation im Browser verantwortlich ist und je nach Auswahl ebenfalls Einfluss auf die Lizenzkosten hat.

9.7 Anforderungsanalyse

In diesem Teil des Dokuments befinden sich sowohl funktionale und als auch qualitative Anforderungen an das System. Die Anforderungen sind nach folgendem Schema mit einem eindeutigen Identifikator versehen, damit darauf referenziert werden kann:

RE-<TYP><UseCaseID>-<Anforderungsnummer>. Die vollständige Anforderungsliste ist im beigelegten Excel File "Anforderungsanalyse.xlsx" zu finden.

Anforderungen mit der Bezeichnung RE-<TYP><UseCaseID>-**I**-<Anforderungsnummer> sind Anforderungen, welche während der Entwicklung hinzugekommen sind. Die Anforderungsliste stellt also gewissermassen auch gleichzeitig das Backlog dar.

Die Anforderungen an das System stammen aus vorhandenen Unterlagen und Interviews mit *EK*. Weitere Systemmerkmale werden durch Bachmann Support GmbH bei einem Brainstorming gesammelt, gewichtet, bewertet und zu Anforderungen formuliert.

9.7.1 Funktionale Anforderungen

Die Tabelle 9.2 enthält die an das System gestellten funktionalen Anforderungen.

ID	Name	Beschreibung	UseCase
RE-F1-1	Addressen aus FlexBüro exportieren	Die aufgerufenen Adressen im FlexBüro sollen auf Klick auf das <i>Marker</i> -Icon in die Kartenapp exportiert werden.	UC1
RE-F1-2	Kartenapp innerhalb FlexBüro	Die Webapp soll in einem neuen Fenster innerhalb von FlexBüro geöffnet werden.	UC1
RE-F1-3	Geocoding	Die hochgeladenen Adressen müssen in der App geocodiert werden (Lat,Lng muss ermittelt werden).	UC1
RE-F1-4	Nicht geocodierbare Adressen	Nicht geocodierbare oder lokalisierbare Adressen werden inaktiv geschaltet.	UC1
RE-F1-5	Aktive Adressen als <i>Marker</i>	Aktive Adressen werden als <i>Marker</i> (nummeriert) auf der Karte dargestellt.	UC1
RE-F2-1	Ausgangsroute	Klick auf "Route zeigen" erstellt die Route durch alle gegebenen <i>WP</i> (aktive) und zeigt diese auf der Karte.	UC2
RE-F3-1	Route optimieren	Klick auf "Route optimieren" berechnet die optimale Route (so schnell und genau als möglich).	UC3
RE-F4-1	Optimierte Route anzeigen (Liste)	Die <i>WP</i> werden im GUI umgeordnet, gemäß der neu ermittelten Route.	UC4
RE-F4-2	Optimierte Route anzeigen (Karte)	Auf der Karte wird die optimierte Route (Rundreise für gegebene <i>WP</i>) angezeigt.	UC4
RE-F2,3,4-1	Routenberechnung	Routenberechnung erfolgt auf realen Straßen (streetlevel).	UC2,3,4
RE-F1-I-1	Zeile markieren bei Klick auf <i>Marker</i>	Beim Klick auf einen <i>Marker</i> innerhalb der Karte soll dieser <i>WP</i> innerhalb der Liste markiert werden, damit sichtbar wird, um welchen <i>WP</i> es sich handelt.	UC1

Tab. 9.2: Funktionale Anforderungen an das System (Quelle: Eigene Darstellung)

9.7.2 Nicht-funktionale Anforderungen

ID	Name	Beschreibung	UseCase
RE-Q0	Sicherheit	Kundendaten dürfen auf keinen Fall von Dritten eingesehen werden können.	-
RE-Q1	Anpassbarkeit Sprache	Das GUI der App soll leicht um weitere Sprachen erweitert werden können (in einem ersten Schritt aber nur Englisch sein).	UC1

ID	Name	Beschreibung	UseCase
RE-Q2	Unabhängigkeit	Falls möglich sollen Opensource Produkte eingesetzt werden, um Lizenzkosten zu sparen und Abhängigkeiten zu Firmen (wie beispielsweise Google) zu vermeiden.	UC2
RE-Q3-1	Korrektheit ≤ 15	Die Berechnung soll für 13 WP noch die perfekte Lösung ergeben.	UC3
RE-Q3-2	Korrektheit > 15	Bei über 13 WP soll die bestmögliche Näherungslösung gefunden werden.	UC3
RE-Q3-3	Geschwindigkeit	Die Lösungsfindung darf keinesfalls zwei Minuten überschreiten.	UC3
RE-Q3-4	Anpassbarkeit / Erweiterbarkeit	Die Ermittlung der besten Route, soll erweiterbar sein, damit zu einem späteren Zeitpunkt weitere Parameter eingefügt werden können (Bsp. Verkehrsaufkommen).	UC3
RE-Q3-5	Mandanten	Die Applikation soll von mehreren Mandanten von Bachmann Support GmbH betrieben werden können.	UC3
RE-Q3-6	Testbarkeit	Die TSP Berechnung soll einfach getestet werden können.	UC3
RE-Q3-7	Zuverlässigkeit	Die Applikation soll zuverlässig laufen.	UC3
RE-Q3-8	Unabhängigkeit von Kartendienst	Die Berechnung des TSP soll soweit als möglich unabhängig von einem Kartenanbieter und Map Client implementiert sein, damit ein Wechsel auf einen neuen Kartenanbieter schnell vollzogen werden kann.	UC3
RE-Q3-9	Skalierbarkeit Problemgrösse	Bei grösserer Anzahl WP soll die Applikation entsprechend skalieren.	UC3
RE-Q3-10	Skalierbarkeit Requestload	Die Applikation soll auch unter grosser Last noch die gewünschte Performance liefern (siehe RE-Q3-3).	UC3
RE-Q1,2,3-1	Usability: Benutzerbarkeit der Routenplanungs-app	Der Benutzer soll mit zwei wenigen Klicks zur optimierten Route seiner gewählten WP gelangen. Das GUI soll intuitiv bedienbar sein.	UC1,2,3
RE-Q1,2,3-2	Usability: Statusanzeige bei langen Berechnungen	Ladebalken sollen dem User den Status von langen Berechnungen anzeigen.	UC1,2,4
RE-Q1,2,3-3	Ohne Plugins	Die gesamte Applikation soll ohne jegliche Plugins (auf Seite des Kunden) lauffähig sein. Plugins auf Server sollen mit guter Begründung möglich sein.	UC1,2,4
RE-Q1,2,3-4	Geschützter Code	Der Quellcode der Applikation soll weitgehend geschützt werden können.	UC1,2,4
RE-R2-1	Lizenzkosten	Softwarelizenzen und Kosten für Services sollen möglichst gering gehalten werden.	UC2
RE-R-1	Entwicklungskosten	Entwicklungskosten sollen gering gehalten werden.	UC1,2,3,4

Tab. 9.3: Nicht-funktionale Anforderungen an das System (*Quelle: Eigene Darstellung*)

9.7.3 Anwendungsfälle (Use Cases)

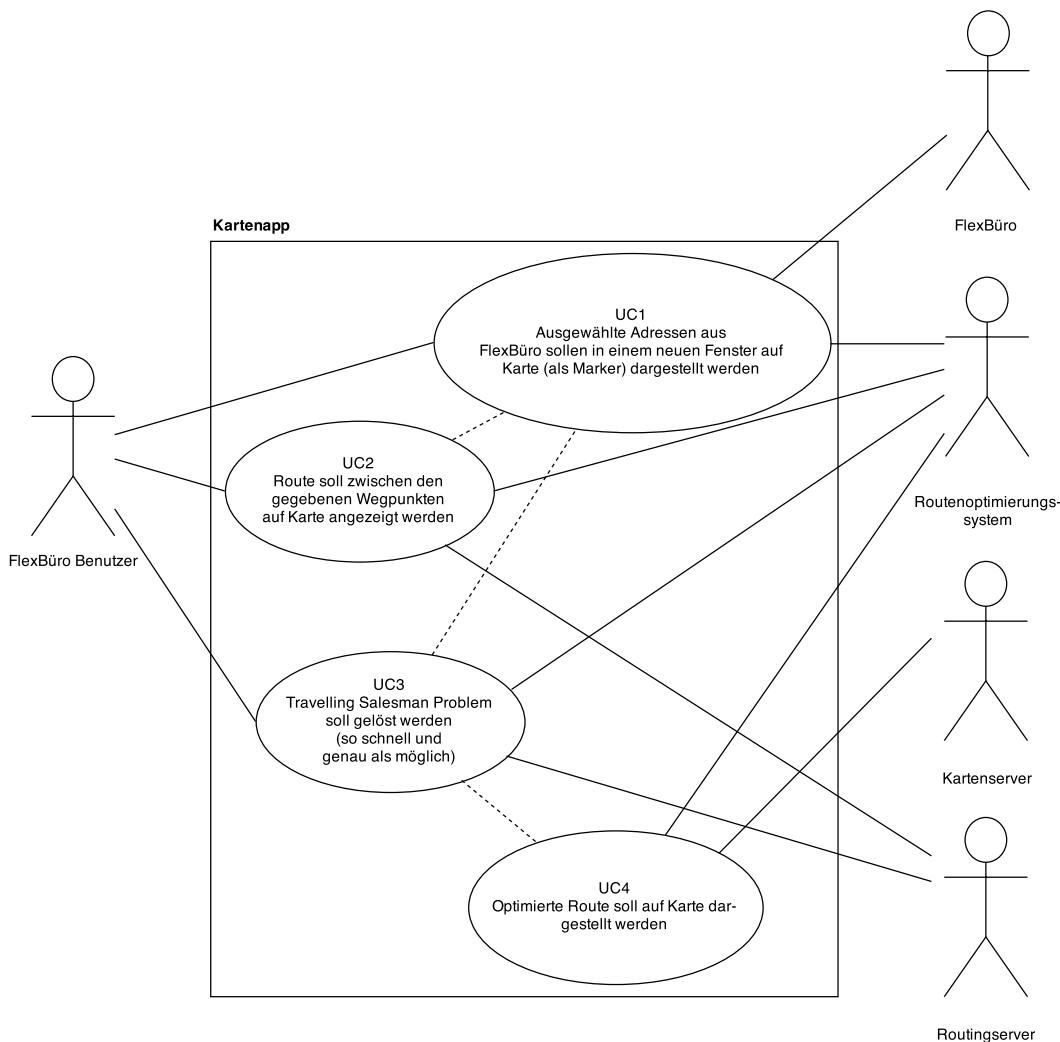


Abb. 9.2: Use Case-Diagramm (*Quelle: Eigene Darstellung*)

Im Use-Case-Diagramm sind die einzelnen Use Cases der Applikation aufgezeigt. Der Akteur "FlexBüro Benutzer" entspricht *EK* und "FlexBüro" stellt das *CRM* Programm FlexBüro dar. Des Weiteren weist das System den Akteur "Routenoptimierungssystem" auf, welcher die Applikation für die Kartendarstellung, Routendarstellung und Routenberechnung bildet und die Dienste der weiteren Akteure "Kartenserver" und "Routingserver" beansprucht.

Aus den Use Cases und den Anforderungen der vorangehenden Abschnitten ist der Mockup in Abb. 9.3 der zu erstellenden Applikation abgeleitet: In der Listenansicht des *CRM-Systems* FlexBüro können die gewünschten Adressen selektiert und mit Klick auf das *Marker* Symbol in der unteren Menüleiste in einem Zusatzfenster in der Karte

dargestellt werden. Danach folgt die Routenberechnung und weitere Visualisierung innerhalb dieses Fensters.

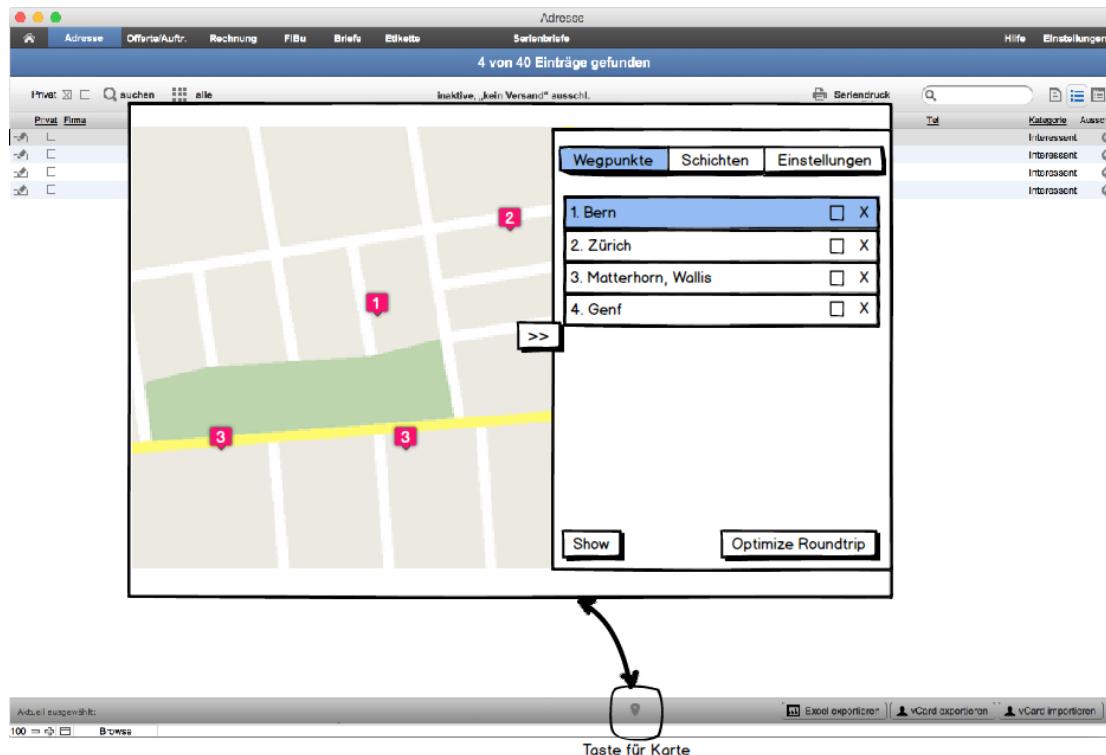


Abb. 9.3: Mockup der Applikation (Quelle: Eigene Darstellung)

9.8 Entscheidungen

9.8.1 Technologien

Da die Anforderung RE-Q1,2,3-3 (System ohne Plugins) besteht und die meisten *Map Clients* (siehe Abschnitt 9.8.3) in JavaScript geschrieben sind, wird für die Umsetzung der Applikation JavaScript eingesetzt. Zudem ist JavaScript sowohl auf einem Server als auch im Browser lauffähig. JavaScript basierte Webserver existieren mittlerweile einige. Eine Liste kann auf Wikipedia [86] gefunden werden, wobei die Evaluation nicht Bestandteil dieser Arbeit ist und für die Implementation ohne weitere Begründung Node.js ausgewählt wurde. Zur Darstellung werden neben JavaScript HTML und CSS eingesetzt.

9.8.2 Karten- und Routingserver

Wegen der Randbedingung der "weltweiten Karte" sind lediglich Kartenanbieter mit weltweitem Kartenmaterial in die Evaluation einbezogen worden. Neben der Möglichkeit,

Routen auf der gesamten Welt anzeigen zu können, existieren dadurch auch grössere Entwickler-Communities für die Projekte. Aufgrund der Technologieentscheidung wurden auch nur Produkte in die Auswahl aufgenommen, welche eine JavaScript-Anbindung zur Verfügung stellen. In der Tab. 9.4 sind die verfügbaren Kartenprovider mit deren wichtigsten Eigenschaften, wie die Möglichkeit des Routings und Geocodings sowie der Lizenzkosten, aufgelistet.

Produkt	Routing verfügbar	Geocoding verfügbar	Lizenzkosten	Anbieter
ArcGIS	Ja	Ja	ArcGIS for Home Programm (100USD), sonst sehr hohe Lizenzkosten	ESRI [16]
Baidu Api	Ja	Ja	-	Baidu [2]
Google Maps	Ja	Ja	Business kostenpflichtig, Privat kostenlos	Google [26]
Microsoft Bing Maps	Ja	Ja	Business kostenpflichtig, Privat oder öffentlich zugänglich kostenlos	Bing, Microsoft [8]
OSM (Open Street Map)	Ja	Ja	Opensource (kostenlos)	OSM [58]
Yahoo! Maps (Nokia HERE)	Ja	Ja	Lizenzmodell mit Transaktionen und Limits	HERE [33]
NASA World Wind	Nein	Nein	kostenlos	NASA [49]
TOMTOM Api	Ja	Ja	Kosten auf Anfrage	TOMTOM [74]
ViaMichelin Api	Ja	Ja	Kosten auf Anfrage	Viamichelin [79]

Tab. 9.4: Karten- und Routinganbieter mit API für Webapplikationen (Quelle: Eigene Darstellung)

Anmerkungen: Die Baidu Api und deren Dokumentation ist momentan lediglich in chinesisch verfügbar. Da Bachmann Support GmbH keine zusätzlichen Lizenzkosten für die Applikation aufwenden kann, wurde in der internen Sitzung vom 23. Dezember 2014 beschlossen, dass das Kartenmaterial und das Routing von Openstreetmap eingesetzt wird. Der grosse Vorteil gegenüber den anderen Anbietern besteht darin, dass ein eigener Kartenserver und Routingserver mit den Daten von Openstreetmap aufgesetzt werden kann, da Openstreetmap ein *Opensource*-Produkt ist. Dies ermöglicht die Konfiguration und Anpassung auf die eigenen Bedürfnisse. Diese Möglichkeit wird aber an dieser Stelle lediglich als Option für die zukünftige Erweiterung des Systems erwähnt und ist keinesfalls Bestandteil der Arbeit.

9.8.3 Map Clients

Ein ausführlicher *Map Client*-Vergleich kann der Webseite von German Carrillo [24] entnommen werden. Basierend darauf wurde OpenLayers als *Map Client* ausgewählt, wobei die Applikation so zu implementieren ist, dass diese Komponente gegebenenfalls ausgetauscht werden kann, ohne die gesamte Applikation neu schreiben zu müssen. Der Entscheid für die Verwendung von OpenLayers ist hauptsächlich auf die sehr gute Integration mit den Openstreetmap-Daten, den grossen Funktionsumfang und die gute Dokumentation zurückzuführen.

9.9 Risikoanalyse

Mit der hier angefügten Risikoanalyse sollen die Projektrisiken und der Erfolg des Projekts besser abgeschätzt werden können. Zudem soll sie helfen, die Anforderungen an das System gegebenenfalls anzupassen, damit das System in der vorgegebenen Zeit fertiggestellt werden kann. Es wird versucht, die Risikoquellen zu ermitteln und diese dann nach Eintrittswahrscheinlichkeit einzustufen, um so eine Risikoprioritätszahl zu erlangen. Im daraus resultierenden Risikoportfolio ist ersichtlich, wie gravierend die einzelnen potenziellen Schäden und deren Eintrittswahrscheinlichkeit sind und wie die Situation über das gesamte Projekt gesehen aussieht.

Stufe	Wahrscheinlichkeit	Interpretation
1	$0 \leq p \leq 0.25$	Es ist eher unwahrscheinlich, dass das Risiko eintritt, aber nicht auszuschliessen.
2	$0.25 < p \leq 0.5$	Das Risiko wird eher nicht eintreten, es ist aber dennoch möglich.
3	$0.5 < p \leq 0.75$	Das Risiko wird eher eintreten, es ist aber keineswegs sicher.
4	$0.75 < p \leq 1$	Das Risiko wird mit hoher Sicherheit eintreten.

Tab. 9.5: Eintrittswahrscheinlichkeitsstufen der Projektrisiken (Quelle: Hindel et al. [34])

In der Tabelle 9.5 sind die einzelnen Eintrittswahrscheinlichkeitsstufen der Risiken aufgeführt. Die angegebenen Stufen werden in den nachfolgenden Tabellen und Berechnungen verwendet.

Um diese Risikoquellen einer Schadenshöhe zuzuordnen, werden gemäss nachfolgender Tabelle die Schadenshöhen für die beiden Kriterien "Schaden hinsichtlich Endprodukt des Projekts" und "Schaden hinsichtlich Projektdauerüberschreitung" definiert und genauer ausgeführt.

Stufe	Schaden hinsichtlich Endprodukt des Projekts	Schaden hinsichtlich Projektdauerüberschreitung
1	Kaum merkbare Einschränkungen für Benutzer des Systems.	Verzögerung kann wieder aufgeholt und der Endtermin eingehalten werden.
2	Die Applikation funktioniert im Wesentlichen, der Endbenutzer bemerkt in einigen Fällen das fehlerhafte Verhalten des Systems, das aufgrund von fehlerhaften Funktionen oder nicht umgesetzten Funktionen herrührt.	Verzögerung kann mit Mehraufwand wieder aufgeholt und der Endtermin eingehalten werden.
3	Die Applikation funktioniert nur ansatzweise und wesentliche Komponenten funktionieren nicht.	Die Projektdauer kann nicht eingehalten werden und die wesentlichen Komponenten müssen nach Projektabschluss noch nachgerüstet werden (Überschreitung der Projektdauer $\leq 20\%$).
4	Die Routingapplikation kann nicht umgesetzt werden und das Endprodukt ist nicht brauchbar.	Die Projektdauer wird überschritten und das Projekt kann nicht fertiggestellt werden.

Tab. 9.6: Charakteristik der Schadenshöhe (*Quelle: Eigene Darstellung*)

Die Zusammenführung und Auswertung der Tabellen 9.5 und 9.6 führt zur Tabelle 9.7. Sie enthält die sogenannte Risikozahl, welche das erfasste Risiko nach der Eintrittswahrscheinlichkeit und Schadenshöhe einstuft. Die Berechnung für die Risikozahl ist $\text{Eintrittswahrscheinlichkeit} * \text{Schadenshöhe}$. Je höher die Risikozahl ist, desto ernstzunehmender ist dieses Risiko, da es sowohl mit hoher Wahrscheinlichkeit eintritt, als auch eine grosse Schadenshöhe verursachen kann.

Nr.	Beschreibung	Schadenshöhe	Eintrittswahrscheinlichkeit	Risikozahl	Risikoquelle
1	Unzureichende Anforderungsanalyse.	3	1	3	Anforderungsrisiko
	Die Anforderungen werden von <i>EK</i> und der Firma Bachmann Support GmbH definiert. Bachmann Support GmbH ist eine erfahrene Firma in diesem Bereich, daher wird die Eintrittswahrscheinlichkeit als gering eingestuft.				
2	Zu viele Anforderungen.	3	1	3	Anforderungsrisiko
	Es existieren nicht viele Anforderungen und diese können gut koordiniert werden.				
3	Routingserver Dienste und Anbieter sind zu teuer.	4	1	4	Anforderungsrisiko
	Der Schaden wäre enorm, jedoch existieren auch alternative Anbieter und <i>Opensource</i> Lösungen.				
4	Komplexität der Problemstellung ist zu hoch.	2	2	4	Anwendungsrisiko
	Zur Lösung des Problems ist fundiertes mathematisches und technisches Wissen von Nöten.				
5	Unrealistische Terminsetzung und Meilensteine.	1	1	1	Auftrag-/Abwicklungsrisiko
	Die Erfahrung von vielen Projekten mit dem selben Umfang hilft die Meilensteine und Terminsetzungen angemessen vorzunehmen.				

6	Limitierte Webtechnologien.	4	3	12	Technik
	Webtechnologien haben auch in der heutigen Zeit noch einige Defizite, diese müssen geschickt umgangen werden.				
7	Unklar ob geforderte Performanz gewährleistet werden kann.	4	4	16	Technik
	Die Performanz von Webtechnologien ist in Bezug auf die rechenintensive Lösung von <i>NP-vollständigen</i> Problemen womöglich nicht die optimale Technologie.				

Tab. 9.7: Risikoliste (*Quelle: Eigene Darstellung*)

Um die Risiken der Tab. 9.7 übersichtlicher darzustellen, kann ein Risikoportfolio erstellt werden, welches in der Tab. 9.8 dargestellt ist. Dunkel eingefärbt sind Bereiche, welche eine hohe Risikozahl aufweisen.

Schadenshöhe

Stufe 4 (hoch)	3	-	6	7	
Stufe 3	1, 2	-	-	-	
Stufe 2	-	4	-	-	
Stufe 1 (gering)	5	-	-	-	
	Stufe 1 (gering)	Stufe 2	Stufe 3	Stufe 4 (hoch)	Wahrscheinlichkeit

Tab. 9.8: Risikoportfolio (*Quelle: Eigene Darstellung*)

Es ist ersichtlich, dass den Risiken mit den Nummern 6 und 7 besondere Beachtung zu schenken sind. Bei beiden Risikofaktoren handelt es sich aber um Bestandteile des Untersuchungsgebiet der Bachelorthesis und werden daher ohnehin mit grosser Aufmerksamkeit verfolgt.

10 Existierende Applikationen (R0)

Dieses Kapitel erfasst im Rahmen einer Marktanalyse die zum jetzigen Zeitpunkt existierenden, im Internet verfügbaren *TSP*-Berechnungslösungen und vergleicht diese untereinander. Die Kennzahlen sind zusammen mit den in Kapitel 9 "Requirements Engineering (R1)" aufgeführten Anforderungen erarbeitet. Aus den gewonnenen Erkenntnissen der Marktanalyse sollen Verbesserungs- und Optimierungsideen abgeleitet werden, wovon die zu implementierende Lösung profitieren soll.

10.1 Kennzahlen

Technologie: Der verwendete Kartenanbieter und Routingserver (gemäss Abschnitt 9.8.2) sowie der verwendete *Map Client*.

Architektur: Die Kennzahl "Architektur" wird aufgrund der in Kapitel 5 "Webtechnologien" erarbeiteten Fakten und resultierenden Erkenntnisse bewertet.

Algorithmus zur Lösungsfindung: Soweit dies publiziert und ersichtlich ist oder in Erfahrung gebracht werden kann, wird der verwendete Algorithmus angegeben. Anhand des Algorithmus kann die zu erwartende Genauigkeit abgeschätzt werden.

Begrenzungen: Diese Kennzahl gibt die maximale Anzahl *WP* an, welche erfasst werden können und allfällige Limitierungen bezüglich des Kartenausschnitts.

Geschwindigkeit: Eng an die vorangehenden Kennzahlen geknüpft ist die Berechnungsgeschwindigkeit. Da jede Applikation unterschiedliche Limitierungen hat, kann keine einheitliche Route für den Vergleich verwendet werden. Daher wird die Dauer für die maximale Anzahl *WP* gemessen, welche noch kostenlos berechnet werden kann (verwendetes Gerät ist im Anhang aufgeführt: MacBook).

Kosten: Da die Lizenzkosten für den Auftraggeber Bachmann Support GmbH von grosser Relevanz sind, ist die Kennzahl der Kosten essentiell.

Spezial: Diese Kennzahl soll besondere Funktionalitäten aufzeigen und Aussage über die Usability machen.

Name	Technologie	Architektur	Algorithmus	Geschwindigkeit [s]	Begrenzungen	Kosten	Spezial	Link
RouteXL	Google Maps	Server, Details unbekannt	Unbekannt	20 WP: 52	max. 150 WP	20 WP kostenlos 5€ pro Tag 35€ pro Monat	Mehrere WP können miteinander hinzugefügt werden	[62]
Routino	OSM, Leaflet	Server, Details unbekannt	Unbekannt	3	9 WP, nur für das Vereinte Königreich von England (UK)	Kostenlos	Viele Optionen, GUI unübersichtlich	[63]
Optimap	Google Maps	Client, normal	Abhängig von WP: bis 15 WP (Dynamic Programming) ab 16 WP Nähierung mit Ant-Colony-, 2-Opt- und k-Opt-Algorithmen	15 WP: 43 40 WP: 75	100 WP	Kostenlos	Mehrere WP können miteinander hinzugefügt werden	[54]
Uni Heidelberg	OSM, OpenLayers 2.12	-	-	-	-	Kostenlos	Geocoding funktioniert nicht, somit kann die Lösung nicht genutzt werden; Gebiete die nicht befahren werden sollen	[76]
Findthebestroute	Google Maps	Client, normal	Brute-Force	2	10 WP	Kostenlos	Nicht benutzerfreundliches GUI	[19]

Tab. 10.1: Marktübersicht (Quelle: Eigene Darstellung)

Name	Technologie	Architektur	Algorithmus	Geschwindigkeit [s]	Begrenzungen	Kosten	Spezial	Link
Drivingrouteplanner	Google Maps, HERE, MapQuest, Distanz Matrix	Client, normal, mit Caching	Unbekannt	39 WP: 120	keine Limite	Kostenlos	Sehr viele Optionen (schnellste und kürzeste Route); Route editierbar; Kein Drag and Drop	[13]
Mapquest	Mapquest	Client, normal	Brute-Force	5	24 WP	Kostenlos	Verkehrsaufkommen	[41]
TourPI	OpenLayers 1.4.0, OSM	Server, Details unbekannt	Unbekannt	25	10 WP	Privat: 30 CHF Organisationen: 300 CHF	XML Web API, Live Routen - berechnung	[75]
Speedyroute	Google Maps	Server, Details unbekannt	Unbekannt	4	500 WP	5 Routen mit 8 WP pro Tag kostenlos 1 Tag: 5,00 € 1 Woche: 13,50 € 1 Monat: 42,50 €	XML Web API, Live Routen - berechnung	[69]
Multiroute	Microsoft Bing Maps	Server, Details unbekannt	Unbekannt	8	7 WP kostenlos, Total Anzahl WP unbekannt	60 Minuten: 1,99 € 24 Stunden: 4,99 € 1 Monat: 49,00 € 1 Jahr: 299,00 € API: ab 892,50 €	GUI unübersichtlich	[48]
Ontimehub	Google Maps	Server, Details unbekannt	Unbekannt	-	-	User/Monat ab 4,50 € API, User/Monat ab 29,90 €	Webseite ist langsam; kompliziert, da sehr viele Einstellungsmöglichkeiten	[51]

Tab. 10.1: Marktübersicht (Fortsetzung)

Aus Tab. 10.1 kann gefolgert werden, dass Lösungen deren Berechnung serverseitig erfolgen, für die Endkunden teurer sind als diejenige, welche auf clientseitigen Technologien aufbauen. Serverseitige Lösungen verursachen vermutlich auch grössere Kosten bei den Betreibern, da für die Berechnungen Serverressourcen nötig sind. Die Korrelation zwischen kostenpflichtigen Services und serverseitiger Berechnung kann aber auch darauf zurückzuführen sein, dass kommerzielle Produkte serverseitige Berechnungen bevorzugen, da der Code geschützt werden kann und eine Kontrolle der Benutzung möglich ist, welche nicht umgangen werden kann.

Aus der Marktanalyse geht ebenfalls hervor, dass die meisten Applikationen den Kartendienst und *Map Client* von Google einsetzen. Dieser Fakt kann auf die Einfachheit der *API* und der Marktpositionierung von Google mit dem eigenen Google-Maps-Dienst zurückgeführt werden.

Da nur wenige Anbieter die Berechnung des *TSP* auf dem Client ausführen, können kaum Aussagen über die eingesetzten Algorithmen gemacht werden. Interessant ist die Implementation von Optimap, welche aufgrund von der gegebenen Anzahl *WP* entscheidet, welchen Algorithmus zur Optimierung verwendet wird.

Die Begrenzungen der Anzahl *WP* sind sehr unterschiedlich und bei den meisten Applikationen können nur Routen mit weniger als 10 bis 20 *WP* kostenlos optimiert werden. Die Usability der meisten Applikationen genügen den hohen Ansprüchen von Bachmann Support GmbH nicht, da die Bedienung zu kompliziert und somit nicht benutzerfreundlich ist.

11 Lösungsmöglichkeiten von ***NP-vollständigen Problemen im Web***

Im Teil II "Theoretische Grundlagen" sind Architekturkonzepte aus der Informatik und Algorithmen zur Lösung des *TSP* vorgestellt worden. Zusammen mit den Erkenntnissen aus der Marktanalyse in Kapitel 10 soll im Folgenden geprüft werden, inwiefern diese Konzepte auch im Webbereich angewendet werden können.

Das Gesamtkonzept beschreibt mögliche Architekturvarianten zur Lösung von *NP-vollständigen* Problemen. Es wird versucht, die in den Grundlagen erarbeiteten Architekturen in einem allgemeingültigen Konzept zu vereinen und als Entscheidungsbaum darzustellen. Die enthaltenen Konzepte werden in einem nächsten Schritt weiter verfeinert und in *POCs* auf deren Umsetzbarkeit geprüft. Das gesamte Konzept basiert auf der Tatsache, dass *NP-vollständige* Probleme rechenintensiv sind und die in Kapitel 8 beschriebenen Anforderungskonflikte gelten. Die vorhandene Rechenleistung soll optimal genutzt werden, um die Berechnung möglichst effizient bewältigen zu können. Deshalb wurden in der Informatik Konzepte, wie parallel ablaufende Algorithmen und verteilte Systeme eingeführt, welche in diesem Kapitel auf das Webumfeld adaptiert werden sollen.

Die Mindestanforderung, um das Gesamtkonzept auf den Webbereich anwenden zu können, ist ein Client (Webbrowser) und ein Server (Webserver). Ebenfalls werden Algorithmen zur Lösung des Problems vorausgesetzt.

Die verwendeten Symbole für die beiden Entscheidungsdiagramme in Abb. 11.2 und Abb. 11.3 sind in Abb. 11.1 erklärt.



Abb. 11.1: Symbollegende für die Konzepte (*Quelle: Eigene Darstellung*)

Im Entscheidungskonzept in Abb. 11.2 wird von der Ausgangslage und den bereits beschriebenen Mindestanforderungen über mehrere Entscheidungsknoten und Entscheidungswege ein Lösungsvorschlag hergeleitet. Als Ausgangslage dient ein *NP-vollständiges* Problem, welches gelöst werden soll. In einem ersten Schritt wird versucht, das Problem in kleinere Teilprobleme zu unterteilen. Anhand der Problemgröße können beispielsweise sinnvolle Problemgruppen erstellt werden. Auf jedes dieser Teilprobleme wird nun das Konzept in Abb. 11.3 angewendet. Sobald alle Teilprobleme durchlaufen sind, wird die Gesamtlösung des Problems durch die Vereinigung aller Teillösungen gebildet.

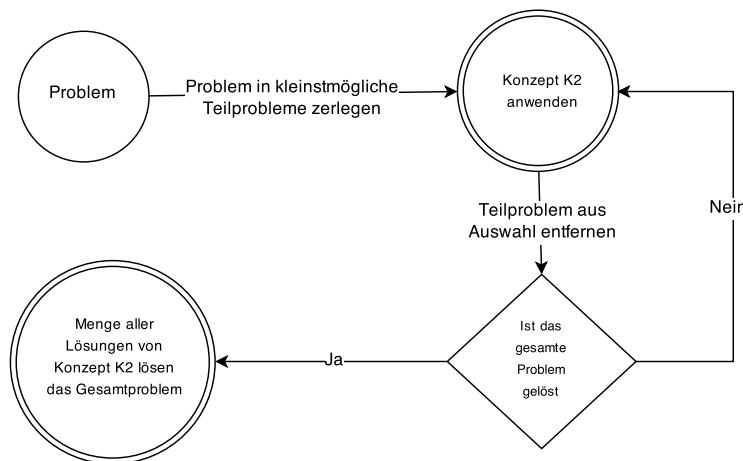


Abb. 11.2: Entscheidungsdiagramm (K1) zur Zerlegung von NP-vollständigen Problemen
(Quelle: Eigene Darstellung)

Das Konzept ist aus den Überlegungen, dass Teilprobleme gegebenenfalls durch unterschiedliche Lösungsvarianten effizienter gelöst werden können entstanden. Je nach Entscheidungen im Konzept Abb. 11.3 ist es aber dennoch möglich, dass das gesamte Problem schliesslich durch dieselbe Architekturvariante gelöst wird, da alle Teilprobleme mit derselben Architekturvariante gelöst wurden.

Das Entscheidungsdiagramm Abb. 11.3 soll helfen, eine geeignete Architektur für den aktuellen Problemfall herauszufinden. Grundsätzlich kann zwischen exakten Lösungsverfahren und Näherungsverfahren unterschieden werden. Diese können wiederum auf lediglich einem Client, auf mehreren Clients, auf einem Server, auf mehreren Servern oder auf einem mit Client und Servern gemischten, verteilten System berechnet werden. Dieselben Berechnungsvarianten existieren auch für die Lösungsfindung über Näherungsverfahren. Zusätzlich muss entschieden werden, ob eine Parallelisierung einen Effizienzgewinn bringt. Abhängig sind diese Entscheidungen von den verfügbaren Algorithmen, deren Umsetzungsaufwand, Kompetenzen der Programmierer, verfügbaren Ressourcen und Anforderungen der Hersteller und Kunden an das System.

In Anbetracht der verfügbaren Ressourcen lässt sich folgern, dass immer von der kleinstmöglichen Lösung ausgegangen werden soll. Kleinstmöglich bedeutet, dass jeweils zuerst eine Architektur nur auf dem Client und nicht parallelisiert angestrebt werden soll (L2). Die nächst grössere Architektur stellt die auf einem Workerthread im Client betriebene, nicht parallelisierte Lösung L3 dar, gefolgt von der auf einem Server (nicht parallelisiert L5 und parallelisiert L6) bis hin zur Lösung in einem verteilten System (L7 - L9). Diese Entscheidung wird durch niedrige Entwicklungs- und Betriebskosten begründet, und dass für kleine Problemgrössen möglichst wenig Overhead, beispielsweise in Form von Kommunikationsverlusten, generiert werden soll.

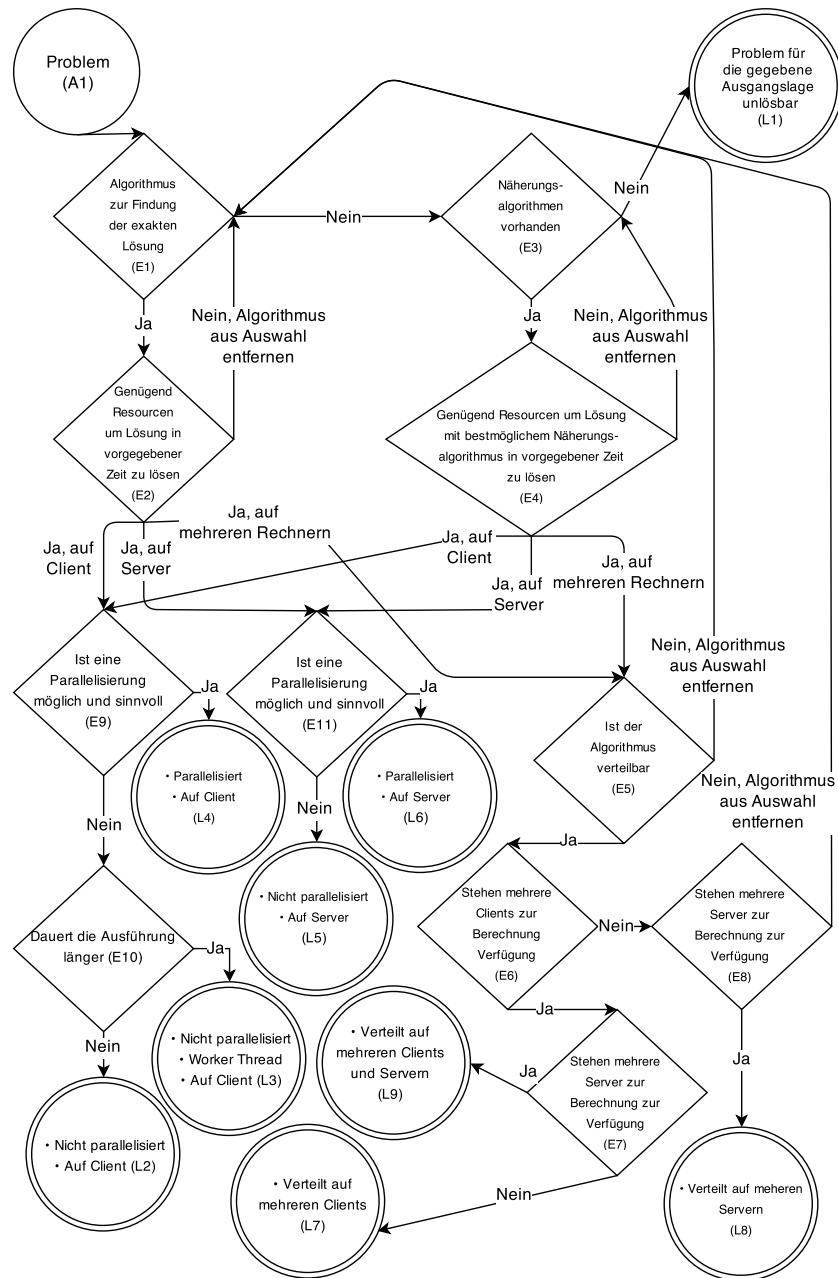


Abb. 11.3: Entscheidungsdiagramm (K2) zur Ermittlung der Architektur zur Lösung von NP-vollständigen Problemen in Webapplikationen (Quelle: Eigene Darstellung)

Ablauf des Entscheidungsdiagramms in Abb. 11.3

Vom Ausgangspunkt A1 führt der Entscheidungsbaum zum ersten Entscheidungsknoten E1. In diesem Knoten muss entschieden werden, ob für das vorliegende Problem Algorithmen zur Ermittlung der exakten Lösung existieren. Literaturrecherchen sind für diese Entscheidung unerlässlich und können gegebenenfalls mit Erforschung neuer Algorithmen ergänzt werden. Für die meisten Probleme wird mindestens ein exakter Algorithmus vorliegen, da mittels Brute-Force-Algorithmus die Lösung bestimmt werden kann. Eine Ausnahme bildet beispielsweise das Suchproblem für die Berechnung der

Zahl π , da es sich dabei um eine nicht endliche Zahl handelt.

Falls ein exakter Algorithmus vorliegt, muss im Knoten E2 evaluiert werden, ob die Lösung des Problems mit den gegebenen Ressourcen möglich ist. Vorab soll geprüft werden, ob genügend personelle und monetäre Ressourcen sowie Fachwissen für die Implementation des Algorithmus vorhanden sind. Eine Experteneinschätzung soll zudem zu entscheiden helfen, auf welchen Rechnern genügend Rechenleistung vorhanden ist, um die Lösung in der vorgegebenen Zeit zu berechnen. Zur Auswahl stehen "Client", "Server", "auf mehreren Rechnern" und "Nein". Falls nicht genügend Ressourcen verfügbar sind, wird dieser Algorithmus aus der Auswahl der exakten Algorithmen entfernt. Dieser Vorgang wird wiederholt, bis für einen Algorithmus genügend Ressourcen verfügbar sind oder keine exakten Algorithmen mehr in der Auswahl vorhanden sind. Falls kein exakter Algorithmus übrig bleibt, erfolgt dasselbe Vorgehen im Entscheidungsknoten E3, um das Problem mittels eines Näherungsalgorithmus zu lösen. Sind genügend Ressourcen auf einem Client oder einem Server vorhanden, muss geprüft werden, ob eine Parallelisierung möglich und auch sinnvoll ist. Es soll geprüft werden, ob sich der Mehraufwand zur Entwicklung der parallelisierten Lösung lohnt und ob diese den gewünschten Performancegewinn bringt. Ist die Parallelisierung auf einem Client nicht möglich oder nicht sinnvoll (E9), muss evaluiert werden (E10), ob eine Ausführung der Berechnung in einem Workerthread (L3) geeigneter ist als die Ausführung der gesamten Applikation und Berechnung in nur einem Thread (L2). Das würde bedeuten, dass die Applikation in zwei *Threads* ablaufen würde. Die Applikation mit dem *GUI* und der Programmlogik in einem Thread und die Berechnung des *TSP* in einem zweiten Thread. Sind nur auf einem Server genügend Ressourcen vorhanden und die Parallelisierung nicht sinnvoll oder möglich (E11), wird die Nicht-Parallelisierte-Lösung (L5) auf einem Server vorgeschlagen.

Falls für einen Algorithmus nur auf mehreren Rechnern zusammen genügend Ressourcen bereitstehen, muss in E5 entschieden werden, ob der Algorithmus auch verteilbar ist. Nicht verteilbare Algorithmen werden dann wieder aus der Auswahl entfernt und der Entscheidungsweg führt zum Knoten E1 zurück. Andernfalls wird in E6 geprüft, ob genügend Clients für eine erfolgreiche Verteilung zur Verfügung stehen. Falls nicht genügend Clients vorhanden sind wird in E8 noch geprüft, ob genügend Server für eine Verteilung auf mehreren Servern verfügbar sind (L8) oder ob eine Mischform zwischen mehreren Servern und Clients umsetzbar ist (L9).

12 Anwendung auf das TSP

Das in Kapitel 11 erarbeitete Konzept zur Ermittlung der Architektur zur Lösung von *NP-vollständigen* Problemen im Webumfeld wird in diesem Kapitel auf das *TSP* angewendet. Durch die Erkenntnisse in diesem Kapitel wird danach das resultierende Konzept erarbeitet und in Kapitel 15 dargestellt. Bevor die einzelnen Konzepte ausgearbeitet werden, sollen *POCs* die Machbarkeit prüfen, damit technologische und konzeptionelle Probleme vorab eliminiert werden können. Damit die erarbeiteten Konzepte vergleichbar sind, wird im Abschnitt 12.2 ein Algorithmus ausgewählt, welcher allen Anforderungen aus der Anforderungsanalyse gerecht wird.

12.1 Vorversuche

Damit realistische Konzepte für die Umsetzung erarbeitet werden können, werden einige Vorversuche durchgeführt. Der Zugriff auf die *POCs* ist in Anhang A beschrieben, wobei die einzelnen *POCs* innerhalb der nachfolgenden Kapitel eingeführt und genauer beschrieben werden. Die Konzeptentwicklung und die Erstellung der Vorversuche finden in einem iterativen Prozess statt, da die Ideen und Erkenntnisse jeweils voneinander abhängig sind und wiederum den nächsten Schritt beeinflussen.

12.1.1 Parallelle Berechnung im Web

Zuerst sollen einige Versuche und Tests betrachtet werden, um die Technologie der *Webworker*, welche das Multithreading im Browser ermöglichen, genauer zu prüfen und deren Limitierungen auszuloten.

12.1.1.1 Verschachtelte *Webworker*

Ein Algorithmus kann auf verschiedenste Arten parallelisiert implementiert werden. Eine Variante erfordert eine verschachtelte Erzeugung von *Threads* für die Generierung und Berechnung der Knoten. Das heisst, innerhalb eines *Threads* werden wiederum neue *Threads* erzeugt. Recherchen und der Versuch POC-1d ergaben, dass kein Browser, ausser *Firefox*, verschachtelte (englisch: nested) *Webworker* unterstützt [25]. Dies schränkt die Auswahl und die Implementierungsmöglichkeiten der im Web einsetzbaren Algorithmen entscheidend ein. Der Test POC-1e zeigt, dass aber mehrere *Webworker* gleichzeitig gestartet werden können.

12.1.1.2 Performanz der Berechnung

Der wichtigste Aspekt der Lösung von *NP-vollständigen* Problemen ist die Geschwindigkeit der Berechnung. Um diese Geschwindigkeit und das Verbesserungspotential mit dem Einsatz von *Webworker* aufzuzeigen, muss zuerst festgestellt werden, wie gross die Zusatzkosten der Threaderzeugung, der Kommunikation zwischen den *Threads* und der Threadbeendigung sind. Ein Thread wird in JavaScript in einem *Webworker* abgebildet. Das Beispiel POC-1b vergleicht die Performanz der Ausführung von JavaScript Code im Browser mit und ohne *Webworker*. Die Abb. 12.1 zeigt die grafische Auswertung der einzelnen Ausführungsarten. "Normal" bezeichnet die Ausführung direkt im Browser und ohne Verwendung von *Webworker* (*Threads*). "Webworker (file)" steht für die Ausführung des Codes aus einer anderen JavaScript Datei in einem *Webworker* und bei "Webworker (embedded)" wurde der Code für den *Webworker* über einen JavaScript Blob Container eingebunden. Der Blob Container stellt eine virtuelle JavaScript Datei dar, worin der Code ausgeführt werden kann, welcher sich eigentlich in derselben Datei wie der Hauptthread befindet. Dadurch muss keine externe Datei geladen werden, wodurch die Lade-Latenz der externen Datei zu eliminieren versucht wird. Die Zeitdauer "Kommunikation (file)" stellt dar, wie lange eine Nachricht benötigt, um vom Hauptthread an den *Webworker* und wieder zurück gesendet zu werden. Bei "Kommunikation (embedded)" wird dasselbe Vorgehen gemessen, jedoch mit Ausführung des *Webworkers* in einem bereits beschriebenen Blob Container.

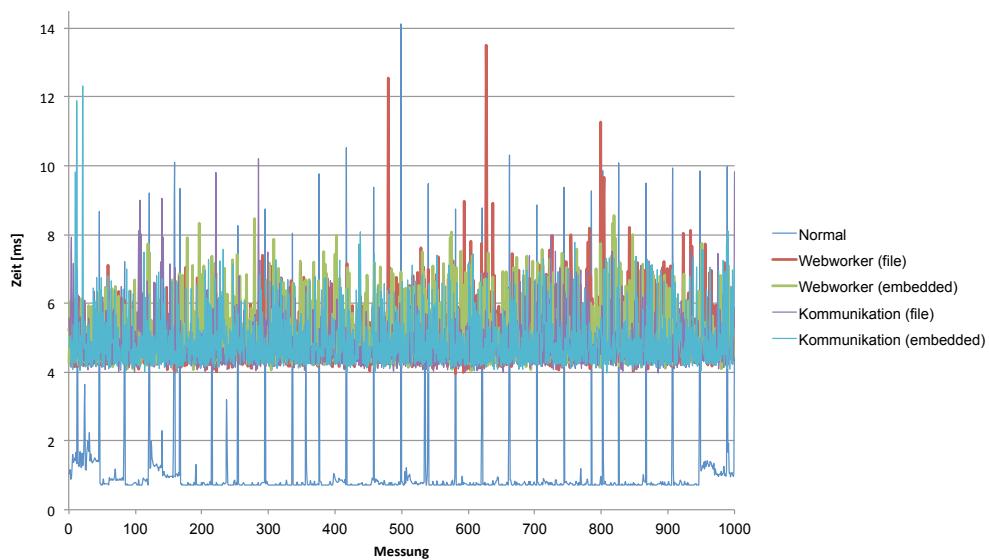


Abb. 12.1: Messungen der Performanz von *Webworker* (gemessen auf einem MacBook, gemäss Anhang, mit POC-1c) (Quelle: Eigene Darstellung, mit Excel)

Die Auswertungen der Daten aus Abb. 12.1 in Tab. 12.1 zeigen, dass die *Webworker*-Erzeugung und *Webworker*-Beendigung, inklusive einer Nachricht an den *Webworker* und dessen Antwort, im Durchschnitt eine Verzögerung von 5.101 ms ergeben und

die Ausführung direkt aus einem Blob Container geringfügig langsamer ist. Zusätzlich zum kleinen Geschwindigkeitsvorteil bietet die externe Datei den Vorteil, dass der Code besser strukturiert werden kann. Der Geschwindigkeitsvorteil deutet auf ein gutes Caching des Browsers hin und die externe Datei somit nicht jedes Mal neu geladen wird. Aus der Abbildung geht ebenfalls hervor, dass die Erzeugung und Beendigung des *Webworkers* gegenüber der Kommunikation nur eine untergeordnete Rolle spielt, da sozusagen der gesamte Zeitaufwand für den Nachrichtenaustausch benötigt wird.

Typ	Normal	<i>Webworker</i> (file)	<i>Webworker</i> (embedded)	Kommuni-kation (file)	Kommuni-kation (embedded)
Median [ms]	0.739	4.791	4.915	4.549	4.593
Durchschnitt [ms]	1.087	5.101	5.185	4.852	4.971
Min [ms]	0.716	3.958	4.079	3.984	3.99
Max [ms]	14.127	13.507	8.555	10.206	12.323

Tab. 12.1: Auswertung der *Webworker* Performance-Messungen (gemessen auf einem MacBook, gemäss Anhang, mit POC-1c) (Quelle: Eigene Darstellung)

Bei den Peaks in den Messungen kann von Geschwindigkeitseinbussen aufgrund von anderen Prozessen ausgegangen werden, da die Peaks sowohl bei der Ausführung im Modus "Normal", als auch bei den *Webworker*n aufgetreten sind.

12.1.1.3 Probleme

Bei der Implementation der POC-1a bis POC-1e zeigte sich, dass das *Debuggen* bei *Webworker*n schwierig ist. Beispielsweise funktioniert der Befehl *console.log()*, womit Informationen auf der Console im Browser ausgegeben werden können, nicht aus einem *Webworker* heraus. Die Ausnahme bildet *Firefox*, der den *console.log()* Befehl zum Hauptthread umleitet. Jedoch kann beim *Firefox* die Methode *console.log()* nicht überschrieben werden und führt zu folgendem Fehler: "Die Webkonsolen-Logging-API (*console.log*, *console.info*, *console.warn*, *console.error*) wurde von einem Skript auf dieser Seite deaktiviert" (siehe POC-1a). Das bedeutet, dass mittels einer Browserweiche unterschieden werden muss, ob es sich um *Firefox* oder einen anderen Browser handelt. Bei den anderen Browsern muss der *concole.log()* Befehl manuell umgeleitet werden, damit die Meldung im Hauptthread in der Console angezeigt wird. Die Umleitung erfolgt über die *Webworker*-Nachrichten, was dazu führt, dass die Reihenfolge der Nachrichten nicht der effektiven Reihenfolge der Ausführung des *console.log()* Befehls entspricht, da die *Webworker* parallel und unabhängig voneinander ablaufen und zudem die Nachrichten asynchron versendet werden.

12.1.2 Verteilte Berechnung unter Clients im Web

Um ein verteiltes System im Webumfeld zu realisieren muss die Kommunikation zwischen den einzelnen Komponenten des Systems gewährleistet sein. Die Kommunikation ist aber langsam und sollte daher minimiert werden. Falls die Kommunikation unvermeidbar ist, sollten möglichst kleine Datenmengen verschickt werden. Der Fokus dieses Abschnitts richtet sich auf die verteilte Berechnung unter Clients, da dadurch keine zusätzlichen Lizenzkosten (RE-R2-1) entstehen.

Um gemeinsame Ressourcen zu realisieren, kann ein Publisher-Subscriber-Pattern mit Kanälen verwendet werden. Alle beteiligten Rechner (sowohl Clients als auch Server sind möglich) schreiben sich hierfür in einem gemeinsamen Kanal ein. Dadurch kann eine Änderung an alle gesendet werden. Um aber die im vorangehenden Abschnitt 12.1.2.1 beschriebenen Probleme der gemeinsamen Datenverwaltung im Web zu umgehen, darf eine Änderung lediglich vom Hauptrechner aus geschickt werden. Daraus ergibt sich folgendes Pattern:

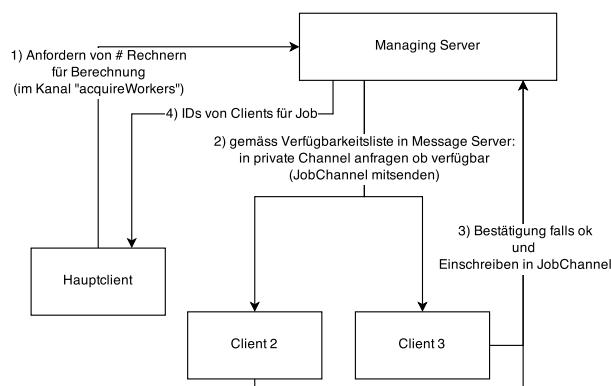


Abb. 12.2: Kommunikationsdiagramm des Konzepts für die gemeinsame Datennutzung bei verteilten Systemen mit dem Publisher-Subscriber-Pattern (Quelle: Eigene Darstellung)

Ausgangslage ist ein Managing-Server, welcher das Messagehandling mit verschiedenen Kanälen (sogenannte Channels) unterstützt und die Jobvergabe koordiniert. Zuerst existiert lediglich ein Channel "acquireWorkers". Dort schreiben alle sich am System anmeldenden Rechner ein. Falls ein Rechner Ressourcen für eine Berechnung benötigt (im Folgenden als Hauptclient bezeichnet), sendet dieser im Kanal "acquireWorkers" eine Anfrage mit dem auszuführenden Job und den Parametern für die einzelnen Clients. Der Managing-Server verteilt dann den Job auf die vorhandenen Clients und schickt die von den Clients während der Berechnung erhaltenen Daten zurück an den Hauptclient. Der Hauptclient wertet die Daten aus und verteilt diese anschliessend wiederum über den Managing-Server in einem gemeinsamen Job-Kanal an alle anderen Clients, welche an der Berechnung beteiligt sind. Das Beispiel POC-2b zeigt diese

Funktionalität unter Verwendung von socket.io, welches bidirektionale, auf Nachrichten basierte Kommunikation in Echtzeit ermöglicht. Socket.io verwendet dafür Websockets (Abschnitt 5.1.1.2), falls der Browser dies unterstützt. Ansonsten wird Long-Polling eingesetzt, wobei wiederholt AJAX-Anfragen gemacht werden um Nachrichten abzufragen (Abschnitt 5.1.1.1). Faye, ein anderes Kommunikationsframework, kann nicht verwendet werden, wenn Nachrichten aus einem *Webworker* heraus verschickt werden. Der Grund dafür ist, dass Faye mit den document und window Objekt arbeitet, welche in *Webworker* nicht verfügbar sind (POC-2a).

Um den Einsatz eines verteilten Client-Systems zu beurteilen, wird zuerst die Kommunikationsgeschwindigkeit im POC-2c gemessen und in Abb. 12.3 dargestellt. Die dafür verwendete Versuchskonfiguration sieht folgendermassen aus:

- Es sind zwei Browserfenster in Chrome auf einem MacBook geöffnet, welche auf der Seite des POC-2b sind. Genauere Details über die Verbindungsgeschwindigkeiten können dem Anhang C "Verwendete Soft- und Hardware" entnommen werden.
- In einem Browserfenster wird die Anzahl Versuche eingestellt (Textbox) und der Versuch gestartet.

Der erste Messwert "Job" stellt die gesamte Dauer von der Erzeugung eines Jobs bis und mit der Beendigung des Jobs dar. Dieser Zusatzaufwand muss für jeden Client, der für die Berechnung eingesetzt wird, berücksichtigt werden. Zusätzlich muss noch der Kommunikationsaufwand für die gemeinsame Verwendung der Daten eingerechnet werden. Dieser Aufwand wird im Messwert "Kommunikation" visualisiert. Gemessen wurde dabei die Dauer der Übermittlung von einem JavaScript Objekt mit der Versandzeit von einem Client an den Haupthread (über den Managing-Server) und zurück an den Client. Da der Managing-Server die gemeinsamen Daten direkt an die anderen Clients und an den Hauptclient verteilt, kann davon ausgegangen werden, dass die gemessene Dauer die Zeit für das Verteilen der Daten an alle Clients gilt (siehe auch Abb. 12.4 für die gemeinsame Datenverwendung in verteilten und parallelen Berechnungen). Da für die Messungen lediglich ein Objekt mit der Startzeit der Messung versendet wurde, muss beim Versenden von grösseren Datenmengen mit einer langsameren Kommunikation gerechnet werden.

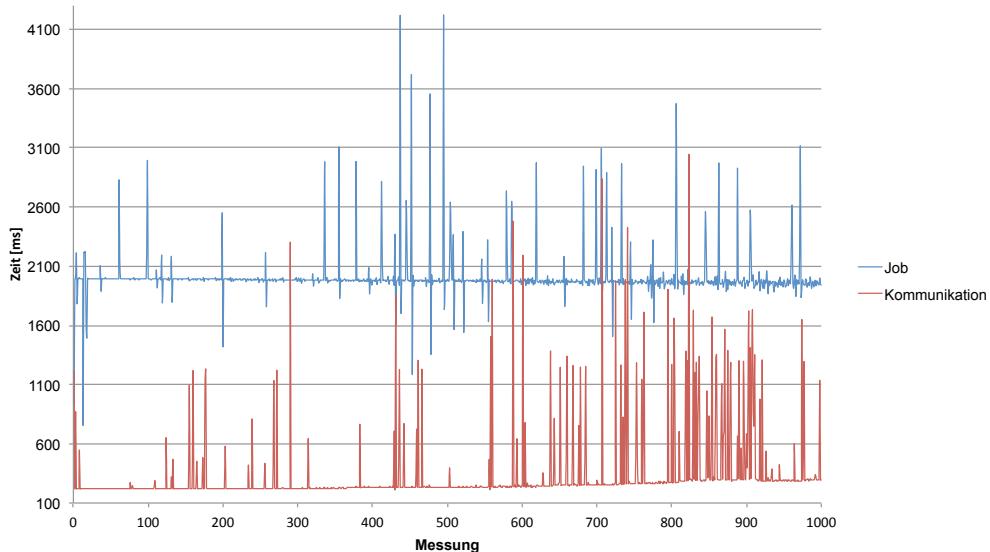


Abb. 12.3: Performance-Messungen der Nachrichtenübertragung bei verteilten Clients aus POC-2c (Quelle: Eigene Darstellung, mit Excel)

Typ	Job	Kommunikation
Median [ms]	1981	236
Durchschnitt [ms]	2003	339
Min [ms]	755	217
Max [ms]	4222	3045

Tab. 12.2: Auswertung der Performance-Messungen der Nachrichtenübertragung bei verteilten Clients aus POC-2c (Quelle: Eigene Darstellung)

Die Resultate in Tab. 12.2 zeigen, dass für jeden Client, welcher für die Berechnung eingesetzt wird, ein Zusatzaufwand von durchschnittlich 2003 Millisekunden aufgewendet werden muss, um die gemeinsamen Kommunikationskanäle zu erstellen und die Initialisierung des Clients vorzunehmen. Um später die gemeinsamen Daten an alle Clients zu schicken wird bei jedem Verteilen der gemeinsam genutzten Daten erneuter Zeitaufwand von durchschnittlich 339 Millisekunden nötig.

12.1.2.1 Gemeinsam im Web verwendete Daten unter Clients

Um Konflikte und Locks zu verhindern werden in anderen Programmiersprachen Transaktionen oder "synchronized" Methoden und Variablen eingesetzt. JavaScript bietet keinerlei solche Mechanismen, da bis zur Einführung von *Webworkers* nur Single-Thread Applikationen entwickelt werden konnten. Dieses Problem kann nur dadurch umgangen werden, wenn ein Browser in einem Thread die Master-Funktion übernimmt und alle anderen Worker und Browser als Slaves fungieren. Somit kann sichergestellt werden, dass auf die gemeinsam verwendeten Daten lediglich in diesem Thread zugegriffen und diese

dort bearbeitet werden. Auch eintreffende Messages werden sequentiell abgearbeitet, obschon diese asynchron eintreffen.

12.2 Algorithmenwahl (R3)

Damit die verschiedenen Architekturvarianten des Entscheidungskonzepts untereinander verglichen werden können, soll ein Algorithmus gefunden werden, welcher in allen Architekturvarianten implementiert werden kann. Daher erfolgt die Algorithmenwahl für die Implementation der *POCs* aufgrund folgender Parametern, welche aus der Anforderungsanalyse in Kapitel 9 und dem Entscheidungskonzept in Kapitel 11 abgeleitet sind:

Der Algorithmus

- muss die exakte Lösung berechnen und nicht nur eine Näherungslösung (wegen RE-Q3-1 und RE-Q3-6).
- muss das *ATSP* lösen können (wegen RE-F2,3,4-1).
- muss parallelisierbar sein (wegen Konzept in Abb. 11.3).
- muss verteilbar sein (wegen Konzept in Abb. 11.3). Dies ist automatisch erfüllt, falls der Algorithmus parallelisierbar ist.
- muss mit einer Distanzmatrix arbeiten können (wegen RE-Q3-4 und RE-Q3-8).
- sollte eine Statusmeldung der Berechnung ermöglichen (wegen RE-Q1,2,3-2).

Name	Exakt-heit	ATSP	Parallelisierbarkeit	Distanz-matrix	Status-meldung	Komplexität	Quellen
Branch-and-Bound	Exakt	Ja	Ja	Ja	Ja	$\mathcal{O}(n!)$	[81], [82], [83], [84]
Brute-Force	Exakt	Ja	Nein	Ja	Ja	$\mathcal{O}(n!)$	-
Held-Karp	Exakt	Ja	nichts dazu gefunden	Ja	Ja	$\mathcal{O}(n^2 * 2^n)$	[68]
Nearest-Neighbour	Näherung	Ja	Ja	Ja	Ja	$\mathcal{O}(n^2)$	[85], [21]
2-Opt, k-Opt, v-Opt	Näherung	Ja	Ja	Ja	Nein	$\mathcal{O}(n^2)$	[78]
Ant-Colony-Optimization	Näherung	Ja	Ja	Ja	Nein	-	[42], [50]

Tab. 12.3: Vergleich einiger Algorithmen zur Lösung des *TSP* (Quelle: Eigene Darstellung)

Die Auswahl fällt auf den Branch-and-Bound-Algorithmus, da dieser die obig genannten Punkte erfüllt (Tab. 12.3) und auch bei jedem Berechnungsschritt a priori bekannt

ist, wie viele neue Knoten auf der nächsten Stufe generiert werden und zu berechnen sind. Dies erleichtert auch die Parallelisierung und Verteilung des Algorithmus, da eine bestimmte Anzahl an *Threads / Clients* zur Berechnung akquiriert werden können. Da *Webworker* keine Verschachtelung unterstützen (Abschnitt 12.1.1.1), muss eine Implementation mit einer flachen Verteilungsstruktur gefunden werden. Der Entscheidungsbaum weist die maximale Anzahl Knoten auf der Stufe 2 auf, weshalb sich eine Verteilung dieser Knoten auf verschiedene *Threads*, respektive Geräte, empfiehlt. Die Suche der Lösung erfolgt über eine Breitensuche, da die verschiedenen *Threads / Rechner* gleichzeitig die einzelnen Knoten auswerten und dann jeweils die nächsttieferen Ebene berechnen. Dies ermöglicht, dass lediglich der Lower Bound und die Reihenfolge der *WP* dieser Zwischenlösung unter den einzelnen *Threads / Clients* verteilt werden muss und so die Grösse der gesendeten Daten (Kommunikation) klein gehalten wird.

Um den Zusatzaufwand der Kommunikation möglichst gering zu halten, wird auf eine gemeinsame Queue (Warteschlange) verzichtet. Denn bei einer gemeinsamen Queue würden alle zu berechnenden Knoten in diese Queue geschrieben, welche bei jeder Änderung an alle *Threads / Clients* verteilt werden müssten. Neben Mehrfachberechnung derselben Knoten, wegen der nicht aktuellen Daten aufgrund der Übertragungsverzögerung, würde die grosse Datenmenge der Queue enormen Kommunikationsaufwand bedeuten. Stattdessen wird jeweils eine lokale Queue pro Knoten gespeichert und nur der gemeinsame Lower Bound und die zugehörige Lösung als Array von Integern in den gemeinsamen Daten gespeichert. Die gemeinsamen Daten befinden sich dabei auf dem Hauptclient und werden bei Änderungen an die anderen Clients geschickt. Der Grund dafür wurde bereits in Abschnitt 12.1.2.1 erläutert.

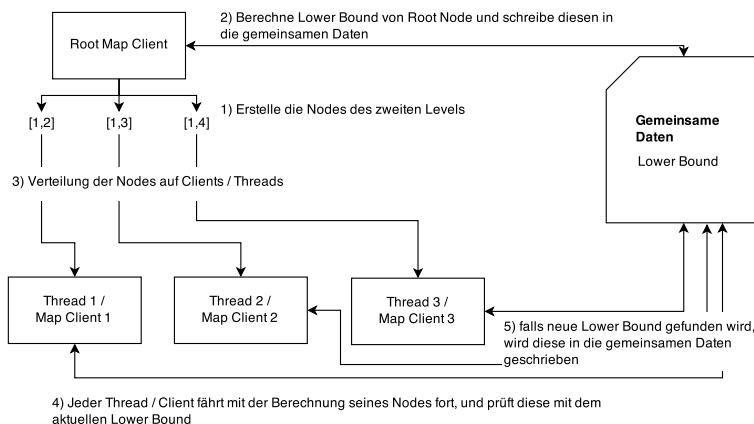


Abb. 12.4: Parallele/Verteilte Berechnung des TSP mit Branch-and-Bound mit $n = 4$
(Quelle: Eigene Darstellung)

Der Ablauf der parallelen Berechnung gleicht dem Ablauf der verteilten Berechnung stark. Daher sind beide Berechnungsarten gemeinsam in Abb. 12.4 dargestellt: Nachdem

der Hauptthread / Hauptclient die Knoten der zweiten Stufe generiert hat (1), schreibt er den berechneten Lower Bound und die *WP* Reihenfolge von der Ausgangsroute in die gemeinsam zugreifbaren Daten (2). Jeder dieser Knoten wird auf einen Thread / Client verteilt (3), welcher den zugeteilten Ast berechnet (4): Jeder Thread / Client generiert bei Erhalt des Ausgangsknoten die Knoten der nächst tiefer liegenden Ebene des Baums und speichert diese aufsteigend geordnet nach dem Lower Bound in einer Queue (Warteschlange), falls der Lower Bound des berechneten Knoten kleiner als derjenige der derzeitigen Zwischenlösung in den gemeinsamen Daten ist. Dann entfernt und berechnet der Thread / Client den ersten Knoten der Queue und erstellt wiederum die nächste Ebene für diesen Knoten und speichert die neu generierten Knoten in der Queue, falls der Lower Bound des Knoten kleiner als die derzeitige Zwischenlösung in den gemeinsamen Daten ist. Dieser Vorgang wiederholt sich bis alle Knoten des Baums erstellt und durchlaufen sind. Falls also ein Knoten bereits einen grösseren Lower Bound als die bisher beste gefundene Zwischenlösung hat, wird der Knoten verworfen und der Ast dieses Knotens nicht weiter verfolgt. Dadurch können viele Berechnungen eingespart werden. Dennoch erfolgen mehr Berechnungen als bei der Ausführung in einem Thread / auf nur einem Client, da die Benachrichtigung über einen neuen Lower Bound erst verspätet eintrifft und einige Knoten in der Zwischenzeit berechnet wurden bevor diese verworfen worden waren.

Sobald ein Blatt (ein Knoten der untersten Ebene des Baumes) mit kleinerem Lower Bound als der bestehende Lower Bound in den gemeinsamen Daten gefunden wird, wird die neue Zwischenlösung in die gemeinsamen Daten geschrieben (5):

Bei der parallelen Berechnung befinden sich die gemeinsamen Daten im Hauptthread. Die *Webworker* melden neue Zwischenlösungen über Nachrichten an den Hauptthread, welcher diese wiederum an alle anderen *Threads* schickt.

Das Konzept für die gemeinsame Datenhaltung bei verteilten Systemen unter Clients gestaltet sich etwas schwieriger. Das Grundkonzept der gemeinsamen Datennutzung wurde bereits in Abschnitt 12.1.2 beschrieben. Die gemeinsamen Daten befinden sich auf dem Hauptclient. Sobald ein Client eine neue Zwischenlösung findet, wird diese dem Managing-Server geschickt (siehe auch Abb. 12.2). Dieser wiederum sendet die Daten an den Hauptclient weiter, welcher den Job aufgegeben hat. Der Hauptclient speichert die neue Zwischenlösung, falls nicht in der Zwischenzeit bereits eine bessere Zwischenlösung eingegangen und gespeichert wurde. Falls die erhaltene Zwischenlösung besser ist als die bisherige, wird die neue Zwischenlösung über den Managing-Server an alle sich im Job-Channel befindenden Clients geschickt. Jeder Client nimmt wiederum die neue Zwischenlösung entgegen und speichert diese, falls der Client nicht bereits eine bessere Zwischenlösung gefunden hat.

12.3 Lösungskonzepte für das TSP im Web

Im folgenden Kapitel wird das in Kapitel 11 erarbeitete Gesamtkonzept für die Integration von *NP-vollständigen* Problemen in Webapplikationen auf das *TSP* angewendet und dadurch verifiziert oder widerlegt.

Im Abschnitt 12.3.1 werden mögliche Implementationsvarianten ausgearbeitet und verglichen. Ein anderer Ansatz stellt die dynamische Algorithmenwahl dar, welche in Abschnitt 12.3.2 erwähnt wird.

12.3.1 Technologie-/Architekturkonzepte (R2)

Nach der Auswahl des Algorithmus im Abschnitt 12.2 und den notwendigen Services in der Anforderungsanalyse in Abschnitt 9.8.2 und Abschnitt 9.8.3, werden nachfolgend Technologiekonzepte und deren möglichen Architekturvarianten zur Berechnung des *TSP* im Webumfeld ausgearbeitet. Alle Konzepte setzen einen Service für die Routenberechnung zwischen zwei *WP*, respektive Distanzmatrizen für mehrere *WP* und einen Service für das Kartenmaterial voraus. Für die Darstellung der Karte mit den Markern und der Route wird im Mindesten ein Webclient, auf welchem die Komponente "Browserapplikation" betrieben wird, benötigt. Diese Komponente greift auf einen Karten- und Routingservice zu. Daraus ergeben sich die Einflussfaktoren "Berechnung der Distanzen" und "Kommunikation zum Service". Da diese Faktoren bei allen Konzepten Einfluss nehmen, werden sie in den Konzepten bewusst ausgeklammert. Eine weiterer essentieller Bestandteil aller Architekturen ist die Komponente "Webserver", welche die Applikation zur Verfügung stellt.

Der Fokus liegt auf denjenigen Konzepten, welche mehrheitlich Clients einsetzen, da dadurch weniger Server und Services benötigt werden und die Lizenzkosten tief gehalten werden können (RE-R2-1). Zudem generiert eine steigende Kundenzahl dadurch keine Zusatzkosten für die Bachmann Support GmbH (RE-Q3-10).

Die gesamte Zeit für das Lösen des *TSP* im Web mit n *WP* setzt sich wie folgt zusammen:

$$T(n) = T_{geoc}(n) + 2 * T_{coordT}(n) + T_{costMatrix}(n) + T_{TSP}(n) + T_{vMap}(n) + T_O(n)$$

T_{geoc} : Zeit für das Geocodieren der erfassten Adressen

T_{coordT} : Zeit für die Koordinatentransformation

$T_{costMatrix}$: Zeit für die Berechnung und das Laden der Kostenmatrix

T_{TSP} : Zeit für die Berechnung des *TSP*

T_{vMap} : Zeit für die Visualisierung der *WP* und Route auf der Karte

T_O : Zusätzlicher Zeit-Overhead je nach Architekturkonzept

Einflussfaktoren:

R : Rechnerleistung

C : Kommunikationsgeschwindigkeit zu den Servern

Die Gesamtzeit $T(n)$ gilt es zu minimieren. Da die Koordinatentransformation eine nicht ressourcenintensive Rechenoperation ist, kann sie vernachlässigt werden. Auch die Visualisierung der WP ist nicht zeitaufwändig, was in POC-3b gezeigt wird und somit auch nicht berücksichtigt werden muss. Nachfolgend wird in Klammern bei jedem Konzept angegeben, welche Lösung dieses Konzept aus dem Entscheidungsdiagramm aus Abb. 11.3 darstellt:

12.3.1.1 Clientseitige Konzepte

Für die clientseitigen Konzepte wird lediglich ein normaler Webserver vorausgesetzt, welcher die Ressourcen (beispielsweise *HTML*, *CSS* und JavaScript Dateien) den Browsern bereitstellt. Der *Map Client* bezieht die Daten für die Berechnung (Distanzmatrix) direkt vom Routingserver und löst das *TSP* direkt im Browser. Ein grosser Vorteil dieser Architekturvarianten besteht darin, dass keine zusätzlichen Kommunikationsverluste anfallen. Die gesamte Berechnung erfolgt auf dem Webclient (im Browser), wo auch die Ein- und Ausgabe erfolgt. Es kann sowohl vor- als auch nachteilig sein, dass lediglich die vorhandenen Ressourcen des Webclients zur Verfügung stehen. Aus Sicht des Applikationsbetreibers können Ressourcen eingespart werden, da keine zusätzlichen Server in Betrieb genommen werden müssen. Aus Sicht der Benutzer hingegen kann es bei schwacher Rechnerleistung, wie dies zum Beispiel auf älteren Smartphones oder alten Computern der Fall ist, zu erheblichen Einschränkungen führen. Falls aber ein genügend schneller Client-Rechner vorhanden ist, kann es natürlich wiederum als Vorteil gesehen werden, dass keine Kommunikation zum Server nötig ist.

Konzept 1: Sequenzielle, clientseitige Berechnung (L2)

Die Lösung des *TSP* erfolgt vollständig im Browser des Benutzers, welcher die Route optimieren will. Funktionen für die Darstellung des GUI und die Berechnung erfolgen im selben Thread, wie dies bei herkömmlichen JavaScript Applikationen der Fall ist.

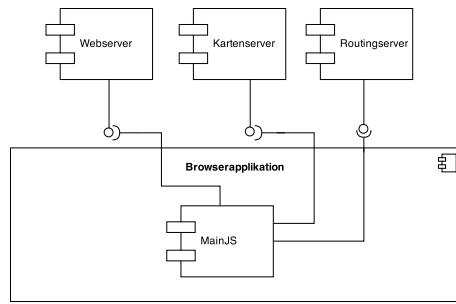


Abb. 12.5: Bausteinsicht des Konzept 1 (*Quelle: Eigene Darstellung*)

Neben den für alle Konzepte benötigten Komponenten "Kartenserver" und "Routingserver" wird nur noch eine JavaScript Komponente innerhalb der Browserapplikation benötigt. Diese Applikationskomponente ist sowohl für das *GUI* und die Routenoptimierung zuständig. Daher wird während der gesamten Berechnung das *GUI* blockiert und falls die Berechnung zu lange dauert, stürzt der Browser ab. Der grosse Vorteil stellt der nicht vorhandene Overhead dar. Die Berechnung erfolgt ohne weitere Verzögerung oder Kommunikationsverluste:

$$T_O(n) = 0$$

Konzept 2: Sequentielle, clientseitige Berechnung in separatem Thread (L3)

Die Lösung des *TSP* erfolgt vollständig im Browser des Benutzers, welcher die Route optimieren will. Es wird neben dem *GUI* (Hauptthread) ein Arbeitsthread erstellt, welcher das *GUI* nicht blockiert.

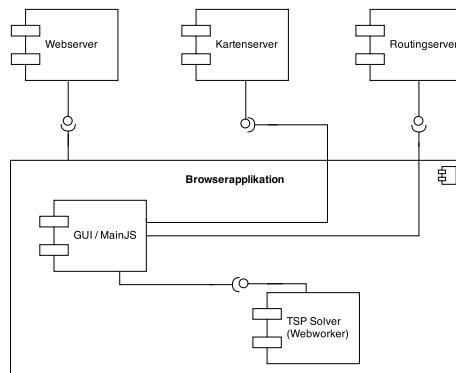


Abb. 12.6: Bausteinsicht des Konzepts 2 (*Quelle: Eigene Darstellung*)

Nachdem im Hauptthread die Distanzmatrix geladen wurde, wird die Berechnung an den *TSP* Solver innerhalb eines *Webworkers* übergeben. Dieser löst das *NP-vollständige*

Problem und sendet in einer Nachricht die optimierte Route zurück an den Hauptthread. Für diese Architekturvariante ergibt sich der folgende zusätzliche Zeitaufwand:

$$T_{Overhead}(n) = T_{ThreadCreation} + T_{ThreadRemoval} + 2 * T_{Communication}$$

Danach wird die neue Route mittels dem GUI / MainJS - Thread im *Map Client* visualisiert.

Konzept 3: Clientseitige Berechnung mit mehreren Threads (L4)

Die Lösung des *TSP* erfolgt auch in diesem Konzept vollständig im Browser des Benutzers, welcher die Route optimieren will.

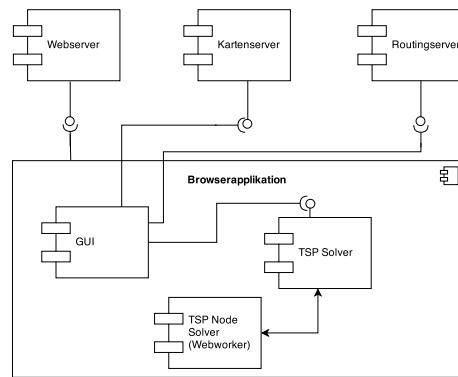


Abb. 12.7: Bausteinsicht des Konzepts 3 (Quelle: Eigene Darstellung)

Für die Berechnung werden mehrere Threads erzeugt, in welchen die Komponente "TSP Node Solver", ausgeführt wird. Die Parallelisierung erfolgt mit dem Branch-and-Bound-Algorithmus, wie dies in Abschnitt 12.2 beschrieben ist. Damit diese Architekturvariante eine Effizienzsteigerung bringt, muss der folgende Zusatzaufwand überwunden werden:

$$T_{Overhead}(n) = (n - 1) * (T_{ThreadCreation} + T_{ThreadRemoval}) + T_{Communication}$$

Aufgrund der gewählten Implementationsvariante des Branch-and-Bound-Algorithmus können neben dem Hauptthread maximal $(n - 1)$ zusätzliche Threads erzeugt werden. Zusätzlich entstehen Kommunikationsverluste $T_{Communication}$, da gefundene Lösungen im Zuge der gemeinsamen Daten an alle anderen Threads versendet werden müssen. Da aber die Threads parallel erzeugt werden, ist bei der absoluten Betrachtung des Zusatzaufwandes die Anzahl der Threads nicht relevant. Natürlich können mehrere Threads nicht exakt zur selben Zeit und exakt gleich schnell erzeugt werden. Dennoch kann die Formel vereinfacht wie folgt dargestellt werden:

$$T_{Overhead}(n) = T_{ThreadCreation} + T_{ThreadRemoval} + T_{Communication}$$

Die Anzahl der nötigen Verteilungen ist dabei abhängig von der gegebenen Distanzmatrix und kann daher nicht genau vorausgesagt werden.

12.3.1.2 Serverseitige Konzepte

Für die serverseitige Umsetzung existieren unzählige Implementationsmöglichkeiten. Nachdem die Anfrage von einem Webserver, beispielsweise über eine REST Schnittstelle entgegen genommen wird, können die Berechnungen in einer beliebigen Programmiersprache und auf unterschiedlichste Art (Design-/Architekturen) berechnet werden. Um die Auswahl eingrenzen und die Vergleichbarkeit zu den anderen Konzepten weitgehend beibehalten zu können, soll der Algorithmus zur Lösung auf dem Server ebenfalls in JavaScript implementiert werden. Als angenehmen Nebeneffekt ist die Wiederverwendbarkeit des Codes aus den clientseitigen Konzepten anzumerken.

Ein wichtiger Vorteil der serverseitigen Konzepte ist, dass die Rechenleistung des Servers bekannt ist. Jedoch führt das Zentralisieren auf einen Server dazu, dass bei mehreren gleichzeitigen Anfragen unter Umständen nicht genügend Rechenleistung vorhanden ist.

Konzept 4: Sequentielle, serverseitige Berechnung (L5)

Die Lösung des *TSP* erfolgt vollständig auf dem Webserver. Die Ausgangsdaten werden mittels einer *AJAX Anfrage* an den Server übermittelt, welcher das Problem in einem einzigen Thread und Prozess löst und das Ergebnis anschliessend zurück an die Browserkomponente schickt.

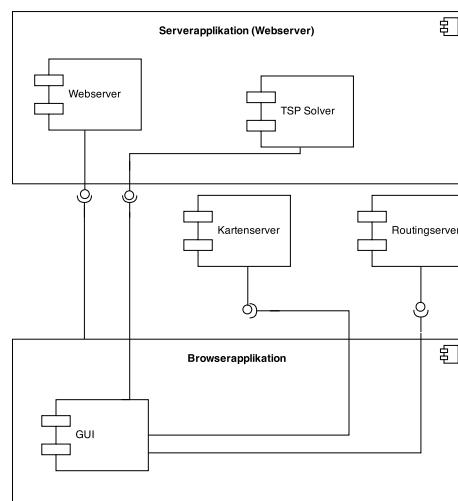


Abb. 12.8: Bausteinsicht des Konzepts 4 (Quelle: Eigene Darstellung)

Der gesamte Overhead stellt die Übermittlung der Ausgangslage des *TSP* und das Zurücksenden der berechneten Lösung dar:

$$T_{Overhead}(n) = 2 * T_{Transmition}$$

Konzept 5: Serverseitige Berechnung mit mehreren Threads (L6)

Dieses Konzept vereint die serverseitige Ausführung und die Parallelisierung des Algorithmus. Die Distanzmatrix wird mittels *AJAX*-Anfrage an die Serverapplikation geschickt, welche das Problem parallelisiert, ähnlich wie in Konzept 3, löst. Der Unterschied zum Konzept 3 ist, dass die Parallelisierung auf dem Server nicht durch *Webworker*, sondern in *Node.js* durch Threads realisiert wird. Der Ablauf der Parallelisierung ist im Abschnitt 12.2 zu finden.

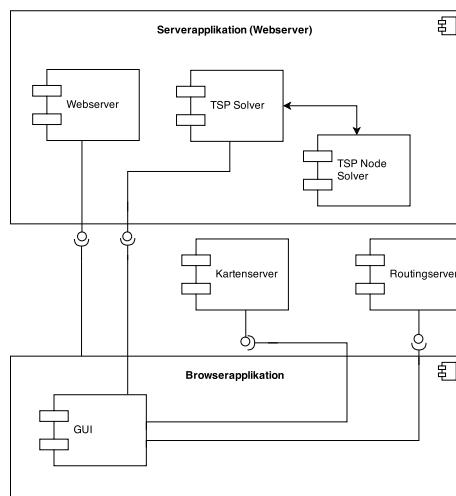


Abb. 12.9: Bausteinsicht des Konzepts 4 (*Quelle: Eigene Darstellung*)

Da dieses Konzept die Kombination von Konzept 3 und Konzept 4 darstellt, können auch deren Overheads addiert werden:

$$T_{Overhead}(n) = 2 * T_{Transmition} + (n - 1) * (T_{ThreadCreation} + T_{ThreadRemoval}) + T_{Communication}$$

Die Thread Erzeugung und Auflösung erfolgt parallel, weshalb der absolute Zusatzaufwand kleiner ausfällt:

$$T_{Overhead}(n) = 2 * T_{Transmition} + T_{ThreadCreation} + T_{ThreadRemoval} + T_{Communication}$$

12.3.1.3 Verteilte Konzepte

Die verteilten Systeme verfolgen die Idee, dass mehrere Rechner gemeinsam die nötigen Ressourcen aufbringen, damit das Problem effizient gelöst werden kann. Der Kommunikationsaufwand darf dabei aber nicht vernachlässigt werden, da vor allem das HTTP-Protokoll einen grossen Overhead hat.

Konzept 6: Verteiltes, clientseitiges System (L7)

Beim clientseitig verteilten Konzept werden die einzelnen Knoten für die Berechnung auf andere Clients ausgelagert, wie dies in Abschnitt 12.1.2 und Abschnitt 12.2 beschrieben ist. Damit dieses Konzept erfolgversprechend ist, müssen viele Clients gleichzeitig auf dieser Webseite sein, welche Rechenleistung bereitstellen. Da die Applikation für mehrere Mandanten ausgelegt werden soll, kann aber davon ausgegangen werden, dass dies der Fall sein wird.

Die Komponente "Browserapplikation" enthält die Komponente "GUI", den "TSP Solver" und den "TSP Node Solver". Bei der Ausführung wird auf dem Hauptclient das GUI gestartet, welches die nötigen Informationen für die Distanzmatrix und das Kartensmaterial vom Routingserver und dem Kartenserver bezieht und damit die Komponente "TSP Solver" aufruft. Innerhalb dieser Komponente wird mittels dem Managing-Server das verteilte System angestoßen (Ablauf ist in Abschnitt 12.1.2 erläutert). Die Komponente "TSP Node Solver" wird dann von dem Managing-Server auf den anderen Clients im System für die Berechnung der einzelnen Knoten verwendet. Das Resultat wird am Schluss wieder zum "TSP Solver" zurückgeschickt.

Der zusätzliche Zeitaufwand für dieses Konzept setzt sich aus den einzelnen Zusatzaufwänden für die Erstellung und Verbindung zu jedem Client und der Kommunikation unter den einzelnen Clients zusammen:

$$T_{Overhead}(n) = (n - 1) * T_{Job} + T_{Communication}$$

Da aber die einzelnen Job-Initialisierungen gleichzeitig erfolgen, reduziert sich die absolute Zusatzzeit:

$$T_{Overhead}(n) = T_{Job} + T_{Communication}$$

Bei verteilten Systemen ist ein sinnvolles Verteilen auf die einzelnen Clients erforderlich. Dies wird in dieser Arbeit bewusst ausgeklammert, damit das Konzept in seinen Grundzügen auf dessen Umsetzbarkeit geprüft werden kann, ohne die Komplexität noch weiter zu erhöhen.

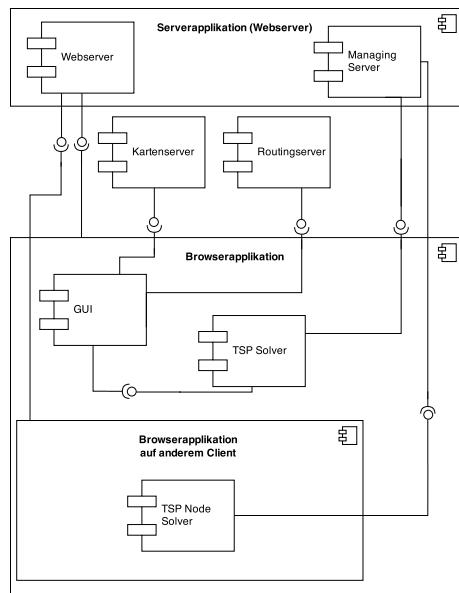


Abb. 12.10: Bausteinsicht des Konzepts 6 (*Quelle: Eigene Darstellung*)

Konzept 7: Verteiltes, serverseitiges System (L8)

Für die serverseitige Implementation eines verteilten Systems existieren verschiedene Möglichkeiten. Da aber bereits im Voraus bekannt ist, dass das System keine hohen Lizenzkosten verursachen darf (RE-R2-1), wird dieses Konzept nicht weiter ausgearbeitet.

Konzept 8: Verteiltes, server- und clientseitiges System (L9)

Damit Server und Clients in ein verteiltes System eingebunden werden können, müssen diese untereinander kommunizieren können. Die Kommunikation unter Servern kann beliebig implementiert sein, um jedoch Clients einzubinden muss auf das HTTP-Protokoll zurückgegriffen werden. Dieses Konzept gleicht somit stark dem Konzept 6, mit der Ausnahme, dass einzelne Clients durch Server ersetzt werden. Auch dieses Konzept wird aufgrund der entstehenden hohen Lizenzkosten (RE-R2-1) für den Betrieb mehrerer Server nicht weiter verfolgt.

Konzept 9: Bestehenden Service beanspruchen

Die Variante der Einbindung eines bestehenden Services sollte nicht ausser Acht gelassen werden. Durch das Auslagern der Problematik kann zwar der grosse Einarbeitungs- und Entwicklungsaufwand eingespart werden. Es wird jedoch eine Abhängigkeit geschaffen (widerspricht RE-Q2) und die Erweiter- und Änderbarkeit der Lösung ist nicht gewährleistet, welche aber ebenfalls eine Anforderung (RE-Q3-4) an das System

darstellt. Zudem fallen vermutlich bei Anbindung eines Services zusätzliche Lizenz- oder Benutzungskosten an. Weiter können Anbieter des Services jederzeit Änderungen am Programm vornehmen oder sogar den Dienst einstellen. Aus diesen Gründen wird auch dieses Konzept nicht weiter verfolgt.

12.3.1.4 Einschätzung der Konzepte

Unter der Bedingung, dass Konzepte mit vergleichsweise hohem zusätzlichen Zeitaufwand besser für grössere Problemgrössen geeignet sind, soll an dieser Stelle eine erste Prognose für den Erfolg der Konzepte gemacht werden. Anhand der $T_{Overhead}(n)$ der einzelnen Konzepte ist abschätzbar, welches die Mindestproblemgrösse ist, bei der das Konzept eingesetzt werden kann. Das erste Konzept ist für sehr kleine Problemgrössen geeignet, da kein Overhead besteht. Jedoch wird schon bei einigen *WP* der Browser abstürzen, respektive nicht mehr bedienbar sein, da das komplette *GUI* blockiert wird. Das Konzept 2 sollte dieses Problem beheben, da die Berechnung im separaten Berechnungsthread erfolgt. Die Obergrenze der Problemgrösse liegt aber nicht viel höher, da die Berechnung nicht effizienter erfolgt, sondern lediglich in einem anderen Thread. Eine effektive Steigerung der Geschwindigkeit müsste das Konzept 3 bringen, da parallel mehrere Berechnungen gleichzeitig erfolgen. Wegen dem grossen zusätzlichen Zeitaufwand wird bei zu kleinen Problemgrössen trotz der effizienteren Berechnung in mehreren Threads kein Geschwindigkeitszuwachs zu erwarten sein. Die Berechnung könnte allenfalls sogar langsamer werden, falls der Overhead grösser als der Zeitgewinn der Berechnung ist. Das Konzept 4, welches die Berechnung serverseitig ausführt, wird für etwas grössere Problemgrössen als die sequenziellen, clientseitigen Konzepte geeignet sein, aber nicht an die Effizienz der parallelisierten Lösung herankommen. Zudem muss bei diesem Konzept in Betracht gezogen werden, dass viele Benutzer gleichzeitig eine Berechnung auslösen können und gegebenenfalls die Ressourcen des Servers nicht genügen, um alle Anfragen genügend schnell zu bearbeiten. Auch das multithreaded, serverseitige Konzept 5 hat diese Einschränkung. Dennoch kann davon ausgegangen werden, dass dieses Konzept für etwas grössere Problemgrössen besser geeignet ist, da auch hier die Parallelisierung die Berechnung beschleunigt. Die komplexesten Konzepte der verteilten Systeme sollten sich für grosse Problemgrössen eignen, da viele Rechner die Ressourcen einbringen können. Nach den Geschwindigkeitsmessungen in den Versuchen in Abschnitt 12.1.2 kann aber davon ausgegangen werden, dass diese Konzepte auch für Problemgrössen über 12 *WP* nicht mehr effizient sind, da die gemeinsam genutzten Daten unter den einzelnen Clients ausgetauscht werden müssen. Dieselbe Voraussetzung ist auch beim Konzept 8 anzutreffen. Lediglich das nur auf Servern basierte Konzept eines verteilten Systems könnte diese Schwachstelle umgehen, indem die Server über

eigene, schlankerere Protokolle kommunizieren.

Nach diesen ersten Einschätzungen werden die Konzepte in Kapitel 13 mittels *POCs* genauer analysiert und ausgewertet.

12.3.2 Konzept der Algorithmenauswahl

Neben der Architekturwahl ist auch die Algorithmenwahl entscheidend für die Lösung des *TSP*. Damit die Architekturen untereinander verglichen werden können, wurde ein Algorithmus ausgewählt, der für alle Konzepte verwendet werden kann. Für die Implementation einer Webapplikation, die das *TSP* löst, kann aber eine zur Laufzeit stattfindende Algorithmenwahl den Vorteil bringen, dass bis zu einer gewissen Problemgrösse die exakte Lösung und bei grösseren Problemgrössen eine Näherungslösung berechnet werden kann. Dadurch kann beispielsweise trotz steigender Anzahl *WP* innerhalb der vorgegebenen Zeit eine Lösung berechnet werden. In POC-3a wurde versucht Browserindikatoren zusammenzustellen, um die Rechengeschwindigkeit des Browsers zu ermitteln. Leider sind die Daten, wie die Anzahl verfügbaren Rechenkerne, nicht in allen Browsern und auf allen Betriebssystemen verfügbar, weshalb im Moment die Auswahl des Algorithmus lediglich aufgrund der Anzahl *WP* realisierbar ist.

13 POCs der Architekturvarianten (R6)

Dieses Kapitel enthält eine Übersicht über die implementierten Teile, welche als *POCs* der ausgearbeiteten Konzepte dienen. Der Zugriff auf die *POCs* und den Quellcode ist in Anhang A beschrieben. Die *POCs* liefern Fakten und Erkenntnisse für die im nachfolgenden Kapitel 14 aufgeführte Nutzwertanalyse.

Die einzelnen *POCs* der Architekturkonzepte sind in einer gemeinsamen Applikation umgesetzt (POC-4a), welche in Abb. 13.1 dargestellt ist. Auf der linken Seite kann eine Distanzmatrix eingegeben oder eine randomisierte Matrix generiert werden. Danach kann auf der rechten Seite der Algorithmus und die Art der Ausführung angegeben werden. Unterhalb der horizontalen Linie erscheinen dann die Ergebnisse und falls eine verteilte Berechnung ausgewählt wurde ist das Log der Berechnung zu sehen.

Abb. 13.1: Der *TSP-Rechner* aus POC-4a (Quelle: Eigene Darstellung)

13.1 Implementation

In den nachfolgenden Abschnitten werden die Überlegungen und Erkenntnisse aus der Erstellung der *POCs* aufgeführt.

Konzept 1

Beim Konzept 1, dessen *POC* zuerst erstellt wurde, tauchte das unerwartete Problem auf, dass beim Serialisieren eines Arrays die Werte Infinity zu Null umgewandelt wurden (POC-0a). Dies beeinflusste die Rechnung insofern, dass der Lower Bound immer kleiner war als der Vorangehende, da der Wert in der Distanzmatrix Null anstelle von unendlich war. Durch eine kleine Anpassung der Lower Bound Berechnung konnte dieses Problem schnell gelöst werden.

Ein weiterer Stolperstein stellte das Generieren von Testdaten dar. Falls nämlich eine Matrix mit vollständig zufälligen Werten erstellt wird, können einzelne Teilstrecken sehr

schnell ausgeschlossen werden, da die Unterschiede der Distanzen so gross sind und der Lower Bound der aktuell besten Lösung schnell überschritten wird. Diese Verzerrung der Messresultate konnte eliminiert werden, indem die Distanzmatrix folgendermassen generiert wurde: Es muss gelten $|d(c_i, c_j) - d(c_j, c_i)| \leq 5$, wobei c_i und c_j Zufallszahlen zwischen 100000 und 200000 sind. Natürlich sind diese Testdaten noch immer nicht perfekt, jedoch genügten sie für die vorgenommenen Messungen. Recherchen im Internet ergaben, dass das Generieren von Testdaten für das *TSP* ein eigenes Forschungsgebiet darstellt und im Rahmen dieser Arbeit nicht weiter verfolgt werden kann.

Konzept 2

Die Umsetzung des Konzept 2 mit der Berechnung im *GUI* Thread brachte keine unerwarteten Probleme mit sich, da die Eigenschaften und Eigenheiten der *Webworker* dank der Vorversuche im Abschnitt 12.1 bereits bekannt waren.

Konzept 3

Die Umsetzung der multithreaded Lösung des Konzept 3 brachte bei den ersten Messungen keinen Geschwindigkeitsvorteil gegenüber der Lösung mit dem *GUI* Thread. Nachforschungen ergaben, dass durch die verwendete "while Schleife" im *Webworker* keine Nachrichten der anderen Threads entgegen genommen wurden, da der gesamte Thread durch die Schleife blockiert wurde. Dadurch konnten lediglich Knoten verworfen werden, falls eine bessere Lösung in demselben Thread gefunden wurde, nicht aber falls ein anderer Thread eine bessere Lösung gefunden hatte. Um dieses Blockieren zu umgehen, kann in JavaScript mit dem Befehl "setTimeout" gearbeitet werden, welcher eine Funktion nach Ablauf einer vordefinierten Zeitspanne ausführt. Der Thread wird während dem Warten auf das Ausführen der Funktion freigegeben, wodurch Nachrichten von anderen Threads nicht blockiert und im Thread angenommen werden.

Zwar brachte die Parallelisierung eine gewisse Geschwindigkeitssteigerung, jedoch wurde eine bessere Leistung erwartet. Deshalb wurde nach Wegen gesucht, um die Leistung noch weiter zu erhöhen. Die naheliegende Lösung ist das Limitieren von den zu erzeugenden Threads. Das heisst beispielsweise, dass bei einer Problemgrösse von $n = 5$ nicht wie bisher $(5 - 1) = 4$ Threads erzeugt werden, sondern lediglich zwei Threads und die Knoten auf diese verteilt werden. Diese Lösung wurde dann auch umgesetzt und es kann in POC-4a bei der Auswahl der parallelisierten (multithreaded) Berechnung angegeben werden, wie viele Threads maximal erzeugt werden sollen. Die Berechnung ist dadurch bedeutend schneller. Nach einigen Versuchen stellte sich heraus, dass die Anzahl Threads für die effizienteste Berechnung in den meisten Fällen bei einem bis drei Threads liegt. Die voreilige Annahme, dass die Variante der Implementation mit

dem *GUI* Thread dasselbe ist, wie das multithreaded System mit der Limitierung auf einen Thread, ist falsch. Die Begründung liegt darin, dass beim GUI Thread die komplette Berechnung im separaten Thread ausgeführt wird und beim Multithreading erst die Berechnung der Knoten. Die einzelnen Knoten der Stufe 2 werden somit noch im Hauptthread generiert, währenddessen bereits die Knotenberechnung im anderen Thread startet.

Konzept 4

Die Applikation konnte ohne grosse Anpassungen auf dem Webserver, welcher Node.js installiert hat, zum Laufen gebracht werden. Die Geschwindigkeitsvorteile sind aber wie erwartet nur geringfügig.

Konzept 5

Für die Implementation des serverseitigen, multithreaded Systems aus Konzept 5 musste auf Prozesse anstelle von Threads gewechselt werden, da Node.js lediglich Prozesse ohne zusätzliche Plugins unterstützt. Dies wird vermutlich auch der Grund für die langsamere Berechnung auf dem Server als auf dem Client sein, da Threads leichtgewichtiger sind als Prozesse. Ansonsten änderte sich am Konzept aber nichts.

Konzept 6

Das letzte umgesetzte *POC*, das verteilte System unter Clients aus Konzept 6, stellte die grösste Herausforderung dar. Bereits mehrere Threads erschweren das *Debuggen* wesentlich. Beim verteilten System wird das *Debuggen* noch weiter durch die Verteilung auf verschiedenen Clients erschwert. Für die Umsetzung wurde das Semaphoren-Konzept eingesetzt, wodurch jeder Client eine bestimmte Anzahl an Jobs entgegen nehmen kann. Zur Vereinfachung ist diese Zahl immer auf eins gesetzt. Der Managing-Server weiss wie viele Jobs jeder Client gleichzeitig abarbeiten kann und verteilt entsprechend das zu berechnende Problem an die Clients, welche freie Kapazität haben. Nicht berücksichtigt wurde bei der Umsetzung, was passiert, falls ein Client während der Berechnung beendet wird. Falls bei der Aufgabe eines Jobs an das verteilte System keine Clients mit verfügbarer Kapazität bereitstehen, wartet der Managing-Server mit der Jobvergabe bis Clients für die Berechnung frei werden. Der Geschwindigkeitsvorteil dieser Lösung ist leider auch nicht wie erwartet ausgefallen. Zurückzuführen ist dies auf die langsame Kommunikation unter den Clients. Dennoch kann davon ausgegangen werden, dass sich in Zukunft solche Systeme etablieren werden, da grosse Mengen an Rechenleistung bestehen, welche zwischen den einzelnen Operationen nur zum Teil genutzt werden oder total ungenutzt sind.

13.2 Debugging

Für das *Debuggen* der Implementation werden die Entwicklerwerkzeuge (DevTools) von *Chrome* und *Firefox* eingesetzt. Die FireFox DevTools (Abb. 13.2) bieten einen Schritt für Schritt Debugging Modus an, jedoch werden darin *Webworker* noch nicht unterstützt.

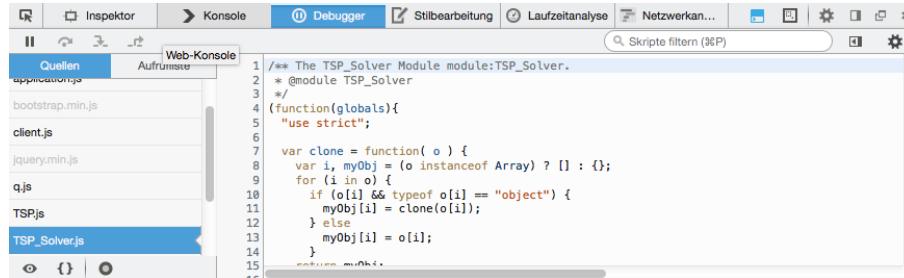


Abb. 13.2: FireFox DevTools Debugging (*Quelle: Eigene Darstellung*)

Eine grosse Hilfe bieten die Chrome DevTools mit dem Profiling an. Darin wird sichtbar, wie lange die Applikation bei der Berechnung für die einzelne Abschnitte und Funktionen benötigt. Ein Ausschnitt aus einem Profile wird in Abb. 13.3 dargestellt. Es sticht heraus, dass beim Ausführen dieses *Webworkers* eine lange Wartezeit ("idle") besteht. Diese Wartezeit entsteht durch das Warten und Versenden von Nachrichten an den Hauptthread. Falls die Implementation ohne die Funktion "setTimeout" realisiert ist, wird die Wartezeit stark verkürzt, da der Thread während der Berechnung blockiert wird und nur zu Beginn und am Ende der Ausführung auf Nachrichten wartet oder Nachrichten versendet. Dies ist in Abschnitt 13.1 genauer beschrieben.

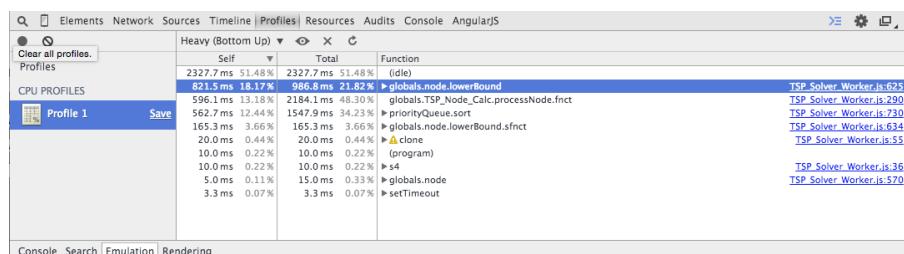


Abb. 13.3: Chrome DevTools Profiling (*Quelle: Eigene Darstellung*)

13.3 Dokumentation

Der Quellcode ist dokumentiert und beschreibt die einzelnen Komponenten. Ebenfalls wurde aus den Kommentaren mittels JSDoc die Dokumentation der Applikation generiert, welche in Abb. 13.4 zu sehen ist.

Class: TSP

TSP

```
new TSP(distanceTable, waypoints)
```

Represents a Traveling Salesman Problem

Parameters:

Name	Type	Description
distanceTable	array	Distance table
waypoints	array	Waypoints array (currently not needed)

Source:

[TSP_Solver.js, line 84](#)

Home

Modules

- Application
- Benchmark
- RandomWaypoints
- TSP_Solver
- TSP_Solver_Worker

Classes

- tsp
- TSP_Node_Calc
- TSP
- TSP_Solver

Methods

```
solve(algo, mode, threadCount, successFnct, errorFnct,
progressFnct)
```

Solve the TSP

Parameters:

Name	Type	Description
algo	mixed	true (boolean) to autodetect best fitting algorithm for the tsp. DefaultValue / DnD / Dynamic / Antecedent / I2 / Nearestneighbour

Abb. 13.4: Dokumentation generiert mit JSDoc aus den Kommentaren des Quellcodes (Quelle: Eigene Darstellung)

13.4 Testing

Das Testing erfolgt mit dem JavaScript Testframework jasmine. Leider konnte kein Testframework gefunden werden, welches *Webworker* vollumfänglich unterstützt, weshalb nur die Umsetzung des Konzepts 1 mit dem Framework getestet werden konnte. Die Tests werden über die Commandline ausgeführt, wie in Abb. 13.5 dargestellt. Die Lösungen der anderen Konzepte wurden dann mittels einzelner Aufrufe getestet und mit der Lösung von Konzept 1 verglichen.

```
klickagent:tsp_solver roman$ jasmine
Started
.....
8 specs, 0 failures
Finished in 2.49 seconds
klickagent:tsp_solver roman$
```

Abb. 13.5: Testing mit dem Framework jasmine (Quelle: Eigene Darstellung)

13.5 Benchmark

Nach der Umsetzung der einzelnen *POCs* wurde in POC-4b ein Benchmark-Tool entwickelt, damit die einzelnen Architekturvarianten geprüft werden können. Die Resultate sind unter dem Punkt Resultate in POC-4b zu finden und bilden die Grundlage für das Bewertungskriterium "Performance" in der Nutzwertanalyse in Kapitel 14. Alle

Messungen wurden mit der in Anhang C beschriebenen Konfiguration gemacht. Damit die Berechnungsgeschwindigkeiten der *POCs* der Konzepte noch mit einer anderen Programmiersprache verglichen werden können, wird eine leicht abgeänderte Implementation der in Java geschriebenen Lösung von Wiener [82] und Wiener [83] eingesetzt. Die verwendete Distanzmatrix ist in JavaScript-Notation in Codeblock 13.1 dargestellt und die Messresultate sind in Tab. 13.1 aufgeführt.

```

1  [
2  [0,52465,346893,127437,236079,301124,604885,110635],
3  [52148,0,385785,170320,215131,298413,556421,62171],
4  [344070,381291,0,245103,419249,270764,876449,439461],
5  [127497,170265,246866,0,305372,319633,722685,228435],
6  [238797,216763,414772,309241,0,182961,460005,186789],
7  [302204,299750,267211,320065,183225,0,640425,349328],
8  [605009,556551,869801,723181,457403,637990,0,499097],
9  [110158,61700,443795,228330,190328,348160,499089,0]
10 ]

```

Codeblock 13.1: Generierte Distanzmatrix in JavaScript-Notation

Programmiersprache	Berechnungszeit [ms], singlethreaded	Berechnungszeit [ms], multithreaded, 7 Threads
JavaScript	3349	985
Java	6	3

Tab. 13.1: Gegenüberstellung der Geschwindigkeit des Algorithmus in Java und JavaScript
(Quelle: Eigene Darstellung)

Die Berechnungszeit zeigt, dass mittels der auf Java basierten singlethreaded Applikation die Lösung 500 Mal schneller gefunden wird als mit JavaScript. Bei der Berechnung mit mehreren Threads ist die Ausführgeschwindigkeit im angegebenen Beispiel 333 Mal schneller als in JavaScript. Die Messungen wurden auf demselben Rechner gemacht unter Verwendung von Chrome für die Ausführung von JavaScript und die Eclipse IDE für Java. Der direkte Vergleich beider Programmiersprachen ist natürlich aufgrund der nicht exakt gleichen Umsetzung des Algorithmus verfälscht, jedoch zeigt der sehr grosse Unterschied, dass Berechnungen direkt im Browser nicht an die Leistung von systemnahen Programmiersprachen kommen können.

14 Nutzwertanalyse (R4)

Mit der Nutzwertanalyse werden die erarbeiteten Konzepte wissenschaftlich bewertet, um diejenigen Konzepte zu finden, welche sich für die Implementation im finalen Konzept eignen. Die Punktevergabe geschieht nach den in Tab. 14.1 angegebenen Bewertungskriterien, die aus der Anforderungsanalyse in Kapitel 9 und dem Entscheidungskonzept in Kapitel 11 stammen. Neben dem Namen und der Beschreibung wird die Gewichtung des Kriteriums gesetzt, um wichtigeren Kriterien grösseren Einfluss zu geben. In der Bewertungsskala werden die zu verteilenden Punkte pro Kriterium genauer erläutert, welche sich immer von 1 Punkt (schlecht) bis 3 Punkten (gut) erstrecken. Für das finale Konzept werden nur noch diejenigen Konzepte berücksichtigt, welche mehr als 80% des Punktemaximums (mindestens 65 Punkte) erreicht haben.

Name	Beschreibung	Ur-sprung	Ge-wich-tung	Bewertungsskala
Lizenzkosten	Anfallende Kosten für Lizenzen von Services und Software	RE-Q2, RE-R2-1	3	1: sehr hohe Kosten (> 1000 Franken pro Monat) 2: mittlere Kosten (> 100 und ≤ 1000 Franken pro Monat) 3: keine Kosten
Infrastruktur	Zusätzlich für den Betrieb der Webseite zu der bereits vorhanden Client-Server Architektur benötigte Infrastruktur	Im Rahmen der Um-setzung ausge-klammert	2	1: > 3 neue Geräte / Komponenten 2: 1 und ≤ 3 neue Geräte / Komponenten 3: keine neuen Geräte / Komponenten
Erweiterbarkeit	Einschätzung wie einfach eine Erweiterung der Bedingungen für die Routenberechnung implementiert werden können	RE-Q3-4	2	1: Änderungen am Algorithmus und der Infrastruktur nötig 2: Änderungen am Algorithmus / der Infrastruktur / dem Service nötig 3: lediglich Distanzmatrix muss angepasst werden
Skalierbarkeit (Problemgrösse)	Systemverhalten bei steigender Problemgrösse	RE-Q3-10	2	1: ≤ 14 WP 2: > 15 WP und < 18 WP 3: ≥ 19 WP
Skalierbarkeit (Lastverhalten)	Systemverhalten bei vielen gleichzeitigen Anfragen	RE-Q3-9	3	1: Die Systemleistung wird stark beeinträchtigt und liefert die Ergebnisse mehr als 20 Sekunden später 2: Die Systemleistung wird weniger beeinflusst und liefert die Ergebnisse mehr als 5 Sekunden später 3: Das System erleidet keine spürbaren Geschwindigkeitseinbussen

Name	Beschreibung	Ur-sprung	Ge-wich-tung	Bewertungsskala
Performance	Durchschnittliche Berechnungszeit für 11 WP, berechnet mit dem Branch-and-Bound-Algorithmus aus Abschnitt 3.3.1.2	RE-Q3-9	3	1: ≥ 30 Sekunden 2: > 5 Sekunden und < 30 Sekunden 3: ≤ 5 Sekunden
Programmkopierschutz	Schützbarkeit der Berechnungsapplikation	RE-Q1,2,3-4	1	1: Gesamter Quellcode ist im Browser einsehbar 2: Entscheidende Teile laufen lediglich serverseitig ab 3: Nicht nötig, da Dienst beansprucht wird
Integration	Bewertung der Integrationseinfachheit in einer Applikation	RE-Q2, RE-Q3-4	2	1: Keine klare Schnittstelle vorhanden 2: Eine sich ändernde Schnittstelle, oder es kann kein Einfluss auf die Schnittstelle genommen werden 3: Klar definierte Schnittstelle vorhanden und es kann gegebenenfalls Einfluss auf die Schnittstelle genommen werden
Ausfallsicherheit	Ausfallsicherheit bezogen auf das Gesamtsystem (der Webserver, welcher die nötigen Daten für clientseitige Systeme bereitstellt ausgenommen). Point of Failure	RE-Q3-7	2	1: Falls ein Rechner ausfällt, fällt das gesamte System für alle Benutzer aus 2: Falls ein Rechner ausfällt, fällt das gesamte System für einige Benutzer aus 3: Falls ein Rechner ausfällt, fällt das System nur für einen Benutzer aus
Implementationsaufwand, KnowHow	Nötiges KnowHow und der Implementationsaufwand für die Entwicklung des Systems	Indirekt durch RE-R-1	3	1: Hohe Komplexität und mehrere Fachbereiche müssen bekannt sein 2: Mittlere Komplexität mit wenigen Fachbereichen (< 3) 3: Keine zusätzlichen Kenntnisse nötig, neben Webseitenprogrammierung mit JavaScript (AJAX Requests vorausgesetzt)
Benutzbarkeit (Usability)	Verhalten des GUIs während Berechnungen (hängt von der Implementation der Applikation ab)	RE-Q1,2,3-1, RE-Q1,2,3-2	2	1: GUI blockiert komplett 2: GUI blockiert kurzzeitig 3: GUI blockiert nie
Testbarkeit	Testbarkeit und Möglichkeiten für Debugging bei der Umsetzung des Konzepts	RE-Q3-6	2	1: GUI nicht testbar 2: GUI teilweise testbar 3: GUI testbar

Tab. 14.1: Bewertungskriterien Nutzwertanalyse (Quelle: Eigene Darstellung)

	Ge-wich-tung	Konzept 1	Konzept 2	Konzept 3	Konzept 4	Konzept 5	Konzept 6	Konzept 7	Konzept 8	Konzept 9
Beschreibung		Seq Client	Client multithreaded	Seq Server	Server multithreaded	verteilt Client	verteilt Server	verteilt gemischt	Service	
Lizenzkosten	3	3	3	3	3	3	3	1	2	1
Infrastruktur	2	3	3	3	3	3	3	1	1	3
Erweiterbar-keit	2	3	3	3	3	3	3	3	3	1
Skalierbarkeit (Problemgrös-se)	2	1	1	2	1	1	1	3	1	2
Skalierbarkeit (Lastverhal-ten)	3	3	3	3	1	1	2	3	3	1
Performance	3	2	2	3	2	2	1	3	2	2
Integration	2	3	3	3	3	3	3	3	3	2
Ausfallsicher-heit	2	3	3	3	1	1	2	3	3	1
Implementati-onsaufwan-d/KnowHow	3	3	2	1	3	2	1	1	1	3
Testbarkeit	2	3	2	1	3	2	1	1	1	3
Programmko-pierschutz	1	1	1	1	3	3	1	3	2	3
Benutzbarkeit (Usability)	2	1	3	3	3	3	3	3	3	3
Total	-	68	70	67	64	59	54	61	55	54

Tab. 14.2: Nutzwertanalyse der Konzepte aus Abb. 11.2 und Abb. 11.3 (Quelle: Eigene Darstellung)

Aus der Zeile Total in Tab. 14.2 ist abzulesen, dass für das finale Konzept die Konzepte 1, 2 und 3 die geeignetsten sind, da diese in der Nutzwertanalyse 65 Punkte oder mehr erzielt haben. Anzumerken ist, dass die Werte der Konzepte 7, 8 und 9 lediglich Einschätzungen sind und nicht auf einem *POC* basieren.

Nachfolgend sollen die Bewertungskriterien und deren Punktevergabe kurz erläutert werden:

Lizenzkosten

Die Lizenzkosten für die Konzepte 4 und 5 wurden, wie die clientseitigen Konzepte 1, 2 und 3 mit 3 Punkten bewertet, da neben dem bereits benötigten Webserver keine weiteren Serverkosten anfallen. Anzumerken ist aber, dass für die Ausführung der JavaScript Applikation auf dem Server ein Server mit Node.js Unterstützung erforderlich ist. Dieselbe Argumentation gilt auch für den Punkt "Infrastruktur".

Erweiterbarkeit

Die Erweiterbarkeit ist wegen der Implementation mit der Distanzmatrix bei allen Konzepten ausser dem Einsatz eines Services gegeben.

Skalierbarkeit (Problemgrösse)

Die Skalierbarkeit bezogen auf die Problemgrösse wurde aufgrund der *POCs* ermittelt. Die Punktevergabe erfolgte anhand der maximalen Anzahl an *WP*, für welche das Zeitlimit aus den Anforderungen noch eingehalten werden konnte. Dies erfüllte keines der umgesetzten Konzepte vollständig. Es kann davon ausgegangen werden, dass ein optimal aufgesetztes, verteiltes System von mehreren Servern, gemäss Konzept 7, diese Anforderung erfüllen kann. Neben dem Service erreicht nur noch das Konzept 3 mit mehreren Threads die Anforderung teilweise.

Skalierbarkeit (Lastverhalten)

Dieselbe Berechnungsgeschwindigkeit unter zunehmender Last des Gesamtsystems ist bei allen clientseitigen Konzepten gewährleistet, da die Anzahl der Benutzer und Anfragen keinen Einfluss auf die Berechnung des Einzelnen haben. Auch bei verteilten Systemen ist der Einfluss nicht relevant. Bei serverseitigen oder der Einbindung eines Services ist der Einfluss aber hoch.

Performance

Die Performance korreliert stark mit der Skalierbarkeit bezogen auf die Problemgrösse und ist beim Konzept 3 und Konzept 7 am besten.

Integration

Die Integrationsfähigkeit der Lösung in eine Applikation ist bei allen sehr gut, lediglich beim Service besteht die Einschränkung, dass die Schnittstelle nicht selber beeinflusst werden kann.

Ausfallsicherheit

Die Ausfallsicherheit ist wiederum bei allen clientseitigen Konzepten gegeben, da nur der eine Client ausfällt und nicht das gesamte System beeinträchtigt wird. Bei serverseitigen Systemen und dem Service bringt der Ausfall des Gerätes oder des Services die gesamte Berechnung aller Anfragen zum Erliegen.

Implementationsaufwand / Know-How

Der Implementationsaufwand und das nötige Know-How für die Konzepte ohne Parallelisierung und Verteilung sind am geringsten. Solche mit Parallelisierung erfordern mehr und diejenigen Konzepte mit verteilten Systemen erfordern am meisten Implementationsaufwand und Know-How.

Testbarkeit

Für die Testbarkeit gilt exakt dieselbe Punkteverteilung, weil die *Webworker* im Browser und Prozesse auf dem Server weniger gut testbar sind und die verteilten Systeme kaum sinnvoll getestet werden können (siehe Abschnitt 13.2).

Programmkopierschutz

Der Programmkopierschutz ist lediglich auf serverseitigen Konzepten und dem verteilten System unter Servern komplett gegeben, da der Code nur auf dem Server ausgeführt wird. Bei den anderen Konzepten ist der Kopierschutz nicht gegeben.

Benutzbarkeit

Das letzte bewertete Kriterium stellt die Benutzbarkeit dar. Diese ist lediglich beim Konzept 1 eingeschränkt, da dort alles in einem Thread ohne der Verwendung von asynchronen Aufrufen erfolgt.

Teil IV

Ergebnisse

Der letzte Teil enthält das resultierende Konzept, das daraus entwickelte Produkt, die Erkenntnisse aus der Arbeit und mögliche weiterführende Untersuchungsgebiete.

15 Resultate (R5,R6)

Aus der Nutzwertanalyse und den *POCs* geht hervor, dass ein verteiltes System unter Webclients keine, und die Parallelisierung nur unter bestimmten Bedingungen eine Effizienzsteigerung bringt. Somit kann die These, dass bekannte Konzepte zur Lösung des *TSP* auch auf den Webbereich angewendet werden können, nur zum Teil bewiesen werden. Bei verteilten Systemen unter Clients wird der Kommunikationsaufwand beim gewählten Branch-and-Bound-Algorithmus zu gross, sodass die zusätzlichen Ressourcen der anderen Clients keine Zeitminderung der Berechnung erwirken können. Die Parallelisierung bringt nur bei 2-3 Workerthreads eine Effizienzsteigerung, wobei auch diese stark von den Eingangswerten der Distanzmatrix abhängt. Falls die Distanzen zu den einzelnen *WP* weit auseinander liegen, können Teillösungen früher ausgeschlossen werden, und der Algorithmus findet die Gesamtlösung schneller. Die Workerthreads von JavaScript eignen sich demnach mehr für die Separierung des *GUIs* von der Berechnung, als für das Parallelisieren von Applikationen für die Berechnung bei grösseren Problemgrössen. Mit den Workerthreads kann erreicht werden, dass das *GUI* nicht blockiert wird und Statusmeldungen der Berechnungen auf dem *GUI* angezeigt werden können, während die Berechnung im separaten Thread erfolgt. Die Verteilung auf mehrere Server wurde aufgrund der Anforderung der Minimierung der Lizenzkosten nicht weiter verfolgt. Dasselbe gilt für die Verteilung auf mehrere Clients und Server.

Eine weitere wichtige Erkenntnis brachte der Vergleich von der Berechnung in JavaScript mit der Berechnung in der systemnäheren Programmiersprache Java. Dieser zeigte, dass Berechnungen in JavaScript 500 Mal langsamer ausgeführt werden als in Java. Dies bekräftigt wiederum das Konzept der Aufteilung des Problems in Teilprobleme und der dynamischen Auswahl des Algorithmus aufgrund der Anzahl *WP* während der Laufzeit.

Das aus den Resultaten abgeleitete Konzept in Abb. 15.1 beschreibt das Vorgehen der Algorithmen- und Architekturwahl für die Implementation. Das Problem wird zuerst in sinnvolle Teilprobleme unterteilt. Für das *TSP* ist eine Unterteilung nach Anzahl *WP* sinnvoll. Als Erstes wird versucht, das Problem mit einem exakten Algorithmus (E1) zu lösen. Sind genügend Ressourcen und genügend Zeit für die Berechnung vorhanden, kann das Problem sequenziell gelöst werden. Falls die Berechnung länger als die erlaubte Browserausführzeit von JavaScript Code (E5) dauert, ist eine Implementation in einem Worker Thread (L2) nötig. Ansonsten ist eine Berechnung direkt im Hauptthread möglich (L1). Sind nicht genügend Ressourcen oder Zeit vorhanden, wird eine Parallelisierung mit 2-3 *Webworkers* nötig (L3). Dasselbe Verfahren wird, falls keine exakten Algorithmen

in Frage kommen, für die Näherungsalgorithmen angewendet, um die entsprechende Architektur für die vorhandenen Ressourcen herauszufinden.

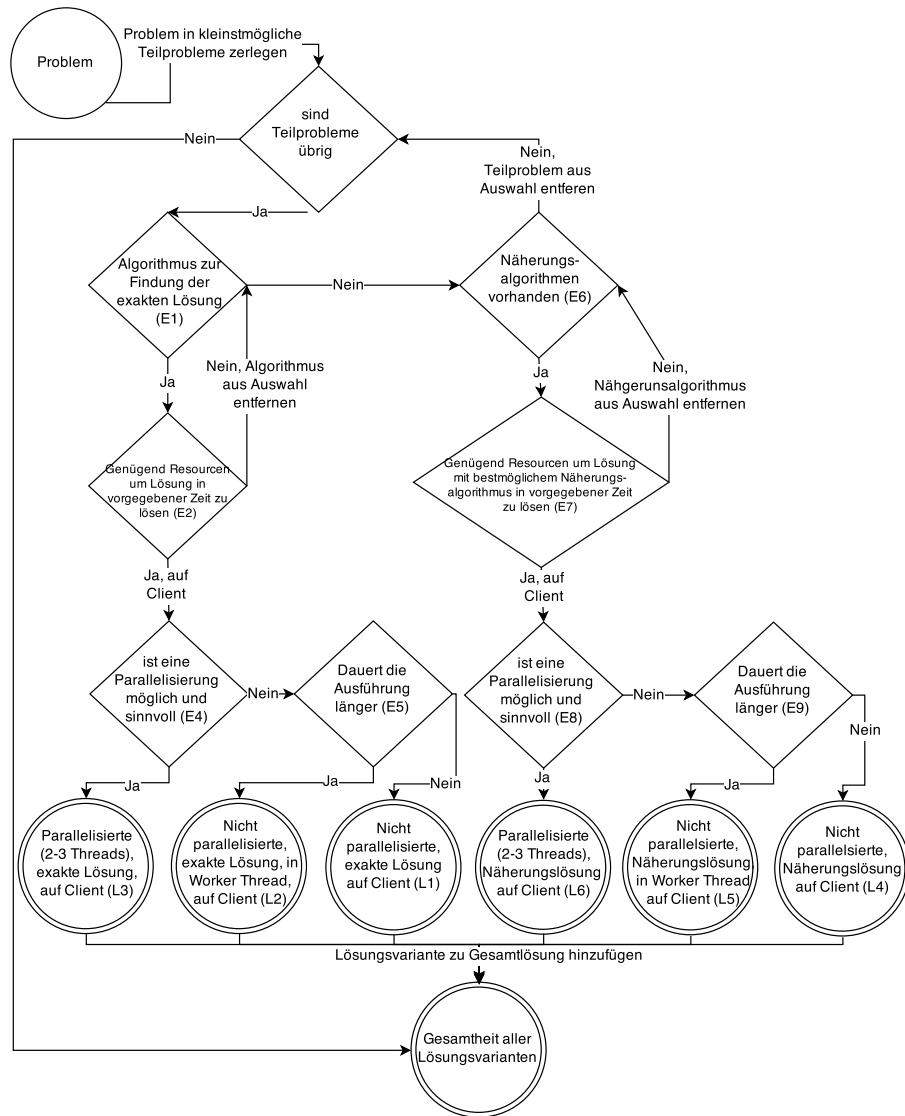


Abb. 15.1: Resultierendes Entscheidungsdiagramm zur Ermittlung der Architektur (Quelle: Eigene Darstellung)

Die Anwendung des Konzepts in Abb. 15.1 auf das *TSP* führt zu folgender Lösung, welche im POC-5a umgesetzt wurde und auch im finalen Produkt eingesetzt wird. Das finale Produkt wird in Kapitel 16 vorgestellt.

Bereich WP	Algorithmus	Architektur	Lösung aus Abb. 15.1
$n < 4$	Dynamic (Held-Karp)	Konzept 1, JavaScript normal	L1
$n < 20$	Dynamic (Held-Karp)	Konzept 2, JavaScript in Webworker	L4
$n < 300$	K3, Ant-Colony	Konzept 2, JavaScript in Webworker	L4

Bereich <i>WP</i>	Algorithmus	Architektur	Lösung aus Abb. 15.1
$n \geq 300$	Nearest Neighbour	Konzept 2, JavaScript in Webworker	L4

Tab. 15.1: Anwendung des Konzepts auf das *TSP* (*Quelle: Eigene Darstellung*)

Falls weniger als vier *WP* angegeben wurden, bietet sich die Lösung mit dem Dynamic-Algorithmus direkt im Browser im selben Thread an (Konzept 1), da kein Zusatzaufwand für die Übermittlung oder ähnliches entsteht. Ebenfalls ist die Berechnung für bis zu 19 *WP* mit dem Dynamic Algorithmus sinnvoll, jedoch unter Verwendung eines Threads für die Berechnung, damit das GUI der Applikation nicht blockiert. Für 20 bis 299 *WP* eignet sich ebenfalls eine Berechnung im GUI Thread, aber unter der Verwendung der beiden Algorithmen 3-Opt und anschliessender Optimierung mittels dem Ant-Colony-Algorithmus. Alle Probleme mit 300 oder mehr *WP* werden im separaten Berechnungsthread mittels dem Nearest-Neighbour-Algorithmus berechnet.

15.1 Kritische Betrachtung der Resultate

Das resultierende Konzept funktioniert einwandfrei und kann im produktiven System eingesetzt werden. Wichtig anzumerken ist aber, dass die Architekturwahl bedeutend kleineren Einfluss auf die Berechnungsgeschwindigkeit und die benötigten Ressourcen hat als die Wahl des Algorithmus. So kann die korrekte Lösung für 10 *WP* mit dem exakten Algorithmus Held-Karp schneller als mit dem Branch-and-Bound-Algorithmus errechnet werden, da dieser Algorithmus eine grössere Effizienz aufweist. Dadurch könnte für die Umsetzung des Systems die Implementation des Branch-and-Bound-Algorithmus komplett weggelassen werden, da der Held-Karp-Algorithmus von 10 bis 19 *WP* genügend schnell die Lösung errechnet. Der Branch-and-Bound ist keinesfalls der schnellste Algorithmus zur Berechnung, jedoch wurde dieser aufgrund der einfachen Parallelisierbarkeit ausgewählt, damit die einzelnen Konzepte untereinander verglichen werden konnten. Um allenfalls eine weitere Geschwindigkeitssteigerung zu erzielen, sollte eine Parallelisierung des Held-Karp-Algorithmus in Betracht gezogen werden.

Eine weitere Schwäche der Umsetzung stellt die Nichtbeachtung der Browserperfomance dar. Die Geschwindigkeitsmessungen sind ausschliesslich auf den Testgeräten erfolgt, wodurch keinerlei Aussage über den Einsatz in einer produktiven Umgebung gemacht werden kann.

Abschliessend soll erwähnt werden, dass für die Geschwindigkeitsmessungen nur zufällige Werte verwendet wurden. Um genauere Aussagen über den Einsatz mit realen Daten machen zu können, müssten qualitativ bessere Testdaten erzeugt werden.

16 Produkt (R6)

Das finale Produkt für den Endkunden *EK*, für welches die Anforderungsanalyse in Kapitel 9 erarbeitet wurde, konnte erfolgreich implementiert werden und ist unter POC-6a-d zu finden. Da die Umsetzung nicht Teil der Bachelorarbeit ist, wird lediglich das Produkt, nicht aber der Quellcode publiziert.



Abb. 16.1: Screenshot der finalen Routenplanungsapplikation (Quelle: Eigene Darstellung)

Die Integration in die *CRM*-Applikation FlexBüro konnte ebenfalls erfolgreich umgesetzt werden und kann wie folgt eingesehen werden:

Um den Starter zu FlexBüro auf dem Demo-Server von Bachmann Support GmbH zu öffnen, wird eine FileMaker Pro 13 Voll- oder Demoversion benötigt. Eine Demoversion kann kostenlos geladen werden unter:

- Mac: http://www.flexbuero.ch/downloads/fmp13_demo.dmg
- Windows: http://www.flexbuero.ch/downloads/fmp13_demo.exe

Die Starterdatei kann von http://www.app-sync.com/api/1.0/additional_files/FlexBuero_Onlinedemo12_Starter.fmp12.zip heruntergeladen werden und muss nach dem Entzeppen mit dem zuvor installierten FileMaker geöffnet werden, um Zugriff auf die Demonstrationsversion von FlexBüro zu bekommen. Nach Erstellung eines neuen Benutzers kann über den Starter das Modul "Adressen" geöffnet werden. In der Listenansicht kann ganz unten das "Marker"-Symbol angeklickt werden, um für die aktuelle Adressselektion die *Marker* auf der Karte anzuzeigen. Danach kann im neuen Fenster mit der Kartenansicht die Route mittels Klick auf die Taste "Optimierung" verbessert werden.

17 Reflexion und Fazit

Die Ziele der vorliegenden Bachelorarbeit wurden ausnahmslos erreicht und die *POCs* konnten wie geplant umgesetzt werden. Zusätzlich konnte aus dem in der Arbeit erstellten Konzept das Endsystem abgeleitet und zur vollsten Zufriedenheit von *EK* und der Firma Bachmann Support GmbH umgesetzt werden. Dennoch weist das Konzept einige Verbesserungsmöglichkeiten auf. Obwohl vor allem die Algorithmenwahl einen grossen Einfluss auf die Berechnungsgeschwindigkeit des Algorithmus hat, wurde der Fokus relativ stark auf die Architektur gelegt. Begründet werden kann dieses Vorgehen dadurch, dass das Augenmerk hauptsächlich auf die Umsetzung der Algorithmen im Webumfeld und den technischen Möglichkeiten in diesem Bereich lag und nicht auf der Entwicklung neuer Algorithmen und deren Implementation. Obschon die Implementation mit einem anderen Algorithmus einiges effizienter arbeiten würde, konnten die gegebenen Anforderungen mit dem gewählten Vorgehen zufriedenstellend erfüllt werden. In der Umsetzung des finalen Konzepts und dem Endprodukt konnte diese Verbesserung bereits berücksichtigt und die Lösung weiter verbessert werden.

Die erarbeiteten Konzepte brachten für die Parallelisierung und Verteilung der Berechnung vorerst nicht die gewünschten Erfolge, weshalb das Konzept der Parallelisierung während der Umsetzung des *POCs* angepasst wurde, damit auch eine obere Grenze für die Anzahl der zu erstellenden Threads angegeben werden kann. Diese Anpassung führte zur gewünschten Geschwindigkeitssteigerung, welche in den anschliessenden Messungen bestätigt werden konnte. Aufgrund der Erkenntnisse aus der Parallelisierung schwand die Hoffnung auf eine hochperformante Lösung mittels einem verteilten System unter Clients, was dann auch im *POC* belegt wurde. Das *POC* des verteilten Systems unter Clients demonstriert aber dennoch die Umsetzbarkeit. Es zeigt, dass diese Architektur für Berechnungen, welche bei der Verteilung wenig Kommunikation benötigen, durchaus sinnvoll eingesetzt werden kann.

Während der Umsetzung der *POCs* wurde zudem festgestellt, dass das Generieren von Testdaten für Performancemessungen von grosser Wichtigkeit ist. Anfangs wählte ich eine Matrix voller zufälliger Werte und es wurde schnell sichtbar, dass bei den Berechnungszeiten grosse Unterschiede entstanden. Nach der Anpassung des Algorithmus zur Testdatengenerierung entsprachen die Ergebnisse den Erwartungen.

Eine weitere wichtige Erkenntnis stellt das Generieren der Distanzmatrix für eine grössere Anzahl an *WP* dar. Bei der Google Maps API kann nur eine Distanzmatrix für 10 *WP* angefordert werden. Dies bedeutet, dass die gesamte Distanzmatrix für alle *WP* durch mehrere Anfragen zusammengestellt werden muss, was zusätzlichen Zeitaufwand zur Folge hat. Bei *OSM* ist hingegen kein Limit angegeben.

Neben den gewonnenen Erkenntnissen wird auch klar, dass viele weiterführende Untersuchungsgebiete bestehen, welche im nachfolgenden Kapitel beschrieben werden.

18 Weiterführende Untersuchungsgebiete

Es verbleiben neben Verbesserungsvorschlägen auch weiterführende Untersuchungsgebiete. Die Untersuchung von verteilten Systemen auf verschiedenen Servern wurde in der Arbeit bewusst ausgeklammert, da Kosten für die benötigten Server anfallen würden und diese von Bachmann Support GmbH nicht getragen werden können. Falls aber ein eigenständiger, kostenpflichtiger Dienst für die Routenberechnung entwickelt werden soll, kann dies eine spannende Option darstellen, da damit ein gut skalierbares System entworfen werden kann. Auch das verteilte System unter Clients kann noch weiter verfeinert werden, indem eine maximale Anzahl an Clients für die Berechnung angegeben werden kann, wie dies beim Konzept 3, bei der Parallelisierung im Webbrowser, umgesetzt wurde. Dadurch würde der Kommunikationsoverhead reduziert, was gegebenenfalls zur gewünschten Verbesserung führt.

Für die Optimierung der Berechnung könnte die fortlaufende Optimierung während der Eingabe der *WP* eine Idee für eine grosse Anzahl *WP* sein. Dies würde aber eine manuelle Eingabe der *WP* erfordern und funktioniert nicht beim Import mehrerer *WP*. Auch würden viele unnötige Berechnungen erfolgen und viele Ressourcen benötigt, da für jeden eingegebenen *WP* das gesamte *TSP* gelöst werden muss. Eine andere Möglichkeit zur schnelleren Berechnung der Lösung stellt das Zusammenfassen von nahegelegenen *WP* dar, damit die Anzahl *WP* verringert werden kann.

Es soll ebenfalls erwähnt sein, dass auch für die gewählte Implementation des Branch-and-Bound-Algorithmus Optimierungen und Variationen möglich sind, um die Berechnungsgeschwindigkeit zu verbessern.

Für die Reduktion der Zeit der Distanzmatrixerzeugung könnten die Luftliniendistanzen anstelle des Streetlevelroutings genommen werden, da vor allem für weit auseinanderliegende *WP* diese Näherung genügend genau ist, um die Route optimieren zu können. Auch ein effizientes Caching der Distanzen und der Routenberechnungen könnte entscheidende Geschwindigkeitsgewinne bringen.

Auch für die automatische Algorithmen- und Architekturwahl während der Berechnung werden in Zukunft genauere Angaben im Browser zur Verfügung stehen. So ist in Chrome bereits ersichtlich, wie viele Kerne für die Berechnung auf dem Rechner vorhanden sind und wie lange die Seite benötigt, um dargestellt zu werden. Basierend darauf können neue Konzepte ausgearbeitet werden, damit auch für grössere Problemgrössen mit $n > 20$ *WP* und auf schwächeren Geräten die exakte Lösung des Problems ausreichend schnell gefunden werden kann.

Abschliessend kann mit Sicherheit gesagt werden, dass noch viele Optimierungsmöglichkeiten bestehen und im Rahmen dieser Arbeit nur ein Teil der Problematik beleuchtet werden konnte.

19 Schlusswort

Während dem halben Jahr, vom Kick-Off am 3. Dezember 2014 bis zur Abgabe der Arbeit am 14. Mai 2015, wurden wöchentlich mindestens zwei Arbeitstage in das Projekt investiert. Die Arbeit bereitete mir viel Freude und ich konnte viele Erfahrungen aus dem Studium und der Berufswelt in das Projekt einfließen lassen. Vor allem die Marktanalyse und ausführliche Anforderungsanalyse halfen, die nötigen Eingrenzungen für das zu entwickelnde System festzulegen.

Die ausführliche Zeitplanung zu Beginn der Arbeit ermöglichte eine gute Arbeitsteilung und die Ausführung erfolgte weitgehend plangemäss. Lediglich die *POCs* benötigten etwas mehr Zeit als erwartet und mussten zudem zeitlich etwas vorgezogen werden, damit die Ergebnisse für die Nutzwertanalyse bereit standen. Die Implementation des resultierenden Konzepts gestaltete sich dafür etwas leichter, da die zuvor erstellten *POCs* wieder verwendet werden konnten.

Trotz des Rückschlages, der nicht ganz so wie erhofft funktionierenden Parallelisierung und Verteilung der Berechnung, konnte eine gute Lösung gefunden werden, welche vollumfänglich funktionstüchtig ist und allen Anforderungen des Endkunden entspricht. Diese Erfahrung zeigt erneut, dass ein Umdenken innerhalb von Projekten nötig sein kann und es sinnvoll ist, alternative Lösungen einzubeziehen.

Am meisten Freude bereitete mir an der Bachelorarbeit, dass ich das gesamte Projekt mit einem Praxisbezug umsetzen konnte. Obwohl die finale Umsetzung des Systems nicht im Rahmen der Bachelorarbeit erfolgte, arbeitete ich während der gesamten Arbeit auf dieses Ziel hin, was mich sehr motivierte. Die erfolgreiche Fertigstellung des Systems bildet den krönenden Abschluss der sehr intensiven, aber auch interessanten Zeit während der Bachelorarbeit.

Teil V

Anhang

A Proof of Concepts (*POCs*)

Alle *POCs* sind sowohl auf dem beigelegten Datenträger (Anhang B) als auch auf der Webseite zur Arbeit¹ zu finden und in Abb. A.1 dargestellt. In der Arbeit werden die Referenzen zu den Beispielen jeweils mit POC-Ref angegeben. POC-1a verweist somit auf das Beispiel 1a der Proof of Concepts.

Ausser der Übersichtsseite sind alle *POCs* und die Dokumentation auf Englisch verfasst.

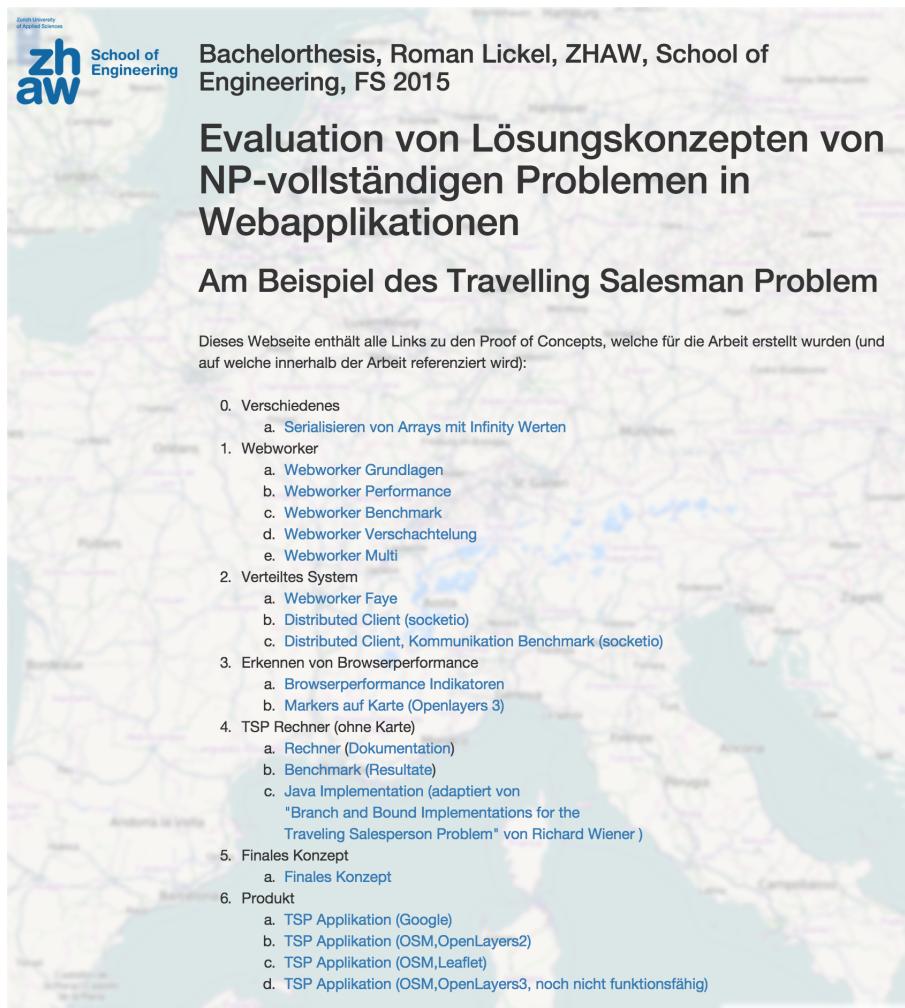


Abb. A.1: Zusammenstellung aller erarbeiteten *POCs* (Quelle: Eigene Darstellung)

¹<http://ba.klickagent.ch/>

B Quellcode

Der gesamte Quellcode der für die Arbeit erstellten *POCs* ist ebenfalls auf Github publiziert¹.

¹https://github.com/klickagent/atsp_solver

C Verwendete Soft- und Hardware

- Dokumentation
 - Texmaker 4.1 für Latex (TeXLive-2013, Intel 64)
 - draw.io
 - Excel for Mac 2011 (Diagramme, Auswertungen)
- Entwicklung
 - Espresso 2.2.2
 - Versionierung mit GIT¹
 - JavaScript Testing mit jasmine 2.2
 - JavaScript Dokumentation mit JSDoc 3
 - Google Chrome 42.0.2311.50 beta (64-bit)
 - FireFox Developer Edition 38.0a2 (2015-03-06)
 - Eclipse IDE for Java Developers, Version: Luna Service Release 1a (4.4.1)
- Technologiestack
 - JavaScript: Node.js v0.12.0, v1.9.1, chance.js v0.7.3 ², socket.io v1.3.5, OpenLayers 2, OpenLayers 3, Google Maps JavaScript API v3, LeafletJS 0.7.3, Optimap v4, Kris Kowal's queue v1.0.0 ³
 - Bootstrap v3.3.4 (CSS,JS)
- Geräte
 - MacBook Pro (Retina, 15-inch, Late 2013), Macos 10.10.2
 - 2 * VMware, Ubuntu, 64-bit 4 Kern Ubuntu in Kanada (CloudAtCost ⁴)
- Netzwerk
 - MacBook Pro über Wireless-Lan (AirPort Extreme, 802.11g) am Internet angeschlossen. Verbindungsgeschwindigkeit des Providers Swisscom: Maximal 10 Mbit/s Download, Maximal 2 Mbit/s Upload (gemessen ⁵: 10.529 Mbit/s Download und 1.396 Mbit/s Upload)
 - Internetanbindung der CloudAtCost-Server: unbekannt

¹<http://github.com>

²<http://chancejs.com/>

³<https://github.com/kriskowal/q>

⁴<http://www.cloudatcost.com/>

⁵<http://www.wieistmeineip.ch/speedtest/ergebnis.php>

D Verzeichnisse

D.1 Glossar

AJAX siehe Abschnitt 5.1.1.1.

Anfrage Eine Anfrage stellt die Anfrage eines Clients an einen Server im Client-Server-Modell dar. Darin werden Inhalte oder andere *Ressourcen* angefordert.

API API steht für application programming interface und bezeichnet eine Programmierschnittstelle.

asynchron Die Definition von asynchron lautet: nicht synchron, nicht gleichzeitig, nicht mit gleicher Geschwindigkeit [ab]laufend.¹

ATSP Asymmetrisches Travelling Salesman Problem, siehe Abschnitt 3.2.

Callback Funktion Eine Callback Funktion ist eine Rückruf-Funktion. Dieses Programmiermuster wird in JavaScript eingesetzt um Funktionen als Parameter mitzugeben, welche innerhalb der aufgerufenen Funktion ausgeführt werden.

Chrome Chrome ist ein Webbrowser von der Firma Google.

CPU CPU ist die Abkürzung für central processing unit, also die zentrale Recheneinheit eines Computers oder einem anderen elektronischen Gerät.

CRM CRM ist die Abkürzung für customer relationship management und bezeichnet die Verwaltung und den Umgang mit Kundeninformationen.

CSS CSS steht für cascading style sheets. Diese Sprache wird für das Styling von HTML Dokumenten verwendet.

Debuggen Der Prozess der Fehlerfindung in Programmen wird Debugging bezeichnet. Der Programmcode wird dabei mittels einem Hilfstoß, dem Debugger, schrittweise durchlaufen, wobei auf den Callstack (abfolge der Funktionen) und die zu diesem Zeitpunkt definierten Variablen zugegriffen werden kann um den Fehler zu finden. Das Verb debuggen bedeutet einen Programmfehler in einem Softwareprogramm beseitigen.²

EBS Das EBS ist das Webportal der ZHAW (Filiale Zürich).

EK EK ist die Abkürzung für Endkunde. Gemeint ist der Endkunde der Firma Bachmann Support GmbH, welcher das Produkt in Auftrag gegeben hat.

¹<http://www.duden.de/suchen/dudenonline/asynchron>

²<http://www.duden.de/rechtschreibung/debuggen>

FileMaker FileMaker ist ein Datenbanksystem, das auf Windows und MacOs betrieben werden kann.

Firefox Firefox ist ein Webbrowser der Mozilla Corporation.

FlexBüro FlexBüro ist eine von Bachmann Support GmbH entwickelte CRM-Lösung und basiert auf *FileMaker*.

Google Maps Google Maps ist ein Produkt der Google Inc. Es bietet Kartenmaterial sowie diverse Frameworks für die Einbindung der Karten in Applikationen.

GUI Die Abkürzung GUI steht für graphical user interface. Es stellt die grafische Benutzeroberfläche dar und bildet somit die Schnittstelle zwischen einem System und dem Anwender/Benutzer.

Heuristik siehe Abschnitt 3.3.

HTML siehe Abschnitt 5.2.

HTTP siehe Abschnitt 5.1.1.

HTTPS siehe Abschnitt 5.1.1.

IP IP steht für Internet Protokoll und ist Teil des *OSI-Modells*. Es ist auf Stufe 3 anzusiedeln.

Map Client siehe Abschnitt 7.3.

Marker Als Marker werden in der Arbeit die Standortmarken für die einzelnen *WP* auf der Karte bezeichnet.

Node Eine Node ist ein Knoten. Es kann sich je nach Kontext sowohl um ein Element in einem Baum handeln, als auch um ein Element in einer Netzwerkstruktur.

NP-vollständig siehe Kapitel 3.

Opensource Opensource bedeutet, dass der Quellcode frei zur Verfügung und unter eigens dafür erstellten Lizenzvereinbarungen angeboten wird.

OSI-Modell OSI steht für open system interconnection. Das OSI-Modell definiert ein Framework für Netzwerke, basiert auf 7 Stufen.

Peer Das englische Wort "Peer" bedeutet auf Deutsch "der Ebenbürtige". Ein Peer ist ein Client-Rechner im Peer-To-Peer-Modell.

POC POC ist die Abkürzung für Proof of Concept und stellt beispielsweise ein Prototyp einer Applikation dar, um gewisse Konzepte zu beweisen.

Problem des Handelsreisenden siehe *Travelling Salesman Problem*.

Request siehe *Anfrage*.

Ressource Je nach Kontext bedeutet Ressource Rechenleistung oder es steht für einen Service, einen Dienst, Daten, oder einen Drucker.

REST siehe Abschnitt 6.2.

Routing Routing steht für die Berechnung der Wegstrecke zwischen zwei *WP*. Es ist verwandt mit dem deutschen Wort Route, stammt vom Englischen und bedeutet übersetzt Wegweiser respektive Routenplanung.

Service Ein Service ist ein Dienst von einem anderen Anbieter.

Stack Stack steht für Stapel und wird in der IT für die Sammlung aller benötigten Komponenten für ein System verwendet.

Stakeholder Personen und Firmen, welche in irgendeiner Weise Einfluss auf das System haben, oder das System in irgendeiner Weise benutzen, oder darauf zugreifen.

STSP Symmetrisches Travelling Salesman Problem, siehe Abschnitt 3.2.

Thread Ein Thread (englisch für "Faden") ist ein leichtgewichtiger Prozess.

Tier Physikalische Schichten, welche Softwareabstraktionsstufen der Architektur darstellen.

Travelling Salesman Problem Das Travelling Salesman Problem ist ein Optimierungsproblem. Mehr Informationen sind im Abschnitt 3.2 zu finden.

TSP siehe *Travelling Salesman Problem*.

Webworker Siehe Abschnitt 6.3.1.

WP Wegpunkt

D.2 Abbildungsverzeichnis

1.1	Lösungsvorgehen (<i>Quelle: Eigene Darstellung</i>)	4
2.1	Zeitplanung als Gantt-Diagramm (<i>Quelle: Eigene Darstellung</i>)	6
3.1	Branch-and-Bound Tree Space mit $n = 3$ (<i>Quelle: Eigene Darstellung</i>)	11
3.2	Ansatz des Branch-and-Bound Tree Space mit $n = 5$ (<i>Quelle: Eigene Darstellung</i>)	11
4.1	Das Geoid (<i>Quelle: Kale [39]</i>)	13
4.2	Längen- und Breitengrade (<i>Quelle: Illinois State University [37]</i>)	14
5.1	Mengendiagramm der Begriffe URL, URI, URN (<i>Quelle: David P. Heitmeyer [12]</i>)	16
5.2	Erfolgreiche HTTP-GET Anfrage (<i>Quelle: Dunkel et al. [15]</i>)	17
5.3	Vergleich nicht-persistente und persistente <i>HTTP</i> Verbindung (<i>Quelle: Serpanos and Wolf [66]</i>)	17
7.1	Karten Unterteilung bei Kartenservern (<i>Quelle: Bing [7]</i>)	25
8.1	Anforderungskonflikt Berechnung des <i>TSP</i> im Webumfeld (<i>Quelle: Eigene Darstellung</i>)	28
9.1	Kontextabgrenzung des Systems (<i>Quelle: Eigene Darstellung</i>)	32
9.2	Use Case-Diagramm (<i>Quelle: Eigene Darstellung</i>)	36
9.3	Mockup der Applikation (<i>Quelle: Eigene Darstellung</i>)	37
11.1	Symbollegende für die Konzepte (<i>Quelle: Eigene Darstellung</i>)	46
11.2	Entscheidungsdiagramm (K1) zur Zerlegung von <i>NP-vollständigen</i> Problemen (<i>Quelle: Eigene Darstellung</i>)	47
11.3	Entscheidungsdiagramm (K2) zur Ermittlung der Architektur zur Lösung von <i>NP-vollständigen</i> Problemen in Webapplikationen (<i>Quelle: Eigene Darstellung</i>)	48
12.1	Messungen der Performanz von <i>Webworkers</i> (gemessen auf einem MacBook, gemäss Anhang, mit POC-1c) (<i>Quelle: Eigene Darstellung, mit Excel</i>)	51
12.2	Kommunikationsdiagramm des Konzepts für die gemeinsame Datennutzung bei verteilten Systemen mit dem Publisher-Subscriber-Pattern (<i>Quelle: Eigene Darstellung</i>)	53
12.3	Performance-Messungen der Nachrichtenübertragung bei verteilten Clients aus POC-2c (<i>Quelle: Eigene Darstellung, mit Excel</i>)	55
12.4	Parallele/Verteilte Berechnung des <i>TSP</i> mit Branch-and-Bound mit $n = 4$ (<i>Quelle: Eigene Darstellung</i>)	57
12.5	Bausteinsicht des Konzept 1 (<i>Quelle: Eigene Darstellung</i>)	61
12.6	Bausteinsicht des Konzepts 2 (<i>Quelle: Eigene Darstellung</i>)	61
12.7	Bausteinsicht des Konzepts 3 (<i>Quelle: Eigene Darstellung</i>)	62
12.8	Bausteinsicht des Konzepts 4 (<i>Quelle: Eigene Darstellung</i>)	63
12.9	Bausteinsicht des Konzepts 4 (<i>Quelle: Eigene Darstellung</i>)	64
12.10	Bausteinsicht des Konzepts 6 (<i>Quelle: Eigene Darstellung</i>)	66
13.1	Der <i>TSP-Rechner</i> aus POC-4a (<i>Quelle: Eigene Darstellung</i>)	69
13.2	Firefox DevTools Debugging (<i>Quelle: Eigene Darstellung</i>)	72
13.3	Chrome DevTools Profiling (<i>Quelle: Eigene Darstellung</i>)	72

13.4 Dokumentation generiert mit JSDoc aus den Kommentaren des Quellcodes (<i>Quelle: Eigene Darstellung</i>)	73
13.5 Testing mit dem Framework jasmine (<i>Quelle: Eigene Darstellung</i>)	73
15.1 Resultierendes Entscheidungsdiagramm zur Ermittlung der Architektur (<i>Quelle: Eigene Darstellung</i>)	82
16.1 Screenshot der finalen Routenplanungsapplikation (<i>Quelle: Eigene Darstellung</i>)	84
A.1 Zusammenstellung aller erarbeiteten <i>POCs</i> (<i>Quelle: Eigene Darstellung</i>)	89

D.3 Tabellenverzeichnis

2.1 Administrative Termine der Bachelorarbeit (<i>Quelle: Eigene Darstellung</i>)	5
2.2 Grobe Aufwandsschätzung (<i>Quelle: Eigene Darstellung</i>)	5
6.1 Datenelemente von REST, übersetzt in Deutsch aus Fielding [18]	23
9.1 Stakeholderliste (<i>Quelle: Eigene Darstellung</i>)	31
9.2 Funktionale Anforderungen an das System (<i>Quelle: Eigene Darstellung</i>)	34
9.3 Nicht-funktionale Anforderungen an das System (<i>Quelle: Eigene Darstellung</i>)	35
9.4 Karten- und Routinganbieter mit <i>API</i> für Webapplikationen (<i>Quelle: Eigene Darstellung</i>)	38
9.5 Eintrittswahrscheinlichkeitsstufen der Projektrisiken (<i>Quelle: Hindel et al. [34]</i>)	39
9.6 Charakteristik der Schadenshöhe (<i>Quelle: Eigene Darstellung</i>)	40
9.7 Risikoliste (<i>Quelle: Eigene Darstellung</i>)	41
9.8 Risikoportfolio (<i>Quelle: Eigene Darstellung</i>)	41
10.1 Marktübersicht (<i>Quelle: Eigene Darstellung</i>)	43
10.1 Marktübersicht (<i>Fortsetzung</i>)	44
12.1 Auswertung der <i>Webworker</i> Performance-Messungen (gemessen auf einem MacBook, gemäss Anhang, mit POC-1c) (<i>Quelle: Eigene Darstellung</i>)	52
12.2 Auswertung der Performance-Messungen der Nachrichtenübertragung bei verteilten Clients aus POC-2c (<i>Quelle: Eigene Darstellung</i>)	55
12.3 Vergleich einiger Algorithmen zur Lösung des <i>TSP</i> (<i>Quelle: Eigene Darstellung</i>)	56
13.1 Gegenüberstellung der Geschwindigkeit des Algorithmus in Java und JavaScript (<i>Quelle: Eigene Darstellung</i>)	74
14.1 Bewertungskriterien Nutzwertanalyse (<i>Quelle: Eigene Darstellung</i>)	76
14.2 Nutzwertanalyse der Konzepte aus Abb. 11.2 und Abb. 11.3 (<i>Quelle: Eigene Darstellung</i>)	77
15.1 Anwendung des Konzepts auf das <i>TSP</i> (<i>Quelle: Eigene Darstellung</i>)	83

D.4 Quellenverzeichnis

- [1] Alexis Deveria (2014). Can I use... Support tables for HTML5, CSS3, etc. <http://caniuse.com/#feat=webworkers>. [Online; Zugriff 16.02.2015].
- [2] Baidu (2015). <http://developer.baidu.com/map/>. [Online; Zugriff 05.01.2015].
- [3] Barbara Wiley, National Geospatial-Intelligence Agency, United States of America. GPS Geodetic Reference System, WGS 84. <http://www.unoosa.org/pdf/icg/2009/icg-4/37d.pdf>. [Online; Zugriff 09.03.2015].
- [4] Barcelona Field Studies Centre S.L. (2013). Google Maps Projection. <http://googlecompass.com/GoogleMapsProjection.htm>. [Online; Zugriff 09.03.2015].
- [5] Berners-Lee, T. (1994). Universal Resource Identifiers in WWW. <http://tools.ietf.org/html/rfc1630>. CERN, [Online; Zugriff 03.01.2015].
- [6] Berry, K. (2009). Distributed and grid computing via the browser. http://www.csc.villanova.edu/~tway/courses/csc3990/f2009/csrs2009/Kevin_Berry_Grid_Computing_CSRS_2009.pdf. [Online; Zugriff 16.02.2015].
- [7] Bing, M. Bing maps tile system. <https://msdn.microsoft.com/en-us/library/bb259689.aspx>. [Online; Zugriff 15.02.2015].
- [8] Bing, Microsoft (2015). <http://www.microsoft.com/maps/default.aspx>. [Online; Zugriff 05.01.2015].
- [9] Cirasella, J., Johnson, D. S., McGeoch, L. A., and Zhang, W. (2000). The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators, and Tests. Online.
- [10] Clausen, J. (1999). Branch and bound algorithms - principles and examples. http://janders.eecg.toronto.edu/1387/readings/b_and_b.pdf. [Online; Zugriff 20.01.2015].
- [11] Consortium, O. G. (2010). Kml. <http://www.opengeospatial.org/standards/kml/>. [Online; Zugriff 16.02.2015].
- [12] David P. Heitmeyer (2015). Names and Locations: URLs, URIs, and URNs. http://cscis12.dce.harvard.edu/lecture_notes/2009/20090623/slides24.html. [Online; Zugriff 15.02.2015].
- [13] Drivingrouteplanner. <http://www.drivingrouteplanner.com/>. [Online; Zugriff 30.11.2014].
- [14] DrScholl (2001). OpenNap: Open Source Napster Server. <http://opennap.sourceforge.net/>. [Online; Zugriff 03.01.2015].
- [15] Dunkel, J., Eberhart, A., Fischer, S., Kleiner, C., and Koschel, A. (2008). *System-Architekturen für verteilte Anwendungen*. Hanser.
- [16] ESRI (2015). <http://www.esri.com/software/arcgis/arcgisonline>. [Online; Zugriff 05.01.2015].
- [17] Fielding, R. T. (2000a). Architectural Styles and the Design of Network-based Software Architectures. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. University of California, IRVINE, [Online; Zugriff 30.12.2014].
- [18] Fielding, R. T. (2000b). Architectural Styles and the Design of Network-based Software Architectures. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. University of California, IRVINE, [Online; Zugriff 30.12.2014].
- [19] Findthebestroute. <http://findthebestroute.com/>. [Online; Zugriff 30.11.2014].
- [20] Gall, N. (2014). WOA: Putting the Web Back in Web Services. http://blogs.gartner.com/nick_gall/2008/11/19/woa-putting-the-web-back-in-web-services/, <http://www.gartner.com/it-glossary/web-oriented-architecture-woa>. [Online; Zugriff 30.12.2014].

- [21] Gao, Y., Chen, L., Chen, G., and Chen, C. (2006). Efficient parallel processing for k-nearest-neighbor search in spatial databases.
- [22] Garey, M. R. and Johnson, D. S. (1978). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman.
- [23] Garey, M. R. and Johnson, D. S. (2010). *P, NP, and NP-Completeness The Basics of Computational Complexity*. Cambridge University Press.
- [24] German Carrillo (2012). <http://geotux.tuxfamily.org/index.php/en/geo-blogs/item/291-comparacion-clientes-web-v6>. [Online; Zugriff 05.01.2015].
- [25] Google (2010). Issue 31666. <https://code.google.com/p/chromium/issues/detail?id=31666>. [Online; Zugriff 16.02.2015].
- [26] Google (2015). <https://developers.google.com/maps/documentation/business/>. [Online; Zugriff 05.01.2015].
- [27] Google Inc. Google Earth projection - Earth Help. <https://support.google.com/earth/answer/148110?hl=en>. [Online; Zugriff 09.03.2015].
- [28] Grigorik, I. (2013). *High Performance Browser Networking: What Every Web Developer Should Know about Networking and Web Performance*. O'Reilly Media, Inc.
- [29] Gross, J. L., Yellen, J., and Zhang, P. (2013). *Handbook of Graph Theory, Second Edition*. CRC Press.
- [30] Gutin, G. (2007). *The traveling salesman problem and its variations*. Springer.
- [31] Gutin, G. and Punnen, A. P. (2002). *The Traveling Salesman Problem and Its Variations*. Springer Science and Business Media.
- [32] Held, M. and Karp, R. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. mathematical programming 1.
- [33] HERE (2015). <https://developer.here.com/>. [Online; Zugriff 05.01.2015].
- [34] Hindel, B., Hörmann, K., Müller, M., and Schmied, J. (2009). *Basiswissen Software-Projektmanagement*. dpunkt.verlag.
- [35] Hussmann, S. and Lutz-Westphal, B. (2007). *Kombinatorische Optimierung erleben: In Studium und Unterricht*. Springer.
- [36] I. Fette and A. Melnikov (2011). RFC 6455 - The WebSocket Protocol. <https://tools.ietf.org/html/rfc6455>. [Online; Zugriff 08.03.2015].
- [37] Illinois State University (2015). <http://education.illinoisstate.edu/>. [Online; Zugriff 15.03.2015].
- [38] Joyent, Inc (2014). Node.js v0.12.0 Manual & Documentation. https://nodejs.org/api/child_process.html. [Online; Zugriff 09.03.2015].
- [39] Kale, H.-G. (2008). *Einführung in die höhere Geodäsie*. ETH Zürich.
- [40] Lastovetsky, A. L. (2003). *Parallel Computing on Heterogeneous Networks*. John Wiley & Sons.
- [41] Mapquest. <http://www.mapquest.com/>. [Online; Zugriff 30.11.2014].
- [42] Marco, D., Gambardella, and Maria, L. (1997). Ant colonies for the travelling salesman problem. *BioSystems*, 43(2):73–81.
- [43] Matsudaira, K. (2012). Scalable web architecture and distributed systems. <http://www.aosabook.org/en/distsys.html>. [Online; Zugriff 16.02.2015].
- [44] Michiels, W., Aarts, E., and Korst, J. (2007). *Theoretical Aspects of Local Search*. Springer Science and Business Media.
- [45] MIT (2003). Big O Notation. http://web.mit.edu/16.070/www/lecture/big_o.pdf.
- [46] Monmonier, M. (2010). *Rhumb Lines and Map Wars: A Social History of the Mercator Projection*. University of Chicago Press.

- [47] Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., and Ouksel, A. M. (2007). *Databases, Information Systems, and Peer-to-Peer Computing: International Workshops, DBISP2P 2005/2006, Trondheim, Norway, August 28-29, 2006, Revised ... Applications, incl. Internet/Web, and HCI*. Springer.
- [48] Multiroute. <https://www.multiroute.de/?locale=en>. [Online; Zugriff 30.11.2014].
- [49] NASA (2015). <http://worldwind.arc.nasa.gov/>. [Online; Zugriff 05.01.2015].
- [50] Neumann, F., Sudholt, D., and Witt, C. (2009). Computational complexity of ant colony optimization and its hybridization. <http://ls2-www.cs.tu-dortmund.de/~sudholt/chapterAC009.pdf>. [Online; Zugriff 20.01.2015].
- [51] Onetimehub. <http://www.onetimehub.com/Dispatcher>. [Online; Zugriff 30.11.2014].
- [52] Openstreetmap. Projection - OpenStreetMap Wiki. <http://wiki.openstreetmap.org/wiki/Projection>. [Online; Zugriff 09.03.2015].
- [53] Openstreetmap (2015). Routing/online routers. http://wiki.openstreetmap.org/wiki/Routing/online_routers. [Online; Zugriff 16.02.2015].
- [54] Optimap (2014). <http://gebweb.net/optimap/>. [Online; Zugriff 30.11.2014].
- [55] Oram, A. (2001). *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly Media.
- [56] O'Reilly, T. (2009). *What is web 2.0*. O'Reilly Media, Inc.".
- [57] osgeo. The open source geospatial foundation. <http://www.osgeo.org/>. [Online; Zugriff 15.02.2015].
- [58] OSM (2015). <http://www.openstreetmap.org/copyright>. [Online; Zugriff 05.01.2015].
- [59] Parviainen, S. (2001). Example of dynamic programming algorithm for the tsp. <http://www.mafy.lut.fi/study/DiscreteOpt/tspd.pdf>.
- [60] R. Fielding (1999). HTTP/1.1: Status Code Definitions. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>. [Online; Zugriff 08.03.2015].
- [61] R. Fielding and J. Gettys and J. Mogul and H. Frystyk and L. Masinter and P. Leach and T. Berners-Lee (1999). Hypertext Transfer Protocol – HTTP/1.1. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>. UC Irvine, Compaq/W3C, Compaq, W3C/MIT, Xerox, Microsoft, W3C/MIT, [Online; Zugriff 03.01.2015].
- [62] Routexl (2014). <http://www.routexl.com/>. [Online; Zugriff 30.11.2014].
- [63] Routino (2014). <http://www.routino.org/uk-leaflet/router.html>. [Online; Zugriff 30.11.2014].
- [64] Schull, A. and Springer, T. (2012). *Verteilte Systeme*. Springer Vieweg, Springer Verlag.
- [65] Schulte, R. (2003). *The Growing Role of Events in Enterprise Applications, Technischer Bericht*. Gartner Research.
- [66] Serpanos, D. and Wolf, T. (2011). *Architecture of Network Systems*. Elsevier.
- [67] Shinjo, Y., Guo, F., Kaneko, N., Matsuyama, T., Taniuchi, T., and Sato, A. (2011). A distributed web browser as a platform for running collaborative applications. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6144813>. [Online; Zugriff 16.02.2015].
- [68] Simha, R. The traveling salesman problem (tsp). <http://www.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html>. [Online; Zugriff 20.01.2015].
- [69] Speedyroute. <https://www.speedyroute.com/>. [Online; Zugriff 30.11.2014].
- [70] Stevens, W. R. and Rago, S. A. (2013). *Advanced Programming in the UNIX Environment, 3rd Edition*. Pearson Education, Inc.

- [71] Tanenbaum, A. and Steen, M. V. (2007). *Verteilte Systeme - Prinzipien und Paradigmen*. Pearson Studium, 2. Auflage.
- [72] Tennant, R. (1992). *Internet basics*. ERIC Clearinghouse on Information and Technology.
- [73] Tiburcio, P. G. S. and Spohn, M. A. (2010). ad hoc Grid: An Adaptive and Self-Organizing Peer-to-Peer Computing Grid. 10th IEEE International Conference on Computer and Information Technology.
- [74] TOMTOM (2015). <http://developer.tomtom.com/docs/read/Home>. [Online; Zugriff 05.01.2015].
- [75] TourPl. <http://www.tourpl.ch/>. [Online; Zugriff 30.11.2014].
- [76] Universität Heidelberg (2014). <http://koenigstuhl.geog.uni-heidelberg.de/ors/>. [Online; Zugriff 30.11.2014].
- [77] URI Planning Interest Group, W3C/IETF (2001). URIs, URLs, and URNs: Clarifications and Recommendations 1.0. <http://www.w3.org/TR/uri-clarification/>. CERN, [Online; Zugriff 03.01.2015].
- [78] Verhoeven, M., Aarts, E., and Swinkels, P. (1995). A parallel 2-opt algorithm for the traveling salesman problem. http://ac.els-cdn.com/0167739X9400059N/1-s2.0-0167739X9400059N-main.pdf?_tid=9a877184-d15d-11e4-b859-0000aacb35f&acdnat=1427116248_802e4a4335d104943e0ab334f60e9dce. [Online; Zugriff 20.01.2015].
- [79] Viamichelin (2015). <http://dev.viamichelin.com/tutorial-rest.html>. [Online; Zugriff 05.01.2015].
- [80] W3Schools (2014). HTML5 Web Workers. http://www.w3schools.com/html/html5_webworkers.asp. [Online; Zugriff 16.02.2015].
- [81] Wiener, R. (2003a). Branch and bound implementations for the traveling salesperson problem | part 1: A solution with nodes containing partial tours with constraints. http://www.jot.fm/issues/issue_2003_03/column7.pdf. [Online; Zugriff 20.01.2015].
- [82] Wiener, R. (2003b). Branch and bound implementations for the traveling salesperson problem | part 2: Single threaded solution with many inexpensive nodes. http://www.jot.fm/issues/issue_2003_05/column7.pdf. [Online; Zugriff 20.01.2015].
- [83] Wiener, R. (2003c). Branch and bound implementations for the traveling salesperson problem | part 3: Multi-threaded solution with many inexpensive nodes. http://www.jot.fm/issues/issue_2003_07/column8.pdf. [Online; Zugriff 20.01.2015].
- [84] Wiener, R. (2003d). Branch and bound implementations for the traveling salesperson problem | part 4: Distributed processing solution using rmi. http://www.jot.fm/issues/issue_2003_11/column5.pdf. [Online; Zugriff 20.01.2015].
- [85] Wiener, R. (2008). Surbhi chaudhry. <https://parasol.tamu.edu/people/surbhic/NNS.pdf>. [Online; Zugriff 20.01.2015].
- [86] Wikipedia (2015). Comparison of server-side JavaScript solutions - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Comparison_of_server-side_JavaScript_solutions. [Online; Zugriff 15.02.2015].
- [87] Williamson, D. P. (1990). Analysis of the held-karp heuristic for the traveling salesman problem.

E Eigenständigkeitserklärung

Hiermit bestätigt der Unterzeichnende, dass die Bachelorarbeit mit dem Thema "Evaluation von Lösungskonzepten von NP-vollständigen Problemen der Informatik in Webapplikationen am Beispiel des Travelling Salesman Problems." gemäss freigegebener Aufgabenstellung, mit Freigabe vom 12. November 2014, ohne jede fremde Hilfe im Rahmen des gültigen Reglements selbstständig ausgeführt wurde.

Uster, den 10. Mai 2015

Roman Lickel