

Johannes Schmidt
Thomas Riechert
Sören Auer
(Hrsg.)

SKIL 2012
Dritte Studentenkonzferenz Informatik Leipzig

Dritte Studentenkonzferenz Informatik Leipzig 2012
Leipzig, Deutschland, 25. September 2012
Tagungsband

Leipziger Beiträge zur Informatik: Band XXXIV

Herausgeber

Johannes Schmidt
Thomas Riechert
Sören Auer

{jschmidt|riechert|auer}@informatik.uni-leipzig.de

Universität Leipzig
Institut für Informatik
Abteilung Betriebliche Informationssysteme
Augustusplatz 10
04109 Leipzig

Institut für Angewandte Informatik (InfAI) e.V.
an der Universität Leipzig
Grimmaische Str. 12
04109 Leipzig

Series Editor

Klaus-Peter Fähnrich

Weitere Informationen zur Studentenkonzferenz Informatik Leipzig finden Sie im Internet unter der Adresse
<http://skil.informatik.uni-leipzig.de>.

SKIL 2012 – Dritte Studentenkonzferenz Informatik Leipzig 2012

Leipzig, Deutschland, 25. September 2012

Tagungsband

Johannes Schmidt, Thomas Riechert, Sören Auer (Hrsg.) – Leipzig, 2012

Der Einband wurde von Maxi Bley gestaltet und mit der Schriftart Yanone Kaffeesatz (Creative-Commons-Lizenz) von Jan Gerner gesetzt.

ISBN 978-3-941608-21-4

Konferenzorganisation (Chair)

Johannes Schmidt, Thomas Riechert, Sören Auer
E-Mail: skil@informatik.uni-leipzig.de

Programmkomitee

Natanael Arndt (Student, Universität Leipzig)
Christoph Augenstein (Universität Leipzig)
Alexander Bau (HTWK Leipzig)
Michael Becker (Universität Leipzig)
Thorsten Berger (Universität Leipzig)
Stefan Bohn (Universität Leipzig)
Henry Borasch (Student, HTWK Leipzig)
Martin Clauß (Universität Leipzig)
Tilo Dietrich (Universität Leipzig)
Thomas Efer (Universität Leipzig)
Ivan Ermilov (Universität Leipzig)
Marvin Fromhold (Unister GmbH)
Michael Hartung (Universität Leipzig)
Olaf Herden (Duale Hochschule Baden-Württemberg Stuttgart)
Walter Hower (Hochschule Albstadt-Sigmaringen)
Axel Hummel (Universität Leipzig)
Martin Junghans (Student, Universität Leipzig)
Stephan Klingner (Universität Leipzig)
Robert Kunkel (Universität Leipzig)
Ralf Laue (Westsächsische Hochschule Zwickau)
Klaus Lyko (Student, Universität Leipzig)
Kyrill Meyer (Universität Leipzig)
Lars-Peter Meyer (Universität Leipzig)
Richard Müller (Universität Leipzig)
Florian Pilz (Student, HTWK Leipzig)
Rubén Navarro Piris (Unister GmbH)
Martin Roth (Universität Leipzig)
Thomas Schmid (Universität Leipzig)
Michael Schmidt (ipoque GmbH)
Peter Scholz (Universität Leipzig)
Frank Schumacher (Universität Leipzig)
Antonius van Hoof (Duale Hochschule Baden-Württemberg Stuttgart)
Stefan Veit (itCampus Software- und Systemhaus GmbH)

Organisatoren und Durchführung der SKIL 2012

Universität Leipzig

UNIVERSITÄT LEIPZIG

Institut für Angewandte Informatik (InfAI)
e.V. an der Universität Leipzig



Forschungsgruppe Service Science
and Technology



Forschungsgruppe Agile Knowledge
Engineering and Semantic Web



Unterstützer der SKIL 2012

Gesellschaft für Informatik e.V. (GI)



Fachschaftsrat Wirtschaftswissenschaften
– Universität Leipzig



Fachschaftsrat Informatik
– Universität Leipzig



Sponsoren der SKIL 2012

Wir bedanken uns sehr herzlich bei allen Sponsoren für ihre Unterstützung.

Senacor Technologies AG



ECG Erdgas-Consult GmbH



O'Reilly Verlag & Co.KG



Unister Holding GmbH



Provedo Software GmbH



Leipziger Informatik-Verbund (LIV)
an der Universität Leipzig



Vorwort zum Tagungsband der SKIL 2012

In der Buchreihe „Leipziger Beiträge zur Informatik“ erscheinen Berichte aus Forschungsvorhaben, Herausgeberbände im Bereich innovativer und sich etablierender Forschungsgebiete, Habilitationsschriften und Dissertationen sowie Konferenz-Proceedings und herausragende studentische Arbeiten. Der Wert dieser durch den „Leipziger Informatik Verbund“ (LIV) als Zusammenschluss und Interessenverbund verschiedener Informatik-Einrichtungen im Jahr 2003 begründeten Reihe liegt darin, zeitnah und umfassend über abgeschlossene oder laufende wissenschaftliche Arbeiten sowie über neu entstehende Forschungsfelder zu berichten. Die Reihe stellt die innovative Themenvielfalt in den Herausgeberbänden neben die hohe wissenschaftliche Durchdringung in Habilitationen und Dissertationen. Zudem ergänzt sie forschungsrelevante Bereiche mit praxisorientierten technischen Beiträgen und Dokumentationen.

Die verschiedenen Beiträge entstammen dem Hintergrund der Angewandten Informatik und Wirtschaftsinformatik. Schwerpunkte liegen dabei in den Bereiche betriebliche Informationssysteme, Content- und Wissensmanagement sowie IT-gestützter intra- und interorganisationale Kooperation mittels Internet-Technologien. Besondere Beachtung findet auch – entsprechend seiner Bedeutung für den Standort – der Dienstleistungssektor, wobei ein besonderer Schwerpunkt auf Dienstleistungen mit einem hohen IT-Anteil gelegt wird.

Seit dem Jahr 2007 kooperiert der LIV in der inhaltlichen Ausgestaltung der Reihe mit dem Institut für Angewandte Informatik e.V. (InfAI), welches als An-Institut an der Universität Leipzig die gemeinnützige Förderung von Wissenschaft und Forschung auf den Gebieten der Informatik und der Wirtschaftsinformatik verfolgt.

Der vorliegende 34. Band der Reihe enthält die Tagungsbeiträge der Dritten Studentenkonzferenz Informatik Leipzig 2012 (SKIL 2012). Die Studentenkonzferenz des Instituts für Informatik der Universität Leipzig richtet sich an alle Studierende der Informatik sowie angrenzender Disziplinen mit Schwerpunkt Informatik. Die Konferenz bietet Studierenden eine Plattform, ihre Projekte und Forschungsvorhaben einem akademischen Publikum vorzustellen. Die Organisation der Konferenz unterscheidet sich nur wenig von wissenschaftlichen Fachkonferenzen. Die Einreichung der Beiträge erfolgte mit Hilfe eines Conference Systems. Die in diesem Buch vorgestellten Beiträge wurden durch das Programmkomitee im Peer-Review bewertet und am Konferenztag in einem Vortrag präsentiert.

Leipzig, den 25. September 2012

Klaus-Peter Fähnrich (Series Editor)
Sprecher des LIV



ECG Erdgas-Consult

ECG. ERFOLG PROGRAMMIERT.

SOFTWARE FÜR DEN ENERGIEMARKT



TALENT
WANTED!

ECG. Bahnbrechende Softwarelösungen sind unsere Stärke, kreative Köpfe die Basis unseres Erfolges. Als eines der führenden Systemhäuser der Energiebranche realisieren wir individuelle Beratungs- und Softwareprojekte in ganz Europa. Ideenreichtum, Selbstverwirklichung und kontinuierliche Weiterbildungsmöglichkeiten beeinflussen dabei unsere Arbeit in einem jungen dynamischen Team und lassen uns gemeinsam erfolgreiche Kundenlösungen umsetzen.

WIR SUCHEN.

- Studenten, Absolventen und Professionals der Informatik, Wirtschaftsinformatik, Wirtschaftswissenschaften und Wirtschaftsmathematik
- Teamplayer mit analytischen und konzeptionellen Fähigkeiten
- nach Möglichkeit Programmierkenntnisse Java und/oder .Net

**IHR SPRUNG
INS UNTERNEHMEN.**

- Direkteinstieg
- Studentische Mitarbeit
- Praktikum
- Abschlussarbeit

AKTUELLE STELLENANZEIGEN FINDEN SIE UNTER www.ecg-leipzig.de/karriere



Inhaltsverzeichnis

Full Paper

Ein Algorithmus für die Erzeugung Facebook-ähnlicher Netzwerkstrukturen basierend auf dem Barabási-Albert-Modell

Nick Robert Kempka und Annett Schumann 1

REx – eine Webapplikation zur Visualisierung der Evolution von Ontologien in den Lebenswissenschaften

Victor Christen 13

Integration von Geodaten in ein Planungssystem

Björn Buchwald 25

A Lexeme-Clustering Algorithm for Unsupervised Learning of Morphology

Maciej Janicki 37

Systematic review about the academic impact of DBpedia

Marcus Nitzschke 49

Visualisierung einer Taxonomie als Treemap und Optimierung der Generierung in einer Web-Applikation

Max Kießling 55

Metriken und optimale Einsatzszenarien für Garbage Collectoren der Java HotSpot Virtual Machine

Michael Schmeißer 63

Generierung von Serviceverträgen auf Basis objektorientierter ereignisgesteuerter Prozessketten

Jörg Hartmann 75

Relevanzbasiertes Preprocessing für automatische Theorembeweiser

Mario Frank 87

Untersuchung von Beschleunigungsverfahren für den Dijkstra-Algorithmus im Zusammenhang mit multimodalem Routing

David Georg Reichelt 99

Künstliche Intelligenz auf Basis von „Baby Machines“: Geschichte und Architektur BMA1

Florian Golemo 113

Poster Abstracts

Softwareentwicklung ohne Implementierung von Entwurfsmustern

Nils Rixin 125

Angezapft Technische Möglichkeiten einer heimlichen Online-Durchsuchung und der Versuch ihrer rechtlichen Bändigung

Rainer Rehak 127

Verbesserung der Performanz der SPARQL-Komponente vom DL-Learner

Didier Cherix 129

Ein Algorithmus für die Erzeugung Facebook-ähnlicher Netzwerkstrukturen basierend auf dem Barabási-Albert-Modell

Nick Kempka

n.kempka@studserv.uni-leipzig.de
Universität Leipzig, Bachelor Studiengang Informatik

Annett Schumann

annett.schumann@stud.htwk-leipzig.de
Hochschule für Wirtschaft, Technik und Kultur Leipzig
Master Studiengang General Management

Zusammenfassung: Soziale Netzwerke gewinnen in unserer heutigen Gesellschaft zunehmend auch für wirtschaftliche Zwecke an Bedeutung. Insbesondere Facebook, als weltweit größtes soziales Netzwerk, ist dabei für Unternehmen als Werbeplattform interessant. Unternehmen können den Erfolg derartiger Werbekampagnen nur unzulänglich beurteilen und beispielsweise die Anzahl der dadurch gewonnenen Neukunden kaum abschätzen. Als Entscheidungsunterstützung dienen unter anderem Simulationen, welche in der Lage sind, die Facebook-Kommunikation und -Interaktion nachzuvollziehen. Um eine solche simulative Prognose erstellen zu können, ist es notwendig, die Struktur des Facebook-Netzes zu kennen. In dem folgenden Beitrag wird ein Algorithmus beschrieben, welcher Facebook-ähnliche Graphstrukturen erzeugt. Als Grundlage für den Algorithmus dient das Barabási-Albert-Modell. Die erzeugten Graphen werden mit echten Facebook-Netzen verglichen und die Ergebnisse diskutiert.

1 Einleitung

Online-Händler sind häufig mit Entscheidungsfragen hinsichtlich ihrer medialen Webpräsenz konfrontiert, die weitreichende Auswirkungen auf ihr Unternehmen haben, deren Nutzen-Abschätzung jedoch häufig problematisch ist. Ziel des Forschungsprojekts Sim-Progno [Sim12] ist es, ein geeignetes Simulationsframework zu entwickeln, welches den Händler bei seiner Entscheidungsfindung unterstützt. Dazu werden verschiedene Simulationsmodule erarbeitet, welche abgegrenzte Fragestellungen betrachten. Ein Simulationsmodul beschäftigt sich mit der Kosten-Nutzen-Abwägung einer Facebook-Präsenz für ein Unternehmen. Dieses Modul wurde agentenbasiert [Klü06] modelliert und umgesetzt. Es wird dabei ein Zusammenspiel einzelner Agenten, welche in diesem Fall Facebook-Nutzer repräsentieren, betrachtet und ausgewertet. Um eine realitätsnahe Simulation zu ermöglichen, müssen in dieser die Agenten durch ihre Verknüpfung untereinander ein Netz schaffen, welches dem Facebook-Netz in Struktur und Eigenschaften möglichst gleich-

kommt. So kann die Nutzerinteraktion und virale Verbreitung von Nachrichten innerhalb des Facebook-Netzes simuliert werden. Dabei wird ermittelt, wie Unternehmensnachrichten durch Nutzer verbreitet werden, beispielsweise durch das Weiterleiten oder Empfehlen dieser Nachrichten, und wie viele Freunde dieser Nutzer davon wirklich erreicht werden. Die daraus resultierenden Werbeeffekte müssen dann mit den Kosten der Betreuung einer Facebook-Unternehmensseite abgewogen werden.

Im Zuge dieses Projektes entstand die Aufgabe Graphen zu konstruieren, welche die Eigenschaften der Vernetzung der Facebook-Nutzer widerspiegeln. Außerdem bestand die Anforderung einer variabel einstellbaren Nutzeranzahl (Knotenanzahl), um verschieden große Netze erzeugen zu können, sowie die eines erweiterbaren Graphen. Diese Kriterien führten zu der Entscheidung, synthetische Netze mithilfe eines Algorithmus zu erzeugen. Im weiteren Verlauf gehen wir zunächst auf Eigenschaften und sich anbietende Umsetzungsmöglichkeiten eines solchen Graphen ein. Anschließend wird unser Algorithmus und dessen Kalibrierung beschrieben. Die daraus resultierenden Graphen werden mit denen echter Netze verglichen und das Ergebnis diskutiert. Final schließt sich das Fazit der Umsetzung an und es werden zukünftige Realisierungs- und Verbesserungsmöglichkeiten in Ausblick gestellt.

2 Hintergrund

2.1 Graph-Struktur von Facebook

Die Vernetzung der Benutzer untereinander ist in sozialen Netzwerken, wie Facebook, durch Graphen beschreibbar. Dabei ist der Graph G ein Paar (V, E) , wobei $v \in V$ die Knoten und $e \in E$ die Kanten sind, welche geordnete Paare von Knoten (v_i, v_j) darstellen. Es gilt $E \subseteq V \times V$. Die Menge von Knoten (Benutzern) stellen durch Verknüpfung mittels Kanten (Freundschaftsbeziehungen) das Verbindungsnetz G dar. Ein solcher Graph eines sozialen Netzwerkes verfügt über verschiedene Eigenschaften, welche Aussagen über Größe und Struktur des Netzes und den Verknüpfungscharakter der einzelnen Knoten liefern.

So gibt der durchschnittliche Knotengrad die durchschnittliche Anzahl der Verbindungen aller Knoten mit jeweils anderen Knoten im Netz an $\sum_i \frac{deg(v_i)}{n}$, wobei n die Anzahl der Knoten ist und $deg(v_i)$ der Grad des Knotens v_i . In Facebook ist diese Eigenschaft mit dem Begriff der durchschnittlichen Anzahl an Freundschaftsbeziehungen aller Benutzer belegt und liegt bei einem Wert von 130 [all12]. Eng an den Begriff des Knotengrades ist auch die Dichte $dn(G) = \frac{2|E|}{|V|(|V|-1)}$ des sozialen Netzwerks als Anzahl der tatsächlichen Beziehungen im Verhältnis zur Anzahl aller möglichen Beziehungen im Netz gekoppelt [Mü08]. Der Durchmesser (Diameter) bezeichnet den größten Abstand zwischen zwei beliebigen Knoten im Netz, wenn der Abstand der Knoten dabei dem kürzesten Weg entspricht. Beim gesamten Facebook-Netz selbst liegt dieser bei sechs [ZT09]. Die durchschnittliche Pfadlänge $l_G = \frac{1}{|V|(|V|-1)} \sum_{i,j} Pfad(v_i, v_j)$ ist ein Maß für die durchschnittliche Anzahl an Knoten, welche entlang eines Pfades passiert werden müssen, um von jedem beliebigen Knoten

zu jedem anderen beliebigen Knoten zu gelangen. Facebook liefert dabei Werte zwischen vier und sechs [WBS⁺09]. Mittels dieser Eigenschaften können Aussagen über die Stärke der Vernetzung getroffen werden. Je mehr die Benutzer untereinander vernetzt sind, desto geringer fällt beispielsweise die durchschnittliche Pfadlänge aus. In Verbindung mit diesen Beobachtungen steht auch der Clusterkoeffizient, welcher Aussagen über die Cliquenbildung (Clustering) trifft und damit die Transitivität der Verknüpfungen bemisst. Dies bedeutet, wenn innerhalb eines Netzes ein Knoten v_i mit einem Knoten v_j verbunden ist, selbiger Knoten v_j wiederum mit Knoten v_k , so sind auch die Knoten v_i und v_k verbunden. v_i , v_j und v_k bilden dann eine Clique. Facebook weist dabei Werte zwischen 0.133 und 0.195 auf [WBS⁺09].

Um darüber hinaus den Verknüpfungscharakter eines Netzes zu analysieren, wurde unter anderem eine bevorzugte Bindung untersucht. Diese unterstellt, dass neue Knoten an bereits gut vernetzte Teilnehmer in einem Netzwerk ankoppeln, was Facebook als Eigenschaft zugesprochen wird. Untersucht wurde ebenfalls die Eigenschaft der Skalenfreiheit. Von skalenfreien Netzen spricht man dann, wenn die Anzahl von Verbindungen pro Knoten (Knotengrad) bei Betrachtung aller Knoten des Netzes nach einem Potenzgesetz verteilt ist. Die Verteilung der Grade nach ihrer Häufigkeit wird nachfolgend mit dem Begriff der Knotengradverteilung überschrieben. Das Facebook-Netz folgt dabei einer Potenzfunktion und zeigt demzufolge annähernd diese Eigenschaft der Skalenfreiheit, was durch verschiedene Studien belegt werden konnte [WBS⁺09, CDMF⁺12, MMG⁺07].

2.2 Graph-Modelle

Zur Entwicklung von Netzwerken gibt es bereits verschiedene Modelle. Namhafte Modelle sind unter anderen das Watts-Strogatz [WS98], das Barabási-Albert [BA99] sowie das Erdős-Rényi-Modell [ER59]. Alle drei erzeugen Zufallsgraphen, die bestimmte Eigenschaften aufweisen, welche auch in sozialen Netzwerken wiedergefunden werden können. Das Watts-Strogatz-Modell hat jedoch den Nachteil, dass es auf eine konstante Anzahl an Knoten angewiesen ist und somit nicht nachträglich erweitert werden kann. Zudem hat es das Manko, dass es keine skalenfreie Gradverteilung generiert. Dies ist auch im Erdős-Rényi-Modell der Fall. Im Barabási-Albert-Modell hingegen werden diese beiden Kriterien berücksichtigt. Zudem wird im Letztgenannten das Prinzip der bevorzugten Bindung umgesetzt, welches das Modell zu einem geeigneten Kandidaten für die künstliche Erzeugung sozialer Netze macht.

2.3 Implementierungen des Barabási-Albert-Modells

Auf Grund der weiten Verbreitung des Barabási-Albert-Modells ist dieses bereits implementiert worden. Bekannte Umsetzungen sind in der R-Bibliothek iGraph [iGr12] und im JUNG-Framework [JUN12] zu finden. Beide Implementierungen bieten zwar die Möglichkeit der Konfiguration, sind jedoch für unsere Zwecke nicht geeignet. Bei dem Versuch

die iGraph-Funktion nach unseren Vorstellungen zu konfigurieren, haben wir festgestellt, dass es nicht möglich ist multiple Kanten zwischen zwei Knoten zu unterbinden. Da eine Facebook-Freundschaft zwischen zwei Nutzern nicht mehrfach besteht, sind multiple Kanten nicht zweckdienlich.

In der aktuellen Version iGraph 0.6, veröffentlicht am 11. Juni, 2012, soll es möglich sein dies zu unterbinden. Jedoch lag die Version zu Beginn unserer Arbeit noch nicht vor. Im JUNG-Framework können parallele Kanten grundsätzlich unterbunden werden. Jedoch hat der Generator mit unserer Konfiguration bereits bei kleinen Netzwerken keine Ergebnisse geliefert, was vermutlich durch eine Endlosschleife bedingt ist. Da die Komplexität des Grundgerüsts des Barabási-Albert-Modells überschaubar ist, haben wir von einer Abwandlung bestehender Implementierungen abgesehen und eine komplett eigene Umsetzung vorgenommen. Dadurch konnten wir unsere Konfigurationsvorstellungen umsetzen und für eine gute Einbettung in die Facebook-Simulation sorgen.

3 Algorithmus

3.1 Vorgehensweise

Der Ablauf unseres Algorithmus (vgl. Listing 1) orientiert sich an einer Idee von Barabási et al. [BJN⁺02]. Er erhält als Eingabeparameter die Anzahl der gewünschten Knoten N und gibt einen Graph in dieser Größe aus. Da eine Facebook-Freundschaft bidirektional ist, ist der Graph ungerichtet. Die Anzahl der Kanten ist bei der Eingabe ungewiss, da der Algorithmus für die Berechnung der Kanten Zufallszahlen nutzt. Im Folgenden werden der Aufbau und die Verfahrensweise des Algorithmus beschrieben.

Das Netzwerk wird schrittweise aufgebaut. In jedem Schritt wird ein Knoten und eine unbestimmte Anzahl an Kanten hinzugefügt. Das Entstehen der Kanten lässt sich in zwei Abschnitte separieren. Im ersten Abschnitt werden die Kanten zwischen den bestehenden Knoten mit dem neu hinzukommenden Knoten generiert. Der zweite Abschnitt sorgt für eine dichtere Verknüpfung innerhalb der bestehenden Knoten. Hierbei wird der Knoten, der im jeweiligen Schritt hinzugefügt wurde, nicht beachtet.

Zu Beginn werden zwei Werte s_1 und s_2 berechnet. s_1 ist die Summe aller Grade der Knoten im Netzwerk $s_1 = \sum_i \text{deg}(\text{Knoten}_i)$. Die Berechnung von s_2 geschieht, indem zunächst das Produkt der Grade aller Knotenpaare bestimmt wird und diese dann aufsummiert werden. Formal ergibt dies $s_2 = \sum_{i,j} \text{deg}(\text{Knoten}_i) \text{deg}(\text{Knoten}_j)$.

Für jeden vorhandenen Knoten_i wird $P = \beta \frac{\text{deg}(\text{Knoten}_i)}{s_1}$ berechnet. Hierbei ist β ein Einflussfaktor, auf welchen wir im Abschnitt 3.2 ausführlicher eingehen werden. Anschließend wird eine Zufallszahl z bestimmt. Ist z kleiner gleich P , so wird der neue Knoten_N mit Knoten_i verbunden. Wird ein neuer Knoten mit keinem der alten Knoten verbunden, wird der Vorgang wiederholt, bis mindestens eine Kante erzeugt wurde. In der Realität gibt es zwar Facebook-Accounts, die keine Freundschaftsbeziehungen und somit

Listing 1: Pseudocode: Graph-Algorithmus

```

1 PROGRAM Graphalgorithmus
3   verbinde(Knoten1, Knoten2)
4   LOOP (Anzahl gewünschter Knoten)
5     A:=Anzahl der aktuellen Knoten
6     N:=A+1
7      $s_1 := \sum_i \text{deg}(\text{Knoten}_i)$ 
8      $s_2 := \sum_{i,j} \text{deg}(\text{Knoten}_i) \text{deg}(\text{Knoten}_j)$ 
9     WHILE(KnotenN hat noch keine Kante)
10      FOR(i:=1, i≤A, i:=i+1)
11         $p := \beta \frac{\text{deg}(\text{Knoten}_i)}{s_1}$ 
12        z:=Zufallszahl zwischen 0 und 1
13        IF(z≤p) verbinde(Knoteni, KnotenN) END IF
14      END FOR
15      Teste, ob KnotenN mindestens eine Kante hat
16    END WHILE
17    FOR(i:=1, i≤A, i:=i+1)
18      FOR(j:=1, j≤A, j:=j+1)
19        IF(keine Kante zwischen Knoteni und Knotenj)
20           $p := \alpha A \frac{\text{deg}(\text{Knoten}_i) \text{deg}(\text{Knoten}_j)}{s_2}$ 
21          z:=Zufallszahl zwischen 0 und 1
22          IF(z≤p) verbinde(Knoteni, Knotenj) END IF
23        END IF
24      END FOR
25    END FOR
26  END LOOP
27 END PROGRAM Graphalgorithmus

```

keine Kanten haben, diese spielen jedoch bei der Nutzer-Interaktion keine Rolle und werden deshalb nicht von der geplanten Simulation beachtet. Der Algorithmus startet mit zwei bereits verbundenen Knoten, da ansonsten P im ersten Schritt Null wäre, was wiederum zu einer Endlosschleife führen würde.

Für jedes Knotenpaar Knoten_i und Knoten_j , welches nicht miteinander verbunden ist, wird $P = \alpha A \frac{\text{deg}(\text{Knoten}_i) \text{deg}(\text{Knoten}_j)}{s_2}$ berechnet. Hierbei ist A die Anzahl der Knoten im vorangegangenen Schritt. Auf den Einflussfaktor α wird im Abschnitt 3.2 genauer Bezug genommen. Daran anschließend wird erneut eine Zufallszahl z bestimmt. Ist z kleiner gleich P , so werden Knoten_i und Knoten_j verknüpft.

3.2 Kalibrierung

Bei der Berechnung der Wahrscheinlichkeit, ob ein Knoten mit einem anderen verbunden wird, besteht die Möglichkeit mit Hilfe der Parameter α und β Einfluss auf die Entwicklung des Graphen zu nehmen. Eine Erhöhung dieser Werte führt zu einer Erhöhung der Kantenanzahl, welche eine dichtere Struktur des Graphen bewirkt.

Vergleicht man den Einfluss von α und β auf den Graphen, so gelangt man zu der Feststellung, dass α einen wesentlich höheren Einfluss auf den Clusterkoeffizienten hat als β . Erhöht man also die Anzahl der Kanten durch ein größeres α , so erzielt man einen wesentlich höheren Clusterkoeffizient als durch ein größeres β . Der durchschnittliche Grad der Knoten, die Dichte, der Diameter und die durchschnittliche Pfadlänge des Graphen werden von α und β nahezu gleichwertig beeinflusst.

Die Tabelle 1 zeigt exemplarisch, dass bei gleicher Kantenanzahl durch die Erhöhung von α ein etwa dreifacher Clusterkoeffizient erreicht wird. Um diesen Vergleich möglich zu machen, haben wir uns auf feste Knoten- und Kantenzahlen festgelegt und diese mit unterschiedlichen Kalibrierungen des Algorithmus erreicht. Dass diese Differenzen der Clusterkoeffizienten nicht auf Grund der Zufallszahlen innerhalb des Algorithmus entstanden sind, wird in Abschnitt 4.1 deutlich, in dem wir auf die Stabilität der Ergebnisgraphen eingehen.

Tabelle 1: Kalibrierungsvergleich (CC:=Clusterkoeffizient, D:=Diameter)

Knoten	Konfiguration	Kanten	Knotengrad	Dichte	CC	D	Pfadlänge
1000	$\alpha = 0.001 \beta = 10$	10283	20.566	0.0206	0.1733	5	3.069
	$\alpha = 0.02 \beta = 2$	10235	20.472	0.0205	0.4574	5	2.975
5000	$\alpha = 0.001 \beta = 10$	61566	24.626	0.0049	0.0717	5	2.794
	$\alpha = 0.0045 \beta = 2$	61050	24.42	0.0049	0.2337	5	2.901

4 Ergebnisse und Diskussion

4.1 Vergleich der synthetischen mit echten Facebook-Netzen

Bei dem Versuch, die Güte des Algorithmus zumindest partiell nachzuprüfen, haben wir aus einer Studie zum Thema „Social Structure of Facebook Networks“ [TMP11] ein Datenset von echten Facebook-Graphen erhalten, welches Facebook-Freundschaftsnetzwerke an einhundert amerikanischen Hochschulen und Universitäten hinsichtlich bestimmter Eigenschaften untersuchte. Im Fokus unserer Betrachtung stand die Vernetzung der Studenten innerhalb dieser Graphen als Vergleichsmöglichkeit zu unseren synthetischen Graphen. Im Folgenden wird dieser Vergleich aufgezeigt.

Die Tabelle 2 zeigt Eigenschaften ausgewählter echter Netze im Vergleich zu unseren synthetisch erzeugten Netzen. Bei den echten Netzen handelt es sich um die Facebook- Ver-

netzungen basierend auf Daten einer Hochschule aus Michigan mit 3748 Knoten, einer Hochschule UC mit 6833 Knoten sowie der Hochschule UCF mit 14936 Knoten. Im zugrunde liegenden Datenset sind die Hochschulen unter der Bezifferung 67 (Michigan), 64 (UC), beziehungsweise 52 (UCF) zu finden. Die Anzahl der Knoten steht dabei für die Anzahl der bei Facebook registrierten Hochschulangehörigen, die Bezifferung der Hochschulen ergeben sich aus der Durchnummerierung der 100 gesammelten Hochschul-Datensätze. In der Gegenüberstellung der Netze nach ihrer Größe ist klar zu erkennen, dass bis auf tendenziell kleinere Durchmesser und durchschnittliche Pfadlängen die Möglichkeit besteht, unseren Algorithmus so zu konfigurieren, dass er Graphen mit sehr ähnlichen Werten, wie die der Universitäts-Netze erzeugen kann.

Tabelle 2: Vergleich der Grapheigenschaften (CC:=Clusterkoeffizient, D:=Diameter)

Knoten	Netz	Kanten	Knotengrad	Dichte	CC	D	Pfadlänge
3748	echt (Michigan 67)	81903	43,70	0,0117	0,563176	7	2,839429
	synth. ($\alpha = 0,01; \beta = 2$)	78504	41,89	0,0110	0,560000	5	2,696000
6833	echt (UC 64)	155332	45,47	0,0066	0,540510	8	3,014991
	synth. ($\alpha = 0,008; \beta = 2$)	161338	47,22	0,0069	0,515600	6	2,762600
14936	echt (UCF 52)	428989	57,43	0,0038	0,465201	6	2,845651
	synth. ($\alpha = 0,004; \beta = 2$)	402779	53,93	0,0036	0,443600	6	2,845253

Hierbei ist noch zu erwähnen, dass der Algorithmus, trotz Zufallszahlen, stabile Ergebnisse liefert. Um dies zu überprüfen, haben wir einige Netze mit gleicher Knotenanzahl und Kalibrierung erzeugt und diese verglichen. Die Tabelle 3 zeigt dies exemplarisch auf. Dort sieht man einige Eigenschaften zu sechs analysierten Graphen, die eine Knotenanzahl von 6833 aufweisen und mit gleicher Kalibrierung ($\alpha = 0,008; \beta = 2$) erzeugt wurden. Dieses Experiment wurde mit einigen Graphengrößen und Kalibrierungen (bis etwa 20000 Knoten) durchgeführt und führte immer zu dem Ergebnis stabiler Werte.

Tabelle 3: Stabile Ergebnisse unter gleicher Kalibrierung

Kanten	Knotengrad	Dichte	CC	D	Pfadlänge
161647	47.3136	0.0069	0.522	6	2.7645
161796	47.3572	0.0069	0.518	6	2.7619
161338	47.2232	0.0069	0.5156	6	2.7626
162262	47.4936	0.0069	0.5174	6	2.765
162147	47.4599	0.0069	0.52	6	2.7583

Ein weiterer interessanter Punkt für unseren Vergleich ist die Knotengradverteilung. Abbildung 1 zeigt diese exemplarisch für das echte Facebook-Netz UC (64) mit 6833 Knoten sowie das darauf basierende generierte mit den Parametern $\alpha=0,008$ sowie $\beta=2$. Die Punkte repräsentieren dabei die diskreten Werte der Knotengrade, die Kurven zeigen die zugehörigen Regressionsgraphen. Die Regressionsanalyse wurde durchgeführt, um einen Zusammenhang zwischen der Verteilung der diskreten Werte des Grades (unabhängige Variable) und dessen jeweiliger Häufigkeit (abhängige Variable) aufzuzeigen. Dazu war es notwendig, vorerst den linearen Zusammenhang dieser beiden Größen in einer Regres-

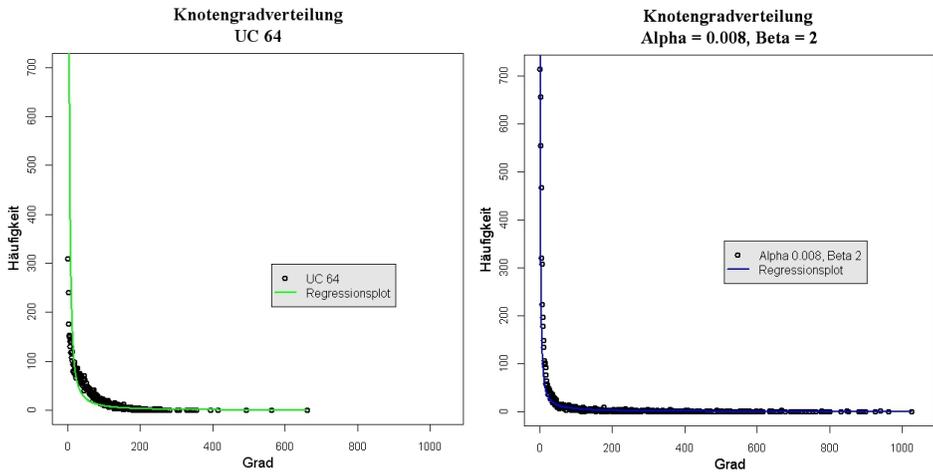


Abbildung 1: Vergleich der Knotengradverteilung

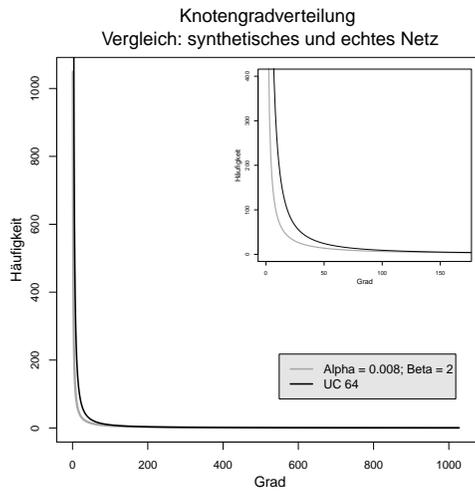


Abbildung 2: Vergleich der Regressionsgraphen

sionsgeraden durch das Logarithmieren des Grades sowie der Häufigkeit zu generieren. Anschließend konnte daraus mittels des errechneten linearen Zusammenhanges zwischen $\log(\text{Grad})$ und $\log(\text{Häufigkeit})$ der Anstiegsparameter sowie der Achsenabschnitt der Regressionsgeraden berechnet werden, welche im Regressionsgraphen die Potenzen bilden. Dies ermöglichte uns schließlich, daraus den Regressionsgraphen abzuleiten. Diese dienen uns dazu, in einer Vergleichsgrafik die noch bestehenden Unzulänglichkeiten und Abweichungen ablesen und daraus künftig für unsere synthetische Netzerzeugung Schlussfolgerungen hinsichtlich der Anpassung ziehen zu können.

Tabelle 4: Vergleich der Knotengradverteilung in Abschnitten (Gradkategorien)

Gradkategorie	synthetisches Netz	relative Häufigkeit	in %	echtes Facebook-Netz	relative Häufigkeit	in %
1-50	5545	0,8115	81,15	4552	0,6662	66,62
51-100	485	0,0710	7,10	1495	0,2188	21,88
101-150	208	0,0304	3,04	516	0,0755	7,55
151-200	132	0,0193	1,93	183	0,0268	2,68
201-250	107	0,0157	1,57	56	0,0082	0,82
251-300	83	0,0121	1,21	17	0,0025	0,25
301-350	54	0,0079	0,79	8	0,0012	0,12
351-400	44	0,0064	0,64	2	0,0003	0,03
401-450	41	0,0060	0,60	1	0,0001	0,01
451-500	20	0,0029	0,29	1	0,0001	0,01
501-550	28	0,0041	0,41	0	0,0000	0,00
551-600	27	0,0040	0,40	1	0,0001	0,01
601-650	13	0,0019	0,19	0	0,0000	0,00
651-700	13	0,0019	0,19	1	0,0001	0,01
701-750	8	0,0012	0,12	0	0	0
751-800	10	0,0015	0,15	0	0	0
801-850	4	0,0006	0,06	0	0	0
851-900	6	0,0009	0,09	0	0	0
901-950	3	0,0004	0,04	0	0	0
951-1000	1	0,0001	0,01	0	0	0
1001-1050	1	0,0001	0,01	0	0	0
Summe	6833	1	100	6833	1	100

Abbildung 2 zeigt den Vergleich des Regressionsgraphen unseres synthetischen mit dem echten Facebook-Teilnetz. Hierbei ist ein hoher Grad an Übereinstimmung festzustellen. Vergleicht man beide Plots der Knotengradverteilung aus Abbildung 1 so stellt man einen Unterschied im Bereich der Knoten mit hohem Grad (> 300) fest. Dies ist auf eine bekannte Schwäche des Barabási-Albert-Modells zurückzuführen, welches im Verhältnis zur Beobachtung echter Netze, ein geringes Clustering aufweist [Yan05]. Im Folgenden wird der Unterschied der Gradverteilung analysiert.

Tabelle 4 zeigt die Häufigkeitsverteilung der Verknüpfungsgrade sowohl im realen Facebook-Netz als auch in unserem synthetisch generierten. Hierfür wurden für die Analyse die Knotengradverteilungen in Einzelkategorien eingestuft, wobei die Kategorien Knotengrade in 50er-Schritten durchlaufen. Dabei wird deutlich, dass der Großteil der generierten Knoten (95%) beziehungsweise im echten Netz der Facebook-User (99%) in beiden Netzen Verknüpfungsgrade in einem Bereich zwischen 1 und 250 aufweist. Weiterhin fällt auf, dass die Streuung im synthetisch erzeugten Netz durchaus breiter gefächert ist als im realen Netz und sich hierbei durch den angewandten Algorithmus bis zu 1027 Verknüpfungen ergeben haben. Im realen Facebook-Netz hingegen zeigten die Freundschaftsbeziehungen bei der Auswertung von 6833 Usern nur einen Verbindungsgrad bis zu 660. Der Vergleich der Knotengradverteilung zeigt diese Auffälligkeit deutlich. Die

Ausläufer der Punktwolke im künstlich generierten Netz haben immerhin noch eine Häufigkeit von 4% für enorm hohe Verknüpfungsgrade zwischen 250 und 1027, wohingegen diese des echten Netzes kaum mehr 0,2% erreichen.

4.2 Allgemeine Problematik der Modellierung eines Facebook-Netzes

Unsere vorausgegangene Recherche und Analyse des Facebook-Netzes hat gezeigt, dass es sehr schwierig ist, einen repräsentativen Graphen für das Facebook-Netz zu erstellen. Damit sind sowohl künstliche als auch echte Graphen gemeint. Selbst wenn es möglich wäre, das gesamte Facebook-Netz zu erfassen, so würde trotz allem auch die Masse an ungenutzten, Fake- sowie Spam-Accounts [Fac12] die meisten Aussagen über dieses Netz verfälschen, da diese Accounts nicht im herkömmlichen Sinne genutzt werden und ihr Einfluss im Gesamtnetz kaum einzuordnen ist. Ließe sich eine Möglichkeit finden, diese Falsch-Accounts heraus zu filtern, so bliebe weiterhin die technische Frage bestehen, wie solch eine Form von Daten in so einer großen Menge überhaupt in annehmbarer Zeit erfasst werden soll. Das Facebook-Netz ist dynamisch und wächst rapide. Wochenlange Erfassungsmethoden würden beispielsweise dazu führen, dass zu Beginn des Scans ein User erfasst wird, der nur sehr wenige Freunde hat, am Ende des Scans jedoch diese Freundesanzahl um ein vielfaches gestiegen ist und damit neue Freundschaftsbeziehungen nicht erfasst wurden. Die Idee einen Teilgraphen zu erstellen, wirft das Problem auf, sich für einen solchen zu entscheiden. Es stellt sich die Frage, welcher Teilgraph repräsentativ für den Gesamtgraph sein kann. Eine ausführliche Ausarbeitung des Themas „Crawling des Facebook-Netzes“ bieten Gjoka et al. [GKBM10].

5 Fazit und Ausblick

Das in Abschnitt 4.2 dargelegte generelle Problem der Analyse des Facebook-Netzes führt dazu, dass sich Eigenschaften wie beispielsweise die Skalenfreiheit nur schwierig feststellen lassen. Daraus resultiert, dass es nur wenige fundierte Eckpunkte gibt, anhand derer ein Modell erstellt oder überprüft werden kann, welches einen repräsentativen Graphen für das gesamte Facebook-Netz erstellt. Die Flexibilität des Algorithmus bietet jedoch die Möglichkeit, die synthetischen Graphen in ihrer Struktur so zu verändern, dass sie bestimmte Kriterien erfüllen, was sich positiv auf eine zukünftige Neu-Konfiguration und Anpassung für bestimmte Zwecke auswirkt. Wir haben die Konfigurationen bis jetzt durch Schätzungen der Parameter α und β vorgenommen. Zukünftig ist eine genaue Auswertung dieser geplant, um ein komfortables Konfigurieren zu ermöglichen. Betrachtet man den Vergleich unserer künstlichen Netze mit echten Facebook-Netzen, so werden diese zwar nicht exakt abgebildet, weisen jedoch eine starke Ähnlichkeit auf. Die Knotengradverteilung stimmt tendenziell überein, sollte in Zukunft jedoch in ihren Randbereichen optimiert werden. Ein weiteres Ziel ist die Erzeugung größerer synthetischer Netze. Dies bringt eine Verbesserung der Performanz des Algorithmus und der Analyse-Verfahren mit sich. Außerdem müssen Vergleichsmöglichkeiten für größere Graphen gefunden werden.

Die künstlichen Graphen lassen sich hervorragend für die Simulation des SimProgno-Projekts nutzen. Aus technischer Sicht sind diesbezüglich keine Hindernisse zu finden. Die Qualität der Ergebnisse der Simulation hängt stark von der Qualität der Netze ab, was die Wichtigkeit einer geeigneten Netzkonfiguration unterstreicht. Da die Entwicklung der Simulation noch nicht vollständig ist, lässt sich über diese Aspekte jedoch vorerst noch keine genaue Aussage treffen.

Danksagung

Ein besonderer Dank gilt Axel Hummel, welcher uns nicht nur bei der Umsetzung des vorgestellten Projekts, sondern auch bei dem Verfassen dieses Beitrags tatkräftig unterstützt hat. Dieser Beitrag entstand im Rahmen des Verbundprojekts SimProgno (Förderkennzeichen: 01IS10042C). SimProgno wird gefördert vom Bundesministerium für Bildung und Forschung.

Literatur

- [all12] allfacebook.de – Blog, 2012. http://allfacebook.de/zahlen_fakten/infografik-facebook-2012-nutzerzahlen-fakten/. Zuletzt geprüft am 06.07.2012.
- [BA99] A. L. Barabási und R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
- [BJN⁺02] A. L. Barabási, H. Jeong, Z. Neda, E. Ravasz, A. Schubert und T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(3-4):590–614, 2002.
- [CDMF⁺12] S. Catanese, P. De Meo, E. Ferrara, G. Fiumara und A. Provetti. Extraction And Analysis Of Facebook Friendship Relations. *Computational Social Networks: Mining And Visualization*, 2012.
- [ER59] P. Erdős und A. Rényi. On Random Graphs I. *Publ. Math. Debrecen*, 6:290, 1959.
- [Fac12] Facebook Report, 2012. http://www.sec.gov/Archives/edgar/data/1326801/000119312512325997/d371464d10q.htm#tx371464_9. Zuletzt geprüft am 21.08.2012.
- [GKBM10] Minas Gjoka, Maciej Kurant, Carter T. Butts und Athina Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *Proceedings of IEEE INFOCOM '10*, San Diego, CA, March 2010.
- [iGr12] iGraph – a free software package for creating and manipulating graphs, 2012. <http://igraph.sourceforge.net/>. Zuletzt geprüft am 05.06.2012.
- [JUN12] JUNG Framework – General framework for the modeling, analysis, and visualization of graphs, 2012. <http://jung.sourceforge.net/>. Zuletzt geprüft am 05.06.2012.

- [Klü06] Franziska Klügl. Multiagentensimulation. *Informatik Spektrum*, 29(6):412–415, 2006.
- [MMG⁺07] Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel und Bobby Bhattacharjee. Measurement and analysis of online social networks. In Constantine Dovrolis und Matthew Roughan, Hrsg., *Internet Measurement Conference*, Seiten 29–42. ACM, 2007.
- [Mü08] Claudia Müller. *Graphentheoretische Analyse der Evolution von Wiki-basierten Netzwerken für selbstorganisiertes Wissensmanagement*. Dissertation, University of Potsdam, 2008.
- [Sim12] SimProgno – Ein integratives Framework zur multidimensionalen Modellierung, Simulation und Prognostik von E-Commerce-Wirkungsketten, 2012. <http://www.simprogno.de/>. Zuletzt geprüft am 05.06.2012.
- [TMP11] Amanda L. Traud, Peter J. Mucha und Mason A. Porter. Social Structure of Facebook Networks. *CoRR*, abs/1102.2166, 2011.
- [WBS⁺09] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy und Ben Y. Zhao. User interactions in social networks and their implications. In Wolfgang Schröder-Preikschat, John Wilkes und Rebecca Isaacs, Hrsg., *EuroSys*, Seiten 205–218. ACM, 2009.
- [WS98] D. J. Watts und S. H. Strogatz. Collective dynamics of “small-world” networks. *Nature*, 393:440–442, 1998.
- [Yan05] Shi-Jie Yang. Exploring complex networks by walking on them. *Physical Review E*, 71(1):016107, 2005.
- [ZT09] Lei Zhang und Wanqing Tu. Six Degrees of Separation in Online Society. *Proceedings of the WebSci’09: Society On-Line*, März 2009.

REx – eine Webapplikation zur Visualisierung der Evolution von Ontologien in den Lebenswissenschaften

Victor Christen

mam08bfa@studserv.uni-leipzig.de

Universität Leipzig, Master Studiengang Informatik

Zusammenfassung: Ontologien sind im Bereich der Lebenswissenschaften für die maschinelle Analyse und Auswertung von Daten unerlässlich. Mithilfe von Ontologien lassen sich molekular-biologische Objekte wie Proteine oder Gene konsistent und einheitlich beschreiben (annotieren), so dass ein maschineninterpretierbarer und semantischer Rahmen für ein Themengebiet spezifiziert werden kann. Durch neue Anforderungen sowie geändertes Domänenwissen unterliegen sie einem ständigen Veränderungsprozess (Evolution). Die Webanwendung REx (Region Evolution Explorer) ermöglicht die Visualisierung von Veränderungen innerhalb einer Ontologie. Dabei werden für veröffentlichte Versionen innerhalb eines Zeitraums Regionen mit hoher (geringer) Änderungsintensität ermittelt. Nutzer können sich diese Regionen über verschiedene Workflows visualisieren lassen, um somit die Evolution insbesondere von großen Ontologien kompakter und besser zu verstehen. REx wurde anhand zahlreicher, großer Ontologien in den Lebenswissenschaften evaluiert und ist unter <http://dbs.uni-leipzig.de/rex> verfügbar.

1 Einleitung

Aufgrund des exponentiellen Anstiegs des Datenvolumens im Bereich der Informationsverarbeitung ist eine semantische Strukturierung der Daten unerlässlich, um eine eindeutige Interpretation zu gewährleisten. Mithilfe von Ontologien ist eine flexible semantische Anreicherung der Daten möglich. Eine Ontologie ist nach Gruber [Gru95] eine gemeinsame explizite Spezifikation einer Konzeptualisierung. Durch den Einsatz einer Ontologie ist ein einheitliches Format der annotierten Daten gewährleistet, so dass verschiedene Institutionen oder Anwendungen diese verwenden und interpretieren können. Insbesondere in den Lebenswissenschaften spielen Ontologien eine essentielle Rolle [BS06]. Sie werden primär für die einheitliche Annotation molekular-biologischer Objekte verwendet. So werden bspw. Proteine in der UniProt Wissensbasis [BAW⁺05] mithilfe von Konzepten aus der Gene Ontology (GO) [Gen08] annotiert, um u.a. deren molekulare Funktionen oder Beteiligung an biologischen Prozessen zu spezifizieren. Solche Wissensbasen können für weitere Analysen verwendet werden, z. B. für die Vorhersage von Proteinstrukturen [Ste03] oder für funktionelle Genexpressionsanalysen [HSL09].

Jede Ontologie repräsentiert eine Domäne, die einen Realweltausschnitt darstellt. Speziell in den Lebenswissenschaften verändern sich die Ontologien in regelmäßigen Abständen,

um neue und geänderte Forschungserkenntnisse in einer Domäne in die entsprechende Ontologie zu integrieren [HKR08]. Des Weiteren müssen Ontologien bspw. verändert werden, um Designfehler aus früheren Versionen zu beseitigen. Infolgedessen werden ständig neue Versionen einer Ontologie veröffentlicht, welche dann durch Endanwender verwendet werden können. Die Veröffentlichung neuer Versionen erfordert die Identifikation der Änderungen der neuen Version bzgl. der Älteren, um festzustellen, ob ein Nutzer aufgrund der durchgeführten Änderungen betroffen ist und ggf. Daten anpassen oder Analysen wiederholen muss.

Da Ontologien in den Lebenswissenschaften oft sehr groß und komplex sind (z. B. umfasst die GO mehr als 30.000 Konzepte), können mögliche Auswirkungen häufig nicht direkt oder nur mit viel manuellem Aufwand ermittelt werden. Derartiges Wissen könnte jedoch gewinnbringend eingesetzt werden. Hat sich bspw. ein spezieller Teil einer Ontologie nicht verändert (d. h. er ist stabil geblieben) wären Nutzer und Anwendungen die diesen Teil verwenden von Änderungen nicht betroffen. Andersherum müssten Datenquellen oder Applikationen, welche einen stark veränderten (instabilen) Ontologieteil verwenden ggf. angepasst oder migriert werden. Gleichmaßen könnten sich Entwickler und Koordinatoren in einem Ontologieprojekt darüber informieren, welche Ontologieteile in den letzten Jahren stark oder eher marginal verändert wurden, um daraus weitere Entwicklungsschritte für künftige Arbeiten abzuleiten und zu planen. Ein entsprechendes Tool für derartige Analysen müsste einerseits die Größe der zu analysierenden Ontologien beachten und sollte andererseits möglichst intuitive und kompakte Darstellungsformen und Visualisierungen zur verständlichen Erfassung der Ontologieevolution anbieten.

Die vorgestellte Webapplikation REx (Region Evolution Explorer¹) visualisiert die Intensität der Veränderungen einer Ontologie über einen bestimmten Zeitraum mittels Ontologieregionen in struktureller und statistischer Form. Es werden die folgenden Beiträge geleistet:

- Konzeption und Implementierung der Webanwendung auf Basis eines Repository für Ontologieversionen und eines Algorithmus zur Erkennung von stabilen bzw. instabilen Ontologieregionen
- Bereitstellung von zwei Workflows zur (1) graphbasierten sowie (2) statistischen Analyse der Ontologieevolution mittels Ontologieregionen
- Evaluierung und Test der Webanwendung für zahlreiche große Ontologien aus den Lebenswissenschaften

Der Beitrag ist wie folgt gegliedert. Im nächsten Abschnitt werden Grundlagen vermittelt und das in der Webapplikation verwendete Verfahren für die Analyse der Evolution von Ontologien vorgestellt. Der dritte Abschnitt befasst sich mit der Webapplikation REx. Dazu werden die beiden Workflows näher vorgestellt und Informationen zur Architektur präsentiert. Eine Evaluierung von REx findet im vierten Abschnitt statt. Im fünften Abschnitt werden verwandte Arbeiten diskutiert. Eine Zusammenfassung sowie ein Ausblick bilden den Abschluss des Beitrags.

¹Region Evolution Explorer: <http://dbs.uni-leipzig.de/rex>

2 Grundlagen – Ontologie, Evolution und Regionen

2.1 Ontologie und Ontologieversionen

Formal wird eine Ontologie O durch eine Menge von Konzepten C , Relationen R und Attributen A beschrieben: $O = (C, R, A)$. Konzepte besitzen einen eindeutigen Identifizierer (z. B. URI oder accession number). Relationen erweitern die Semantik einer Ontologie, indem Konzepte miteinander in Beziehung gesetzt werden können. Die *is_a* Relation, die eine Subklassen Beziehung zwischen zwei Konzepten definiert, ist die essentiellste Relation. Neben dieser Relation existieren andere Relationen wie z. B. *part_of* oder domänenspezifische wie z. B. *regulates*. Attribute wie Name, Synonym oder Definition werden genutzt, um ein Konzept näher zu beschreiben.

Eine Ontologie ist keine statische Spezifikation. In regelmäßigen Abständen werden Änderungen vorgenommen, um neue Erkenntnisse oder Fehler einzubeziehen bzw. zu beseitigen. Für eine Analyse der Veränderungen einer Ontologie ist eine Versionierung unerlässlich. Eine Version v einer Ontologie O ist eine Momentaufnahme zu einem bestimmten Zeitpunkt und wird als O_v bezeichnet. Eine Version ist solange gültig bis eine neue Version veröffentlicht wird. Die Zeitpunkte von Versionen stellen somit eine Ordnungsrelation dar, d. h. jede Version (mit Ausnahme der Ersten bzw. Letzten) besitzt exakt eine Vorgänger- bzw. Nachfolgeversion.

2.2 Regionenalgorithmus

Eine Ontologieregion OR ist ein Teilgraph der Ontologie, der ein Wurzelkonzept und dessen Nachfolgekonzepte beinhaltet, die durch eine *is_a* Relation verknüpft sind. Im Folgenden wird kurz erläutert wie (in)stabile Regionen zwischen zwei Ontologieversionen O_v und O_{v+1} berechnet werden können. Für eine ausführliche Darstellung inkl. Beispiel wird auf [HGKR10] verwiesen.

Zunächst müssen die Änderungen zwischen O_v und O_{v+1} mittels eines Diff-Algorithmus identifiziert werden. Der Diff von zwei Versionen O_v und O_{v+1} beschreibt die Menge von Änderungsoperationen, welche angewandt auf O_v die Version O_{v+1} erzeugen würden. Es wird zwischen einfachen Änderungen (z. B. Einfügen eines Konzepts) und komplexen Änderungen (z. B. Merge mehrerer Konzepte in ein Konzept) unterschieden. Für die Berechnung des Diffs existieren diverse Verfahren wie z. B. PromptDiff [NM02] oder COntoDiff [HGR12]. Basierend auf dem Diff zwischen beiden Versionen wird die Intensität der Veränderung mithilfe des Regionenalgorithmus ermittelt. Für die Berechnung der Intensität wird ein Kostenmodell verwendet, das jeder Änderungsoperation Kosten zuweist, z. B. Kosten zwei für das Löschen eines Konzepts oder eins für das Hinzufügen einer Relation.

Zu Beginn werden sowohl in O_v als auch in O_{v+1} den gelöschten Konzepten bzw. hinzugefügten Konzepten die entsprechenden Kosten zugewiesen. Die Kosten für veränderte Relationen werden den beteiligten Konzepten zugewiesen. Diese Kosten entsprechen einer direkten Veränderung eines Konzepts und werden als lokale Kosten lc bezeichnet.

Des Weiteren wird ein Konzept durch die Konzepte innerhalb seiner Region beeinflusst. Diese Art des Einflusses wird durch die aggregierten Kosten ac repräsentiert. Die aggregierten Kosten ac eines Konzepts c berechnen sich aus den aggregierten Kosten seiner Kinderkonzepte sowie den eigenen lokalen Kosten lc . Die aggregierten Kosten eines Kinderkonzepts c' werden anteilig an dessen Elternkonzepten ($parents(c')$) propagiert:

$$ac(c) = \sum_{c' \in children(c)} \frac{ac(c')}{|parents(c')|} + lc(c)$$

Die Berechnung der aggregierten Kosten aller Konzepte in den beiden Ontologieversionen O_v und O_{v+1} beginnt jeweils bei den Blattkonzepten und wird entlang der is_a Struktur bis zu den Wurzelkonzepten fortgesetzt. Um den Einfluss der Löschung von Elementen ebenfalls in der Version O_{v+1} nachvollziehen zu können, müssen die Kosten aus O_v nach O_{v+1} transferiert werden. Für jedes Konzept, welches sowohl in O_v als auch in O_{v+1} existiert, werden die aggregierten Kosten aus beiden Versionen zusammengefasst.

Ein signifikantes Maß für die Veränderung (Stabilität) einer Region OR sind die durchschnittlichen Kosten avg_costs . Diese Kosten relativieren die aggregierten Kosten der Wurzel ($root$) von OR bezüglich der Größe der Region: $avg_costs = \frac{ac(root)}{|OR|}$. Regionen mit hohen avg_costs weisen somit starke Veränderungen auf und können als instabil klassifiziert werden. Für $avg_costs=0$ liegen keine Änderungen in einer Region vor.

Die Analyse für einen Zeitraum, der mehrere Versionen beinhaltet, basiert auf der iterativen Ausführung der beschriebenen Methode. In jeder Iteration mit Ausnahme der ersten wird die neue Version der vorherigen Iteration als die alte angesehen und mit der darauf folgenden Version die Berechnung durchgeführt. Der Algorithmus endet, wenn die Berechnungen für die letzte Version im Zeitraum beendet sind. Somit werden sukzessive die aggregierten Kosten aus allen Versionen aufgesammelt und in der letzten Version kann eine Regionenerkennung stattfinden.

3 REX – Region Evolution Explorer

REx ist eine Webapplikation, die mithilfe des Regionalgorithmus die Intensität der Evolution von Ontologien visualisiert. Die Visualisierung gliedert sich in die strukturelle Repräsentation der Ontologie in Form eines Graphen und in die statistische Repräsentation. Aufgrund der immensen Größe des resultierenden Graphen ist eine ausschließliche statische Repräsentation ungeeignet, da der Anwender nicht in der Lage ist innerhalb der Ontologie zu navigieren. Das prinzipielle Designprinzip ist das „*Information Seeking Mantra*“ [Kei02], das den Anwender in den Explorationsprozess integriert. REX ist mithilfe des GWT (Google Web Toolkit) und den erweiternden Frameworks SmartGWT² und ExtGWT³ sowie den Graphbibliotheken Flare⁴ und InfoVis⁵ implementiert. Im Folgenden werden die zwei Hauptworkflows näher erläutert und die Architektur beschrieben.

²<http://code.google.com/p/smartgwt/>

³<http://www.sencha.com/products/extgwt>

⁴<http://flare.prefuse.org/>

⁵<http://thejit.org/>

3.1 Funktionalitäten

Generell gliedert sich jede Funktionalität in die Spezifikation der notwendigen Parameter, die Berechnung der spezifischen Größen des Regionenalgorithmus und die graphische Präsentation.

3.1.1 Graphrepräsentation

Der Anwender ist in der Lage sich einen Überblick über die gesamte Struktur inklusive der Analyseergebnisse zu verschaffen und Subregionen eines Konzepts in einem weiteren Graph näher zu analysieren. Neben der graphischen Repräsentation können die Analyseergebnisse in tabellarischer Form betrachtet werden. Der prinzipielle Ablauf der Applikation für die Repräsentation ist in Abb. 1 dargestellt.

Zu Beginn der Analyse werden die Eingabeparameter im *Input Panel* spezifiziert, die aus dem Namen und dem Typ der Ontologie sowie dem zu analysierenden Zeitraum bestehen. Nach Beendigung der Berechnungen wird die gesamte Ontologie als Graph im *Overview Panel* dargestellt. Basierend auf den berechneten Durchschnittskosten *avg_costs* werden die Konzepte mittels einer Grün-Gelb-Rot-Farbskala entsprechend eingefärbt. Für die visuelle Exploration ist der Anwender in der Lage mit dem geometrischen Zoom bzw. mit dem Fisheye-Zoom Teilgebiete detaillierter zu betrachten. Im *Control Panel* ist der ge-

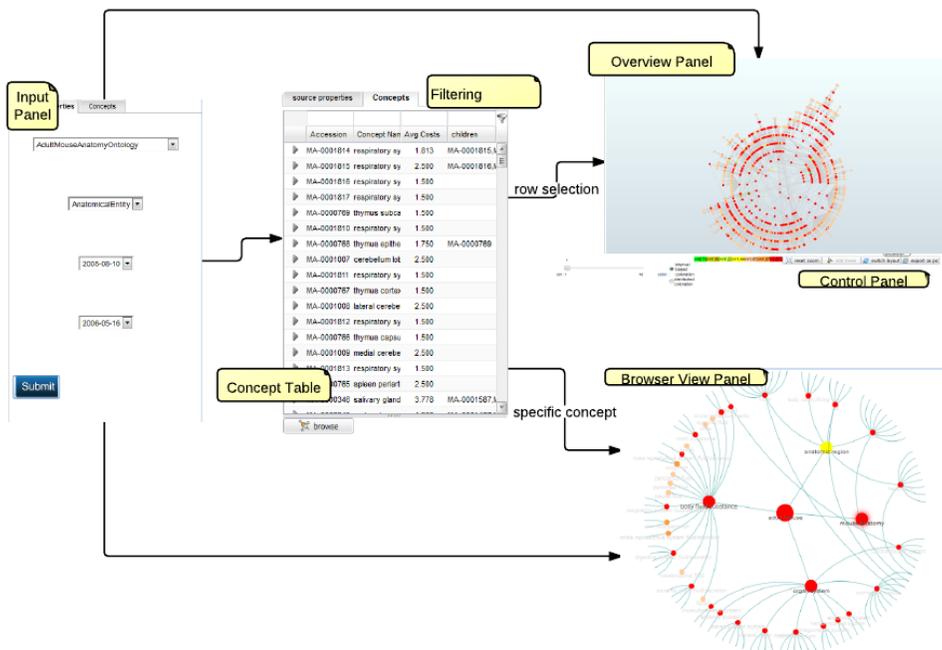


Abbildung 1: Graph-Workflow

ometrische Zoom einstellbar. Durch Auswahl eines Konzepts wird der Graph, von diesem Knoten aus, nach dem Fisheye-Zoom Prinzip [SB94] verzerrt.

Bei einer hohen strukturellen Komplexität einer Ontologie ist es möglich, dass der Graph nicht exakt visuell zu erfassen ist, deshalb ist im *Browser View Panel* eine zweite Ansicht der Ontologie verfügbar, dessen Komplexität durch einen Tiefenfilter der Größe 3 minimiert ist. Die Farbgebung der Konzepte, entspricht der selbigen wie im *Overview Panel*. Dieser Graph bietet eine dynamische Navigation durch die Ontologie, indem bei einem gewählten Konzept ein neuer Graph mit Tiefe 3 und diesem Konzept als Wurzel animiert erstellt wird. Die Elternkonzepte werden ebenfalls angezeigt, um innerhalb der Ontologie navigieren zu können.

Mithilfe der *Concept Table* können Informationen, wie z. B. der Name, die Kosten oder die Kinder eines Konzepts der Ontologie tabellarisch betrachtet werden. Die Tabelle ist sortierbar und darzustellende Informationen können über eine Filteroption eingeschränkt werden. Die Selektion eines Konzepts in der Tabelle wird im Graph durch Hervorhebung des Knotens gekennzeichnet, sodass die textuelle Information intuitiv mit dem Graph der Ontologie verknüpft werden kann. Des Weiteren kann ein selektiertes Konzept als Wurzelkonzept des Graphen im *Browser View Panel* verwendet werden.

3.1.2 Statistikrepräsentation

Die Evolution von Ontologien kann ebenfalls mithilfe von Diagrammen analysiert werden. REX bietet dafür quantitative sowie Sliding-Window Statistiken an. Die Grundlage der quantitativen Statistiken ist die Anzahl der Änderungsoperationen (Diff) zwischen aufeinanderfolgenden Versionen einer Ontologie. Des Weiteren besteht die Möglichkeit in der Sliding-Window Statistik die Evolution anhand des Verlaufs der durchschnittlichen Kosten für eine Region über einen gewählten Zeitraum mithilfe des Sliding-Window Verfahren zu veranschaulichen. Der prinzipielle Ablauf ist in Abb. 2 dargestellt.

Der Anwender hat bei der Visualisierung einer quantitativen Statistik die Wahl zwischen dem Vergleich von zwei Ontologien über einen Zeitraum und dem Vergleich von einer Ontologie über zwei Zeiträume. Je nach gewählter Option müssen die Namen der Ontologien bzw. Zeiträume im vorgesehenen Eingabepanel spezifiziert werden. Das resultierende Diagramm beinhaltet zwei Graphen, je nach gewählter Option entsprechen diese zwei Ontologien bzw. zwei Zeiträumen. Die Koordinaten eines Graphen sind durch die Zeit und die Anzahl der Änderungsoperationen gegeben. Mithilfe der generierten Diagramme können sich Nutzer zunächst einen ersten Überblick über die Evolution verschaffen, um anschließend auf Basis dessen ausgewählte Zeiträume und Regionen näher zu inspizieren.

Für die zweite Statistik wird ein Sliding-Window Verfahren verwendet. Der Anwender spezifiziert die Ontologie, den Zeitraum und die Konzepte, deren Analyseergebnisse visualisiert werden sollen. Darüber hinaus müssen für das Verfahren spezifische Parameter wie Fenstergröße und Schrittweite angegeben werden. Das Sliding-Window Verfahren berechnet schrittweise die durchschnittlichen Kosten innerhalb des Fensters für den gesamten Zeitraum. Das Fensterende wird zu Beginn mit dem spezifizierten Startzeitpunkt des gewählten Gesamtzeitraums gleichgesetzt. Nun werden mithilfe des Regionenalgorithmus

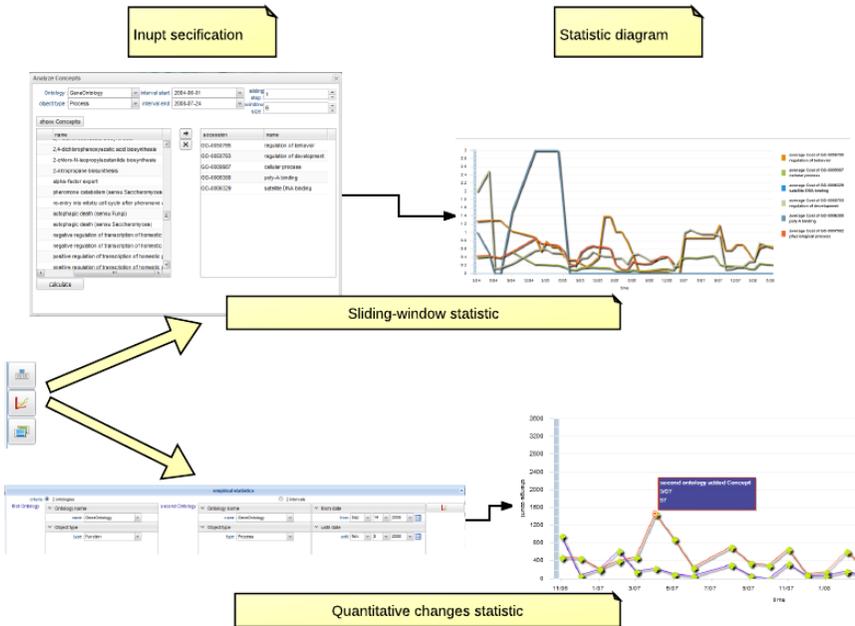


Abbildung 2: Statistik-Workflow

die durchschnittlichen Kosten für jedes Konzept für den fensterspezifischen Zeitraum ermittelt. Die ermittelten Kosten werden mit dem Endzeitpunkt des Fensters gespeichert. Danach wird das Fenster um die spezifizierte Schrittweite in die Zukunft geschoben und das Verfahren berechnet erneut die Kosten für die neue Position des Fensters. Das Verfahren endet, wenn das Fenster das Ende des Analysezeitraums erreicht hat. Die Koordinaten des resultierenden Graphen repräsentieren die entstandenen Kosten in Abhängigkeit vom Zeitpunkt.

3.2 Architektur

Aufgrund des hohen Grades der Interaktion, die durch die Menge an Kontroll- und Anzeigekomponenten geboten wird, zählt die Applikation zu den Rich Internet Applications. Prinzipiell wurden die Pakete bzw. Klassen sowohl auf Clientseite als auch auf Serverseite nach den einzelnen Workflows modular realisiert. Der Aufbau der Applikation gliedert sich in drei große Subkomponenten. Die Client-Komponente beinhaltet alle Klassen, die die Clientfunktionalitäten realisieren, d. h.. die Implementierung der Weboberfläche und das Event-Handling für die Interaktion. Generell ist die Oberfläche mit allen Kontrollelementen mithilfe des GWT, ExtGWT und SmartGWT implementiert. Die einzelnen Graphen wurden in ActionScript basierend auf der Flare Graphbibliothek sowie in JavaScript mithilfe der InfoVis Bibliothek realisiert und in die GWT-Umgebung einge-

bunden. Die Flare Bibliothek basiert auf dem IVR Modell (Information Visualization Reference Model), das eine Dekomposition des Visualisierungsprozesses definiert. Die Teilprozesse beinhalten die Datenspeicherung, das Mappen der Rohdaten auf Objekte visueller Form und die Speicherung der Informationen für das Aussehen der Objekte sowie das Rendering. Diese Bibliothek bietet eine hohe Diversität an Möglichkeiten für die Gestaltung der Visualisierung, sodass sich der Fisheye-Zoom oder der geometrische Zoom des Gesamtgraphen realisieren ließen. Das Layout des Übersichtsgraphen ist ein Radial Hierarchical Layout [BK01]. Dieses Layout ordnet die Knoten radial an, wobei mit zunehmender Tiefe bzgl. des Graphen die Entfernung der Knoten vom Zentrum zu nimmt. Die Präsentation der Ontologie im *Browser View Panel* ist durch einen Hypergraph realisiert [HL99]. Mithilfe dieser Darstellung wird der Fokus auf das Konzept im Zentrum konzentriert, welches der Anwender spezifiziert hat. Die Kommunikation der eingebetteten Komponenten und der GWT-Umgebung basiert auf JSNI (JavaScript Native Interface) Methoden, die es erlauben direkt JavaScript zu implementieren, um somit auf die einzelnen Komponenten im DOM-Baum zu zugreifen.

Die serverseitige Implementierung realisiert die Datenanfragebehandlung für die einzelnen Workflows und umfasst für die Berechnung der Regionen den Regionalalgorithmus. Diese Berechnungen werden sowohl für die Sliding-Window Statistik als auch für die zu visualisierenden Daten benötigt. Um dem Anwender eine große Flexibilität bzgl. der Parameterspezifikation zu gewähren, erfolgt die Berechnung der Regionen ad hoc. Eine Materialisierung der Analyseergebnisse von potenziellen Konfigurationen hätte den Vorteil performante Anfragen zu gewährleisten, würde aber entweder einen immensen Speicheraufwand beanspruchen oder eine Einschränkung der Eingabeparameter bedeuten. Des Weiteren werden für den Regionalalgorithmus, die quantitativen Statistiken und die anzuzeigenden Informationen SQL-Queries bereit gestellt, so dass die Ontologieversionen oder andere relevante Daten von der Datenbank extrahiert werden können. Eine Datenbank zur Verwaltung von Ontologieversionen stellt das Backend der Applikation dar.

4 Evaluation

Um die Anwendbarkeit der Applikation zu testen, erfolgte eine Evaluation anhand unterschiedlich großer Ontologien aus OnEX [HKGR09]. Beispielfhaft werden nachfolgend Ergebnisse für die Teilontologie Molekulare Funktionen der GO von 2007 bis 2009 präsentiert, welche 9214 Konzepte und 10604 Relationen enthält.

Das Ergebnis der strukturellen Repräsentation ist in Abb. 3 ersichtlich. Aufgrund der Rotverteilung(schwarz) ist intuitiv die Instabilität einiger Regionen der Ontologie erkennbar. Konzepte, die rot eingefärbt sind, weisen überdurchschnittliche *avg_costs* über 1,5 auf. Im Folgenden erfolgt eine nähere Analyse der Kinderkonzepte des Wurzelkonzepts. Auffällig ist der grün(hellgrau) gefärbte Sektor im rechten oberen Teil des Graphen. Ein Großteil dieses Sektors wird durch die Region *binding* gebildet. Die *avg_costs* betragen lediglich 0,62. Diese geringen Kosten resultieren aus den geringen *avg_costs* der Konzepte die in dieser Region liegen (siehe Abb. 4(a)). Die *avg_costs* dieser Konzepte betragen weniger als 0,25. Aufgrund der niedrigen Kosten ist die Region des

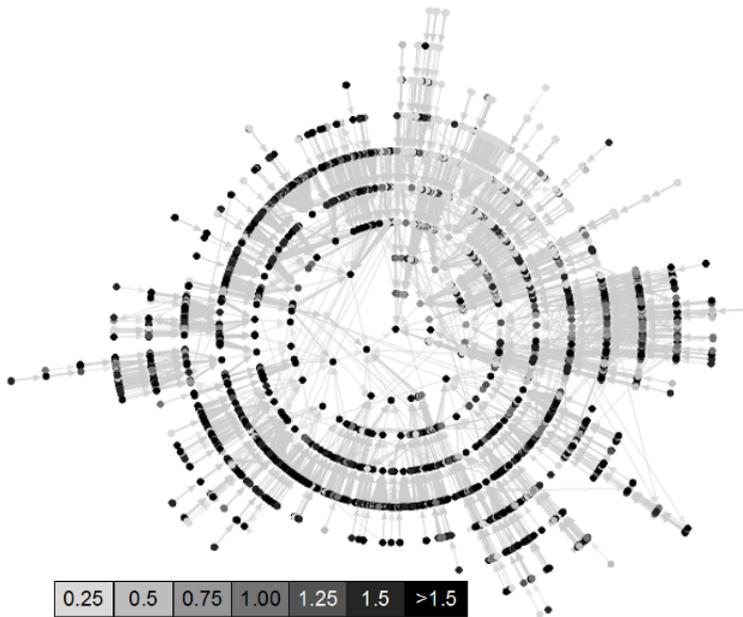


Abbildung 3: Änderungsintensitäten in GO Molekulare Funktionen zwischen 2007 und 2009

Konzepts *binding* als stabil zu erachten. Ebenfalls eine stabile Region ist *transcription regulator activity* mit $avg_costs < 0,5$. Im Gegensatz zu den stabilen Regionen existieren für diesen Zeitraum auch instabile Regionen, z. B. *catalytic activity* (siehe Abb. 4(b)). Insbesondere in den Teilgebieten *DNA polymerase activity*, *peptidase activity* sowie *electron carrier activity* fanden mit $avg_costs > 11$ immense Änderungen statt, welche zur Instabilität der gesamten Region beitragen.

Um festzustellen, ob die Regionen *transcription regulator activity* und *catalytic activity* einer ständigen Veränderung unterlagen oder teilweise stabil waren, soll im Folgenden eine Sliding-Window Analyse über einen längeren Zeitraum durchgeführt werden. Dazu wurden die folgenden Parameter verwendet: der Zeitraum umfasst 2004–2010 bei einer Fenstergröße von sechs Versionen und einer Schrittweite von einer Version. Die Ergebnisse sind in Abb. 4(c) dargestellt. Die Region *catalytic activity* (dunkle Linie) weist häufig Instabilitäten auf, z. B. erfolgten von Juni 2007 bis Januar 2008 zahlreiche Änderungen was sich in einem Anstieg der avg_costs von 0,05 auf 2 widerspiegelt. Seither sind innerhalb der Region weniger Modifikationen, d. h. eine ansteigende Stabilität, zu beobachten (Abfall der avg_costs). Die Region *transcription regulator activity* weist aufgrund der geringen avg_costs von höchstens 0,3 über den gesamten Zeitraum eine eher geringe Änderungsintensität auf. In den letzten Jahren stiegen die avg_costs nie über 0,1, was vermuten lässt, dass die Ontologieentwicklung in diesem Bereich nahezu abgeschlossen ist.

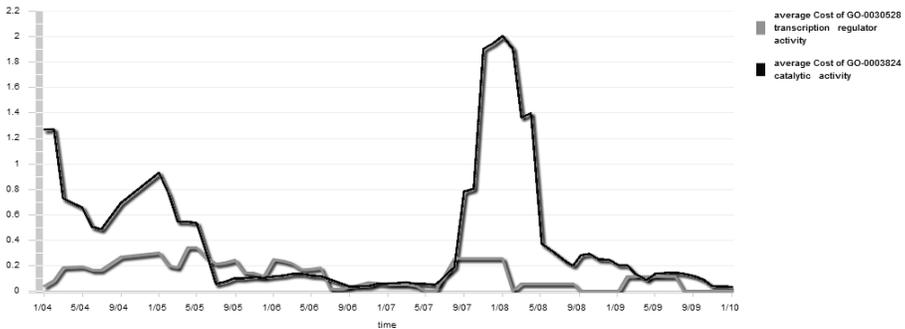
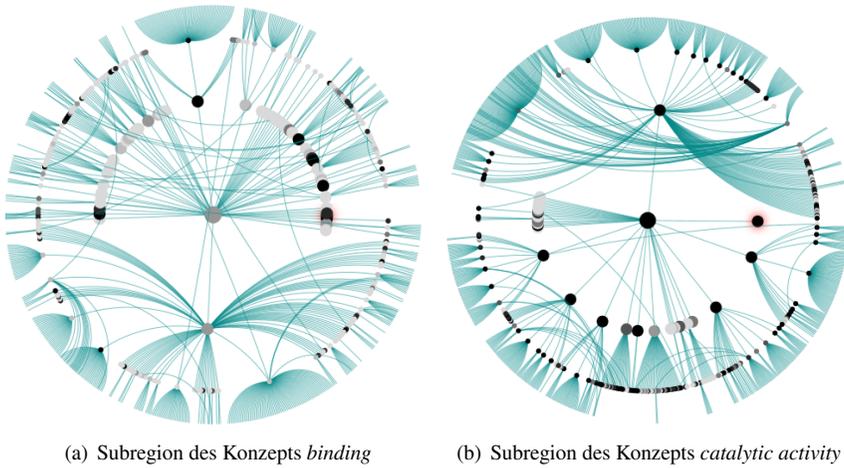


Abbildung 4: Detailanalysen für GO Molekulare Funktionen

5 Verwandte Arbeiten

Verwandte Arbeiten gliedern sich in Arbeiten aus dem Bereich der Ontologieevolution (siehe [FMK⁺08] für Übersichtsartikel) und web-basierten Systemen, welche einen Zugang zu Ontologien erlauben und Inhalte entsprechend visualisieren können.

In den Lebenswissenschaften haben sich mit BioPortal [NSW⁺09] sowie der OBO Foundry [SAR⁺07] zwei Plattformen entwickelt, welche die Verwaltung zahlreicher biomedizinischer Ontologien verfolgen. Es werden primär Ontologien sowie zugehörige Versionen zum Download angeboten, so dass Anbieter ihre Ontologien (inkl. Versionen) über diese Plattformen zentral veröffentlichen können. Für einige Ontologien existieren ebenfalls spezielle Webapplikationen. AmiGO⁶ ist ein Beispiel zur Visualisierung der GO. Durch einen Tree-Browser ist es möglich die Abhängigkeiten bzw. Zusammengehörigkeit

⁶<http://amigo.geneontology.org/cgi-bin/amigo/go.cgi>

von Konzepten durch das interaktive Selektieren zu identifizieren. Des Weiteren lassen sich durch Selektion eines Konzepts weitere Eigenschaften, wie z. B. Name und Beschreibung anzeigen. Der Tree-Browser ist als Single-Rooted-Tree realisiert, so dass die Problematik bezüglich der Darstellung der Polymorphie existiert. In einer Graphansicht können Graphen in Form von Bildern aus Konzepten und Relationen im Tree-Browser generiert werden. Diese Visualisierungsform bietet dadurch keine weiteren Möglichkeiten der Interaktion, so dass eine weitere Exploration nur über den Tree-Browser möglich ist. Die Visualisierung basiert immer auf der neuesten GO Version.

OnEX [HKGR09] ist eine Webapplikation, die auf die Analyse der Evolution von Ontologien in den Lebenswissenschaften spezialisiert ist. Die Präsentation gliedert sich in die Wiedergabe von Evolutionstrends mittels quantitativer Statistiken und Tabellen sowie der Evolution von Konzepten innerhalb einer Ontologie. Die quantitativen Analysen beruhen auf der Anzahl von Änderungsoperationen, die zwischen Versionen einer Ontologie aufgetreten sind. Es besteht die Möglichkeit eines Browsing durch die Historie der Versionen. Dadurch ist es möglich sich sowohl die Veränderung einer Ontologie im Ganzen als auch die Veränderung einzelner Konzepte zu betrachten. Eine baumartige Navigation durch die Inhalte einer Ontologie wird nicht angeboten.

Im Gegensatz zu vorherigen Arbeiten fokussiert diese Arbeit auf die visuelle Präsentation der Evolution von großen Ontologien. Mithilfe eines speziellen Regionalalgorithmus kann die Intensität der Veränderungen für jede Region einer Ontologie berechnet und dynamisch visualisiert werden. So können Nutzer intuitiv stark bzw. schwach veränderte Teile innerhalb einer Ontologie ausmachen und ggf. auch im Detail inspizieren.

6 Zusammenfassung und Ausblick

Diese Arbeit befasste sich mit der Thematik der Datenvisualisierung in Bezug auf die Evolution von Ontologien der Lebenswissenschaften. Die Untersuchung der Evolution von Ontologien ist notwendig, um Forschungstrends zu identifizieren, die Intensität der Veränderungen von Ontologien zu messen und Entwicklungen von Ontologien zu planen bzw. zu überwachen. Aufgrund der hohen Datenmenge ist eine textuelle Repräsentation ungeeignet. Diesbezüglich wurde die Webanwendung REX entwickelt, welche auf Basis eines Regionalalgorithmus Veränderungen innerhalb einer Ontologie aggregiert und visualisiert. REX bietet Anwendern diverse Workflows, um Ergebnisse des Regionalalgorithmus als Graph oder Statistik zu präsentieren.

Derzeit arbeitet REX auf Basis von Ontologieversionen aus dem OnEX Repository. Künftig wäre es wünschenswert beliebige Ontologien mit REX analysieren zu können. Ein Anwender sollte z. B. in der Lage sein eigene Ontologieversionen in REX zu laden, um anschließend für diese Ontologie eine Regionenanalyse durchführen zu können.

Danksagung. Die vorliegende Arbeit entstand auf der Basis meiner Bachelorarbeit. Mein Dank gilt meinen Betreuern Anika Groß und Dr. Michael Hartung, die mir stets helfende Ratschläge und Verbesserungsvorschläge geboten haben.

Literatur

- [BAW⁺05] A. Bairoch, R. Apweiler, C.H. Wu, et al. The universal protein resource (UniProt). *Nucleic acids research*, 33(suppl 1), 2005.
- [BK01] G. Book and N. Keshary. Radial Tree Graph Drawing Algorithm for Representing Large Hierarchies. *University of Connecticut*, 2001.
- [BS06] O. Bodenreider and R. Stevens. Bio-ontologies: current trends and future directions. *Briefings in Bioinformatics*, 7(3), 2006.
- [FMK⁺08] G. Flouris, D. Manakanatas, H. Kondylakis, et al. Ontology change: classification and survey. *The Knowledge Engineering Review*, 23(2), 2008.
- [Gen08] Gene Ontology Consortium. The Gene Ontology project in 2008. *Nucleic Acids Research*, 36(Database Issue), 2008.
- [Gru95] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5), 1995.
- [HGKR10] M. Hartung, A. Groß, T. Kirsten, and E. Rahm. Discovering Evolving Regions in Life Science Ontologies. In *Data Integration in the Life Sciences (DILS)*, 2010.
- [HGR12] M. Hartung, A. Groß, and E. Rahm. COnTo-Diff: Generation of Complex Evolution Mappings for Life Science Ontologies. *Journal of Biomedical Informatics*, 2012.
- [HKGR09] M. Hartung, T. Kirsten, A. Groß, and E. Rahm. OnEX: Exploring changes in life science ontologies. *BMC Bioinformatics*, 10(1), 2009.
- [HKR08] M. Hartung, T. Kirsten, and E. Rahm. Analyzing the evolution of life science ontologies and mappings. In *Data Integration in the Life Sciences (DILS)*, 2008.
- [HL99] R. Haenni and N. Lehmann. Efficient hypertree construction. Technical report, Institute of Informatics, University of Fribourg, 1999.
- [HSL09] D.W. Huang, B.T. Sherman, and R.A. Lempicki. Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists. *Nucleic acids research*, 37(1), 2009.
- [Kei02] D.A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 2002.
- [NM02] N.F. Noy and M.A. Musen. PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions. In *Proc. AAAI/IAAI*, 2002.
- [NSW⁺09] N.F. Noy, N.H. Shah, P.L. Whetzel, et al. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic acids research*, 37(suppl 2), 2009.
- [SAR⁺07] B. Smith, M. Ashburner, C. Rosse, et al. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11), 2007.
- [SB94] M. Sarkar and M.H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12), 1994.
- [Ste03] G. Steger. *Bioinformatik: Methoden zur Vorhersage von RNA-und Proteinstrukturen*. Birkhauser, 2003.

Integration von Geodaten in ein Planungssystem

Björn Buchwald

mam09bhi@studserv.uni-leipzig.de

Universität Leipzig, Bachelor Studiengang Informatik

Zusammenfassung: Dieses Paper beschreibt die Konzeption einer prototypischen Softwarelösung für das „Dinner Hopping“. Dabei sind Teams gegeben, die an bestimmten Orten beheimatet sind. Jedes Team richtet entweder eine Vor-, Haupt oder Nachspeise an. Ziel ist es, jedem Team eine Route, bestehend aus drei Orten zuzuteilen, sodass jedes Team ein komplettes 3-Gänge-Menü einnimmt. Dabei müssen bestimmte Nebenbedingungen eingehalten werden. Es wird auf den Zusammenhang mit dem Travelling Salesman Problem eingegangen. Daraufhin folgt eine Lösungsskizze, welche auf Geokodierung, Routenberechnung, Möglichkeiten der Visualisierung von geografischen Daten in einer Java-Applikation und verwendete Services eingeht. Schließlich findet eine Beschreibung des Routenplanungsalgorithmus statt.

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Konzeption einer prototypischen Softwarelösung, welche durch die Integration von Geodaten ein Planungssystem zur Routenfindung zwischen verschiedenen hintereinander stattfindenden Veranstaltungsorten ermöglicht. Geodaten werden hier als geographische Daten, das heißt Informationen zu Adressen, wie geographische Lage in Breiten- und Längengrad, die Entfernung zwischen zwei Adressen, sowie Informationen über die Route, also den Weg von einer Adresse zu einer anderen, aufgefasst. Diese Daten sollen über verschiedene Services wie Internetdienste oder lokale Datenquellen integriert werden. Um die Problemstellung zu verdeutlichen, wird im Folgenden der konkrete Anwendungsfall näher erläutert.

Das „Dinner Hopping“, eine Veranstaltung der WILMA („Willkommens-Initiative für in Leipzig mitstudierende Ausländer“) findet in regelmäßigen Abständen statt und dient der Erleichterung der Kontaktaufnahme Leipziger Austauschstudenten untereinander und zu deutschen Studenten. Dabei bewirten die Studenten in Zweierteams in ihrer Wohnung für einen Gang eines Drei-Gänge-Menüs vier andere Studenten. Bei den beiden anderen Gängen sind sie zu Gast bei ihren Kommilitonen. Die Herausforderung besteht darin, jedem Team möglichst kurze Wege von einem Gang zum nächsten zu garantieren. Eine genaue Beschreibung der Nebenbedingungen ist in Abschnitt 2 gegeben.

Zurzeit wird die Planung dieser Veranstaltungen per Hand erledigt, was ein sehr zeitaufwändiges Unterfangen darstellt, wenn man bedenkt, dass bis zu 39 Teams teilnehmen. Um diesen Vorgang deutlich zu verkürzen, soll der in dieser Arbeit beschriebene Prototyp konzipiert werden.

Die Arbeit gliedert sich im Weiteren wie folgt. Abschnitt zwei befasst sich mit der sich aus der Einführung ergebenden wissenschaftlichen Problemstellung. Dabei wird weniger auf die Integration von Geodaten eingegangen, sondern vielmehr die wissenschaftliche Grundlage des sich ergebenden Problems beschrieben. In Abschnitt drei wird die algorithmische Umsetzung der Routenplanung und damit ein Lösungsansatz für die beschriebene Problemstellung vorgestellt. Der darauffolgende Abschnitt befasst sich mit der Konzeption des Prototypen und dessen Strukturierung in fünf Teilkomponenten. Abschnitt fünf betrachtet den gezeigten Lösungsansatz kritisch und gibt einen Ausblick auf aus Platzgründen und frühzeitiger Entwicklungsphase nicht behandelte Themen.

2 Problembeschreibung sowie Vergleich des Travelling Salesman Problems mit dem „Dinner Hopping“ Problem

2.1 „Dinner Hopping“ Problem

Die einzuhaltenden Nebenbedingungen des in der Einleitung dargestellten Problems sind nachfolgend beschrieben. Ein Schwerpunkt der Routenberechnung liegt (1) auf der Bestimmung kurzer Routen zwischen den drei Gängen bevorzugt für Teams, welche die Hauptspeise anrichten. (2) Es darf keine doppelten Begegnungen der Teams geben. (3) Der Gang des Gastes muss mit dem Gang des Gastgebers übereinstimmen. Richtet ein Team beispielsweise die Hauptspeise an, so müssen Teams, die bei diesem zu Gast sind, für diesen Gang ebenfalls für die Hauptspeise vorgemerkt sein. Die Anfahrt zum ersten Gang bzw. die Abreise vom letzten Gang können vernachlässigt werden. (4) Jedes Team richtet genau einen Gang selbst an. (5) Der Veranstaltungsort dieses Ganges ist der Heimortort des jeweiligen Teams. (6) Der Gastgeber bewirtet jeweils genau zwei weitere Teams. (7) Ein Team muss jeden Gang (Vor-, Haupt- und Nachspeise) genau einmal durchlaufen. (8) Bei den beiden restlichen Gängen ist es bei einem anderen Team zu Gast.

Da dies für jedes Team gilt, wird von der Annahme ausgegangen, dass die Anzahl der Teams durch drei teilbar ist. Ein Veranstaltungsort, an dem die Vor-, Haupt bzw. Nachspeise stattfindet, wird als Vor-, Haupt- bzw. Nachspeiseort bezeichnet. Die anrichtenden Teams sind entsprechend Vor-, Haupt- bzw. Nachspeiseteams. Diese Einteilung der Teams erfolgt im Prototyp selbst durch den Routenplanungsalgorithmus und ist dementsprechend nicht im Vorfeld gegeben. Weitere Annahmen, die speziell im Lösungsansatz der Routenplanung dieser Arbeit getroffen werden, sind in Abschnitt 5.1 erläutert. Ein Veranstaltungsort entspricht einer Adresse. Durch die Geokodierung wird die Adresse in Breiten- und Längengrad umgewandelt und auf diese Weise ein Veranstaltungsort definiert. Als Entfernung zwischen diesen Orten kann sowohl die reine Luftlinie, als auch die kürzeste auf Verkehrswegen beruhende Strecke gesetzt werden. Auf die Umsetzung der Entfernungsbestimmung wird in Abschnitt 4.2.2 eingegangen. Im Folgenden ist eine komplette Routenplanung mit einer kurzen Erläuterung des Ablaufs der Veranstaltung gegeben. Abb. 1 zeigt eine komplette Planung der Routen für jedes Team. Die Routen sind auf statischen Google-Karten eingezeichnet.

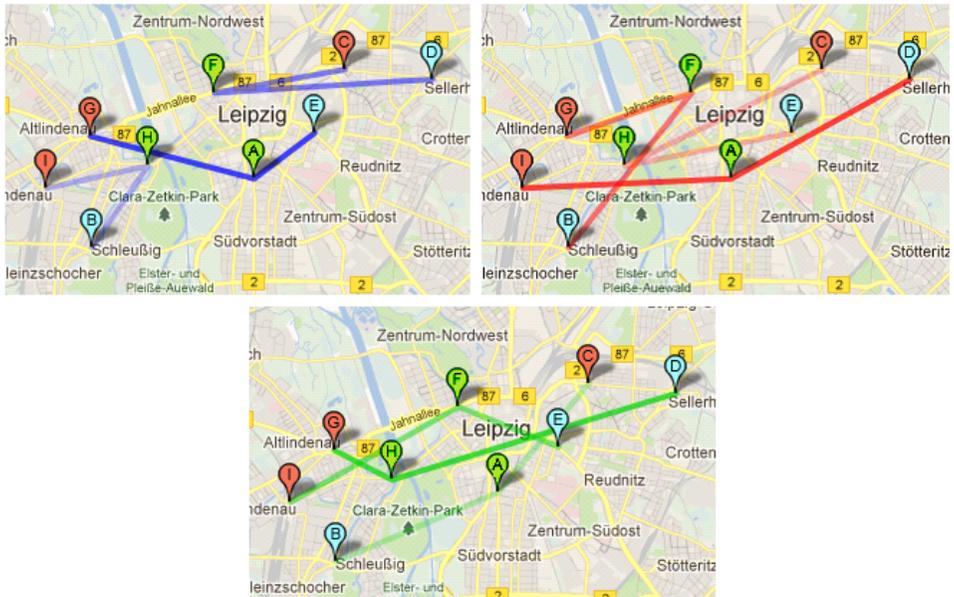


Abbildung 1: Routenplanung für Vorspeiseteams (oben links), Hauptspeiseteams (unten) und Nachspeiseteams (oben rechts)

Das Planungssystem bekam einen Datensatz mit 9 Leipziger Orten zur Bearbeitung. Die Heimatorte der Teams sind durch Buchstaben markiert. Beispielsweise *besitzt Team H den Heimatort H* und richtet dort die Hauptspeise an (siehe (4)). Beim Lesen der Karten ist zu beachten, dass ein Pfad stets bei einem Vorspeiseort (im Beispiel B, D oder E) beginnt und über einem Hauptspeiseort (A, F oder H) zu einem Nachspeiseort (C, G oder I) verläuft, da (7) erfüllt sein muss. Pfade der Vorspeise- (Abb. 1 oben links), Haupt- und Nachspeiseteams beginnen immer an einem Vorspeiseort. Die Visualisierung zeigt, dass die Nebenbedingungen (1) bis (8) des DHP eingehalten werden. Hier ein Beispiel:

Die Route von *Team H als Hauptspeiseteam* beginnt bei Vorspeiseteam D (Pfad von H in Abb. 1 unten ersichtlich). H ist also zu Gast bei D. Danach geht H zu seinem Heimatort, um dort den Hauptgang anzurichten. Dort sind C (Pfad von C in Abb. 1 oben rechts ersichtlich) und B (Pfad in Abb. 1 oben links ersichtlich) als Gäste anwesend. Die Route verläuft schließlich weiter zu Nachspeiseteam G. Hier ist H wiederum Gast.

Eine teilweise Formalisierung des Problems ist im Folgenden gegeben.

Gegeben ist eine Menge von Teams T

$$T = \{t \in \mathbb{N} \mid |T| \bmod 3 = 0\}$$

Analog wird die Menge der Orte O definiert

$$O = \{n \in \mathbb{N} \mid |O| \bmod 3 = 0\}$$

Jedem Team ist genau ein Heimatort zugeordnet, d. h. es existiert eine bijektive Abbildung

$$f : T \rightarrow O$$

Gegeben ist weiterhin eine symmetrische $n \times n$ Entfernungsmatrix

$$M = (m_{i,j}) = (m_{j,i}); i, j \in \{1, \dots, n\}$$

Mit $m_{i,j}$ als Entfernung von Ort i zu Ort j . Da ein Ort eine Adresse einer Stadt beschreibt, sind die folgenden Bedingungen erfüllt. Es existiert immer eine Verbindung zwischen zwei Orten, sodass

$$\forall i, j : 0 < m_{i,j} < \infty; i, j \in \{1, \dots, n\}$$

Die Dreiecksungleichung ist erfüllt.

$$m_{i,j} \leq m_{i,k} + m_{k,j}; i, j, k \in \{1, \dots, n\}; i \neq j, j \neq k, k \neq i$$

Dieses Problem wird im Weiteren als „*Dinner Hopping Problem*“ (*DHP*) bezeichnet. Es lassen sich einige grundlegende Merkmale des DHP erkennen:

- Es sind verschiedene Orte und deren Entfernung gegeben.
- Es existieren verschiedene Personen (Teams), die sich zwischen den Orten von einem Start- zu einem Zielort bewegen.
- Die zurückgelegte Gesamtstrecke aller Teams soll minimal sein. Hier sind weitere Möglichkeiten denkbar, wie die minimale durchschnittliche Strecke pro Team oder die minimale Gesamtstrecke explizit für Hauptpeisetams.
- Städte werden von einer Person (Team) nur einmal besucht.

Auf Grundlage dieser Eigenschaften lassen sich einige Ähnlichkeiten zum „*Travelling Salesman Problem*“ (TSP) bzw. zu Abwandlungen von diesem erkennen. Im Folgenden werden Gemeinsamkeiten und Unterschiede zwischen DHP und TSP aufgezeigt. Des Weiteren wird untersucht, ob das TSP sowie Lösungsansätze von diesem auf das DHP und dessen Lösung übertragbar sind.

2.2 Travelling Salesman Problem (TSP)

Die Definition des TSP im Städtekontext lautet: „Das *TSP* oder auch *Problem des Handlungsreisenden* sucht eine Rundreise über n gegebene Städte. Jede Stadt wird genau einmal besucht. Die zurückgelegte Gesamtstrecke soll minimal sein.“ [Sch97]

Die allgemeine Formulierung des Problems ist folgende: „Ein gerichteter stark zusammenhängender Graph $G = N, A$ ist gegeben. N ist die Menge der *Knoten* $1, 2, \dots, n$ (*Knoten* \equiv *Stadt*) und A ist die Menge der gerichteten *Kanten* (\equiv *Städteverbindungen*), wobei jeder Kante (i, j) Kosten $c_{i,j}$ zugeordnet werden. Gesucht ist ein gerichteter Hamilton-Kreis (Hamilton-Zyklus) Z von G mit minimalen Gesamtkosten $c(Z) := \sum_{(i,j) \in Z} c_{i,j}$.“ [MS09] „Dabei ist ein Hamilton-Kreis eine Abfolge von Knoten und verbindenden Kanten (abwechselnd), die alle Knoten aus G genau einmal betrachtet.“ [MS09] . c ist analog der oben definierten Entfernungsmatrix M mit den Werten $m_{i,j}$ zu betrachten.

Sofort lassen sich Gemeinsamkeiten zu den oben beschriebenen Merkmalen feststellen. Sowohl Orte als auch Entfernungen zwischen diesen sind gegeben. Es existiert eine Person, die sich von einem Start- zu einem Zielort bewegt. Die zurückgelegte Strecke dieser Person soll minimal sein und jede Stadt wird nur einmal besucht.

Die folgenden Eigenschaften werden in der allgemeinen Definition in [Sch97] jedoch nicht zwangsläufig vorausgesetzt. Das DHP verlangt, dass (1) die gegebene Entfernungsmatrix symmetrisch ist. (2) Außerdem muss die Dreiecksungleichung erfüllt sein. (3) Des Weiteren dürfen Entfernungen nicht unendlich sein, d.h. es gibt in jedem Fall eine Verbindung zwischen zwei Orten. Die gegebene Definition ist demnach in Bezug auf das DHP sehr allgemein gehalten und schließt Fälle ein, die das DHP nicht erfüllen. (4) Dazu kommt, dass beim DHP mehrere Handlungsreisende zu betrachten sind. (5) Weiterhin besteht eine Tour eines Reisenden nur aus drei Orten, wobei (6) diese Tour keine Rundreise darstellt, da (7) jedes Team an verschiedenen Orten startet und endet. Es sind also Variationen des TSP gesucht, welche die Eigenschaften (1) bis (7) möglichst genau erfüllen bzw. durch diese charakterisiert werden.

2.3 Multiples Travelling Salesman Problem (mTSP)

„Das mTSP stellt eine Verallgemeinerung des TSP dar. Die verallgemeinerte Variante kann wie folgt definiert werden. Gegeben ist eine Menge von Knoten (Städten) und n Handlungsreisende, die sich an einem Depot befinden. Das mTSP besteht darin, Touren für alle Handlungsreisende zu finden, die an diesem Depot beginnen und an diesem enden, sodass jeder Knoten genau einmal besucht wird und die Gesamtkosten zur Besichtigung aller Knoten minimiert sind.“ (vgl. [Bek06])

Interessant sind hierbei die Variationen „multiple Depots“ und „single Depots“. Der „single Depots“ Fall entspricht dem gerade beschriebenen allgemeinen Fall. „Im „multiple Depots“ Fall existieren mehrere Depots, wobei an jedem Depot eine gewisse Anzahl von Handlungsreisenden starten. Daraufhin sind zwei Varianten denkbar. Entweder kehren die Handlungsreisenden am Ende ihrer Tour zu ihrem Depot zurück oder sie beenden die Tour an einem beliebigen anderen Depot. Dabei muss die anfängliche Anzahl der Handlungsreisenden an den Depots nach Beenden der Touren gleich bleiben.“ (vgl. [Bek06])

Das mTSP geht auf Eigenschaft (4) ein. Es werden mehrere Handlungsreisende betrachtet, was im DHP ebenfalls der Fall ist. Des Weiteren können Routen an verschiedenen Depots enden, was die Eigenschaften (6) und (7) teilweise beinhaltet. Außerdem starten mehrere Handlungsreisende ihre Tour vom selben Ort, was ebenfalls eine Gemeinsamkeit zum DHP darstellt, da immer drei Teams ihre Route am Vorspeiseort beginnen. Da hier allerdings alle Handlungsreisenden ihre Tour am selben Ort beginnen, ist Eigenschaft (6) nicht vollständig enthalten.

Probleme, die (1) bis (3) sowie (5) angehen, sind das symmetrische und metrische TSP, sowie das Euclidian Traveling Salesman Selection Problem (TSSP). Diese werden hier aus Platzgründen nicht näher erläutert. Erstere werden in [MS09], [Rei94] und [Sch00] näher beschrieben. Informationen zum TSSP finden sich in [HM95].

Zusammenfassend lässt sich sagen, dass keines der betrachteten Probleme das DHP vollständig beinhaltet und es aus diesem Grund keine Lösung für eines der Probleme gibt, die das DHP vollständig löst. Trotzdem lassen sich einige Gemeinsamkeiten feststellen, wie die Suche nach einem Weg mit minimaler Summe der Distanzen für den Handlungsreisenden und die Einbeziehung mehrerer Handlungsreisender. Probleme, wie die Ungleichheit der Anfangs- und Zielorte der Teams, sowie das Finden der minimalen Wegstrecke für jedes einzelne Team, konnten auch durch die Variationen des TSP nicht gelöst werden.

3 Algorithmus der Routenberechnung

3.1 Beschreibung des Routenplanungsalgorithmus

Zunächst ist es von großer Bedeutung die Nebenbedingung (1) aus Abschnitt zwei noch einmal genauer zu betrachten. Um diese Bedingung zu erfüllen, wird die Berechnung der Routen für Hauptspeisetams im ersten Schritt durchgeführt. Dies wird wie folgt realisiert:

1. Es erfolgt eine Vorauswahl der Orte¹, welche die geringste Entfernung zu allen anderen Orten aufweisen. Die Annahme ist dabei, dass für diese Orte im späteren Verlauf die potenziell kürzesten Routen entstehen. Für jeden ersten Ort i wird die Summe s_i über die Distanzen zu allen anderen Orten bestimmt. Mit der definierten Entfernungsmatrix in Abschnitt 2 ergibt sich:

$$s_i = \sum_{l=0}^n m_{i,l} = \sum_{l=0}^n m_{l,i}; i \in \{1, \dots, n\}$$

Dabei werden die $n/3$ Orte mit den kleinsten Distanzen ausgewählt und provisorisch als Hauptspeiseorte festgelegt. Diese Orte definieren die Menge H . Die $2n/3$ verbleibenden Orte definieren eine Menge Z . Durch dieses Verhalten werden die Auswahlmöglichkeiten der Hauptspeiseorte minimiert. Auf eine genaue Betrachtung wird in Abschnitt 5.2 verwiesen.

2. Um eine Route über Vor-, Haupt- und Nachspeise zu erstellen, werden im Folgenden für jeden in Schritt 1 ausgewählten Ort $h \in H$ die zwei am nächsten zu ihm gelegenen Folgeorte $z_1, z_2 \in Z, z_1 \neq z_2$ ausgewählt. Das Wählen eines Folgeortes mit kleinster Entfernung entspricht der Nearest Neighbor Heuristik. Es ist darauf zu achten, dass bereits ausgewählte Folgeorte für andere Orte nicht mehr in Frage kommen.
3. In diesem Schritt werden die eigentlichen Vor-, Haupt- und Nachspeiseorte festgelegt. Außerdem erfolgt die endgültige Festlegung der Routen der Hauptspeisetams. Jede Gruppe, bestehend aus h und den zwei zugewiesenen Orten z_1, z_2 , wird so zu einer Route verbunden, dass für die Summe s der Distanzen gilt:

$$s = \min(m_{a,b} + m_{b,c}), a, b, c \in \{z_1, z_2, h\} \wedge a, b, c \text{ verschieden.}$$

¹Es ist zu beachten, dass jedem Team genau ein Ort zugeordnet ist. Ein Ort spezifiziert also auch das Team, welches dort beheimatet ist.

b stellt den „mittleren“ Ort der Route dar. Erst jetzt wird b als Hauptspeiseort und das beheimatete Team als Hauptspeisetem definiert. Die Route von b wird von a nach b über c festgelegt. Die angrenzenden Orte a , c werden zufällig als Vor- und Nachspeiseorte, die dort beheimateten Teams als Vor- und Nachspeisetem definiert. Damit sind die Routen für die Hauptspeisetems festgelegt.

In Schritt 2 erfolgt die Festlegung der Routen für Vor- und Nachspeisetems unter Einhaltung von Bedingung (2) aus Abschnitt 2. Aus diesem Grund wird in jedem Team-Objekt vermerkt, welche Teams auf der Route bereits gesehen wurden. Die Berechnung der Routen für Vorspeisetems wird wie folgt realisiert.

1. Die Vorspeisetems benötigen zuerst einen Folgeort, an dem sie die Hauptspeise einnehmen. Hier kommt wiederum die Nearest Neighbor Heuristik zum Einsatz. Es wird jeweils der Hauptspeiseort mit der kürzesten Entfernung ausgewählt. Wird ein Hauptspeiseort für ein Vorspeisetem ausgewählt, so steht er den anderen Vorspeiseorten nicht mehr zur Auswahl zur Verfügung. Er wird also aus der zugehörigen Liste entfernt. *Aus diesem Grund ist die letztendliche Verteilung der Vorspeiseorte auf die Hauptspeiseorte stark von der Reihenfolge abhängig, in der die Vorspeiseorte durchlaufen werden.* Um flexible Ergebnisse zu erhalten und für eine möglichst faire Verteilung wird diese Reihenfolge zufällig gewählt. Die Menge der Hauptspeiseorte wird dagegen für jeden Vorspeiseort sequentiell durchlaufen. Zusätzlich muss darauf geachtet werden, dass das Hauptspeisetem nicht bereits vom Vorspeisetem gesehen wurde, um Bedingung (2) zu erfüllen. In diesem Fall wird der entsprechende Hauptspeiseort während der Auswahl übersprungen.
2. Als nächstes wird den Vorspeisetems ein Nachspeiseort zugeteilt. Dieser Schritt erfolgt weitgehend analog zu Schritt 1, allerdings werden jetzt die Nachspeiseorte ausgewählt, welche die kürzeste Entfernung zu den in Schritt 1 bestimmten Hauptspeiseorten besitzen. Weiterhin ist zu beachten, dass das Vorspeisetem auf seiner bisherigen Route sowohl an seinem Heimatort ein Hauptspeisetem bewirte hat, als auch am Hauptspeiseort zu Gast bei einem Hauptspeisetem war. Das heißt, alle Nachspeiseorte, die diese beiden Teams zu Gast haben, werden für eine Auswahl ausgeschlossen.

Die Schritte 3 und 4 befassen sich mit der Berechnung von Routen der Nachspeiseorte. Diese verläuft analog den Schritten 1 und 2. Eine Beschreibung wird aus Platzgründen nicht gegeben bzw. kann aus den vorherigen Schritten abgeleitet werden. Bereits an den ersten beiden Schritten ist zu erkennen, dass die Möglichkeiten einen Nachfolgeort auszuwählen durch die zunehmende Menge bereits gesehener Teams pro Team in jedem Schritt abnehmen. Aus diesem Grund besteht die Möglichkeit, dass in einem Schritt kein passender Folgeort für ein Team gefunden wird. Eine erste Maßnahme, dieses Problem zu beheben, ist eine erneute Berechnung mit abgeänderter Reihenfolge des Durchlaufens der Orte² durchzuführen. Außerdem werden die Schritte 1 bis 4 durch rekursive Aufrufe mit Backtracking folgendermaßen realisiert:

²Siehe Hervorhebung in Schritt 1

Da die einzelnen Schritte analog verlaufen, werden sie in ein und derselben Java Methode implementiert. Die Schritte werden als Zustände codiert und der Methode als Parameter übergeben, um anzuzeigen, von welchen Heimatorten zu welchen Folgeorten eine Berechnung stattfinden soll. Des Weiteren wird jeweils die aktuelle Teamliste mit den Zuständen der einzelnen Teams in Form einer Kopie übergeben.

Wird nun im aktuellen Zustand auch nach mehrmaliger Berechnung mit unterschiedlicher Reihenfolge der Orte³ keine konsistente Zuordnung von Folgeorten gefunden, so wird in den vorherigen Zustand zurückgekehrt. Nach diesem Backtracking erfolgt schließlich wiederum eine Neuberechnung von Folgeorten. Für die Anzahl dieser Neuberechnungen pro Zustand kann ein fixer Wert festgelegt werden. Das Backtracking erfolgt rückwirkend bis Schritt 1. Für die Anzahl der Neuberechnung von Hauptspeiserouten kann ebenfalls ein fester Wert angegeben werden. Außerdem besteht die Möglichkeit, den Algorithmus nach einer festgelegten Anzahl berechneter Lösungen zu terminieren.

3.2 Vorstellung von Ergebnissen

Im Folgenden wird die in Abschnitt 2.1 gezeigte Routenplanung genauer beschrieben. Im Beispiel berechnete der Planungsalgorithmus 70 Lösungen innerhalb von 25 Sekunden. Dies ist ausschließlich die Zeit der Routenberechnung. Um die einzelnen Resultate miteinander vergleichen zu können, wurde jeweils die Summe des zurückgelegten Weges aller Teams bestimmt. Die gefundenen Lösungen besaßen Längen von 59888 m bis 62762 m. Diejenige mit der kleinsten Distanzsumme wurde ausgewählt und in Abb. 1 dargestellt. Ein Kritikpunkt dieser Auswahlmethode ist, dass die Hauptspeiseteams nicht in jedem Fall die kürzesten Routen besitzen. Dies kann jedoch durch ein Auswahlverfahren, welches Routenplanungen mit besonders kurzen Routen für Hauptspeiseteams auswählt, gelöst werden. Andere Auswahlverfahren sind denkbar. Die Anzahl der Neuberechnungen von Folgeorten pro Zustand (NpZ) wurde auf 10 festgelegt. Die Neuberechnung von Hauptspeiserouten (NvH) erfolgte 100 Mal.

Das Verhalten des Algorithmus mit unterschiedlichen Anzahlen von Teams soll hier kurz angedeutet werden. Der Algorithmus liefert für 39 Teams sehr schnell Ergebnisse. Bei einer Konfiguration von NpZ gleich 1000 und NvH gleich 100 liegt die Bearbeitungszeit bei unter 1 s. Dieses Verhalten ändert sich bis zu einer Anzahl von 21 Teams nicht signifikant. Werden 15 Teams bearbeitet, so liegt die Bearbeitungszeit bereits bei ca. 6 min. Für eine Konfiguration der Hauptspeiseteams wurde nach fünf gefundenen Lösungen zur nächsten Konfiguration gewechselt. Somit entstanden insgesamt 500 Lösungen. Das beschriebene Verhalten kann folgendermaßen erklärt werden. Je geringer die Anzahl der Teams ist, umso weniger Möglichkeiten existieren die Teams in einer Weise anzuordnen, sodass die Nebenbedingungen (1) bis (8) erfüllt sind. Gibt es weniger Möglichkeiten⁴, so muss jedes Mal, wenn keine konsistente Zuordnung von Orten zu Gängen gefunden wird, ein Backtracking durchgeführt werden. Die Anzahl dieser Schritte steigt damit bei kleineren Teamzahlen bedeutend an. Dieses Verhalten kann abgefangen werden, indem NpZ für

³Siehe Hervorhebung in Schritt 1

⁴Hier ohne Beweis. Weiterführende Verweise werden in Abschnitt 5.2 gegeben.

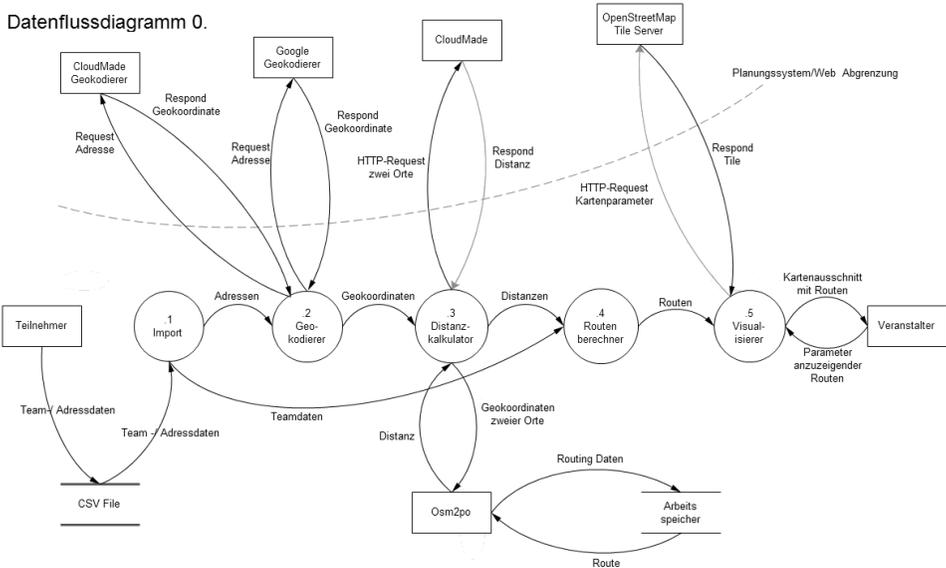


Abbildung 2: Komponenten und Datenaustausch des Planungssystems

kleinere Teamanzahlen verkleinert wird oder bereits nach einer geringeren Anzahl von Lösungen zur nächsten Konfiguration der Hauptspeiseteams gewechselt wird. Des Weiteren sinkt die Anzahl der Möglichkeiten Hauptspeiseteams unterschiedlich anzuordnen. Aus diesem Grund sollte der Parameter NvH für kleinere Teamzahlen ebenfalls klein gehalten werden.

4 Konzepte eines Prototyps

4.1 Beschreibung der Teilkomponenten

Die Bestandteile des Planungssystems und der Datenaustausch zwischen ihnen werden in Abb. 2 gezeigt. Die Beschreibung der verwendeten Dienste, bezeichnet mit Google-Geokodierer, CloudMade, OpenStreetMap Tile Server und Osm2po erfolgt im Abschnitt „Verwendete Dienste“.

Der Teilnehmer trägt zusammen mit seinem Teammitglied Team- und Adressdaten in ein Onlineformular ein. Diese Daten werden intern in einer CSV-Datei gespeichert. Die Import-Komponente liest Team- und Adressdaten aus selbiger Datei ein und stellt sie für die weitere Verarbeitung zur Verfügung.

Das Planungssystem benötigt eine Möglichkeit, die für jedes Team angegebene Adresse in die entsprechende Geokoordinate, d.h. Breiten- und Längengrad umzuwandeln. Auf der Grundlage dieser Koordinaten wird später die Entfernungsbestimmung der Orte zueinan-

der durchgeführt. Diese Aufgabe übernimmt der Geokodierer.

Die Grundlage der Späteren Routenplanung ist die Entfernungsbestimmung zwischen den Veranstaltungsorten. Es werden die Entfernungen von jedem Veranstaltungsort zu jedem anderen Ort benötigt. Nur auf diese Weise können während der Routenplanung Entscheidungen getroffen werden, welche Routen besonders kurz sind und welche Routen von der Lösung ausgeschlossen werden.

Der Routenberechner ist die Kernkomponente des Planungssystems. Anhand der Distanzen des Distanzkalkulators und den Teamdaten des Imports werden hier mögliche Verteilungen von Teams auf Vor-, Haupt- und Nachspeise sowie Routen zwischen den Teams unter Einhaltung der Nebenbedingungen berechnet. Grundlage ist der im vorherigen Abschnitt geschilderte Planungsalgorithmus.

Diese errechneten Daten sind ohne eine passende und übersichtliche Visualisierungsmöglichkeit für den Nutzer ohne Bedeutung. Die Visualisierung verschiedener Routen stellt, angesichts der Anzahl von Teams und Routen, durchaus eine Herausforderung dar. Die Umsetzung des Planungssystems erfolgt mittels Java. Diese bietet Plattformunabhängigkeit. Weiterhin bieten viele Dienstleister von geografischen Daten, deren Integration ein Kernbestandteil des Planungssystems ist, ihre Dienste durch eine Java API an. Dies wird im Abschnitt „Verwendete Dienste“ deutlich.

4.2 Verwendete Dienste

4.2.1 Geokodierung

Für die Geokodierung werden zwei Dienste zur Verfügung gestellt. Zum einen findet der Geokodierer von CloudMade Verwendung. Die CloudMade-Dienste basieren auf OpenStreetMap-Daten und ergänzen diese mit alternativen Quellen. Der Geokodierungsdienst wird als HTTP API realisiert und ist frei zugänglich. Es werden keine Einschränkungen bezüglich einer Begrenzung der Anfragen gemacht.

Der zweite Dienst ist die Geokodierung über Google. Hier wird die Google Geocoding API V2 bereitgestellt. Der Dienst bietet eine Erkennung von Adressen bis auf Hausnummerebene. Allerdings unterliegen die Anzahl der Anfragen sowie die entnommenen Daten einigen Einschränkungen.

4.2.2 Entfernungsbestimmung

Die Berechnung der reinen Luftlinie, welche ebenfalls als Option zur Verfügung steht, berücksichtigt keine Verkehrswege, sodass Ergebnisse nicht mit einer Fahrzeit in Verhältnis gesetzt werden können. Der hier eingesetzte CloudMade Routing Webservice bietet dagegen Echtzeit-Routing und arbeitet auf Grundlage von OpenStreetMap-Daten. Es existiert die Möglichkeit, sowohl Routen per Auto als auch per Fahrrad zu berechnen.

Ein zweiter alternativer Dienst wird mit Osm2po geboten. Osm2po bietet umfangreiche

Möglichkeiten, Daten des OpenStreetMap-Projektes (OSM-Daten) aufzubereiten und auf diesen Routenberechnungen durchzuführen. Das auf Java basierende Tool fungiert als Konverter von OSM-Daten in Graph-Daten. Alle für das Routing relevanten Daten werden lokal gehalten und bei Berechnungen in den Arbeitsspeicher kopiert.

4.2.3 Visualisierung

Die Visualisierung erfolgt mittels Einbettung von OpenStreetMap-Kartenmaterial in das Planungssystem mit Hilfe des Open Source JXMapView⁵. Dabei handelt es sich um eine Swing-Komponente, die auf Grundlage der SwingX-Erweiterung des Swing GUI Toolkit entwickelt wurde. Dadurch wird es ermöglicht, Karten in die eigenen Java-Applikationen zu integrieren.

5 Kritik und Ausblick

5.1 Routenberechnung

Der Algorithmus zur Routenberechnung wurde in Abschnitt 3.1 beschrieben. Es ergeben sich einige Diskussionsansätze. Es ist zu bemerken, dass der Algorithmus einige Möglichkeiten der Erzeugung von Routen nicht betrachtet. Dies sind die folgenden Fälle:

- Zwei Hauptspeiseteams starten von demselben Vorspeiseort
- Zwei Hauptspeiseteam enden in demselben Nachspeiseort
- Zwei Nachspeiseteams starten von demselben Vorspeiseort
- Zwei Vorspeiseteams enden in demselben Nachspeiseort
- Zwei Nachspeiseteams gehen zum gleichen Hauptspeiseort
- Zwei Vorspeiseteams gehen zum gleichen Hauptspeiseort

Wird beispielsweise für ein Vorspeiseteam v1 ein Hauptspeiseort h ausgewählt, so steht h für das danach betrachtete Vorspeiseteam v2 nicht mehr zur Verfügung. Dieses Problem kann behoben werden, indem h nicht sofort aus der Liste möglicher Folgeorte gelöscht wird und im Beispiel für weitere Vorspeiseorte zugänglich bleibt. Hier muss allerdings darauf geachtet werden, dass es maximal zwei Vorspeiseteams geben kann, die bei h zu Gast sind, da maximal drei Teams an einem Ort einen Gang einnehmen. Zudem ist nicht klar, ob auf diese Weise insgesamt kürzere Routen entstehen.

Des Weiteren bringt der Ansatz zur Bestimmung der Hauptspeiseorte einige Nachteile mit sich. Die Verteilung der Veranstaltungsorte hat einen großen Einfluss auf die entstehenden Längen der Routen. Sind die Veranstaltungsorte beispielsweise in einzelnen Clustern

⁵Tutorial unter: <http://today.java.net/article/2007/10/24/building-maps-your-swing-application-jxmapviewer>

angeordnet, so wäre es sinnvoll, die Hauptspeiseteams verteilt auf die einzelnen Cluster zuzuweisen. Daraufhin würden die Routen dieser Teams hauptsächlich innerhalb eines Clusters verlaufen. In einem solchen Fall wären diese Routen unter Umständen die kürzesten. Der hier beschriebene Algorithmus nimmt darauf allerdings keine Rücksicht und ordnet Teams im Zentrum aller Veranstaltungsorte an.

5.2 Ausblick

Interessante weitere Aspekte wären an dieser Stelle, warum der Algorithmus richtige bzw. kürzeste Lösungen ermittelt, wie oft es passiert, dass der Algorithmus für eine bestimmte Aufteilung der Teams keine passende Routenplanung findet und welchen Einfluss die einzelnen Parameter auf die Laufzeit des Algorithmus haben (in Abschnitt 3.2 angedeutet). Des Weiteren ist eine kritische Betrachtung des Algorithmus durchzuführen. Außerdem wurde vor Beginn der Themenbearbeitung eine Recherche zu bisherigen vergleichbaren Arbeiten durchgeführt. Für Ausführungen dieser Aspekte wird auf die parallel zu diesem Paper in Bearbeitung befindliche Bachelorarbeit „Integration von Geodaten in ein Planungssystem“ verwiesen.

Danksagung.

Mein Dank gilt den Betreuern meiner Bachelorarbeit Michael Thieme und Lars Peter Meyer, die sich stets Zeit für Hinweise und Fragen nahmen. Spezieller Dank geht an Ivan Krishanik, der mich mit Anregungen und interessanten Diskussionen meiner Ideen unterstützte.

Literatur

- [Bek06] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.
- [HM95] A. Hamacher and C. Moll. The Euclidian Traveling Salesman Selection Problem. In *Proceedings of the SOR '95, Report No. 95-199*, 1995.
- [MS09] Taieb Mellouli and Leena Suhl. *Optimierungssysteme : Modelle, Verfahren, Software, Anwendungen*. Springer-Lehrbuch. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2009.
- [Rei94] Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, Berlin, Heidelberg, 1994.
- [Sch97] Uwe Schöning. *Algorithmen- Kurz Gefasst*. Spektrum, Akad. Verlag, 1997.
- [Sch00] Walter Schmitting. *Das Travaling-Salesman-Problem: Anwendungen und heuristische Nutzung von Voronoi-/Delaunay-Strukturen zur Lösung euklidischer, zweidimensionaler Traveling-Salesman-Probleme*. Westfälische Wilhelms-Universität Münster, 2000.

A Lexeme-Clustering Algorithm for Unsupervised Learning of Morphology

Maciej Janicki
macjan@o2.pl

University of Leipzig, Master's programme in Computer Science

Abstract: This paper presents an algorithm for grouping morphologically related words, i.e. different inflected forms of the same base word, for the purpose of unsupervised learning of a natural language morphology. Initially, a trie of words is built and each node in the trie is considered a candidate for stem. The suffixes, with which it occurs, are clustered according to mutual information in order to identify inflectional paradigms. The algorithm has been implemented and evaluated for Polish, German, Finnish and Latin testing sets. Applications and topics for further research have been pointed out.

1 Introduction

Morphology is the field of linguistics studying the shape of words, or how words are built from the smallest meaningful components (*morphemes*). With morphology of a given language, we mean the rules of forming words in that language and the relations between words in terms of morphemes used. There exist numerous textbooks for morphology. In this paper, [Has02] was used as a linguistic background.

With *unsupervised learning of morphology*, we mean computer algorithms, that try to derive patterns of word formation and identify morphologically related words using as little data as a plain wordlist or an unannotated corpus, without any knowledge of the language itself. Though its results are still insufficient for practical applications [HB11], developing this field has important implications, both practical – the possibility of building morphological parsers working for many languages with low cost, and theoretical – giving us knowledge about how much of a natural language grammar can be inferred out of plain textual data [Gol06].

This paper is an attempt to solve one of the subproblems of unsupervised learning of morphology: grouping words into lexemes. The implementation of the presented algorithm, as well as the data used for evaluation, are publicly available¹.

¹<https://bitbucket.org/macjan/automorph/>

1.1 Basic definitions

A group of words, which have essentially the same meaning, but occur in different grammatical contexts, is called a *lexeme*, e.g. {*sing, sings, singing, sang, sung*}. Most often, lexemes (rather than all possible words) are the units described in dictionaries.

There are two important morphological relations between words: *inflection* and *derivation*. *Inflection* refers to the rules of forming words, which belong to the same lexeme, like adding the English 3rd person singular suffix *-s* or the progressive suffix *-ing* to a verb. *Derivation* means forming new lexemes, of which both the meaning and the phonological form is derived from the old lexeme, like adding in English the suffix *-er* to the verb *sing* to form a new noun lexeme *singer*. Since the borders between inflection and derivation are not always sharp, in this paper we'll assume, that inflection is an operation, that doesn't change the word's *part of speech* (e.g. verb to noun), while derivation does.

Inflection and derivation can be performed with *morphological operations*, which include *affixation* and *alternation*. Both operate on the *stem* – the morpheme, that is common to all the words belonging to one lexeme and that carries the meaning of the lexeme. *Affixation* means adding a new morpheme (*affix*) to the stem. Depending on its position, the affix can be called *prefix* (before the stem, e.g. English *re-make*), *suffix* (after the stem, e.g. English *sing-s*), *circumfix* (before and after the stem, e.g. German *ge-mach-t*), or *infix* (inside the stem). *Alternation* means the change inside the stem, like English *sing* : *sang*, or German umlauting: *Vater* : *Väter*.

1.2 Goals of this paper

One of the approaches to unsupervised learning of morphology is the so-called *Group and Abstract* approach [HB11]. It consists of two tasks: in the first, the words are grouped in clusters according to some measure, which should yield groups of morphologically related words. The second task is to generalize patterns, that are followed by many clusters, into morphological rules.

This paper presents an algorithm, which performs the first task of the *Group and Abstract* approach. It gets a plain list of words as input and returns a list of lexemes. The algorithm is targeted to work on European languages with suffix-based morphologies with various degrees of complexity. The possibility to apply a similar approach to different types of morphology is left open, but requires some additional research, which is described at the end of the paper.

1.3 Related work

Much work done on the field of unsupervised learning of morphology is described in the survey article [HB11], ranging from the first works in 1960s up to present. Recently, a

lot of research has been done in this field, among which the most attention is given to the methods based on finding the segmentation into morphemes by means of frequency or information-theoretical measures, like [Gol06]. Various methods for lexeme clustering are discussed in [YW00], along with the possibility to combine them. A semantics-based approach is presented in [SJ00].

2 The algorithm

The algorithm for lexeme clustering consists of four steps. In the first, we analyze the frequency of n -grams occurring word-finally, taking those with high frequency as possible candidates for suffixes. The second step is transforming the wordlist into a trie structure, in which we type candidates for stems and their suffix sets. In the third step, suffix sets are clustered into morphological paradigms using the mutual information measure. Finally, for each word, morphologically-related words are extracted from the trie.

2.1 Suffix filtering

Before we start the actual algorithm, we can easily filter out n -grams, that occur at word-final positions, but are certainly not morphological suffixes. We will make use of the following assumption:

Assumption 2.1 *Morphological suffixes need to occur with a variety of stems, i.e. their frequency must be of significant size.*

Experiments show, that all the real morphological suffixes in all tested languages have a frequency greater than 0.001 among n -grams in word-final positions (we measure the frequency separately for each n , i.e. separately for unigrams, bigrams and so on). By filtering out n -grams below that frequency, we get rid of about 99.8% word-final n -grams (for example, out of 530498 endings of length ≤ 9 found in Polish data set, only 773 have the frequency above 0.001). We will consider the remaining n -grams as candidates for morphological suffixes.

2.2 Trie with suffix sets

After filtering out improbable suffixes, we build up a *trie* of words. Tries are well-known data structures, that have already been successfully used in unsupervised learning of morphology (cf. for example [SJ00], [Gol06]). *Trie* is a tree data structure, in which each node represents the set of words, which share a certain common prefix. The children of the node correspond to prefixes, which are longer by one letter, than the prefix of their parent, so that moving from parent to child in the trie means moving one letter forward in a word.

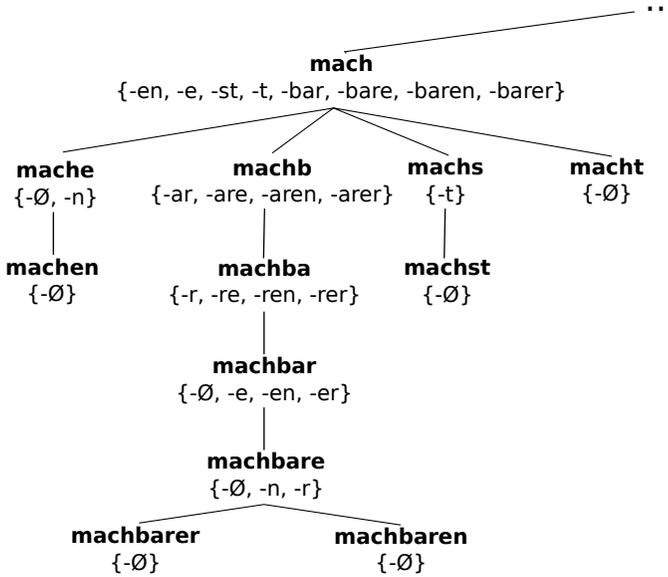


Figure 1: Trie with suffix sets example.

In every node of the trie, we will also store the *suffix set* of its prefix, i.e. the set of suffixes which can be concatenated to the prefix to form a word, that appears in the wordlist. For example, if the word *machen* appears in the wordlist, the ending *-en* should be included in the suffix set of *mach-* and so should *-n* for *mache-* and \emptyset (empty ending) for the node containing the prefix *machen-*. However, in suffix sets, we include **only** endings, that were not filtered out in the previous step. An example fragment of a trie with suffix sets, containing the German words $\{machen, mache, machst, macht, machbar, machbare, machbaren, machbarer\}$, is shown in Figure 1.

Because we don't know where the actual border between stem and suffix is, we can consider every prefix occurring in some node of a trie a candidate for stem and its suffix set members candidates for morphological suffixes.

2.3 Partitioning suffix sets

In each node of a trie, we have a candidate stem and a corresponding suffix set. It's easy to see, that some of the suffix sets will yield words belonging to different lexemes, like the set $\{-en, -e, -st, -t, -bar, -bare, -baren, -barer\}$ ² for the stem *mach-* in German (see Figure 1). Therefore, there arises a need to partition suffix sets into clusters, so that suffixes from every cluster will yield words belonging to the same lexeme, like $\{-en, -e, -st, -t\}$, $\{-bar, -bare, -baren, -barer\}$. Some of such clusters, like $\{-en, -e, -st, -t\}$,

²For simplicity, only a few possible endings were shown.

immediately are morphological paradigms. Others, like $\{-bar, -bare, -baren, -barer\}$, lead to a morphological paradigm $\{-\emptyset, -e, -en, -er\}$ lower in the trie. With such partitioning of the suffix set, we can group words sharing a node in the trie into different lexemes.

We will use the *mutual information* measure to measure dependencies between subsets of a suffix set. Mutual information of two random variables is the amount of information, that they share. For random variables X, Y taking values from the sets \mathcal{X}, \mathcal{Y} , respectively, it is defined as [CT91]:

$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \quad (1)$$

The following assumption will allow us to apply mutual information for suffix set partitioning:

Assumption 2.2 *Parts of the same morphological paradigm share a lot of information, i.e. are highly dependent on each other, while two different morphological paradigms are nearly independent, i.e. the mutual information among them is low.*

Using this assumption, we will define the partitioning of the suffix set, that we want to achieve. To each set of suffixes S , we can assign a random variable X_S , telling us what the probability of finding some subset of S with a random stem is. More precisely, let the elements of S be indexed as follows: $S = \{s_1, s_2, \dots, s_n\}$. Then, the values of the variable X_S are vectors $\mathbf{v} = v_1 v_2 \dots v_n \in \{0, 1\}^n$, where for each v_i , the value 1 represents presence of the corresponding suffix s_i , while the value 0 represents absence of this suffix. Thus, $p(\mathbf{v})$ is defined as the probability, that for a randomly chosen suffix set R holds: $\forall_{i \in \{1, \dots, n\}} (s_i \in R \Leftrightarrow v_i = 1)$.

The entropy of the variable X_S corresponds to the amount of information contained in knowing, which suffixes of S some stem takes or does not take. The mutual information $I(X_S; X_T)$ between variables corresponding to different suffix sets means the amount of information, by which knowing which suffixes of S some stem takes helps to determine which suffixes of T it should take and reverse. In the further part we will use the notion $I(S; T)$ for $I(X_S; X_T)$. Note, that for sets of suffixes S, T, U and their random variables X_S, X_T, X_U , holds: $I(S \cup T; U) = I(X_{S \cup T}; X_U) = I(X_S, X_T; X_U)$.

Let's now define the target partitioning:

Definition 2.1 *Let S be a suffix set (belonging to some stem), $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$ a partitioning of this set and κ a fixed constant. We call \mathcal{S} a partitioning into paradigms, iff:*

1. for each $i \neq j$: $I(S_i, S_j) < \kappa$
2. for each disjoint $A, B : A \cup B = S_i \Rightarrow I(A, B) \geq \kappa$

Those two conditions represent the statements of assumption 2.2: the mutual information is low between two different paradigms and high between two parts of the same paradigm.

Data: a suffix set $S = \{s_1, s_2, \dots, s_n\}$

Result: $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$, being the partitioning of S into paradigms

$\mathcal{S} \leftarrow \{\{s_1\}, \{s_2\}, \dots, \{s_n\}\};$

$k \leftarrow n;$

while true do

$i_{max}, j_{max} \leftarrow 1, 1;$

for $i \leftarrow 1$ **to** k **do**

for $j \leftarrow 1$ **to** k **do**

if $I(S_i, S_j) > I(S_{i_{max}}, S_{j_{max}})$ **then**

$i_{max} \leftarrow i;$

$j_{max} \leftarrow j;$

end

end

end

if $I(S_{i_{max}}, S_{j_{max}}) \geq \kappa$ **then**

$S_{i_{max}} \leftarrow S_{i_{max}} \cup S_{j_{max}};$

$\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_{j_{max}}\};$

$k \leftarrow k - 1;$

else

terminate and return $\mathcal{S};$

end

end

Algorithm 2.1: Computing the partitioning into paradigms

The κ parameter represents the maximum mutual information of two independent inflectional paradigms (we will call it *independence threshold*) and has to be given at input. Formally, if two random variables are independent, then their mutual information should be 0, but also different paradigms are never completely independent. Some paradigms tend to occur together with others (e.g. verb with a corresponding deverbal noun), other exclude each other (e.g. two different verb conjunction paradigms).

The algorithm 2.1 is used to compute the partitioning into paradigms. Initially, each suffix is made a distinct part of the partitioning. Then, in a loop, the parts, which share the highest amount of information are merged. The algorithm terminates when the highest dependency among some two parts is lower then the independency threshold, which means, that all parts are already independent. The following theorem will be used to prove the correctness of the algorithm:

Theorem 2.1 *The algorithm 2.1 computes the partitioning into paradigms, as defined in the definition 2.1.*

Proof. The low dependency between member sets of \mathcal{S} is obvious: the algorithm terminates exactly when there is no more pair of sets with mutual information $\geq \kappa$.

We will now show the high dependency inside parts. Let $S_i \in \mathcal{S}$ and A, B be disjoint subsets of S_i , for which holds: $A \cup B = S_i$. At some iteration of the algorithm, there must

have been such $C \subseteq A, D \subseteq B$, that C and D have been merged, which means $I(C;D) \geq \kappa$. Let $A' = A \setminus C$ and $B' = B \setminus D$.

Using the chain rule for mutual information [CT91] and the non-negativity of mutual information, we obtain:

$$\begin{aligned} I(A;B) &= I(C \cup A'; D \cup B') = I(C; D \cup B') + I(A'; D \cup B' | C) \\ &= I(C;D) + I(C;B'|D) + I(A';D \cup B'|C) \geq I(C;D) \geq \kappa \end{aligned}$$

□

Optimization Because the algorithm 2.1 requires computing mutual information many times, which takes a very long time (iterating over all stems is necessary each time mutual information is computed), some optimizations had to be performed. They include:

- storing mutual information between each pair of parts in a matrix and calculating new values only there, where some sets were merged;
- merging more than one pair of sets in a single iteration: if S_i and S_j are merged, one can merge S_k and S_l in the same iteration, provided that $I(S_k, S_l)$ is greater than each of the values: $I(S_k, S_i \cup S_j)$, $I(S_l, S_i \cup S_j)$, because then merging S_i and S_j will not affect the behaviour of S_k and S_l ;
- storing entropies of all sets seen before in a hash table and computing mutual information from the formula [CT91]:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (2)$$

Note also, that for a set of suffixes S and its random variable X_S , computing $H(X_S)$ out of the definition would require iterating over all possible values of X_S , which is $2^{|S|}$. Instead, we iterate over all stems in the trie and count all subsets of S , that we find in suffix sets. Non-occurrent values don't change the entropy, so they don't have to be included in computation.

2.4 Extracting lexemes

The last step of the algorithm is extracting lexemes from a trie with partitioned suffix sets. Given a word split into stem and suffix, we can look at the node of the trie corresponding to the stem and the part of its suffix set corresponding to the suffix, to find other possible morphological suffixes of the same paradigm, which can occur with the same stem, yielding other words belonging to the same lexeme. There is one problem with this approach: we don't know where the border between stem and suffix is. However, we can iterate over *all* possible stem-suffix pairs for a given word. The intuition lying behind this approach can be expressed in the following assumption, which turned out to work in practice:

Assumption 2.3 *If two words w_1, w_2 belong to different lexemes, at each level of the trie where they share the stem, their endings will belong to different parts of the suffix set.*

Formally, for a word w , let $\sigma(w) := \{(r, s) : |r| > 0 \wedge w = rs\}$ denote the set of *splittings* of the word w , i.e. possible stem-suffix pairs. Let $\mathcal{S}_r = \{S_{r,1}, S_{r,2}, \dots, S_{r,n}\}$ denote the partitioned suffix set corresponding to stem r . Then the set $\text{lex}(w)$, containing all words belonging to the same lexeme as w , is computed as follows:

$$\text{lex}(w) = \bigcup_{(r,s) \in \sigma(w)} \{rs' : \exists_i (\{s, s'\} \subseteq S_{r,i})\} \quad (3)$$

The assumption 2.3 ensures, that we will not overgenerate the lexeme. Note that the nodes in trie, where we pick new words to the lexeme, are likely to be morpheme boundaries. Perhaps one could use the same trie to determine the segmentation into morphemes, which would be an important step towards abstracting morphological paradigms from lexemes.

3 Evaluation

For the purpose of evaluation, testing datasets for four languages were established. Each dataset contains a list of lexemes, which is expected at the output of the algorithm (reference data). The corresponding input data can easily be obtained by splitting the list of lexemes into a list of words.

The datasets for Finnish, German and Latin were extracted from Wiktionary. The pages with inflected word forms usually contain a template of a form:

{{form-of|base-form|inflectional-data}}

(or similar), from which base forms were extracted and words with the same base form were grouped into lexemes. English Wiktionary³ has been used for Latin and Finnish, while German Wiktionary⁴ has been used for German.

As Polish dataset, the *List of inflected words* of a freeware *Polish language dictionary*⁵ was used. The list was slightly modified: words containing non-letter characters (such as apostrophes or hyphens) were removed and adjective forms with prefix *nie-* (eng. *un-*) were separated as different lexemes, since their relation to the original adjective is derivational, not inflectional [Has02].

Note that not all existing forms of a given lexeme occur in the datasets, because not all of them appear in Wiktionaries. Let's call the number of forms of a given lexeme appearing in the data divided by the number of all possible forms of this lexeme, *lexeme coverage*. The average lexeme coverage in the datasets is likely to be higher than that of a corpus⁶

³<http://en.wiktionary.org/>

⁴<http://de.wiktionary.org/>

⁵<http://www.sjp.pl/slownik/odmiany/>

⁶See [HB11], section 3.2.3 for more detailed remarks about lexeme coverage in corpora.

Testing set	Words	Lexemes	κ	Precision	Recall	F-measure
Finnish	9k	1k	0.002	98.6 %	81.3 %	89.1 %
German	54k	20k	0.01	97.7 %	79.1 %	87.4 %
Polish	980k	61k	0.002	96.1 %	79.0 %	86.7 %
Latin (decl.)	210k	22k	0.05	95.4 %	92.4 %	93.9 %
Latin (full)	586k	26k	0.015	85.7 %	41.3 %	55.8 %
Latin (full)	586k	26k	0.02	67.1 %	81.7 %	73.7 %

Table 1: Evaluation results ($k = 1000$).

(because we extract forms from a dictionary), but still far from 1.0. The testing data sets are not very different from real corpora in this feature. However, for the purpose of unsupervised learning of morphology, the lexeme coverage in input data should be as high as possible.

The evaluation was performed as follows: for each word, its lexeme was computed and matched against the lexeme in the reference data. Matches (TP), overgenerated words (FP) and undergenerated words (FN) were counted. These results were summed up for all the words in the wordlist and used to calculate Precision, Recall and F-measure as follows:

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

$$F\text{-measure} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (6)$$

The optimal κ values were determined experimentally for each dataset and correspond to the maximal mutual information value between two different paradigms observed in this dataset. It is not clear, whether optimal κ values could be automatically derived from datasets.

For Latin data set, three tests were performed: one only for declension (nouns and adjectives, without verbs) and two for the full data, with two different κ values. For every other data set, a single test was performed.

Table 1 shows the results of the evaluation. Additional information about the size of testing sets and the value of the κ parameter were included. The full Latin set turned out to be most challenging, with results strongly depending on the κ value. The reason is, that Latin verbal stems take a great number of endings (often over 100) and it's hard to decide, which of them are derivational and which inflectional (e.g. participles).

4 Conclusion

The results show, that the mutual information measure defined on sets of affixes is a good means to discover morphologically related words. This method can be further explored in order to abstract morphological paradigms and thus provide a complete algorithm for unsupervised learning of morphology.

4.1 Problems and further work

The major problem of the algorithm is that it takes a very simple view on morphology: it treats words as concatenations of a stem and a single suffix⁷. Even in the four tested languages there are cases, where this approach is insufficient:

- alternations inside stem, e.g. German umlauting: *Haus-ϕ* : *Häus-er*
- prefixing, e.g. German *mach-en* : *ge-mach-t*
- multiple affixes, e.g. Polish *mówi-ć* (*to speak*) : *mówi-li-by-śmy* (*we would speak*)
- affixes shared across multiple paradigms, e.g. Latin *Aureli-is* : *Aureli-us* vs *Aureli-is* : *Aureli-a* (in our approach the suffix set is partitioned into **disjoint** paradigms)

Although the evaluation results, especially for Finnish with its multi-slot morphology, have shown, that treating multiple suffixes as a single one (a concatenation of those) works well enough, this behaviour still needs to be justified.

Dealing with prefixing and alternations is to be subject of further work. Let us observe, that this would significantly improve the results (more specifically, recall), which are already high. That lets us expect, that some additional research could make this algorithm achieve much better results, possibly above 90% of F-measure.

Another problem is the presence of the κ parameter. The ideal approach requires, that no other data than a list of words need to be supplied. The behaviour of κ , e.g. its dependency on language, and the possibilities to compute a near-to-optimal value out of the wordlist, should be examined. However, it's only one parameter and its meaning is well defined.

The approach of determining morphological classes only out of surface forms has also its limitations. For instance, there are cases in Latin, where one stem can be shared by a masculine and a feminine noun, like name pairs *Aurelius* : *Aurelia*. Such pairs, which should be classified as distinct lexemes, are hard or impossible to distinguish from adjectives, which can take both masculine and feminine endings, like *altus* : *alta*.

⁷As it is also done in some other approaches, for example [Gol06].

4.2 Applications

As mentioned before, the algorithm was developed for the purpose of unsupervised learning of morphology. The next task would be to find or develop a suitable method to abstract morphological paradigms from lexemes. The final result would then be a general method to construct morphological analyzers and morphology tables for many languages with a little effort.

Clustering lexemes can also have applications on its own. For example, it can be used in simple corpus search engines to augment search patterns with inflected forms of words. A clustered wordlist can also be used for language-statistical purposes: statistical measures over number of inflected forms of a lexeme, suffix length etc. could be interesting for linguists.

References

- [CT91] Thomas M. Cover and Joy Thomas. *Elements of Information Theory*. Wiley, 1991.
- [Gol06] John Goldsmith. An algorithm for the unsupervised learning of morphology. *Nat. Lang. Eng.*, 12(4):353–371, December 2006.
- [Has02] M. Haspelmath. *Understanding Morphology*. Understanding Language Series. Arnold, 2002.
- [HB11] Harald Hammarström and Lars Borin. Unsupervised Learning of Morphology. *Computational Linguistics*, 37(2):309–350, 2011.
- [SJ00] Patrick Schone and Daniel Jurafsky. Knowledge-Free Induction of Morphology Using Latent Semantic Analysis. In Claire Cardie, Walter Daelemans, Claire Nedellec, and Erik Tjong Kim Sang, editors, *Proceedings of CoNLL-2000 and LLL-2000*, pages 67–72. Lisbon, Portugal, 2000.
- [YW00] David Yarowsky and Richard Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 207–216, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

Systematic review about the academic impact of DBpedia

Marcus Nitzschke

mam08crr@studserv.uni-leipzig.de

University of Leipzig, Master's programme in Computer Science

Abstract: There is an extensive collection of academic papers and presentations for the Wikipedia project available. This enables one to analyze the evolution of such projects over the years from the beginning. The goal of this paper is to create a similar collection for the DBpedia project. With this base there will be introduced several analyses and a comparison to the Wikipedia project.

1 Introduction

In 2009 there was a large survey to collect journal articles and conference paper concerning the Wikipedia project [Oko09]. The findings of this survey were published,¹ but are, as of now, only partially analyzed. The results showed a continuous growth of the number of journal articles. The number of conference papers grew until 5 years after the founding. Afterwards it began to decrease year by year. With these numbers as a base one can start to interpret the data and find reasons for these characteristics.

The goal of this paper is a similar analysis for the DBpedia project. DBpedia extracts structured content from Wikipedia and republishes this content in a semantically understandable way. This allows one to ask comprehensive and sophisticated questions on top of the Wikipedia data. DBpedia is one of the most important data sets in the *Linked Open Data* cloud.² The project was founded in 2007.

The approach in this systematic review is also imaginable to extend to other projects or research areas. With the ongoing effort in making data publicly available and retrievable by web services the whole approach could be implemented in a semiautomatic framework. This could establish new possibilities to get an overview how active or fulfilled a specific research area or project is.

The paper will first describe how the relevant information was retrieved and processed. Afterwards the results of the different analyses will be presented and finally the results will be discussed.

¹http://en.wikipedia.org/wiki/Wikipedia:Academic_studies_of_Wikipedia (16.08.2012)

²<http://richard.cyaniak.de/2007/10/lod/> (16.08.2012)

2 Data retrieval and processing

The process of data retrieval included four sources. Table 1 gives an overview of these sources and the resulting number of publications.

Table 1: Overview of data sources

source	results
Google Scholar	84
Arnetminer	54
Semantic Web Conference Corpus	49
manually	4

The criteria for inclusion in the analyses were geared to the Wikipedia study. The main criterion was that the term “dbpedia” occurs in the title or – if supported by the source – the abstract of the publication. Later the total amount was reduced to only match conference papers and journal articles which were all peer reviewed.

The following listing describes how the informations of the different data sources were retrieved.

- **Google Scholar:** Google Scholar³ allows to search a multiplicity of academic publications. The search was limited because the intent of this paper is to collect only the most relevant publications concerning DBpedia. The usage of the search term `allintitle:dbpedia` limited the total number of results to that containing “dbpedia” in their title. This search produced 84 results at the moment of writing (06/2012).
- **Arnetminer:** Arnetminer⁴ is a search engine which extends the collection of academic documents with informations about authors, conferences or journals. These informations linked together should enable a complete view of the data. Besides the front end search Arnetminer also provides a RESTful interface.⁵ So it is possible to search the publications by title. For better processing the retrieved data was converted from the JavaScript Object Notation (JSON) to BibTeX by a small script.⁶
- **Semantic Web Conference Corpus:** The Semantic Web Conference Corpus⁷ provides numerous informations about conferences and the presented talks belonging to the Semantic Web topic. This site also offers a SPARQL endpoint which was used to get the relevant publications for this paper. The query retrieved all publications

³<http://scholar.google.com/> (16.08.2012)

⁴<http://arnetminer.org/> (16.08.2012)

⁵http://arnetminer.org/RESTful_service (16.08.2012)

⁶https://raw.githubusercontent.com/kenda/dbpedia_impact/master/scripts/get_arnetminer.py (16.08.2012)

⁷<http://data.semanticweb.org> (16.08.2012)

with the term “dbpedia” in the title or in the abstract. A script⁸ got the resulting URIs and scraped the corresponding html view for the BibTeX representation of the entry.

- **manually:** The data sources listed so far could not cover all important journals like the *Semantic Web Journal* (SWJ). That’s why parts of the results were collected manually. The underlying search term for the SWJ executed by Google was `dbpedia site:semantic-web-journal.net/content`. Hence all published research papers by the SWJ were searched by the term “dbpedia”.

To ensure a structured organization and analysis of the collected data a reference management software, namely Zotero,⁹ was used. This solution has numerous advantages, for example a BibTeX importer and an easy possibility to import results from Google Scholar via a browser plugin.

With the aid of Zotero the data processing first solved duplicates between the publications. This step reduced the total number of data entries from 191 to 130. Six further entries dropped out because of the type of publication, e.g. theses or technical reports, which are not in the scope of this paper. The second step was to complete lacking data fields like the publication date or conference names.

The cleaned data was exported from the Zotero database to the csv format by a SQL script.¹⁰ This csv data is the base for the analyses which are done by Gnu R.¹¹

3 Results

The results contain the different analyses over the period between 2007 and 2012. Because this paper was written in June 2012 there is a lack of publications for this year. Especially conferences which are mostly held in the second half of the year are lacking, e.g. ISWC.

The first analysis shown in Figure 1 plots the total number of publications by year. This is the analogical visualization of the Wikipedia analysis. The barplot shows a minimum number of publications in 2007 of zero and four for the journals and conferences respectively. The maximum number of journals is already reached in 2012 with seven. The maximum number of conference papers was reached in 2011 with 32. But it seems like this number will be exceeded in 2012.

There is an obvious trend that the number of conference papers is constantly growing. The trend of the journal articles is not that obvious because there is an outlier in 2010. But this fact is owed to the small number of journal articles in general. The reason why the number of conference papers is so much greater than the number of journal articles was

⁸https://raw.githubusercontent.com/kenda/dbpedia_impact/master/scripts/get_semweb_org.sh (16.08.2012)

⁹<http://www.zotero.org/> (16.08.2012)

¹⁰https://raw.githubusercontent.com/kenda/dbpedia_impact/master/scripts/zotero_query.sql (16.08.2012)

¹¹<http://www.r-project.org/> (16.08.2012)

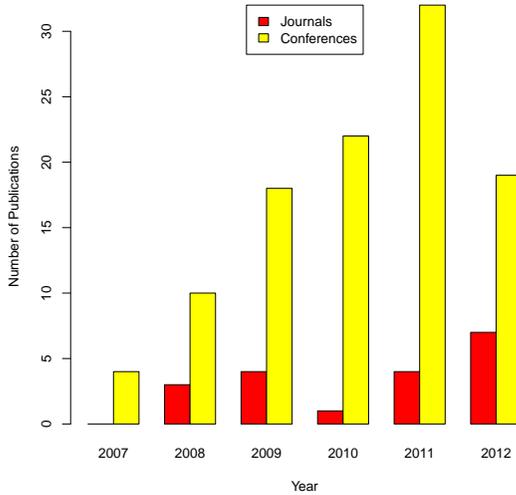


Figure 1: barplot visualizing total number of publications by year

already covered by the Wikipedia analysis. “Journals are not the norm in CS/HCI research. Knowledge is shared through conferences, not journals.” [Kri11] they explain.

The second analysis goes into the details of the journals and conferences. It plots the total number of publications by the specific journal or conference if there are at least two publications.

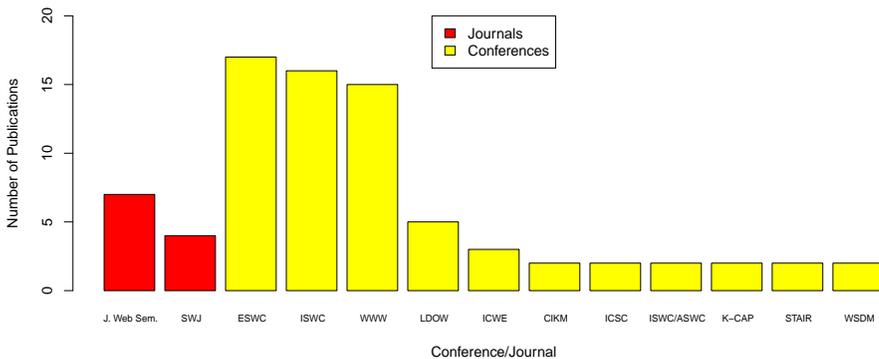


Figure 2: barplot visualizing total number of publications n by conference/journal with $n > 1$

Figure 2 illustrates that there are two journals which published more than one article related to DBpedia. The *Journal of Web Semantics* published seven articles, the *Semantic Web Journal* four articles. The plot also shows that there are three major conferences where DBpedia mattered since 2007. The most papers were published at *ESWC* with a number of 17. *ISWC* and *WWW* published 16 and 15 papers, respectively. It follows a gap of 10 to the *LDOV* conference with five published papers. This plot assumes the *ISWC/ASWC* as a separate conference. Otherwise *ISWC* would have two additional publications and therefore 18 overall.

Table 2 shows the importance of DBpedia at the major conferences in 2011. The table breaks the total number of publications at the conferences down to the number of publications where the term “dbpedia” occurred somewhere in the paper ($dbpedia_{all}$) and the number of publications where the term occurred in the title or abstract ($dbpedia_{head}$).

Table 2: Compared number of publications at ESWC, ISWC, WWW in 2011

	ESWC (%)	ISWC (%)	WWW (%)
total	57 (100)	260 (100)	220 (100)
$dbpedia_{all}$	N/A	75 (28.84)	9 (4.09)
$dbpedia_{head}$	4 (7.02)	9 (5.52)	1 (0.45)

Therefore DBpedia was mentioned in 28.84% and 4.09% of the publications at ISWC and WWW somewhere in the paper. The highest rate of the $dbpedia_{head}$ criterion was reached by ESWC with 7.02% and a number of four papers. There were five more papers at ISWC, however this led to a lower percentage (5.52). The lowest percentage is reached by WWW with 0.45%.

Finally the collection of DBpedia related publications contains two highly cited papers. "*DBpedia: A Nucleus for a Web of Open Data*" [ABK⁺07] was cited 802 times based on Google Scholar. The second paper "*DBpedia - A Crystallization Point for the Web of Data*" [BLK⁺09] was cited 403 times and won the “JWS Most Cited Article 2006-2010 Award” in 2011.

4 Discussion

Compared to the Wikipedia study this paper showed mainly similar characteristics in the total number of publications per year analysis. The major difference is that the number of conference papers related to Wikipedia started to decrease six years after the foundation. This is exactly the year (2008) when DBpedia started to increase the number of conference publications. It will be interesting to see whether the importance of DBpedia will also start to decrease in the next one or two years.

One focus of this paper was to retrieve and process the data as transparent and reproducible as possible. That’s why most of the publications were retrieved through several

web services. Although these services and the additional manual sources cover the bigger part and the most important publications it is hard to promise that all available publications that would observe the requirements of this study were retrieved. But these missing publications probably wouldn't lead to significantly different results.

The data sources also showed different measurements of quality. While Google Scholar contains the highest number of results it also contains the highest number of publications which had to be modified manually either because of lacking fields or the wrong type of publication. Besides Google Scholar this could also be caused by the Zotero plugin. The best quality of publications as well as possibility to retrieve the data was provided by the interface of the Semantic Web Conference Corpus. This showed the strengths of SPARQL and how it can improve searches.

Perspectively it would be desirable to publish more systematic reviews and meta analyses beyond clinical trials like introduced in [Mul94]. Although there is usually no direct advance in the state of the art by systematic reviews, the holistic view can provide new and interesting information about the research area as a whole.

References

- [ABK⁺07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, pages 722–735, 2007.
- [BLK⁺09] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia-A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.
- [Kri11] Travis Kriplean. Request to verify articles for Wikipedia literature review. <http://lists.wikimedia.org/pipermail/wiki-research-l/2011-March/001366.html>, 2011.
- [Mul94] C.D. Mulrow. Systematic reviews: rationale for systematic reviews. *Bmj*, 309(6954):597–599, 1994.
- [Oko09] Chitu Okoli. A Brief Review of Studies of Wikipedia in Peer-Reviewed Journals. In *Proceedings of the 2009 Third International Conference on Digital Society, ICDS '09*, pages 155–160, Washington, DC, USA, 2009. IEEE Computer Society.

Visualisierung einer Taxonomie als Treemap und Optimierung der Generierung in einer Web-Applikation

Max Kießling

max.kiessling@moez.fraunhofer.de

Fraunhofer MOEZ, Bachelor Studiengang Informatik

Zusammenfassung: Unternehmen sollen anhand ihrer Dienstleistungen mithilfe einer Treemap ansprechend visualisiert und so untereinander vergleichbar gemacht werden. Die Visualisierung soll innerhalb einer Web-Applikation genutzt werden und auf Grundlage der dazugehörigen Datenbank dynamisch erstellt werden. Dabei zeigt sich, dass die naive Implementierung sehr lange zur Berechnung benötigt, wodurch eine Optimierung unumgänglich wird. Die Optimierung in 3 Schritten ergab eine Verbesserung um etwa den Faktor 80, wobei auch die Codequalität deutlich verbessert wurde.

1 Motivation

Das Fraunhofer MOEZ Projekt *IP Industry Base* (IPIB) klassifiziert Unternehmen, anhand der von den Unternehmen angebotenen Dienstleistungen, mithilfe der *Intellectual Property Service Taxonomy* [MTBJ12]. Anhand dieser Daten soll die grafische Repräsentation der Unternehmen in einer *Ruby on Rails* (*Rails*) Anwendung dynamisch generiert werden. Das Unternehmen soll mittels einer Treemap [Shn06] visualisiert werden und so eine „Unternehmens-DNA“ entstehen.

Die Treemap beruht auf der Idee hierarchische Daten, also Daten in Baumform, grafisch darzustellen. Hierbei entspricht jeder Knoten des Baumes einem Rechteck, welches wiederum durch die Subelemente des Knotens in kleinere Rechtecke aufgeteilt ist. Die Größe des Rechtecks wird dabei proportional zu einer spezifischen Dimension der Daten abgeleitet. In diesem Fall leitet sich die Größe eines Rechtecks von der Anzahl der Unternehmen ab, welche die dargestellte Dienstleistung anbieten.

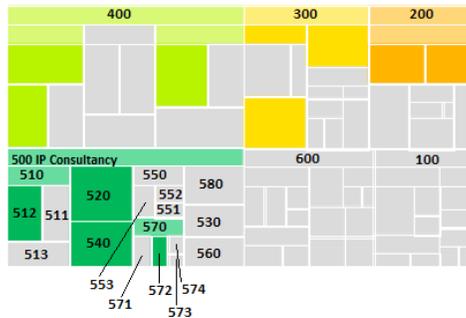


Abbildung 1: Unternehmens-DNA

In Abbildung 1 ist zu sehen, dass sich die Größe der Rechtecke stark unterscheidet. Dies bedeutet, dass Dienstleistungen, wie 520, von mehr Unternehmen angeboten werden, als Dienstleistungen, wie 573, deren Rechtecke kleiner sind. Eine weitere in der Treemap dargestellte Information ist, ob ein Unternehmen die gewählte Dienstleistung direkt anbietet oder ob es sich um eine Verallgemeinerung eines Dienstes handelt. So bedeuten in Abbildung 1 die dunklen Felder, wie 520, dass diese Dienstleistung vom Unternehmen direkt angeboten wird, die hellen Felder, wie 500, dass das Unternehmen eine speziellere Form der Dienstleistung anbietet.

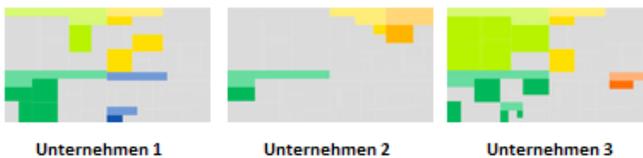


Abbildung 2: Vergleich der Unternehmens-DNA verschiedener Unternehmen

Wie in [TBJ⁺12] ausführlich dargestellt, können Unternehmen hierdurch vom Nutzer in ihrer Angebotsvielfalt einfach erfasst und untereinander verglichen werden. Im konkreten Beispiel, bieten in Abbildung 2 zwar alle Unternehmen Dienstleistungen der Kategorie 500 an (links unten), unterscheiden sich dennoch stark in ihrer Spezialisierung. So ist Unternehmen 2 stark spezialisiert, Unternehmen 1 und 3 hingegen besitzen ein vielfältigeres Angebot an Dienstleistungen.

2 Problem

Im ersten Schritt der Implementierung wird die Funktion naiv implementiert. Die Ladezeit für die Darstellung der Treemap mit der naiven Implementation beträgt 20,53 Sekunden, bei 1488 SQL-Anfragen. Nach Jakob Nielsen [Apa09] sollte die Ladezeit einer Website jedoch nicht mehr als eine Sekunde betragen, da der Nutzer sonst nicht das Gefühl hat frei

auf der Seite navigieren zu können. Eine Performance-Optimierung ist also unumgänglich. Dennoch ist dieses Vorgehen gerechtfertigt. Donald Knuth schrieb bereits 1974: „*premature optimization is the root of all evil.*“ [Knu]. Es ist also sinnvoll, zuerst sauberen Code zu entwickeln und diesen nur wenn nötig zu optimieren.

Zur Generierung der Treemap wird die JavaScript-Bibliothek *JavaScript Information Visualization Tool* (The JIT) [Bel] benutzt. Die Bibliothek berechnet die optimale Größe und Platzierung der Rechtecke und stellt diese grafisch mithilfe des HTML-Canvas-Elements dar. Dazu müssen der darzustellende Baum, Name und Beschreibung der Knoten, sowie deren Verhältnis zu allen anderen Knoten angegeben werden.

Eine Schwierigkeit bei der Erstellung einer Treemap ist eine annähernd optimale Verteilung der Rechtecke zu finden, um den Platz möglichst effizient nutzen zu können. Gleichzeitig soll die Reihenfolge der Ausgabedaten nicht zu stark verändert werden, um eine vorhersehbare Darstellung zu erhalten. Eine gleichbleibende Anordnung der Dienstleistungen ist im Fall der *IPIB* sehr wünschenswert, damit das Aussehen der Treemap nicht nach jeder Änderung in der Datenbasis grundlegend verändert wird. Diese beiden Anforderungen stehen einander jedoch unvereinbar gegenüber. Findet man also eine optimale Verteilung, so ist die Anordnung der Daten bezüglich der Anordnung der Eingabedaten nicht mehr vorhersehbar. Es gibt daher verschiedene Algorithmen, welche eine für das menschliche Empfinden gute Darstellungsmöglichkeit erzeugen.

In unserem Fall benutzen wir den in *The Jit* eingebauten „squared“ Algorithmus [Hod] [BHvW00], dessen Ziel es ist, die Flächen möglichst quadratisch darzustellen. Das Verhältnis von Höhe und Breite soll gegen 1 gehen. Dies erzeugt ein optisch ansprechendes Resultat, bei dem die einzelnen Knoten gut unterscheidbar sind.

Als Problem bleibt, dass das Verhältnis aller Daten, also die Größe aller Rechtecke, sowie deren Farbe in Abhängigkeit vom Unternehmen berechnet werden muss. Dabei soll die Größe eines Rechtecks nicht ausschließlich durch die Anzahl der Unternehmen bestimmt sein, da sonst manche Dienstleistungen nicht mehr sichtbar sind, wenn nur wenige Unternehmen diese anbieten.

Die Größe des Rechtecks soll daher zur Hälfte aus seiner relativen Größe bestehen. Also dem Verhältnis von Unternehmen, die die entsprechenden Dienstleistungen anbieten im Vergleich zur Gesamtzahl der angebotenen Dienstleistungen. Zur anderen Hälfte soll sie aus einer festen Größe bestehen, welche sich aus der Anzahl der Kategorien, die sich auf der gleichen Ebene des Teilbaums befinden, bestimmt. Der erste naive Ansatz ist, die Größe jedes Rechtecks einzeln zu bestimmen. Hierzu muss die Größe des Elternelements bekannt sein. Diese hängt jedoch wiederum von der Anzahl der Unternehmen ab, die eine Dienstleistung, die in der Taxonomie unterhalb des Elternelements angeordnet ist, anbieten. Zusätzlich muss für jedes Element die gesamte Anzahl der Unternehmen, die die betreffende Dienstleistung oder eine untergeordnete Dienstleistung anbieten, neu berechnet werden. Da die Taxonomie- und Dienstleistungs-Objekte in einer Datenbank gespeichert sind, generiert das in Rails umgesetzte Objekt-Relation-Mapping *Active Record* pro Objekt eine Datenbankabfrage. So kommt es bei der naiven Implementierung zu einer großen Zahl von Datenbankabfragen, welche sich zu einem großen Teil wiederholen und sehr viel Zeit benötigen.

3 Lösungsansätze und Umsetzung

Um die Ladezeit zu verringern kann nun die Anzahl der Datenbankabfragen und anderer Redundanzen schrittweise minimiert und zusätzlich Teilergebnisse der Berechnung oder sogar die gesamte Visualisierung zwischengespeichert (gecached) werden. Im ersten Schritt wird versucht, durch schrittweises *Refactoring* die Funktion so zu optimieren, so dass eine akzeptable Ladezeit erreicht wird.

Durch die Optimierung soll jedoch nicht nur die Geschwindigkeit der Berechnung erhöht werden. Ein weiteres Ziel ist es, die Lesbarkeit und Testbarkeit der konkreten Implementierung zu erhöhen. Dies ist wichtig, um auch später die Qualität und Funktionalität sicherstellen zu können. Es werden nach dem Prinzip *Separation of Concerns* [der08] Klassen und Funktionen eines Programms so gestaltet, dass sie möglichst unabhängig voneinander nutzbar sind und sich nur in wenigen Punkten berühren. Gleichzeitig soll gemäß dem Prinzip *Don't Repeat Yourself (DRY)* [HT91] Duplikation von Code vermieden werden. Die Funktionen sollen so kurz und einfach wie möglich gehalten werden¹ was neben der Verständlichkeit auch der besseren Testbarkeit dienen soll. Als Metriken zum Vergleich der Änderungen werden die zyklomatische Komplexität [Wik12], sowie die Codezeilen pro Funktion berechnet. Diese werden mithilfe des Ruby Gems *saikuro* [Blu] bestimmt.

Die Ladezeit-Analyse der Funktion erfolgt mithilfe der lokalen Version der Middleware *New Relic RPM* [new], welche eine Analyse der Anwendung mithilfe der Log-Daten vornimmt. Alle im folgenden angegebenen Werte beziehen sich dabei nur auf die zur Berechnung der Treemap nötigen Funktionen, aber beim vollständigen Aufruf innerhalb des Rails Frameworks. Es wurde der Mittelwert fünf in Folge ausgeführter Messungen berechnet.

Optimierung 1: Zunächst fällt auf, dass einige Rechteckgrößen mehrmals berechnet werden. Dies liegt daran, dass der Algorithmus zur Berechnung der Größe des aktuellen Elements die Größe des übergeordneten Elternelements kennen muss. Dies kann auch rekursiv berechnet werden, wird zu diesem Zeitpunkt jedoch nach der Berechnung nicht gespeichert. Daher wird für jedes Unterelement einer Dienstleistung die Fläche der Dienstleistung neu berechnet, was zu enormen Redundanzen in der Berechnung führt. Um dieses Problem zu lösen ist es also notwendig die bereits berechneten Größen abzuspeichern. Hier gibt es zwei Möglichkeiten. Man kann die berechneten Daten temporär speichern, um sie für die aktuelle Berechnung wiederverwenden zu können. Danach würden die Daten verfallen. Diese Methode würde jedoch eine komplexere Struktur des Algorithmus zur Folge haben, was dem Prinzip kleiner unabhängiger Funktionen widersprechen würde. Daher macht es in diesem Fall Sinn die berechneten Größendaten in der Datenbank zu speichern, da eine einzelne neue Verknüpfung zwischen Unternehmen und Dienstleistung nur in geringem Maße etwas an der Gestalt der Treemap ändert. Cronjobs eignen sich um die Daten in regelmäßigem Abstand erneut zu generieren und gegebenenfalls auf den aktuellen Stand der Datenbasis anzupassen.

Nachdem diese Änderung umgesetzt wurde, verringert sich die Ladezeit der Website drast-

¹Robert C. Martin: "The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that." [Mar08]

isch. Die Ladezeit beträgt nun 2,23 Sekunden, wobei nur noch 459 Datenbankabfragen getätigt werden.

Optimierung 2: Als nächstes kann die Ladezeit der Seite, durch Verbesserungen der Berechnung der Färbung der Flächen, auf 0,61 Sekunden und 174 Datenbankabfragen verringert werden. Vorher wurde die Färbung für jede Fläche separat berechnet. Dazu mussten alle Sub-Dienstleistungen untersucht werden. Hierdurch entstanden viele redundante Operationen. Verbessert werden konnte dies, indem bereits während der Flächenberechnung gespeichert wird, ob die jeweilige Dienstleistung vom Unternehmen angeboten wird. So können alle übergeordneten Dienstleistungen auf den Status der Subelemente zugreifen und so ohne weitere Datenbankzugriffe ihren eigenen Zustand bestimmen.

Optimierung 3: Im letzten Schritt der Analyse fällt auf, dass während der Berechnung mehrmals alle Dienstleistungen von der Datenbank abgerufen werden. Dies ist notwendig, da die Beziehung der Dienstleistungen zueinander, die Taxonomie, im Datenmodell selbst nicht modelliert ist. Diese Beziehung muss anhand der Kategorisierungsnummern bestimmt werden, wodurch es nötig wird alle Dienstleistungen zu betrachten, um sie einer Oberkategorie zuzuordnen zu können. Es muss also jedes Mal eine Anfrage an die Datenbank gestellt werden, die alle Dienstleistungen in Objektform zurückerliefert. *Rails* cached derartige Anfragen zwar, sie kosten dennoch relativ viel Zeit², da der gesamte „Active Record“ Stack durchlaufen werden muss. Die Taxonomie wird daher für die Dauer einer Anfrage im Speicher vorgehalten. Da die Treemap nur den im Augenblick der Anfrage erreichten Zustand der Datenbasis darstellen soll, ist es möglich sie nach einmaliger Abfrage im Speicher vorzuhalten, um so im Laufe der Berechnung immer wieder direkt darauf zugreifen zu können. Dies führt dazu, dass sich die Ladezeit auf 0,25 Sekunden verringert. Es wird nun nur noch eine einzige Datenbankabfrage gestellt.

An dieser Stelle ist es nur noch mit hohem Aufwand und vor allem mit Verlust an Verständlichkeit des Codes möglich weitere Optimierungsmöglichkeiten durch Refactoring zu finden und diese umzusetzen. Dies ist auch nicht mehr notwendig, da die Ladezeit durch die getroffenen Maßnahmen deutlich gesenkt wurde und sich nun auf akzeptablem Niveau befindet.

²1000 Zugriffe auf Objekt im Speicher ist mit der Ruby Bibliothek „Benchmark“ nicht messbar (0 Sekunden). Bei einem durch Active Record gecachtem Objekt wurden 0,54 Sekunden bei ebenfalls 1000 Zugriffen gemessen.

Optimierungsschritt	1	2	3	4
SQL-Anfragen	1488	459	147	1
Ladezeit in Sekunden ³	20,553	2,256	0,61	0,245
Beschleunigungsfaktor	1	9,102	33,693	83,89
durchschnittliche zyklomatische Komplexität	2,72	2,3	2,3	2,3
maximale zyklomatische Komplexität	6	3	3	3
Anzahl Funktionen	11	12	12	14
durchschnittliche Anzahl Codezeilen pro Funktion	11,72	8,75	8,75	6,79
maximale Anzahl Codezeilen einer Funktion	39	17	17	15

Tabelle 1: Darstellung der Optimierungsschritte

4 Fazit

Durch die Auswahl des Java-Script Frameworks und des „squarified“ Layout Algorithmus konnte eine für den Nutzer sehr ansprechende und verständliche Darstellung der Unternehmen erreicht werden. Durch die beschriebenen Änderungen im Vergleich zur naiven Funktion konnte die Antwortzeit der Webseite erheblich verbessert werden. Die Seite lädt nun in unter einer Sekunde, wodurch der Nutzer das Gefühl hat, die Seite frei und flüssig bedienen zu können. Daher ist es zunächst nicht mehr nötig weitere Optimierungsschritte durchzuführen. Zusätzlich werden durch die Optimierungen die einzelnen Funktionen zur Berechnung der Treemap stark entkoppelt, wodurch es nun sehr einfach ist Unit-Tests zu erstellen, um die korrekte Funktionsweise in Zukunft sicher zu stellen.

Danksagung

Ich danke Michael Prilop und Dr. Lutz Maicher vom Fraunhofer MOEZ dafür, dass sie mich sowohl bei der Realisierung des Projekts, als auch bei der Fertigstellung dieses Papers unterstützt haben.

Literatur

- [Apa09] Walter Apai. Interview with Web Usability Guru, Jakob Nielsen. <http://www.webdesignerdepot.com/2009/09/interview-with-web-usability-guru-jakob-nielsen/>, September 2009.
- [Bel] Nicolas Garcia Belmonte. Treemap - Animated Squarified, SliceAndDice and Strip TreeMaps. <http://thejit.org/static/v20/Jit/Examples/Treemap/example1.html>.

³Mittelwert von fünf Messungen im Single-User Betrieb auf dem Testsystem (Intel CoreI7-2600@2x3,40GHz , 6GB Ram, MintLinux 32Bit)

- [BHvW00] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified Treemaps. In *Data Visualization 2000: Proc. Joint Eurographics and IEEE TCVG Symp. on Visualization*, pages 33–42. Springer-Verlag, 2000.
- [Blu] Zev Blut. Saikuro : A Cyclomatic Complexity Analyzer. <http://saikuro.rubyforge.org/>.
- [der08] derekgreer. The Art of Separation of Concerns. <http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/>, March 2008.
- [Hod] Jonathan Hodgson. Squarified Treemaps in XAML & C# using Microsoft Longhorn. <http://www.codeproject.com/Articles/7039/Squarified-Treemaps-in-XAML-C-using-Microsoft-Long>.
- [HT91] Andrew Hunt and David Thomas. The Pragmatic Programmer. In *The Pragmatic Programmer*, page 26. Springer Verlag, 1991.
- [Knu] Donald Knut. Structured Programming with go to Statements. *ACM Journal Computing Surveys*, (4):261.
- [Mar08] Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. page 34. Prentice Hall International, 1 edition, January 2008.
- [MTBJ12] Lutz Maicher, Liina Tonisson, Fabian Batsch, and Pirjo Jha. IP Industry BaseIntellectual Property Services Taxonomy (IPST). <http://ipib.ci.moez.fraunhofer.de/ipst>, 2012.
- [new] New Relic : Web Application Performance Management (APM). <http://newrelic.com/>.
- [Shn06] Ben Shneidermann. Discovering Business Intelligence Using Treemap Visualizations. <http://www.perceptualedge.com/articles/b-eye/treemaps.pdf>, November 2006.
- [TBJ⁺12] Liina Tonisson, Fabian Batsch, Pirjo Jha, Lutz Maicher, and Michael Prilop. Service profiling - a method for data driven competitive intelligence in service industries. In *International Symposium on Services Science*. Leipzig, 2012.
- [Wik12] Wikipedia contributors. McCabe-Metrik, August 2012. Page Version ID: 106562844.

Metriken und optimale Einsatzszenarien für Garbage Collectoren der Java HotSpot Virtual Machine

Michael Schmeißer

michael@skamandros.de

Hochschule für Technik, Wirtschaft und Kultur Leipzig

Master Studiengang Informatik

Zusammenfassung: Diese Arbeit liefert ein Entscheidungsverfahren zur Auswahl eines Garbage Collectors der Java HotSpot Virtual Machine für ein beliebiges Programm, basierend auf einfach zu ermittelnden Entscheidungsgrundlagen. Dafür werden Metriken für Garbage Collectoren vorgestellt und deren Gewinnung mithilfe einer eigens entwickelten Messplattform beschrieben. Mithilfe dieser Messplattform werden für ausgewählte Beispiele Metriken ermittelt und interpretiert. Schließlich werden die anhand der Experimente gewonnenen Erkenntnisse verallgemeinert, wodurch sich Best Practices für die Auswahl eines Garbage Collectors ergeben.

1 Einleitung

Garbage Collection ist nicht selten ein Faktor in Softwareprojekten, dem im Anfangsstadium keine Beachtung geschenkt wird, da die Standardeinstellungen ausreichen, um die entwickelte Software auszuführen. Mit fortschreitendem Projektstatus treten jedoch in einigen Projekten Fragen rund um Garbage Collection auf, die den Aspekt der automatischen Speicherverwaltung in den Fokus rücken. Ein Beispiel für eine solche Frage ist:

Welcher Garbage Collector eignet sich für mein Programm am besten?

Um festzustellen, für welche Klassen von Programmen sich ein Garbage Collector eignet und für welche nicht, sind eingehende Untersuchungen erforderlich. Der in dieser Arbeit verfolgte Ansatz besteht darin, einfache, charakteristische Beispiele zu konstruieren und das Verhalten der Garbage Collectoren bezüglich dieser Beispielprogramme zu messen. Anschließend werden die Ergebnisse der Garbage Collectoren bezogen auf die Charakteristika der untersuchten Programme eingeordnet. [Zor89] beschreibt dieses Vorgehen als ungeeignet, um Garbage Collectoren als Ganzes zu vergleichen. Es eignet sich jedoch, um einzelne Aspekte zu untersuchen, da die Verwendung von komplexeren, realitätsnahen Beispielprogrammen eine Vielzahl von Effekten hervorruft, die kaum oder gar nicht auf einzelne Ursachen zurückgeführt werden können. Diese Arbeit verzichtet deshalb darauf, vergleichende Aussagen über Garbage Collectoren als Ganzes zu treffen und konzentriert sich auf einzelne, konkrete Aspekte.

Für die Sprache Java, welche automatische Speicherverwaltung einsetzt, existieren zahlreiche Untersuchungen, die darauf abzielen, den geeignetsten Garbage Collector für ein Pro-

gramm zu finden. Die Eignung eines Garbage Collectors wird dabei an einer bestimmten Zielmetrik festgestellt. Auch gibt es Untersuchungen, die Techniken aus dem Bereich der künstlichen Intelligenz zur Optimierung der Parameter des Garbage Collectors anwenden oder gar diesen zur Laufzeit umschalten, um ein optimales Verhalten bezüglich einer Zielgröße zu erreichen [SKB04, SBWC07, SKBL11]. Selten jedoch werden die Ursachen erforscht, warum ein Programm unter Verwendung eines bestimmten Garbage Collectors zu dem beobachteten Verhalten des Gesamtsystems führt. Dies rührt meist daher, dass die Motivation für die Untersuchung ein Programm ist, dessen Ausführungsverhalten verbessert werden soll. Ein solches Programm ist allerdings meist zu komplex, um die beobachteten Effekte mit vertretbarem Aufwand auf die Charakteristika des Programms zurückführen zu können, weswegen sich die Untersuchungen auf eine Optimierung des Gesamtsystems konzentrieren.

Der folgende Abschnitt 2 führt zentrale Begriffe ein. Anschließend beschreibt Abschnitt 3 die experimentelle Vorgehensweise, Abschnitt 4 definiert die verwendeten Metriken und geht auf deren Erfassung ein. In Abschnitt 5 werden die im Rahmen der Arbeit durchgeführten Experimente erläutert und deren Ergebnisse analysiert. Schließlich werden die Resultate in Abschnitt 6 zu Kriterien für die Auswahl eines Garbage Collectors verallgemeinert, bevor ein Ausblick auf weitere Forschungsmöglichkeiten die Arbeit abschließt. Eine Beschreibung der Testsysteme, die für die Experimente dieser Arbeit verwendet wurden, ist unter [Sch12] verfügbar.

2 Automatische Speicherverwaltung

2.1 Terminologie

Automatische Speicherverwaltung bezeichnet die selbständige Rückgewinnung von Speicher, der nicht länger von einer Anwendung benötigt wird [Tho10, S. 5]. Selbständig bedeutet dabei, dass der Speicher zu einem beliebigen Zeitpunkt zurückgewonnen ist, spätestens aber wenn er von der Anwendung zwingend benötigt wird. Der Begriff *Heap* bezeichnet den Speicherbereich, der für die dynamische Allokation zur Verfügung steht. Er werde zur Beschreibung des allgemeinen Modells automatischer Speicherverwaltung als *Speichergraph* gemäß Definition 1 betrachtet.

Definition 1: Speichergraph, Wurzeln, vgl. [JHM11, S. 12f.]

Sei $\mathcal{G} = (V, E) \mid E \subseteq V^2$ ein gerichteter Graph, der möglicherweise Zyklen enthalte und *Speichergraph* heiße. Alle Knoten V des Graphen repräsentieren Objekte im Speicher und alle Kanten E repräsentieren Referenzen zwischen Objekten. Eine bestimmte Teilmenge der Knoten werde als *Wurzeln* (\mathcal{R}) bezeichnet.

Die Aufgabe von automatischer Speicherverwaltung besteht darin, den Speicher aller Datenstrukturen freizugeben, die nicht länger benötigt werden. Automatische Speicherverwaltung garantiert dabei, dass kein Speicher freigegeben wird, der noch für die weitere Programmausführung benötigt wird. Um nicht benötigten Speicher zu identifizieren, muss zunächst definiert werden, wann Speicher nicht mehr benötigt wird. Definition 2

beschreibt, wann Objekte so genannter Müll sind und zurückgewonnen werden können.

Definition 2: Erreichbarkeit, Müll (engl. Garbage), vgl. [JHM11, S. 13f.]

Sei „ \rightarrow “ als Adjazenzrelation auf \mathcal{G} definiert, sodass $\forall a, b \in V : a \rightarrow b \iff (a, b) \in E$.
 Es heie dann b erreichbar von a (geschrieben $a \rightarrow^* b$), genau dann, wenn das Paar (a, b) Element der transitiven Hulle der Adjazenzrelation ist:

$$x \rightarrow^* y \iff \exists n \geq 0 \exists x_1, \dots, x_n \in V : x \rightarrow x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n \rightarrow y$$

Ein Objekt o im Speichergraphen heie genau dann *lebend*, wenn es Teil von \mathcal{R} oder von \mathcal{R} erreichbar ist, das heit: $lebend(o) \iff \exists r \in \mathcal{R} : o = r \vee r \rightarrow^* o$

Als Müll (auch: tote Objekte) wird die Menge aller Objekte bezeichnet, die nicht leben:

$$Mll = \{o \mid o \in V \wedge \neg lebend(o)\}$$

Die bisherige Beschreibung der in einem Programm mit automatischer Speicherverwaltung vorhandenen Komponenten lsst sich verfeinern, wie in Definition 3 erlutert wird.

Definition 3: Programm mit automatischer Speicherverwaltung [DLM⁺78]

Ein Programm heit *automatisch speicher verwaltet*, wenn es sich in folgende – nur ber Schnittstellen verbundene – Teile trennen lsst:

- Der *Mutator*, welcher Anweisungen ausfhrt, die den Speichergraphen (Definition 1) durch Allokation neuer und Modifikation existierender Objekte verndern, sich jedoch nicht um Speicherbeschaffung und -rckgewinnung kmmert.
- *Kollektor* und *Allokator*: Der Kollektor fhrt Quelltext zur Identifizierung von Mll (Definition 2) aus und gibt deren Speicher frei. Der Allokator bearbeitet Speicheranfragen des Mutators und ruft gegebenenfalls den Kollektor auf.

Die automatische Speicherverwaltung wird durch den Kollektor und den Allokator realisiert: Der Mutator fordert neuen Speicher vom Allokator an, welcher gegebenenfalls den Kollektor aufruft, um Speicher zurckzugewinnen. Da diese beiden Komponenten des Systems gem Definition 3 vom Mutator durch klare Schnittstellen getrennt sind, lassen sie sich austauschen, ohne dass Anpassungen am Mutator ntig sind. Diese austauschbare Implementierung der automatischen Speicherverwaltung wird als *Garbage Collector* (Definition 4) bezeichnet.

Definition 4: Garbage Collector

Ein *Garbage Collector* ist ein Teil eines Programmes, dessen Aufgabe die Bereitstellung von Speicher fr neue Objekte durch die Rckgewinnung von Speicherressourcen nicht lnger bentigter Speicherzellen ist. Eine zentrale Charakteristik eines Garbage Collectors ist die Realisierung der Bestimmung des nicht lnger bentigten Speicherinhalts – *Garbage* (dt. Mll, Definition 2) genannt. Bezogen auf Definition 3 kann ein Garbage Collector als Realisierung eines Kollektors verstanden werden, oft sind dabei allerdings auch Anpassungen am Allokator ntig.

2.2 Arten von Garbage Collectoren

Garbage Collectoren lassen sich nach einer Vielzahl von Kriterien unterscheiden, die in diesem Abschnitt näher betrachtet werden [Sun06, S. 5]. Eine Unterscheidungsmöglichkeit von Garbage Collectoren besteht darin, wie sie mit toten Objekten umgehen. Einerseits gibt es *konservative* Garbage Collectoren, die lediglich den Speicher freigeben, an dem sich das nicht länger benötigte Objekt befunden hat. Eine andere Lösung stellt die *kopierende* Rückgewinnung dar, bei der alle lebenden Objekte zu einem durchgängig belegten Speicherbereich kompaktiert werden. Dies kann einerseits durch eine Verschiebung innerhalb des Heap-Segments geschehen, in dem sich die Objekte befinden, oder durch Evakuierung der Objekte in ein anderes Heap-Segment, welches danach alle lebenden Objekte beinhaltet.

Für viele Programme ist es heutzutage eine entscheidende Anforderung, dass sie auf Mehrkernarchitekturen skalieren. Diese Anforderung gilt ebenfalls für Garbage Collectoren. Man unterscheidet *serielle* Kollektoren, welche die Bestimmung und Bereinigung des Mülls in einem einzelnen Thread ausführen und *parallele* Kollektoren, die mehrere Threads dafür verwenden. Ein weiteres Unterscheidungsmerkmal besteht darin, wie stark die Interaktion und Beeinflussung des Mutators durch den Kollektor in einem System mit automatischer Speicherverwaltung ist. Während klassische Kollektoren *pausierend* arbeiten, das heißt während des gesamten Aufräumzyklus alle Mutator-Threads anhalten, gibt es mittlerweile auch Kollektoren, die *simultan* arbeiten. Dies bedeutet, dass Kollektor-Threads zum Teil nebenläufig zu Mutator-Threads arbeiten können.

2.3 Automatische Speicherverwaltung in Java

In der Java HotSpot Virtual Machine (HotSpot JVM) existieren 5 verschiedene Garbage Collectoren, welche zum Teil in diesem Abschnitt beschrieben werden [DFHP04, Sun06, S. 7ff.]. Die in dieser Arbeit beschriebenen Garbage Collectoren unterteilen den Heap in mehrere Bereiche, welche Abbildung 1 dargestellt. Sie unterscheiden ferner zwischen zwei verschiedenen Typen von Aufräumzyklen:

Sobald die Allokationsregion *Eden* über zu wenig freien Speicher verfügt, um eine Allokationsanforderung zu befriedigen, wird ein kleiner Aufräumzyklus (engl. minor collection) ausgelöst, bei dem ausschließlich die Partitionen Eden und From-Space der jungen Generation bereinigt werden. Die lebenden Objekte werden in den To-Space evakuiert, welcher im nächsten kleinen Aufräumzyklus seine Rolle mit dem bisherigen From-Space tauscht. Lebende Objekte, welche ein gewisses Alter erreicht haben oder nicht mehr in den To-Space passen, werden in die alte Generation befördert. Reicht dafür der Platz nicht aus, wird ein großer Aufräumzyklus (engl. major collection) ausgelöst. Bei einem großen Aufräumzyklus wird der gesamte Heap bereinigt. In vielen Fällen wird die alte Generation mit einem anderen Verfahren bereinigt als die junge Generation. Dies begründet sich unter anderem darin, dass die erwartete Sterberate in der alten Generation niedriger ist und sich deshalb andere Verfahren besser eignen als für die junge Generation. [Ung84]

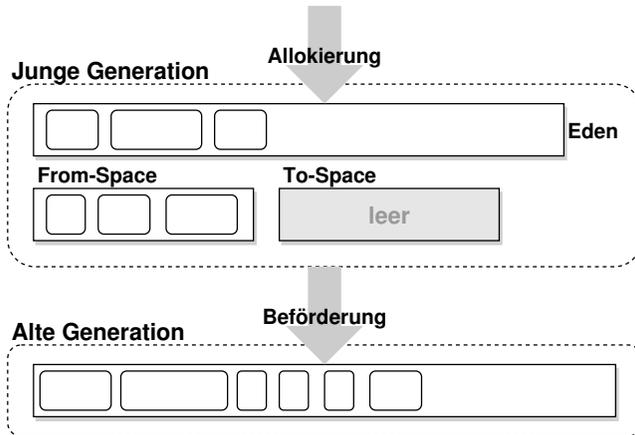


Abbildung 1: Generationenschema der Garbage Collectoren der HotSpot JVM [Sun06, Abb. 1 u. 2, S. 6f.]

Einer der drei in dieser Arbeit betrachteten Garbage Collectoren ist der *Serial Collector*. Dieser Garbage Collector arbeitet seriell, pausierend und kopierend. Bei einem kleinen Aufräumzyklus geht er nach dem bereits beschriebenen Verfahren vor, wobei ein einzelner Thread zum Einsatz kommt. Bei großen Aufräumzyklen führt der Serial Collector die Bereinigung der alten und der permanenten Generation mit einem kompaktierenden, markierenden Verfahren durch.

Bei dem *Parallel Compacting Collector* (Abk.: PC Collector) handelt es sich um einen parallelen, pausierenden und kopierenden Garbage Collector. Im Gegensatz zum Serial Collector werden für die meisten Teilaufgaben mehrere Threads eingesetzt. Zur Bereinigung der jungen Generation setzt der Parallel Compacting Collector nahezu das gleiche Verfahren wie der Serial Collector ein. Der Unterschied besteht darin, dass mehrere Threads gleichzeitig lebende Objekte identifizieren und evakuieren. Für die Bereinigung der alten Generation kommt ein Verfahren zum Einsatz, das in drei Phasen arbeitet – eine der Phasen ist nicht parallelisiert, das heißt, verwendet nur einen Thread.

Für Szenarien, in denen die Antwortzeit eines Programms entscheidend ist, steht in der HotSpot JVM der *Concurrent Mark-Sweep Collector* (Abk.: CMS Collector) zur Verfügung. Bei diesem Garbage Collector werden große Aufräumzyklen hauptsächlich simultan zur Mutatoraktivität durchgeführt. Für kleine Aufräumzyklen kommt das gleiche Verfahren wie beim Parallel Compacting Collector zum Einsatz.

Im in dieser Arbeit ausschließlich betrachteten Servermodus der HotSpot JVM ist der Parallel Compacting Collector als Garbage Collector voreingestellt.¹

¹<http://java.sun.com/docs/hotspot/gc5.0/ergo5.html>

3 Experimentelle Vorgehensweise

Für eine Reihe von Garbage Collectoren spielt die Lebensdauer von Objekten eine Rolle. Es gibt zwei Ansätze, wie die Lebenszeit von Daten definiert werden kann: Zum einen kann die Realzeit in Sekunden zwischen Geburt und Tod des Objekts verwendet werden und zum anderen die Anzahl der Bytes, die während dieses Zeitraums innerhalb des Systems mit automatischer Speicherverwaltung allokiert wurden [JHM11, 9.2 Measuring time, S. 113]. Da sich die Menge der bis zum Tod eines Objektes allokierten Bytes präziser beeinflussen lässt als die vergangene Zeit in Sekunden und überdies in direktem Zusammenhang mit der Belastung der Speicherverwaltungskomponente steht, wird in den Experimenten die Lebenszeit der Testobjekte relativ zum allokierten Speicher beeinflusst. Um dies zu erreichen, werden Containerobjekte verwendet, welchen mittels des Methodenaufrufs `add(Testobjekt, Lebenszeit)` ein Testobjekt übergeben werden kann, das sie dann für eine bestimmte Zeit referenzieren und damit verhindern, dass es zu Müll wird. Alle anderen Referenzen auf die Testobjekte werden unmittelbar nach der Übergabe an das Containerobjekt entfernt, sodass der einzige Referenzpfad von den Wurzeln zu dem jeweiligen Testobjekt über das Containerobjekt hergestellt wird.

Die an die Methode des Containerobjekts übergebene Lebenszeit ist dabei die Anzahl der Aufrufe derselben Methode bis das Containerobjekt die Referenz auf das Testobjekt fallen lässt, wodurch das Testobjekt zu Müll wird. Jedes Testobjekt einer Testserie wird mithilfe der `add(...)`-Methode dem Containerobjekt der Testserie hinzugefügt. Der konkrete Wert für die übergebene Lebenszeit ergibt sich aus dem Index des Testobjektes in der jeweiligen Testserie, welcher als Argument einer der in Gleichung 1 dargestellten Funktionen übergeben wird. rng_x bezeichnet dabei eine zufällige, aber feste Funktion von \mathbb{N} nach $[1, x] \subset \mathbb{N}$.

$$\begin{aligned} \mathbf{L}(x) : \mathbb{N} &\longrightarrow [1, x] \subset \mathbb{N} & \mathbf{L}_c(x) &= \{\forall i \in \mathbb{N} : i \mapsto x\} \\ \mathbf{L}_s(x) &= \left\{ \forall i \in \mathbb{N} : i \mapsto \max_{k \leq x} (k \mid i) \right\} & \mathbf{L}_p(x) &= \left\{ \forall i \in \mathbb{N} : i \mapsto rng_x \left[\left\lfloor \frac{i-1}{\frac{x}{10}} \right\rfloor \right] \right\} \end{aligned} \quad (1)$$

Die Erfassung von Werten für die in Abschnitt 4 beschriebenen Metriken wird von anderen Prozessen gestört, die zeitlich mit der Ausführung des Testprozesses überlappen. Weiterhin ist auch die HotSpot JVM selbst ein komplexes Programm, welches eine Vielzahl zusätzlicher Berechnungen in seinen eigenen Threads durchführt. Als Beispiele lassen sich das Laden von Klassen und die Initialisierung statischer Variablen kurz nach dem Start der HotSpot JVM oder die Optimierung von oft ausgeführten Methoden nennen. Aus diesen Gründen kamen bei den Experimenten in dieser Arbeit folgende Maßnahmen zur Reduzierung von Messfehlern zum Einsatz:

- Durchführung mehrerer Messungen mit Entfernung von Ausreißern, das heißt, den jeweils k größten und kleinsten Werten mit anschließender Mittelwertbildung
- Voranstellung einer Aufwärmphase vor die Tests, in welcher die gleichen Berechnungen wie in den späteren Messungen zehnmal durchgeführt werden
- Zeitlich getrennte Erfassung von Performance-Metriken und berechnungsintensiven Metriken

4 Metriken

Eine für den Nutzer eines Programms mit automatischer Speicherverwaltung oft besonders relevante Metrik ist die *Gesamtausführungszeit* des Programms, das heißt die gesamte Zeit, die für Allokation, Mutation und Kollektion verbraucht wird. Zur Erfassung der Werte für diese Metrik wird die Methode `System.nanoTime()` vor und nach dem jeweiligen Quelltextabschnitt aufgerufen und die Differenz gebildet. Die Verwendung dieser Methode ist das geeignetste Vorgehen zur Messung vergangener Zeit, allerdings garantiert die Methode keine Genauigkeit im Nanosekundenbereich und der Aufruf selbst verbraucht auf manchen Plattformen einige Mikrosekunden [Hol06]. Durch Rundung auf Millisekunden wird diesen Einschränkungen Rechnung getragen.

Programmphasen, in denen die Mutatoren durch einen Kollektor angehalten werden, werden als *Pausenzeit* bezeichnet. Zur Charakterisierung der Pausenzeit existieren verschiedene konkrete Metriken. Eine bloße Messung der durchschnittlichen oder maximalen Pausenzeit ist wenig sinnvoll, weil dadurch nicht erfasst wird, wie lange die Mutatoren tatsächlich Zeit für ihre Berechnungen hatten [JHM11, S. 7]. Deshalb wurden aussagekräftigere Metriken wie beispielsweise die *beschränkte Mutatornutzung* (aus [SMB04], engl. *bounded mutator utilisation*, Abk.: BMU) entwickelt, welche das Verhältnis von Mutator- und Kollektoraktivität charakterisieren.

Die Mutatoraktivität η lässt sich dabei als Quotient aus der Anzahl von Rechenzyklen, in denen der Mutator im Intervall $[a, b]$ aktiv war, $T_m(a, b)$, und der Intervalllänge $x = b - a$ ausdrücken, wie in Gleichung 2 dargestellt.

$$T_m : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; 0 < T_m(a, b) \leq b - a | a, b \in \mathbb{N}; b > a \geq 0 \quad (2)$$

$$\eta(t, x) = \frac{T_m(t, t+x)}{x}$$

Auf der Mutatoraktivität aufbauend definiert Gleichung 3 die *minimale Mutatornutzung* MMU für eine Berechnung C , deren Dauer in Rechenzyklen $|C|$ und eine Intervalllänge x als minimale Mutatoraktivität bezogen auf alle Intervalle der Länge x innerhalb der Ausführung von C [CB01]. Gleichung 4 definiert die beschränkte Mutatornutzung BMU für eine Intervalllänge x , welche im Gegensatz zur MMU monoton steigend ist.

$$MMU_C(x) = \min_{0 \leq t \leq |C| - x} (\eta_C(t, x)) \quad (3)$$

$$BMU(x) = \min_{x' \geq x} (MMU(x')) \quad (4)$$

Bei sowohl BMU als auch MMU gibt das maximale x , für das die Metriken 0 sind, die Länge der größten Mutatorpause an. Der Wert der Metriken für die Gesamtlaufzeit der Berechnung gibt den Gesamtanteil der Mutatoraktivität an.

Um die Werte für die in diesem Abschnitt vorgestellten Metriken zu gewinnen, wird mithilfe von HotSpot JVM Optionen² die Ausgabe der Länge von Phasen aktiviert, in denen der Mutator nicht pausiert beziehungsweise pausiert war. Diese Ausgabe wird in eine Datei umgelenkt und nach der Ausführung zur Berechnung der Metriken ausgewertet.

²-XX:+PrintGCApplicationConcurrentTime und -XX:+PrintGCApplicationStoppedTime

5 Experimente

Zur Analyse des Verhaltens der Garbage Collectoren der HotSpot JVM wird in diesem Abschnitt zunächst ein Programm betrachtet, welches Fibonacci-Bäume³ [Knu98] generiert und anschließend mehrmals Knoten daraus löscht⁴, wobei nach jedem Löschvorgang die AVL-Eigenschaft des Baumes wiederhergestellt wird. Beispielprogramm 1 modelliert durch folgende voneinander unabhängige Berechnungen ein Szenario mit durchgehend starker Nutzung von bis zu zwei Hardware-Threads:

- (A) Eine Berechnung, die 50 000 Fibonacci-Bäume der Stufe 7 generiert und anschließend 7 Knoten aus jedem Baum löscht.
- (B) Eine Berechnung, die 7 500 Fibonacci-Bäume der Stufe 11 generiert und anschließend 11 Knoten aus jedem Baum löscht.

Für die Erzeugung der Lebensdauer der Objekte wird in beiden Berechnungen $L_s(5113)$ verwendet, wobei der Index des Fibonacci-Baums innerhalb der Berechnung als Argument dient. Wie Tabelle 1 zeigt, wird mit dem Parallel Compacting Collector für alle drei betrachteten Heap-Größen das beste Ergebnis für die Gesamtausführungszeit erzielt. Ebenfalls ist zu sehen, dass sowohl der Serial Collector als auch der Parallel Compacting Collector Vorteile gegenüber dem CMS Collector haben. Dieser Effekt lässt sich damit begründen, dass das Testsystem nur über zwei Hardware-Threads verfügt, welche beide die meiste Zeit für die eigentlichen Berechnungen benötigt werden, sodass der CMS Collector keinen Vorteil durch simultane Ausführung der Aufräumzyklen erzielen kann. Tabelle 2 unterstreicht dies für den Fall von drei vorhandenen Hardware-Threads, in welchem der CMS Collector bereits ein besseres Ergebnis als der Serial Collector erzielt.

Heap-Größe	Serial Collector		PC Collector		CMS Collector	
	\bar{T}	σ_T	\bar{T}	σ_T	\bar{T}	σ_T
288 MiB	60 367	33	57 760	127	61 236	140
480 MiB	54 968	25	51 973	355	60 166	208
768 MiB	54 858	30	49 887	264	62 824	135

Tabelle 1: Gesamtausführungszeit T von Beispielprogramm 1 in Millisekunden ermittelt in 50 Versuchen mit Entfernung von jeweils 7 Minima und Maxima auf Testsystem 1

Tabelle 1 zeigt weiterhin eine unerwartete Verschlechterung der Gesamtausführungszeit für den CMS Collector bei einer Erhöhung der Heap-Größe von 480 auf 768 MiB. Es besteht die Vermutung, dass es sich dabei trotz der ergriffenen Maßnahmen zur Eliminierung von Messfehlern um solche handelt, was durch weitere Versuche zu klären wäre. Allgemein ist jedoch festzuhalten, dass die Erhöhung der Heap-Größe von 288 auf 480 MiB eine deutlich größere Reduzierung der Gesamtausführungszeit bewirkt hat als die weitere Erhöhung auf 768 MiB.

³Jeder Knoten enthält einen Balancefaktor, eine Zufallszeichenkette der Länge 20 und eine 32 Bit lange Zahl.

⁴Es wird der jeweils rechtste Knoten gelöscht, das heißt genau der Knoten, der durch verfolgen der Referenzen zum rechten Unterbaum von der Wurzel aus erreicht wird und dessen rechter Unterbaum leer ist.

Zur Ermittlung des bestgeeigneten Garbage Collectors für Systeme mit Einkernprozessoren wurde dasselbe Beispielprogramm auf einem virtuellen Debian-System ausgeführt, dem drei beziehungsweise ein einziger Kern der auf dem Host-System vorhandenen vier Kerne zur Verfügung gestellt wurden. Wie das Ergebnis in Tabelle 2 zeigt, platziert sich in der Konfiguration mit einem Kern der Concurrent Mark-Sweep Collector erneut als letzter der betrachteten hinter den beiden pausierenden Kollektoren. Interessanterweise erzielt jedoch trotz nur eines verfügbaren Hardware-Threads nicht der Serial Collector das beste Ergebnis, sondern liegt weiterhin hinter dem Parallel Compacting Collector. Als Erklärung hierfür kommen die unterschiedlichen Verfahren für große Aufräumzyklen in Frage.

Hardware-Threads	Serial Collector		PC Collector		CMS Collector	
	\bar{T}	σ_T	\bar{T}	σ_T	\bar{T}	σ_T
3	28 806	267	24 709	123	28 561	95
1	44 560	240	42 379	493	48 085	333

Tabelle 2: Gesamtausführungszeit T von Beispielprogramm 1 in Millisekunden ermittelt in 50 Versuchen mit Entfernung von jeweils 7 Minima und Maxima bei 750 MiB Heap-Größe auf Testsystem 4

In einem weiteren Experiment wurden 400 000 Objekte mit einem Gesamtdatenvolumen von circa 16 GiB mithilfe von Protocol Buffers⁵ in Java-Objekte deserialisiert (Beispielprogramm 2). Die deserialisierten Objekte wurden an einen Objektcontainer übergeben, welcher unterschiedliche Funktionen für die Erzeugung der Lebenszeit der Objekte verwendet. Die Ergebnisse bezüglich der Gesamtausführungszeit stellt Tabelle 3 dar. Es stellt sich heraus, dass der Parallel Compacting Collector für L_c und L_p , welche bezüglich steigender Argumente dauerhaft beziehungsweise abschnittsweise konstant sind, das beste Ergebnis erzielt. Für ständig wechselnde Lebenszeiten unter Verwendung von L_s positioniert sich jedoch der Concurrent Mark-Sweep Collector am besten. Dieses Bild bestätigt sich auch auf die BMU(2s) bezogen, wie Tabelle 4 zeigt.

6 Resultat

Um eine Entscheidungsgrundlage für die Auswahl eines Garbage Collectors zu bilden, werden die Garbage Collectoren bezüglich der Experimente in dieser Arbeit folgendermaßen eingeordnet:

$$\text{Symbol} = \begin{cases} ++ & \text{bestes Ergebnis in allen betrachteten Experimenten} \\ + & \text{nie mehr als 10\% schlechter als das beste Ergebnis} \\ \circ & \text{nie mehr als 20\% schlechter als das beste Ergebnis} \\ - & \text{nie mehr als 30\% schlechter als das beste Ergebnis} \\ -- & \text{sonst} \end{cases}$$

⁵Protocol Buffers - Google's data interchange format (<http://code.google.com/p/protobuf/>)

Lebenszeit	Serial Collector		PC Collector		CMS Collector	
	\bar{T}	σ_T	\bar{T}	σ_T	\bar{T}	σ_T
$L_s(4000)$	37 050	28	37 359	25	36 551	60
$L_c(4000)$	13 926	19	13 896	14	23 639	104
$L_p(4000)$	13 097	34	12 951	19	18 857	136

Tabelle 3: Gesamtausführungszeit T von Beispielprogramm 2 in Millisekunden ermittelt bei 32 MiB Heap-Größe in 50 Versuchen mit Entfernung von jeweils 7 Minima und Maxima auf Testsystem 1

Lebenszeit	Serial Collector		PC Collector		CMS Collector	
	\overline{BMU}	σ_{BMU}	\overline{BMU}	σ_{BMU}	\overline{BMU}	σ_{BMU}
$L_s(4000)$	0,921	0,002 2	0,918	0,002 4	0,915	0,009 3
$L_c(4000)$	0,718	0,000 4	0,734	0,001 8	0,418	0,008 6
$L_p(4000)$	0,783	0,001 9	0,787	0,002 4	0,532	0,008 1

Tabelle 4: BMU(2s) von Beispielprogramm 2 ermittelt bei 32 MiB Heap-Größe in 50 Versuchen mit Entfernung von jeweils 7 Minima und Maxima auf Testsystem 1

Es werden für verschiedene Charakteristika jeweils die Ergebnisse aller passenden Experimente betrachtet und das jeweils schlechteste Ergebnis der Garbage Collectoren herangezogen. Tabelle 5 zeigt das Ergebnis der Garbage Collectoren bezüglich der auf dem Testsystem verfügbaren Hardware-Threads im Verhältnis zu den durch das untersuchte Programm maximal auslastbaren Threads. Der Serial Collector erzielt dabei ein nachvollziehbares Ergebnis: Mit mehr zur Verfügung stehenden Hardware-Threads verschlechtert sich sein Ergebnis relativ zum bestplatzierten Parallel Compacting Collector, welcher mehrere Threads nutzt. In Tabelle 6 ist das relative Ergebnis der Garbage Collectoren hinsichtlich der Gesamtausführungszeit bezogen auf die Lebenszeit der Testobjekte zusammengefasst. Es zeigt sich dabei, dass der Serial Collector seine besten Ergebnisse unter Verwendung von L_c erzielt, während der CMS Collector bei Einsatz von L_s sein bestes Resultat erreicht.

Hardware-Threads	Serial Collector	PC Collector	CMS Collector
Mangel	+	++	o
Sättigung	+	++	-
Überschuss	o	+	o

Tabelle 5: Eignung der betrachteten Garbage Collectoren bzgl. Gesamtausführungszeit bezogen auf die zur Verfügung stehenden Hardware-Threads formuliert als Mangel, Sättigung oder Überschuss je nachdem, ob die Anzahl der verfügbaren Hardware-Threads kleiner, gleich oder größer als die maximal durch das Programm auslastbaren Hardware-Threads ist

Das Ergebnis in Hinblick auf die BMU(2s) bezogen auf unterschiedliche Funktionen zur Erzeugung der Lebenszeit der Testdaten ist in Tabelle 7 dargestellt. Unter Verwendung

Lebenszeit	Serial Collector	PC Collector	CMS Collector
konstant	++	++	--
abschnittsweise konstant	+	++	--
wechselnd	o	+	o

Tabelle 6: Eignung der betrachteten Garbage Collectoren bzgl. Gesamtausführungszeit bezogen auf die Lebenszeit der Daten – konstant unter Verwendung von L_c , abschnittsweise konstant unter Verwendung von L_p und wechselnd unter Verwendung von L_s

von L_s erreichten alle Garbage Collectoren im Rahmen der Standardabweichung das gleiche Ergebnis. Für die anderen betrachteten Funktionen zur Erzeugung der Lebenszeit schnitt der CMS Collector jedoch deutlich schlechter ab. Als mögliche Ursache kommt hier nicht nur die veränderte Verteilung der Lebenszeit der Testobjekte in Betracht, sondern auch die Tatsache, dass der erhöhte Aufwand zur Berechnung der Funktionswerte von L_s zu einer höheren Gesamtausführungszeit und damit zu einer langsameren Allokationsgeschwindigkeit führt. An dieser Stelle empfehlen sich weitere Untersuchungen, welche die Effekte der Allokationsgeschwindigkeit von denen der Verteilung der Lebenszeit der Daten getrennt betrachten.

Lebenszeit	Serial Collector	PC Collector	CMS Collector
konstant	+	++	--
abschnittsweise konstant	++	++	--
wechselnd	++	++	++

Tabelle 7: Eignung der betrachteten Garbage Collectoren bzgl. BMU(2s) bezogen auf die Lebenszeit der Daten – konstant unter Verwendung von L_c , abschnittsweise konstant unter Verwendung von L_p und wechselnd unter Verwendung von L_s

Insgesamt lässt sich festhalten, dass der Parallel Compacting Collector in allen betrachteten Experimenten bezüglich der Gesamtausführungszeit nie mehr als 10% schlechter als der bestplatzierte Garbage Collector abschnitt. Bezüglich der BMU(2s) erzielte er stets das beste Ergebnis. Angesichts dieser Resultate ist es sinnvoll, dass die HotSpot JVM diesen Garbage Collector verwendet, wenn nicht explizit ein anderer gewählt wird. Tabelle 7 deutet jedoch an, dass der CMS Collector bezüglich der BMU in bestimmten Fällen bessere Ergebnisse erzielen könnte als die pausierenden Kollektoren. Deshalb bieten sich weitere Untersuchungen bezüglich der dargestellten und anderer Merkmale an, um die erhaltenen Ergebnisse zu sichern und auszubauen. Auch kann sich eine Einbeziehung des im Rahmen dieser Arbeit nicht betrachteten Garbage-First Garbage Collectors der HotSpot JVM als aufschlussreich erweisen.

Danksagung: Für die Betreuung meiner Masterarbeit und die Unterstützung dieses Papers möchte ich mich bei Prof. Dr. Johannes Waldmann, Dr. Andreas Both und Martin Gleditsch herzlich bedanken. Außerdem danke ich meinen Korrekturlesern Maik Riechert, Philip Loche, Christof Pieloth und André Winkler für ihre Unterstützung.

Literatur

- [CB01] Perry Cheng and Guy Blleloch. A Parallel, Real-Time Garbage Collector. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM SIGPLAN Notices 36(5), pages 125–136, Snowbird, UT, June 2001. ACM Press.
- [DFHP04] David Detlefs, Christine Flood, Steven Heller, and Tony Printezis. Garbage-First Garbage Collection. In David F. Bacon and Amer Diwan, editors, *4th International Symposium on Memory Management*, pages 37–48, Vancouver, Canada, October 2004.
- [DLM⁺78] Edsger W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, and E. F. M. Steffens. On-The-Fly Garbage Collection: An exercise in Cooperation. *Communications of the ACM*, 21(11):965–975, Nov 1978.
- [Hol06] David Holmes. Inside the Hotspot VM: Clocks, Timers and Scheduling Events - Part I - Windows. In *David Holmes' Weblog*. October 2006. https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks, abgerufen am 28. Juni 2012.
- [JHM11] Richard Jones, Antony Hosking, and Eliot Moss. *The Garbage Collection Handbook: The Art of Automatic Memory Management*. CRC Applied Algorithms and Data Structures. Chapman & Hall, August 2011.
- [Knu98] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 2 edition, March 1998. S. 417.
- [SBWC07] Jeremy Singer, Gavin Brown, Ian Watson, and John Cavazos. Intelligent Selection of Application-Specific Garbage Collectors. In Greg Morrisett and Mooly Sagiv, editors, *6th International Symposium on Memory Management*, pages 91–102, Montréal, Canada, October 2007. ACM Press.
- [Sch12] Michael Schmeißer. Testsysteme. <http://sharkofmetal.de/tsys.html>, July 2012.
- [SKB04] Sunil Soman, Chandra Krintz, and David Bacon. Dynamic Selection of Application-Specific Garbage Collectors. Technical Report 2004–09, UCSB, January 2004.
- [SKBL11] Jeremy Singer, George Kooor, Gavin Brown, and Mikel Luján. Garbage Collection Auto-Tuning for Java Mapreduce on Multi-cores. In Hans Boehm and David Bacon, editors, *10th International Symposium on Memory Management*, San Jose, CA, June 2011. ACM Press.
- [SMB04] Narendran Sachindran, J. Eliot B. Moss, and E. D. Berger. MC²: High-Performance Garbage Collection for Memory-Constrained Environments. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM SIGPLAN Notices 39(10), pages 81–98, Vancouver, Canada, October 2004. ACM Press.
- [Sun06] Sun Microsystems. Memory Management in the Java HotSpot™ Virtual Machine. http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf, April 2006.
- [Tho10] James Thomas. Incremental Parallel Garbage Collection. Technical report, Imperial College London, June 2010.
- [Ung84] David M. Ungar. Generation Scavenging: A Non-Disruptive High Performance Storage Reclamation Algorithm. In *ACM/SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, ACM SIGPLAN Notices 19(5), pages 157–167, Pittsburgh, PA, April 1984. ACM Press.
- [Zor89] Benjamin G. Zorn. *Comparative Performance Evaluation of Garbage Collection Algorithms*. PhD thesis, University of California, Berkeley, December 1989.

Generierung von Serviceverträgen auf Basis objektorientierter ereignisgesteuerter Prozessketten

Jörg Hartmann
jhartmann@informatik.uni-leipzig.de
Universität Leipzig, Studiengang Diplom Informatik

Zusammenfassung: Eine serviceorientierte Architektur (SOA) ist eine unternehmensweite IT-Architektur, bei der Anwendungen durch Komposition loser gekoppelter Services entstehen. Die Identifikation geeigneter Services steht am Beginn des Servicelebenszyklus. Sie stellt die Grundlage für eine fachliche Spezifikation der Services und bedingt damit den Erfolg einer SOA. Nach Vergleichen publizierter Serviceidentifikationsmethoden ist die Frage nach einer standardisierten Vorgehensweise unzureichend beantwortet. Diese Arbeit stellt eine modellbasierte Methode zur Serviceidentifikation vor. Auf Basis einer Anforderungsspezifikation wird das Servicedesign in einem Meet-In-The-Middle-Ansatz mit Hilfe objektorientierter ereignisgesteuerter Prozessketten und deren Verfeinerungen modelliert. Ein abschließender Schritt sieht die Generierung von Standards aus dem Servicedesign vor. Gekennzeichnet ist die Methode durch automatisierte Reflektion der Änderungsevolution zur Modellierungszeit.

1 Einleitung

Eine serviceorientierte Architektur (SOA) ist eine unternehmensweite IT-Architektur. Ziel einer SOA ist die vereinfachte Anpassung von Softwaresystemen und die flexible Abbildung von Geschäftsprozessen auf die IT [Mül07]. Anwendungen entstehen durch Komposition von lose gekoppelten – minimal abhängigen – Services [FZ09, S. 3]. Ein Service stellt eine autonome und wiederverwendbare Komponente mit bestimmter Funktionalität dar und wird über einen Servicevertrag spezifiziert [Erl07, S. 31f]. Die Bedeutung der Services innerhalb einer SOA forciert die Frage nach einer systematischen Identifikation geeigneter Services. Die Serviceidentifikation steht am Beginn des Servicelebenszyklus [Jos08, S. 171] und stellt die Grundlage einer fachlichen Spezifikation und Implementierung der Services [KK07]. In der Literatur wird die Serviceidentifikation als „[...] eine der herausforderndsten Aufgaben der Umsetzung einer SOA“ [Mül07] hervorgehoben.

Services können Top-Down auf Basis von Anforderungen und Geschäftsprozessen oder Bottom-Up durch Ableitung aus Altanwendungen und existierenden Services identifiziert werden [IB07]. Die Schwierigkeiten der Serviceidentifikation liegen u. a. an einer fehlenden standardisierten Vorgehensweise, notwendigem Business-IT-Alignment, Inkonsistenzen in den Geschäftsprozessmodellen sowie mangelnder Durchgängigkeit der Modellierung von der fachlichen zur technischen Ebene [Mül07, BG09]. Auch liegt die Konzentration existierender Methoden [BKO08, BG09] auf den funktionalen Eigenschaften der Services. Nicht-funktionale Eigenschaften werden meist nicht berücksichtigt [BKO08].

Daraus resultiert eine Lücke in der Spezifikation von Serviceverträgen [Erl07, S. 127].

Forschungsbedarf besteht darin, Identifikationsansätze zu stärker formalisierten und detaillierten Vorgehensweisen weiter zu entwickeln [BKO08] und nicht-funktionale Eigenschaften zu berücksichtigen. In dieser Arbeit wird eine Methode zur Ableitung technischer Serviceverträge vorgestellt. Die Methode beschreibt die Transformation fachlicher Anforderungen zu objektorientierten ereignisgesteuerten Prozessketten (oEPK), deren Verfeinerungen in Modellsichten und die Generierung technischer Serviceverträge.

Nach der thematischen Einführung wird der Begriff Servicevertrag in Kapitel 2 definiert und eine modellbasierte Sichtweise auf Serviceverträge vorgestellt. Die Kapitel 3 und 4 fokussieren den Aufbau und die Anwendung der Methode. Kapitel 5 schließt die Arbeit mit einer Zusammenfassung und ausstehenden Forschungsaufgaben ab.

2 Grundlagen

Ein Service ist eine „Komponente mit wohldefinierter und wiederverwendbarer Funktionalität“ [Mül07], bestehend aus einem Servicevertrag mit einer Menge von Schnittstellen und der Implementierung. Der Servicevertrag „[...] umfasst die vollständige Spezifikation eines Service und wird zwischen einem spezifischen [Service-] Anbieter und einem spezifischen [Service-] Nutzer geschlossen“ [Jos08, S. 37]. Die Implementierung erfüllt den Servicevertrag, bildet die Geschäftslogik ab und stellt benötigte Daten [KBS07, S. 78ff].

Ein Servicevertrag ist aus einem technischen Servicevertrag und einem Service-Level-Agreement zusammengesetzt [Erl09, S. 34f]. Der technische Servicevertrag wird in die abstrakte und konkrete Beschreibung partitioniert. Die abstrakte Beschreibung definiert die Schnittstelle – eine Beschreibung der Operationen, auszutauschende Nachrichten und Datenmodelle. Demgegenüber definiert die konkrete Beschreibung Implementierungs- und Aufrufinformationen. Abstrakte und konkrete Beschreibungen können mit nicht-funktionalen Anforderungen verknüpft werden [Erl09, S. 52]. Ein Service-Level-Agreement ist eine Vereinbarung über die Quality-of-Service [Jos08, S. 35]. Diese differenziert je Servicenutzer und kann deshalb in einem Identifikationsansatz nicht berücksichtigt werden. Aus diesem Grund liegt der Fokus der Arbeit auf technischen Serviceverträgen.

Technische Serviceverträge umfassen die vollständige Spezifikation eines Service. Deshalb ist eine Dekomposition des Informationsgehaltes in Modellsichten vorteilhaft. Birkmeier et al. schlagen zur Beschreibung von Services drei Modellsichten vor. Eine Datensicht zur Definition der Datenstrukturen, eine Funktionssicht zur Verknüpfung von Funktionen mit Daten sowie eine Prozesssicht zur Orchestrierung von Systemfunktionen [BKO08]. Die drei Sichten spezifizieren die abstrakte Beschreibung technischer Serviceverträge. Zur Spezifikation der konkreten Beschreibung schlägt Müller eine Applikations- und Informationsarchitektur vor [Mül07], welche in der Applikationssicht zusammengefasst werden. Nicht-funktionale Anforderungen der abstrakten und konkreten Beschreibungen werden in einer Polycysicht spezifiziert. Zum Abschluss dieses Abschnittes fasst Tabelle 1 die Sichten auf einen technischen Servicevertrag zusammen.

Modellsicht	Beschreibung	Notation
Datensicht	Aufbau, Struktur und Relationen der Daten.	Entity-Relationship-Diagramm UML-Klassendiagramm
Funktionssicht	Verknüpfung der Systemfunktionen mit Daten.	erweiterte ereignisgesteuerte Prozesskette (eEPK)
Kompositionssicht	Geschäftsprozesse mit zeitlich-logischer Abfolge der Systemfunktionen.	objektorientierte ereignisgesteuerte Prozesskette (oEPK) Business-Process-Modeling-Language (BPML)
Applikationssicht	Bindings und Endpoints der Services und benötigte IT-Systeme.	UML-Deploymentdiagramm
Polysicht	Nicht-funktionale Anforderungen.	WS-Policy Editor

Tabelle 1: Zusammenfassung der Modellsichten

3 Prozessgetriebene Generierung von Serviceverträgen

3.1 Architekturüberblick

Die Zielstellung fokussiert eine systematische Identifikation und konsistente Modellierung von Services. In der Literatur werden verwandte Probleme über eine Menge von Abstraktionsebenen gelöst. Bei diesem Ansatz wird die Funktionalität eines Systems mit Hilfe von Geschäftsprozessen illustriert, welche über weitere Ebenen iterativ zu Services verfeinert und mit technischen Anforderungen kombiniert werden [Mül07]. Der Vorteil einer Ebenenarchitektur liegt in der Trennung zwischen technologieabhängiger und -unabhängiger Modellierung [Pin07].

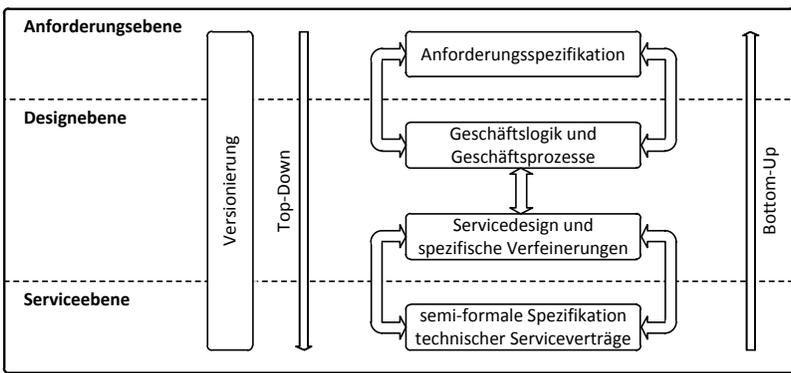


Abbildung 1: Architekturüberblick

Die in dieser Arbeit vorgestellte Methode verfolgt einen Meet-In-The-Middle-Ansatz – einer Kombination von Top-Down und Bottom-Up basierten Ansätzen [Mül07, BKO08]. Die Methode ist auf eine Drei-Ebenen-Architektur gestützt (vgl. Abbildung 1). Auf Ba-

sis einer Anforderungsspezifikation werden in der Anforderungsebene Geschäftsobjekte abgeleitet, welche in der Designebene zu Geschäftsprozessen vervollständigt werden. Anschließend werden die Prozesse über nutzerspezifische Verfeinerungen detailliert. Das auf diese Weise modellierte Servicedesign wird mit Hilfe eines technologieunabhängigen Servicemodells in der Serviceebene auf Standards projiziert. Ergänzt wird die Methode durch ein vertikales Modellrepository.

3.2 Architekturebenen

3.2.1 Anforderungsebene

„Soll der Service die Rolle der Brücke zwischen IT und Fachbereichen wahrnehmen, so muss dieser den Anforderungen der verschiedenen Stakeholder gerecht werden“ [Mül07]. Aus diesem Grund liegt der Methode eine Anforderungsspezifikation – die Menge aller Anforderungen an ein zu entwickelndes System [Par10, S. 32] – zugrunde. Die Anforderungsspezifikation sowie die Transformation der Anforderungen in Geschäftsobjekte ist Gegenstand der Anforderungsebene (vgl. Abbildung 2).

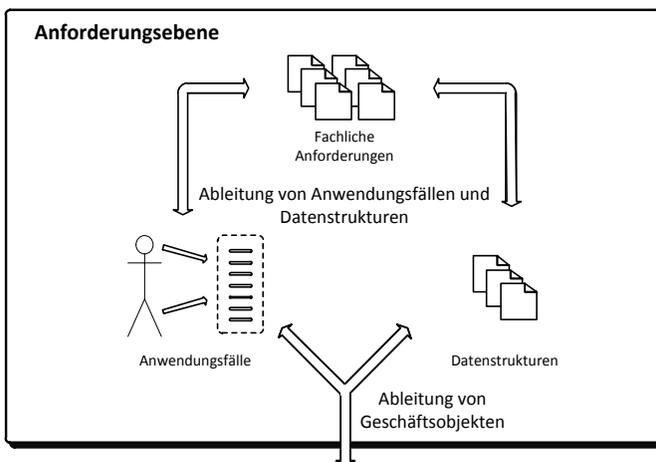


Abbildung 2: Anforderungsebene

Anforderungen beschreiben ein System über drei komplementäre Perspektiven: Daten-, Funktions- und Verhaltensperspektive [Poh07, S. 181ff], die durch nicht-funktionale Anforderungen ergänzt werden [Par10, S. 33]. Eine solche Dekomposition der Anforderungsspezifikation ähnelt den vorgestellten Sichten auf einen Servicevertrag. Daraus resultiert die Fragestellung, wie der Informationsgehalt der Anforderungen mit Hilfe von Geschäftsprozessen in der Kompositionssicht widerspiegelt werden kann. Für diesen Übergang ist eine weitere Abstraktion notwendig. In der Literatur wird dafür ein anwendungsfallorientiertes Vorgehen empfohlen. Use-Cases bilden die Grundlage der Geschäftsprozesse [Par10, S. 34] und erlauben eine Ableitung der groben Servicestruktur [TM07]. Aus diesen

Gründen bilden Use-Cases eine hinreichende Formalisierung natürlich-sprachlicher Anforderungen. Möglichkeiten zur automatisierten Übersetzung von natürlich-sprachlichen zu modellbasierten Anforderungen bietet die Disziplin des Text Minings [BK10].

Die transformierten Anforderungen sind in Form von Geschäftsobjekten weiter zu formalisieren. Diese können in die Kompositionssicht der Designebene integriert und dort über Geschäftsprozesse dargestellt werden. Geschäftsobjekte kapseln einen betriebswirtschaftlichen Zusammenhang in Form von Attributen und Methoden [SNZ97] und stellen auf diese Weise eine Komposition von Daten, Funktionen und Schnittstellen dar [NZ98]. Aufgrund dieser Eigenschaft der Geschäftsobjekte ist eine vertikale Transformation der Anforderungen hin zu Geschäftsobjekten ohne Informationsverlust durchführbar. Zu beachten ist die Reflektion möglicher Änderungen an der Anforderungsspezifikation, wodurch die Rücktransformation der Geschäftsobjekte über Use-Cases und Datenstrukturen zu Anforderungen berücksichtigt werden muss. Zudem ist vorwärtsgerichtete Nachverfolgbarkeit der durch die Geschäftsobjekte beschriebenen Anforderungen notwendig. Die Nachverfolgbarkeit erlaubt die Verfolgung der Anforderungen über die Architekturebenen bis zum Servicevertrag. Ein Mechanismus zur Nachverfolgbarkeit wird im Abschnitt zur Designebene eingeführt.

3.2.2 Designebene

Gegenstand der Designebene (vgl. Abbildung 3) ist die Modellierung des fachlichen Servicedesigns. Die Designebene integriert die Geschäftsobjekte der Anforderungsebene zu Geschäftsprozessen in der Kompositionssicht. Die Geschäftsprozesse werden über nutzerbasierte Verfeinerungen detailliert, wodurch das Servicedesign modelliert wird.

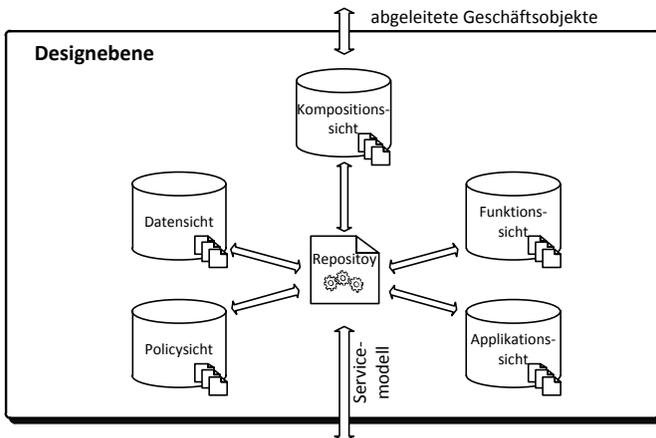


Abbildung 3: Designebene

Zur Modellierung der Kompositionssicht wird die objektorientierte ereignisgesteuerte Prozesskette (oEPK) verwendet. Die Methode ist eine semiformale Modellierungssprache, welche die EPK um objektorientierte Konzepte auf Basis der Unified Modeling Lan-

guage (UML) erweitert [SNZ97, NZ98]. Der Vorteil der oEPKs liegt in der intuitiven und weniger fehleranfälligen Modellierung [HJPN09]. Scheer et al. definieren den Geschäftsprozess im Sinne der oEPK als „[...] ereignisgesteuerte Bearbeitung von Geschäftsobjekten mit dem Ziel der Leistungserstellung“ [SNZ97]. Die Geschäftsobjekte der Anforderungsebene können somit direkt mit der oEPK-Notation visualisiert werden. oEPKs liegen sichtenorientierte Konzepte zugrunde [NFZ98], welche den Verfeinerungsgedanken unterstützen. Geschäftsobjekte zeigen bspw. einen Ausschnitt der Funktions- und Datensicht. Ausgehend von oEPKs der Kompositionssicht sind einzelne Modellelemente in weiteren Modellsichten zu konkretisieren. Die Detaillierung eines Modellelements über die Modellsichten wird als Verfeinerung – eine vertikale Modelltransformation [MvG06] – bezeichnet.

Verfeinerungen bedingen die Konzentration auf Nachverfolgbarkeit der Modellelemente und Konsistenz zwischen den Modellsichten. In beiden Fällen sind Relationen der Modellelemente von Bedeutung. Solche Relationen werden Korrespondenzregeln genannt [Wag09, S. 35]. Eine Korrespondenzregel bildet zwei Metamodellelemente heterogener Metamodelle sowie deren Attribute aufeinander ab. Zur Identifikation einer Korrespondenz auf Modellebene muss eine Attributabbildung als identifizierend markiert werden, da Korrespondenzen auch von den Attributwerten abhängen und nicht allein auf der Metaklasse beruhen. Das Zusammenspiel der Korrespondenzregeln und der Verfeinerungen erlaubt die Nachverfolgbarkeit der Modellelemente.

Änderungen der Modellelemente können mittels einer Modellsynchronisation auf korrespondierende Modellelemente übertragen werden. Dies betrifft das Hinzufügen und Löschen von Modellelementen sowie die Modifikation von Attributwerten. Tritt eine Änderung auf, können Inkonsistenzen durch Interpretation der Korrespondenzregeln behoben werden. Korrespondenzregeln und korrespondierende Modellelemente werden in einem Repository – ein System zur Verwaltung von Metadaten [MKM⁺09] – persistiert. Dies erleichtert die Modellnavigation und unterstützt die Synchronisation.

3.2.3 Serviceebene

Die Serviceebene thematisiert die Generierung technischer Serviceverträge aus dem modellierten Servicedesign. Zusätzlich wird die Ableitung von Geschäftsobjekten und Datenstrukturen aus bestehenden Serviceverträgen mit Hilfe des Bottom-Up-Ansatzes fokussiert. Der Rahmen der Serviceebene ist in Abbildung 4 illustriert.

Das in der Designebene modellierte Servicedesign wird in der Serviceebene auf technische Serviceverträge projiziert. Als Schnittstelle zwischen Design- und Serviceebene dient dabei ein technologieunabhängiges Servicemodell. Das Servicemodell zielt durch Abstraktion auf einfache Erweiterbarkeit hinsichtlich der zu generierenden Artefakte. Zur Vermeidung von Informationsverlusten zwischen den Ebenen muss das Servicemodell die Sichten auf einen Servicevertrag konzeptionell integrieren. Zur Realisierung des Servicemodells dient die Unified Service Description Language (USDL). Die USDL ist eine vollständige Beschreibungssprache zur Vereinheitlichung von Services, die von SAP Research im Rahmen des Forschungsprojektes TEXO – Infrastruktur für internetbasierte Dienste [THE12] – vorgestellt und vor der Standardisierung des W3C steht. USDL inte-

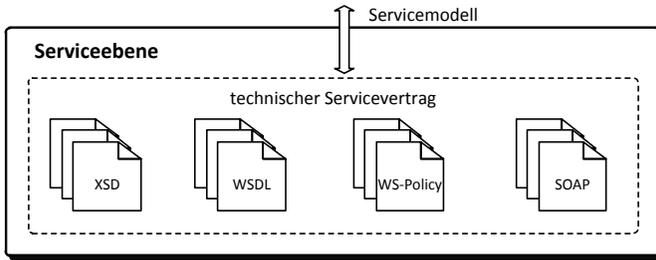


Abbildung 4: Serviceebene

griert die WS-* Standards [BKO11] und ist aus diesem Grund als Vorlage für das Servicemodell geeignet. Eine Modifikation der USDL ist hinsichtlich der Traceability-Informationen notwendig. Die im Repository persistierten Modellelemente müssen im Servicemodell und in den generierten Artefakten referenziert werden.

Zur Generierung der Serviceverträge muss das modellierte Servicedesign vom Repository in das Servicemodell transformiert werden. Die Transformation bedingt eine Abbildung der Kompositionssicht auf das Servicemodell. Aufgrund der Berücksichtigung des Bottom-Up-Ansatzes ist genauer eine bijektive Abbildung heranzuziehen. Die Geschäftsprozesse der Kompositionssicht kapseln die Funktionalitäten einzelner Services. Aus diesem Grund wird jede oEPK auf einen Service abgebildet. Geschäftsobjekte, deren Attribute und Methoden stellen die Schnittstellen samt Parameter und Operationen des Service dar. Die Abbildung der Verfeinerungen auf das Servicemodell ist von den verwendeten Notationen abhängig. Zur Integration der Verfeinerungsinformationen sind daher spezifische Modell-zu-Modell-Transformationen notwendig. Nach der Integration des Servicedesigns in das Servicemodell ist die Generierung technischer Serviceverträge durchführbar. Die Technologieunabhängigkeit des Servicemodells erlaubt die Generierung jedes gängigen Standards. Innerhalb dieser Arbeit werden folgende Standards verwendet: XML Schema Definition (XSD) zur Beschreibung der Typdeklarationen, Web Services Description Language (WSDL) zur Beschreibung der funktionalen Eigenschaften sowie WS-Policy und WS-PolicyAttachment zur Beschreibung nicht-funktionaler Eigenschaften.

4 Anwendung

4.1 Verfeinerung eines Geschäftsprozesses

Ziel dieses Abschnittes ist die Veranschaulichung der Verfeinerungen. Da es in dieser Arbeit nicht möglich ist, einen kompletten Geschäftsprozess zu verfeinern, liegt der Fokus auf einem Teilprozess um die Systemfunktion „AntragAusfüllen“. Abbildung 5 illustriert die Systemfunktion in oEPK-Notation. Ein Betriebsinhaber muss zur Anmeldung eines Gewerbes einen Antrag ausfüllen. Benötigt werden Angaben zur Betriebsstätte und zum Gewerbe selbst. Die Funktion wird vom Betriebsinhaber über einen Webserver genutzt.

Das vorliegende Geschäftsprozessmodell bildet die Kompositionssicht ab.

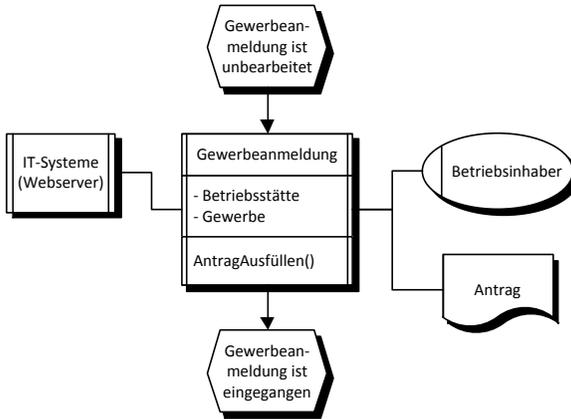


Abbildung 5: oEPK zur Systemfunktion „AntragAusfüllen“

Auf Basis der Kompositionssicht kann der Nutzer Verfeinerungen durchführen. Die Illustration des Verfeinerungsprinzips erfolgt an der Funktions- und Datensicht. Die Funktionssicht verfolgt das Ziel, Systemfunktionen mit Eingabe- und Ausgabedaten zu verknüpfen. Als Notation wird die erweiterte ereignisgesteuerte Prozesskette (eEPK) gewählt. Grundlage der Verfeinerung ist die Abbildung der Kompositionssicht auf die Funktionssicht mit Hilfe der Korrespondenzregeln. Für den Anwendungsfall sind drei Korrespondenzregeln notwendig: Geschäftsattribut \leftrightarrow Cluster, Dokument \leftrightarrow Cluster und Geschäftsmethode \leftrightarrow Funktion. Als identifizierendes Attribut wurde der Name der Modellelemente gewählt. Auf Basis der Korrespondenzregeln kann die Modellsynchronisation eine initiale eEPK erzeugen. Der Nutzer muss lediglich die Zuordnung der Parameter zur Funktion modellieren. Im Beispiel wurden die Geschäftsattribute „Betriebsstätte“ und „Gewerbe“ als Eingabedaten modelliert. Ausgegeben wird ein „Antrag“. Abbildung 6 zeigt die Verfeinerung der Funktionssicht.

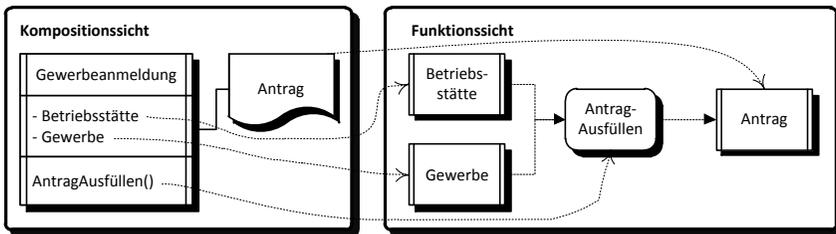


Abbildung 6: Verfeinerung der Funktionssicht

Die genaue Struktur der Eingabe- und Ausgabedaten wird in der Datensicht spezifiziert. Als Notation werden UML-Klassendiagramme verwendet. Grundlage der Verfeinerung bilden die Korrespondenzregeln Geschäftsattribut \leftrightarrow Klasse und Dokument \leftrightarrow Klasse.

Das identifizierende Attribut ist erneut der Name der Modellelemente. Auf Basis dieser Korrespondenzen werden die beiden Geschäftsattribute „Gewerbe“ und „Betriebsstätte“ sowie das Dokument „Antrag“ in Form von UML-Klassen verfeinert. Abbildung 7 zeigt die Verfeinerung der Datensicht.

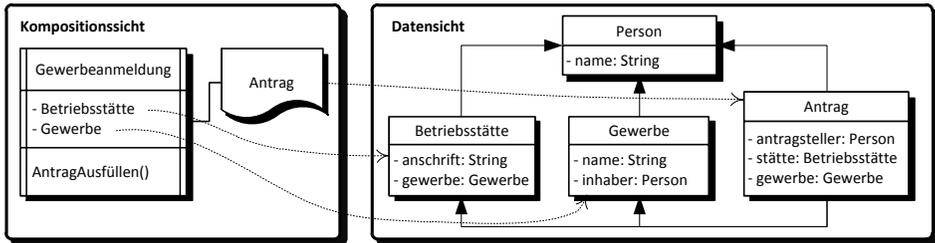


Abbildung 7: Verfeinerung der Datensicht

Das Zusammenspiel der Korrespondenzregeln und der Verfeinerungen erlaubt die Nachverfolgbarkeit der Modellelemente. Durch Interpretation der Korrespondenzregeln ist erkennbar, dass das Geschäftsattribut „Betriebsstätte“, die UML-Klasse „Betriebsstätte“ und das Cluster „Betriebsstätte“ die selbe Entität repräsentieren.

4.2 Generierung der Serviceverträge

Die Generierung technischer Serviceverträge stellt den letzten Schritt zur Umsetzung der Architektur dar. Zunächst ist das Servicedesign in das Servicemodell zu transformieren. Eine Abbildung der Modellsichten auf das Servicemodell ist in Abschnitt 3.2.3 skizziert. Anschließend wird das Servicemodell über eine Modell-zu-Text-Transformation auf Web-Service-Standards projiziert. Listing 1 zeigt einen Ausschnitt des generierten Servicevertrages. Im Beispiel existiert genau ein Service „Gewerbebeanmeldung“, welcher mit der oEPK aus Abbildung 5 korrespondiert. Das gleichnamige Geschäftsobjekt wird auf ein Interface des Service abgebildet. Über das Interface wird die Operation „AntragAusfüllen“ veröffentlicht. Geschäftsattribute werden als Nachrichten abgebildet. Deren Zuordnung zu Operationen ist der Funktionssicht zu entnehmen. Der Verfeinerung der Funktionssicht folgend, werden die Geschäftsattribute „Gewerbe“ und „Betriebsstätte“ als Eingabeparameter für den Request genutzt. Eine genaue Typdeklaration ist in Form eines XSD-Dokumentes referenziert. Die Struktur der dort beschriebenen Datentypen wurde in der Datensicht spezifiziert. Demzufolge ist der Typ „Betriebsstätte“ aus einer „Anschrift“ und einem „Inhaber“ zusammengesetzt.

Listing 1: Ausschnitt des Servicevertrages für die Systemfunktion „AntragAusfüllen“

```

1 <!-- type declaration Betriebsstaette -->
2 <xsd:element name="Betriebsstaette" type="tns:BetriebsstaetteType" />
3 <xsd:complexType name="BetriebsstaetteType">
4   <xsd:sequence>
5     <xsd:element name="Anschrift" type="tns:AnschriftType"
6       minOccurs="1" maxOccurs="1"/>

```

```
7         </xsd:element>
8         <xsd:element name="Inhaber" type="tns:PersonType"
9             minOccurs="1" maxOccurs="1">
10            </xsd:element>
11        </xsd:sequence>
12    </xsd:complexType>
13
14    <!-- message declaration AntragAusfuellen -->
15    <wsdl:message name="AntragAusfuellenRequest">
16        <wsdl:part name="Betriebsstaette" element="types:Betriebsstaette" />
17        <wsdl:part name="Gewerbe" element="types:Gewerbe" />
18    </wsdl:message>
19    <wsdl:message name="AntragAusfuellenResponse">
20        <wsdl:part name="Antrag" element="types:Antrag" />
21    </wsdl:message>
22
23    <!-- interface Gewerbeanmeldung -->
24    <wsdl:portType name="Gewerbeanmeldung">
25        <!-- policy declaration -->
26        <wsp:PolicyReference URI="#SecurityPolicy" />
27        <!-- exposed function AntragAusfuellen -->
28        <wsdl:operation name="AntragAusfuellen">
29            <wsdl:input message="tns:AntragAusfuellenRequest" />
30            <wsdl:output message="tns:AntragAusfuellenResponse" />
31        </wsdl:operation>
32    </wsdl:portType>
33
34    <!-- service Gewerbeanmeldung -->
35    <wsdl:service name="Gewerbeanmeldung">
36        <wsdl:port binding="tns:gewerbeanmeldungSOAP" name="Gewerbeanmeldung">
37            <soap:address location="localhost:8080" />
38        </wsdl:port>
39    </wsdl:service>
```

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Die Arbeit greift die Problemstellung um Serviceidentifikationsmethoden auf. Motiviert durch die Forschungsfrage, bisherige Identifikationsansätze zu einer formalisierten und detaillierten Vorgehensweise weiter zu entwickeln, wurde eine Methode zur Modellierung technischer Serviceverträge vorgestellt. Auf Grundlage einer Anforderungsspezifikation erfolgt mit Hilfe von Geschäftsprozessen in Form der oEPK und deren Verfeinerungen die Modellierung des Servicedesigns. Das Servicedesign wird automatisiert auf technische Serviceverträge abgebildet, wobei die Architektur auch die Rückrichtung ermöglicht. Änderungen können aufgrund der Nachverfolgbarkeit direkt in der Anforderungsspezifikation oder den Serviceverträgen reflektiert und Inkonsistenzen in den Geschäftsprozessmodellen durch Interpretation der Korrespondenzregeln aufgelöst werden. In diesem Sinn erfüllt die vorgestellte Methode einen Meet-In-The-Middle-Ansatz, bei dem Prozessdekomposition und die vorgestellten Modellsichten einbezogen werden.

5.2 Ausblick

Weitere Forschungsaufgaben sind zur Vervollständigung und Erweiterung der Architektur notwendig. Die Vervollständigung der Architektur adressiert die Versionierung. Die Integration eines Modellrepositorys zur Versionierung unterstützt die Nachverfolgung von Änderungen, ist relevant für die Anpassung von IT-Systemen bei Anforderungs- oder Schnittstellenänderungen und ermöglicht kollaboratives Arbeiten. Die Erweiterung der Architektur fokussiert die Implementierung der Services. In diesem Kontext kann die Architektur um eine vierte Ebene erweitert werden, welche den Fokus auf Implementierung und Tests der Services legt. Denkbar ist die Generierung von Quellcode.

Literatur

- [BG09] René Börner and Matthias Goeken. Methods for Service Identification: A Criteria-based Literature Review. In Daniel Moldt, Juan Carlos Augusto, and Ulrich Ultes-Nitsche, editors, *Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 76–84, Italy, May 2009. INSTICC PRESS.
- [BK10] Michael W. Berry and Jacob Kogan, editors. *Text Mining: Applications and Theory*. Wiley, Chichester, UK, 2010.
- [BKO08] Dominik Birkmeier, Sebastian Klöckner, and Sven Overhage. Zur systematischen Identifikation von Services: Kriterien, aktuelle Ansätze und Klassifikation. In *Modellierung betrieblicher Informationssysteme (MobIS)*, pages 255–272, 2008.
- [BKO11] Alistair Barros, Uwe Kylau, and Daniel Oberle. Unified Service Description Language 3.0 (USDL) Overview. Technical report, 2011. letzter Zugriff am 15.06.2012.
- [Erl07] Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, 2007.
- [Erl09] Thomas Erl. *Web service contract design and versioning for SOA*. Prentice Hall, Upper Saddle River, NJ, 2009.
- [FZ09] Patrick Finger and Klaus Zeppenfeld. *SOA und WebServices (Informatik Im Fokus)*. Springer, 2009.
- [HJPN09] Frank Hogrebe, Alexander Jürgens, Sven Pagel, and Markus Nüttgens. EPK-Varianten auf dem Prüfstand: Explorative Studie zur Gebrauchstauglichkeit von eEPK und oEPK. In Markus Nüttgens, Frank J. Rump, Jan Mendling, and Nick Gehrke, editors, *EPK 2009. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. 8. Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreise "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)"*, pages 195–212, Berlin, 2009.
- [IB07] Srikanth Inaganti and Gopala Krishna Behara. Service Identification: BPM and SOA Handshake. BPTrends, March 2007.
- [Jos08] Nicolai Josuttis. *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*. dpunkt.verlag, Heidelberg, 2008.

- [KBS07] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Best Practices für Serviceorientierte Architekturen - Einführung, Umsetzung, Praxis*. mitp, 1 edition, 2007.
- [KK07] Karsten Klose and Ralf Knackstedt. Serviceidentifikation für die Produktionsplanung eines mittelständischen Auftragsfertigers. In Hans-Peter Fröschle and Stefan Reinheimer, editors, *Serviceorientierte Architekturen*, number Heft 253 in Praxis der Wirtschaftsinformatik. HMD, 2007.
- [MKM⁺09] Sascha Müller, Walter Kuhn, Christian Meiler, Ilia Petrov, and Stefan Jablonski. Integratives IT-Architekturmanagement. In Ralf Reussner and Wilhelm Hasselbring, editors, *Handbuch der Software-Architektur*, pages 225–247. dpunkt.verlag, Heidelberg, 2 edition, 2009.
- [Mül07] Thomas Müller. Der Weg zum guten Service. In Gernot Starke and Stefan Tilkov, editors, *SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen*, pages 141–159. dpunkt.verlag, 2007.
- [MvG06] Tom Mens and Pieter van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005).
- [NFZ98] Markus Nüttgens, Thomas Feld, and Volker Zimmermann. Business Process Modeling with EPC and UML: Transformation or Integration? In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language – Technical Aspects and Applications, Workshop*, pages 250–261. Physica-Verlag, 1998.
- [NZ98] Markus Nüttgens and Volker Zimmermann. Geschäftsprozessmodellierung mit der objektorientierten Ereignisgesteuerten Prozeßkette (oEPK). In M. Maicher and H. J. Scheruhn, editors, *Informationsmodellierung – Branchen, Software- und Vorgehensreferenzmodelle und Werkzeuge*, pages 23–36, Wiesbaden, 1998.
- [Par10] Helmuth A. Partsch. *Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme*. Springer, 2 edition, 2010. ISBN: 978-3-642-05357-3.
- [Pin07] Dierk Pingel. Der SOA-Entwicklungsprozess. In Gernot Starke and Stefan Tilkov, editors, *SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen*, pages 187–200. dpunkt.verlag, 2007.
- [Poh07] Klaus Pohl. *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, 2 edition, 2007.
- [SNZ97] August-Wilhelm Scheer, Markus Nüttgens, and Volker Zimmermann. Objektorientierte Ereignisgesteuerte Prozesskette (oEPK) – Methode und Anwendung. *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Heft 141, 1997.
- [THE12] THESEUS. TEXO – Infrastruktur für internetbasierte Dienste, <http://theseus-programm.de/de/texo.php>, 2012. letzter Zugriff am 17.06.2012.
- [TM07] Dirk Tschierschke and Thilo Mezger. Evolution von zentralen Objektmodellen zu einer serviceorientierten Komponentenarchitektur. In Gernot Starke and Stefan Tilkov, editors, *SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen*, pages 161–168. dpunkt.verlag, 2007.
- [Wag09] Robert Wagner. *Inkrementelle Modellsynchronisation*. PhD thesis, Universität Paderborn, 2009.

Relevanzbasiertes Preprocessing für automatische Theorembeweiser

Mario Frank

Mario.Frank@Uni-Potsdam.de

Universität Potsdam, Diplom Studiengang Informatik

Zusammenfassung: Automatisches Theorembeweisen ist ein komplexes Forschungsgebiet mit vielen Anwendungsmöglichkeiten in der Praxis. Der Beweis eines Satzes (Problem oder auch Conjecture genannt) beinhaltet im Allgemeinen die Verwendung von Axiomen oder Axiom-artigen Formeln, die im Folgenden als Axiome bezeichnet werden. Gibt es große Mengen von Axiomen, dann kann die Zeitkomplexität für den Beweis enorm ansteigen. Hier wird ein relevanzbasierter Mechanismus vorgestellt, der zum Ziel hat, die für den Beweis potentiell notwendigen Axiome auszuwählen und dadurch die Axiom-Menge zu verringern ohne auf statistische Verfahren zurückzugreifen.

1 Einführung

Automatische Deduktion ist ein sehr wichtiges Forschungsgebiet und erlaubt z. B. den (teil-)automatischen Nachweis von Eigenschaften in Softwaresystemen und Kommunikationsprotokollen. Zu diesem Zweck werden seit vielen Jahren Theorembeweiser konzipiert und implementiert, die auf Basis eines formalen Kalküls die Gültigkeit von Sätzen unter Zuhilfenahme von Axiomen (bekanntermaßen wahren Sätzen, auch Theoreme genannt) beweisen sollen. Zu den bekanntesten zählen der iProver [Kor08], Vampire [HKV11], E [Sch02]/[Sch04] und leanCoP [OB03], die verschiedene Kalküle nutzen.

Probleme bereiten bei der Beweissuche jedoch große Mengen von Axiomen. Während ein Satz mit einigen wenigen Axiomen automatisch beweisbar ist, hat man unter Umständen einige Tausend oder sogar Millionen zur Verfügung. Will man den Satz in einer vorgegebenen Zeit beweisen, kann eine große Menge von Axiomen die Beweissuche nicht handhabbar machen, da die Anzahl der Beweispfade enorm ansteigt.

Um dieses Problem zu behandeln können Axiom-Auswahlssysteme wie SinE [HV11] genutzt werden, die die für den Beweis relevanten Axiome auswählen. Diese können Eigenschaften von Axiomen heranziehen, die für die Entscheidung über die Relevanz eines Axioms verwendbar sind. Das in diesem Dokument behandelte Konzept basiert auf direkter Verwendbarkeit von Axiomen für einen Beweis durch Konnektion ohne dabei auf statistische Informationen wie Häufigkeiten von Literalen zurückzugreifen und benötigt keine Formeln in Klausel-Normalform. Ein Beweis für einen Satz (Problem, oder auch Conjecture genannt) wird im Allgemeinen mit zwei verschiedenen Ansätzen geführt. Während der eine Ansatz die Conjecture aus den gegebenen Axiomen konstruiert (Konstruktionsbeweis), widerlegt der andere die Negation der Conjecture mithilfe der Axiome (Beweis

durch Widerlegung), wobei die Unerfüllbarkeit der negierten Conjecture bewiesen wird. Zum einfacheren Verständnis des Suchverfahrens wird im Folgenden der Ansatz des Beweises durch Widerlegung verwendet. Die ausgewählten Axiome könnten jedoch für beliebige Beweisverfahren genutzt werden, da die Analyse und Auswahl der Axiome an Kopien der Axiome vorgenommen und der Beweiser die Originale erhält. Betrachten wir einführend eine Vereinfachung eines Standardbeispiels der Logik (Tweety-Problem). Variablen beginnen in diesem Beispiel mit einem großen Buchstaben und Konstanten mit einem kleinen. Das Beispiel in Abbildung 1 zeigt eine Menge von prädikatenlogischen Formeln, wobei die erste unsere Conjecture ist und alle anderen die gegebenen Axiome. Die Conjecture sagt aus, dass Tweety fliegen kann und das erste Axiom bedeutet, dass für alle Instanzen von X (Variablenbelegungen) gilt, dass sie fliegen können, wenn sie Vögel, aber keine Pinguine sind. Alle anderen Axiome definieren die diversen Tiere.

Conjecture: $canFly(tweety)$ Axiome: $\forall X((bird(X) \wedge \neg penguin(X)) \Rightarrow canFly(X))$ $bird(tweety)$... weitere Vögel $penguin(tux)$... weitere Pinguine aber keiner namens Tweety $cat(meez)$... weitere Katzen und andere Tiere
--

Abbildung 1: Prädikatenlogische Formelmenge - Tweety-Problem

Man kann leicht nachvollziehen, dass alle Katzen und andere Tiere, die nicht Pinguine oder Vögel sind für den Beweis der Conjecture irrelevant sind, da in dieser und dem direkt verwendbaren Axiom nur Vögel, Pinguine und Instanzen die fliegen können vorkommen. Ein automatischer Theorembeweiser lädt jedoch alle Axiome und muss den Beweis mit allen Axiomen suchen. Da die Beweiser die Axiome meist in spezielle Normalformen wie die Klausel-Normalform transformieren, entsteht ein sehr großer Aufwand, der mit geeigneter Axiom-Auswahl verringert werden kann.

Im Folgenden betrachten wir einige notwendige theoretische Grundlagen, wie Prädikatenlogik, Normalformen, Unifikation und Konnektion.

2 Theoretische Grundlagen

Grundlagenwissen über Prädikatenlogik erster Ordnung sowie Logik allgemein wird in diesem Dokument vorausgesetzt, wird jedoch wieder aufgegriffen. Während die Syntax der Prädikatenlogik, Normalformen und die Unifikation sich auf die Ausführungen in [Hö09] stützen, wird für das Konzept der Konnektion [Bib92] herangezogen.

2.1 Prädikatenlogik erster Ordnung

Zuallererst betrachten wir die Syntax der Prädikatenlogik.

Definition (Syntax der Prädikatenlogik).

Eine prädikatenlogische Formel besteht aus Mengen von klein geschriebenen n -stelligen Prädikaten (z.B. $p(x, Y)$), Funktionen (z.B. $f(x)$), Konstanten (null-stellige Funktionen, z.B. x) und groß geschriebenen Variablen (z.B. Y). Diese werden auch **Terme** genannt.

Weiterhin enthalten prädikatenlogische Formeln logische Junktoren, wie die Negation (\neg), Konjunktion (\wedge), Disjunktion (\vee), Implikation (\Rightarrow) und Äquivalenz (\Leftrightarrow).

Alle Variablen können allquantifiziert ($\forall X (F)$ "Für alle Instanzen X gilt die Formel F ") oder existenzquantifiziert ($\exists X (F)$ "Es gibt eine Instanz X , für die die Formel F gilt") sein.

Die Argumente der Prädikate und Funktionen können Variablen, Konstanten oder auch n -stellige Funktionen mit $n > 0$ sein. Eine prädikatenlogische Formel kann zur automatisierten Verarbeitung in verschiedene Normalformen überführt werden die wir nachfolgend vorstellen.

2.2 Normalformen

Will man möglichst viele Informationen über die in einer Formel vorhandenen Prädikate sammeln, dann muss die Formel in Negationsnormalform überführt werden.

Definition (Negationsnormalform).

Eine Formel F ist in Negationsnormalform, wenn alle Negationen direkt vor Prädikaten stehen.

Jede prädikatenlogische Formel kann in eine semantisch äquivalente Formel in Negationsnormalform transformiert werden. Dazu müssen alle Implikationen ($A \Rightarrow B \equiv \neg A \vee B$) und Äquivalenzen ($A \Leftrightarrow B \equiv (\neg A \vee B) \wedge (A \vee \neg B)$) eliminiert werden und danach alle Negationen, die nicht vor Prädikaten stehen mithilfe der De Morgan'schen Gesetze verschoben werden. In der Formel vorhandene Quantoren werden dabei unter Umständen ebenfalls negiert ($\neg \forall A(F) \equiv \exists A(\neg F)$, $\neg \exists A(F) \equiv \forall A(\neg F)$). Als Beispiel überführen wir das erste Axiom aus der Einführung in die Negationsnormalform. Abbildung 2 zeigt die Transformation von der Ursprungsformel (1) in die Negationsnormalform (4).

- 1) $\forall X((bird(X) \wedge \neg penguin(X)) \Rightarrow canFly(X))$
- 2) $\forall X(\neg(bird(X) \wedge \neg penguin(X)) \vee canFly(X))$
- 3) $\forall X((\neg bird(X) \vee \neg \neg penguin(X)) \vee canFly(X))$
- 4) $\forall X(\neg bird(X) \vee penguin(X) \vee canFly(X))$

Abbildung 2: Transformation in Negationsnormalform

Nach dieser Transformation können wir für die Prädikate neben dem Namen, der Stelligkeit, der Prädikats-Argumente (Signatur) und deren Typen (Konstante, Funktion oder

Variable) auch die Polarität der Prädikate ermitteln.

Dabei haben negierte Prädikate die negative Polarität (-) und nicht negierte die positive Polarität (+). Diese Informationen können später in der Axiom-Auswahl sinnvoll genutzt werden. Weiterhin können prädikatenlogische Formeln in die Pränexnormalform und dann in die Skolem-Normalform transformiert werden, um nur noch einen Quantoren-Typen zu belassen. Beide Normalformen sollen hier vereinfacht definiert werden.

Definition (Pränexnormalform).

Eine prädikatenlogische Formel F ist in Pränexnormalform, wenn sie in Negationsnormalform vorliegt und die Form $Q_1X_1 \dots Q_nX_n(G)$ hat, wobei Q_i Quantoren und X_i Variablen sind und die Formel G keine Quantoren enthält.

Bei der Transformation müssen unter Umständen Variablen umbenannt werden, damit diese nicht durch verschiedenen Quantoren gebunden sind. Danach kann man die Formel in Skolem-Normalform transformieren.

Definition (Skolem-Normalform).

Eine prädikatenlogische Formel F ist in Skolem-Normalform, wenn sie in Pränexnormalform vorliegt und alle existenzquantifizierten Variablen durch eine einzigartige Funktion ersetzt wurden, sodass alle noch vorhandenen Variablen allquantifiziert sind. Eine Formel in Skolem-Normalform ist erfüllbarkeitsäquivalent zu der ursprünglichen Formel.

Die letzte vorzustellende Normalform ist die Klausel-Normalform.

Definition (Klausel-Normalform).

Eine Formel ist in Klausel-Normalform, wenn sie in Skolem-Normalform vorliegt und nur aus Disjunktionen von Konjunktionen oder Konjunktionen von Disjunktionen vorliegt.

Die Klausel-Normalform-Transformation führt oft zu einer exponentiellen Aufblähung der Ursprungsformel, weshalb der in diesem Dokument vorgestellte Ansatz auf die Klausel-Normalform verzichtet. Die Konnektions-Suche kann auch auf Nicht-Klausel-Normalformen durchgeführt werden, ist jedoch schwerer zu implementieren. Abschließend werden noch die Konzepte Unifikation, schwache Unifikation, und Konnektion eingeführt.

2.3 Unifikation

Seien $p(X)$ und $p(a)$ zwei einstellige Prädikate. Eine Substitution σ ordnet jeder Variable einen Term zu, der wiederum eine Variable, eine Konstante oder eine n -stellige Funktion mit $n > 0$ sein kann. Dann sind $p(X)$ und $p(a)$ genau dann unifizierbar, wenn es eine Substitution gibt, sodass beide Prädikate gleich werden. In diesem Beispiel müsste für die Variable X lediglich a eingesetzt werden.

Formal ist die Unifizierbarkeit wie folgt definiert:

Definition (Unifizierbarkeit).

Zwei Prädikate $p(s_1, s_2, \dots, s_n)$ und $p(t_1, t_2, \dots, t_n)$, wobei s_i und t_i Terme sind, sind genau

dann **unifizierbar**, wenn es für alle i eine Substitution gibt, sodass $\sigma(s_i) = \sigma(t_i)$ gilt.

In diesem Fall wird die Substitution σ auch **Unifikator** genannt.

Ein Unifikator σ_a ist der **allgemeinste Unifikator**, wenn es für jeden anderen Unifikator σ_b eine Substitution θ gibt sodass $\sigma_b = \theta(\sigma_a)$ gilt. Alle anderen Unifikatoren lassen sich also mit Anwendung einer Substitution auf den allgemeinsten Unifikator konstruieren.

Ein Unifikations-Algorithmus kann, selbst wenn er effizient implementiert ist, so zeitintensiv sein, dass die Axiom-Auswahl relativ lange dauert, da bei sehr großen Formelmengen auch viele Unifikationen ausgeführt werden müssen und der allgemeinste Unifikator berechnet wird. Außerdem kann die Unifikation in eine Endlosschleife geraten, wenn durch eine Substitution eine rekursive Substitutionsmöglichkeit entsteht. Um dies zu vermeiden, werden Unifikationen auch mit einem sogenannten Occurs-Check vorgenommen, der überprüft, ob ein Term, der eine Variable substituiert, diese Variable wiederum enthält. Jedoch führt der Occurs-Check zu einem starken Anwachsen der Komplexität der Unifikation. Ein effizienter Unifikations-Algorithmus ist das Martelli-Montanari Verfahren, das auch in [Bib92] beschrieben wird.

Da ein Preprocessing-Algorithmus möglichst schnell sein soll, um dem Theorembeweiser möglichst viel Zeit für den Beweis zu belassen, muss bei großen Axiom-Mengen auf eine Unifikation mit Berechnung des Unifikators verzichtet werden. In diesem Kontext ist viel mehr wichtig, ob zwei Prädikate unifizierbar sind oder nicht. Es sollen also möglichst viele nicht unifizierbare Prädikate erkannt werden. Die schwache Unifikation soll dies bewerkstelligen.

Definition (Schwache Unifizierbarkeit).

Zwei Prädikate $p(s_1, s_2, \dots, s_n)$ und $p(t_1, t_2, \dots, t_n)$, wobei s_i und t_i Terme sind, sind genau dann **schwach unifizierbar**, wenn es für alle i eine lokale Substitution gibt, sodass $s_i = t_i$ gilt.

Eine Substitution ist **lokal**, wenn die Variablenbelegungen der vorherigen Substitutionen nicht betrachtet werden.

Die schwache Unifikation erfolgt in zwei Schritten:

Zuerst wird ein Signatur-Kompatibilitäts-Check vorgenommen, der fehlschlägt, wenn s_i und t_i Konstanten mit $s_i \neq t_i$ sind oder einer der beiden Terme eine Konstante und der andere eine n -stellige Funktion mit $n > 0$ ist.

Diese Überprüfung kann in linearer Laufzeit zur Anzahl der Argumente der Prädikate erfolgen. Danach wird für alle s_i und t_i , wobei diese n -stellige Funktionen ($n > 0$) sind, geprüft, ob diese durch Substitution gleich werden können, ohne bisherige Unifikatoren zu betrachten.

Diese Unifikation ist deutlich schwächer als die gewöhnliche Unifikation, jedoch auch deutlich schneller als diese und reicht für die Suche nach Konnektionen.

2.4 Konnektion

In diesem Rahmen ist es unmöglich, das Konzept des Konnektionskalküls vollständig zu erläutern. Eine detaillierte Beschreibung findet man in [Bib92]. Hier soll zum Verständnis lediglich die Konnektion erläutert werden:

Definition (Konnektion).

Eine **Konnektion** ist ein Paar von unifizierbaren Prädikaten der Form $\{p(s_1, \dots, s_n), \neg p(t_1, \dots, t_n)\}$.

Sei p ein in der negierten Conjecture vorkommendes Prädikat und $\neg p$ ein in einem Axiom vorkommendes, dann ist die Konnektion ein Indiz für die Unerfüllbarkeit der negierten Conjecture und somit der Gültigkeit der Conjecture. Betrachten wir nun das Problem aus der Einführung, das hier negiert wurde:

$$\neg \text{canFly}(\text{tweety})$$

Das Prädikat lässt sich mit dem Prädikat $\text{canFly}(X)$ aus dem ersten Axiom in Abbildung 1 konnektieren. Dies reicht jedoch nicht aus, um die Conjecture zu beweisen, da nun X instanziiert werden muss, um das Axiom mit Fakten zu belegen.

Mit den bislang eingeführten Grundlagen lässt sich ein Konzept für die Axiom-Auswahl formulieren.

3 Konzept

Um eine Axiom-Auswahl vorzunehmen bedarf es einiger Vorarbeit. Alle Axiome sowie die Conjecture müssen analysiert werden. Bei der Analyse wird zumindest eine (eingeschränkte) Negationsnormalform-Transformation vorgenommen, um Informationen über die Polarität der Prädikate zu erhalten. Die Einschränkung betrifft die Auflösung der Äquivalenzen. Abbildung 3 zeigt eine prädikatenlogische Formel und ihre Transformation in Negationsnormalform. Schon dieses Beispiel zeigt, dass die Auflösung von Äquivalen-

$\begin{aligned} & \text{isEven}(\text{two}) \Leftrightarrow (\neg \text{isOdd}(\text{two})) \\ & (\text{isEven}(\text{two}) \Rightarrow (\neg \text{isOdd}(\text{two}))) \wedge ((\neg \text{isOdd}(\text{two})) \Leftarrow \text{isEven}(\text{two})) \\ & (\neg \text{isEven}(\text{two}) \vee (\neg \text{isOdd}(\text{two}))) \wedge (\neg(\neg \text{isOdd}(\text{two})) \vee \text{isEven}(\text{two})) \\ & (\neg \text{isEven}(\text{two}) \vee \neg \text{isOdd}(\text{two})) \wedge (\text{isOdd}(\text{two}) \vee \text{isEven}(\text{two})) \end{aligned}$

Abbildung 3: Negationsnormalform-Erzeugung mit Äquivalenz

zen zu einer starken Aufblähung der Formel führt. Das Beispiel zeigt auch, dass beide in die Äquivalenz einbezogenen Prädikate in der Normalform sowohl negativ, als auch positiv auftreten. Um die Aufblähung zu vermeiden und die Analyse zu beschleunigen, kann man eine neue Polarität (neutral) definieren, die aussagt, dass das Prädikat sowohl positiv, als auch negativ in der Formel vorkommt. Neutrale Prädikate wären also mit positiven, negativen und neutralen konnektierbar. Diese zusätzliche Polarität ändert auch die

Konnektierbarkeit von positiven und negativen Prädikaten. Sie können zusätzlich mit neutralen Prädikaten konnektiert werden. Da die Auflösung von Äquivalenzen ein Kopieren von Prädikaten bedingt, kann die Einführung einer zusätzlichen Polarität Zeit sparen. Jedoch hat dies den Nachteil, dass keine vollständige Skolem-Normalform erzeugt werden kann, die die Axiom-Auswahl einschränken kann. Dies ist darin begründet, dass bei der Bildung der eingeschränkten Negationsnormalform Quantoren, die in Äquivalenzen eingebettet sind, nicht negiert werden wenn die Äquivalenz negiert wird sodass die tatsächliche Art des Quantors nicht erkennbar ist und dieser daher nicht eliminiert werden darf.

3.1 Bilden von Prädikats-Klassen

Betrachten wir erneut unser anfängliches Beispiel aus Abbildung 1. Wie wir bereits festgestellt hatten sind nur Vögel, Pinguine und alle Instanzen, die fliegen können von Interesse. Um die Suche nach Konnektionen zu vereinfachen erscheint eine Bildung von Prädikats-Klassen als sinnvoll. Zu diesem Zweck kann man jedem Prädikat einen qualifizierenden Namen zuweisen, der sich aus der Polarität, dem Namen und der Stelligkeit zusammensetzt. Das Prädikat $\neg canFly(tweety)$ hätte beispielsweise den qualifizierenden Namen “ $\neg canFly/1$ “. Aggregiert man alle Prädikate mit demselben qualifizierenden Namen in einer Liste, dann erhält man dadurch eine Klasse von Prädikaten mit übereinstimmenden Grundeigenschaften.

Während man vorher zuerst nach dem passenden Prädikats-Namen, dann nach der passenden Stelligkeit und der passenden Polarität hätte suchen müssen, kann das nun in einem Suchschritt nach dem qualifizierenden Namen, also der Klasse bewerkstelligt werden. Dies hat zusätzlich den Vorteil, dass die Überprüfung der Anzahl der Argumente der Prädikate, die in der Unifikation vorgenommen wird, unnötig wird. Um die Suche nach Prädikats-Klassen zu beschleunigen, kann man Hash-Tabellen einsetzen, wobei der Schlüssel einer Zeile der qualifizierende Name der Prädikats-Klasse ist und der Wert die Liste der Referenzen auf die Prädikate.

3.2 Axiom-Auswahl

Um die Axiom-Auswahl zu verdeutlichen betrachten wir in Abbildung 4 das (gekürzte) Beispiel aus der Einführung, wobei alle Formeln in Negationsnormalform transformiert wurden.

Conjecture: $\neg canFly(tweety)$
 Axiome:
 $\forall X (\neg bird(X) \vee penguin(X) \vee canFly(X))$
 $bird(tweety)$

Abbildung 4: Tweety-Problem in Negationsnormalform

Hier muss eine Ersetzung der Variablen X vorgenommen werden, sodass die negierte Conjecture widerlegt werden kann. Hier reicht jedoch nicht der Schritt von der negierten Conjecture zum ersten Axiom. Wird die Variable X mit "tweety" belegt, dann heißt das, dass Tweety entweder kein Vogel oder ein Pinguin ist oder fliegen kann. Es entsteht jedoch eine Konnektion zwischen dem ersten Axiom und dem zweiten, sodass belegt werden kann, dass Tweety ein Vogel ist. Zusätzlich gibt es kein Prädikat, das aussagt, dass Tweety ein Pinguin ist. Daher können wir die Belegung von X als korrekt ansehen.

Das Beispiel zeigt, dass eine iterative Suche nach konnektierbaren Prädikaten notwendig ist, um den Beweis zu führen.

Die iterative Suche lässt sich mit folgendem Pseudocode in Abbildung 5 beschreiben, wobei `getPredicateClass` die Prädikats-Klasse zurück gibt, die zur Konnektion des Prädikats geeignet ist (passende Polarität, Stelligkeiten und Name). Hier wird zuerst eine nullte Iteration (eine Liste) erzeugt (Zeilen 3,4) in die die negierte Conjecture eingefügt wird. Dann wird in jeder Iteration für jede Formel der letzten Iteration für jedes Prädikat eine Konnektion gesucht, indem die konnektierbare Prädikats-Klasse geladen wird und das Prädikat mit jedem Element aus der Klasse schwach unifiziert wird (Zeile 12). Die Suche bricht ab, wenn in der letzten Iteration keine neuen Formeln dazugekommen sind, wenn also ein Fixpunkt erreicht wurde (Zeile 5).

```
1  int iteration = 1;
2  List<Iteration> ll;
3  ll.add(new Iteration());
4  ll.get(0).add(conjecture);
5  while (ll.get(iteration - 1).size() != 0){
6      ll.add(new Iteration());
7      for (Formula formula : ll.get(iteration)){
8          for (Predicate predicate : formula){
9              List possiblyConnectable =
10                 getPredicateClass(classTable, predicate);
11                 for (Predicate otherPredicate : possiblyConnectable){
12                     if (isWeakUnifiable(predicate, otherPredicate))
13                         ll.get(iteration).add(otherPredicate.getAxiom());
14                 }
15             }
16         }
17     startTheoremProver(ll);
18     iteration++;
19 }
```

Abbildung 5: Konnektions-Suche in Java-Syntax

Mit diesem Verfahren kann in jeder Iteration ein Theorembeweiser auf der aktuell ausgewählten Axiom-Menge gestartet werden und parallel die nächste Iteration berechnet werden. Wie der Theorembeweiser die erzeugte Axiom-Menge nutzt, hängt von dem spezifischen Beweiser ab. Die meisten erzeugen eine Klausel-Normalform. Einige erzeugen

dann eine Matrix und suchen dann einen Beweis(Konnektionsbeweis), andere führen einen Resolutionsbeweis. Das vorgestellte Konzept hat den Vorteil, dass, wenn der Beweis in Iteration n möglich ist, auch in Iteration $n + x$ möglich ist, da keine Axiome verloren gehen. Schleifen während der Suche können verhindert werden, wenn eine Überprüfung vorgenommen wird, ob ein Axiom bereits ausgewählt wurde.

4 Implementierung und Evaluation des Konzeptes

Eine Implementation des vorgestellten Konzeptes ist die Axiom Relevance Decision Engine (ARDE), die im Beta-Stadium bei der CASC J6¹ in der LTB-Division als Teil von leanCoP-ARDE eingereicht wurde. In dieser Weltmeisterschaft der automatischen Theorembeweiser müssen alle Beweiser die gleichen, zufällig aus der TPTP-Problembibliothek [Sut09] ausgewählten Probleme behandeln und in einer gegebenen Zeit (30-120 Sekunden) lösen.

Die LTB-Division enthält Probleme mit großen Axiom-Mengen, die mehrere Tausend, Hunderttausend oder Millionen Axiome enthalten, wobei für den Beweis nur ein Bruchteil, meist weniger als 1000 Axiome notwendig sind. Daher gibt es nur wenige, dafür optimierte Beweiser, die an dieser Division teilnehmen. Das eingereichte System bestand dabei aus zwei Teilen, die im Folgenden beschrieben werden.

4.1 leanCoP-ARDE

ARDE ist für die Auswahl der Axiome zuständig und ein prototypisches System, das in C++ (> 90 %) und Prolog implementiert ist. Die Teile, die in Prolog implementiert sind, werden via GNU Prolog² fest in den C++-Teil eingebunden, um eine möglichst hohe Geschwindigkeit zu erreichen. Die Geschwindigkeit der Implementierung basiert aber vor allem auf der starken Nutzung von Pointern, Hash-Tabellen und Multithreading (Mehrere Beweiser-Instanzen rechnen parallel). ARDE analysiert alle Axiome, bildet die Negation-normalformen und Prädikats-Klassen und führt die iterative Konnektions-Suche durch. Jedes neu ausgewählte Axiom wird in ein Hashset eingefügt, sodass in späteren Iterationen verhindert werden kann, dass ein Axiom erneut hinzugefügt wird. Prädikate, für die Konnektionen gesucht wurden, werden mit einem Index ihrer letzten Benutzung markiert ("benutzt in Problem Nummer i"), sodass immer nur Konnektionen für Prädikate gesucht werden müssen, deren Benutzungsindex kleiner als der aktuelle Problemindex ist. Am Ende einer jeden Iteration übergibt ARDE die bislang ausgewählten Axiome sowie die Conjecture an leanCoP[Ott08]. leanCoP ist ein schlanker, in Prolog implementierter Theorembeweiser, der als Beweiskalkül den Konnektionskalkül nutzt. Der Beweiser ist sehr effizient, hat jedoch den Nachteil, dass der Prolog-Interpreter beim Laden von mehr als ca. 10000 Axiomen überlastet sein kann.

¹<http://www.tptp.org/CASC/J6/>

²<http://www.gprolog.org/>

4.2 Ergebnis des Wettbewerbs

Mit Nutzung von ARDE konnte leanCoP Beweise führen, die ohne ARDE nicht in der Zeit möglich waren (in der Isabelle-Kategorie zwei), da die Axiom-Menge zu groß war. In der Isabelle-Kategorie (ca. 50-5000 Axiome), einem Teil der LTB-Division konnte leanCoP-ARDE zwei der seit Jahren in der LTB-Division teilnehmenden oder solche Systeme nach-nutzenden anderen Beweiser schlagen und hat im Gegensatz zu diesen auch die Beweisausgaben zu den 17 bewiesenen Problemen erzeugt, was zeitintensiv sein kann. Die geschlagenen Systeme konnten 16 beziehungsweise 7 Beweise führen. In der MRZ@ Turing-Kategorie, einem anderen Teil der CASC, hatte leanCoP-ARDE den besten SOTAC-Wert, auch wenn nur 7 von 400 Problemen in der Zeit gelöst wurden. Je höher der SOTAC-Wert ist, desto mehr Beweise konnten geführt werden, die kein anderer Beweiser führen konnte. Fünf der Sieben Probleme waren von leanCoP-ARDE, aber nicht einmal von den besten Beweisern lösbar. Es zeigte sich dabei, dass eine Anzahl von bis zu fünf Iterationen ausreichend war um einen Beweis zu führen während bis zu 10 durchgeführt wurden.

Es wurden auch einige Schwächen in der Implementierung von ARDE aufgetan, die beseitigt werden müssen. Beispielsweise wurden Gleichheiten, die in den Formeln enthalten sind, in der Axiom-Auswahl nicht betrachtet. Dadurch kam es zu leeren Axiom-Mengen, wenn die Conjecture nicht konnektiert werden konnte ohne Gleichheiten zu betrachten. Große Probleme machte die Tatsache, dass Axiom-Dateien, die nicht in allen Problemen verwendet werden und deshalb nicht als global markiert waren später mehrmals neu geladen werden mussten und so viel Zeit verloren ging, dass teilweise nicht einmal die erste Iteration vor Zeitablauf beendet werden konnte. Auch wurde die Anzahl der Axiome, die an leanCoP weitergegeben werden nicht beschränkt, sodass teilweise deutlich mehr als 10000 Axiome ausgewählt wurden. Es zeigte sich auch, dass die Nutzung der neutralen Polarität zu einer zu schwachen Einschränkung der Axiom-Menge führt und eine Negationsnormalform mit Auflösung der Äquivalenzen und Skolemisierung sinnvoller erscheint. Diese schwerwiegenden Schwächen führten auch dazu, dass leanCoP-ARDE in der Gesamt-Division knapp den letzten Platz einnahm. Das Ergebnis zeigt dennoch, dass ein solches Konzept zur Axiom-Auswahl möglich ist und auch tatsächlich zu Verbesserungen führt, jedoch auch Optimierungsbedarf da ist.

5 Andere Ansätze

Andere in der Division teilnehmende Beweiser nutzen einen anderen Ansatz zur Axiom-Auswahl. Dieser sammelt Informationen über alle in den Axiomen und der Conjecture vorkommenden Literale (Prädikats-, Funktions- und Konstanten-Namen) und ermittelt, in wie vielen der Formeln die Literale enthalten sind. Darauf basierend werden Relationen zwischen Formeln erzeugt, die aussagen, dass gewisse Literale andere Formeln und auch die Conjecture irgendwie definieren, wobei besonders häufige Literale benachteiligt werden und in der Conjecture enthaltene favorisiert werden. Auf diesen Relationen basierend werden dann die Axiome ausgewählt, die für den Beweis genutzt werden sollen. Der Ansatz wurde in SInE [HV11] implementiert und wurde von Theorembeweisern wie Vam-

pire und E integriert. Der Ansatz funktionierte in den letzten Jahren außergewöhnlich gut, ist jedoch sehr stark abhängig von den Literalen und betrachtet weder Unifizierbarkeit, noch Polaritäten. Der Ansatz von ARDE erlaubt im Gegenzug prinzipiell die Nutzung der Literal-Häufigkeit um stärkere Einschränkungen vorzunehmen, wenn der Grund-Ansatz nicht stark genug scheint.

Der Ansatz von ARDE ähnelt dem in [PY03] vorgestellten, das auch Konnektionen sucht und darauf basierend einen Relevanz-Wert für Axiome erzeugt, unterscheidet sich aber vor allem dadurch, dass eine Klausel-Normalform vorausgesetzt wird, deren Erzeugung jedoch äußerst zeit- und speicherintensiv sein kann. Der in [RPS09] vorgestellte Ansatz ordnet die Axiome nach ihrer Relevanz wobei die Relevanz sich unter anderem daraus ergibt, ob diese gemeinsame Prädikate und Funktionen haben. Die in [Pud07] und [SP07] behandelten Verfahren führen eine semantische Axiom-Auswahl durch, indem sie Interpretationen und Modelle für die Formeln bilden, setzen jedoch auch eine Klausel-Normalform voraus.

6 Aktueller Stand und Ausblick

Die Implementierung enthielt, wie erwähnt, einige Fehler und Schwächen, die aktuell ausgebessert werden. Außerdem werden aktuell Konzepte wie die Bildung einer vollständigen Negationsnormalform und Skolem-Normalform getestet, sowie Optimierungen bezüglich der Geschwindigkeit von ARDE vorgenommen. Aktuelle Tests ergaben eine weit schnellere Erzeugung der Iterationen (im Allgemeinen deutlich mehr als doppelt so schnell, meist nur wenige Sekunden) im Vergleich zum eingereichten System mit deutlich stärkerer Einschränkung der Axiom-Menge. Die Beschleunigung entstand dabei unter anderem durch Entfernung von unnötig gewordenen String-Manipulationen. Die Behandlung der Gleichheiten ist aktuell in der Konzeptions-Phase. Die aktuelle Repräsentation der Axiome ist verlustbehaftet bezüglich der Junktoren und erschwert die Möglichkeit zur Einschränkung, was auch in Zukunft behandelt werden soll. Außerdem ist eine Anpassung des Konzeptes für intuitionistische Logik sowie Modallogik in Vorbereitung.

7 Danksagung

Die Implementierung des in Prolog geschriebenen Teiles von ARDE wurde mit Hilfe der Expertise von Jens Otten, dem Autor von leanCoP bewerkstelligt, der mich auch mit Verbesserungsvorschlägen bei der Fertigstellung dieses Dokumentes unterstützt hat. Für seinen, sowie den Einsatz der Reviewer möchte ich mich herzlichst bedanken.

Literatur

[Bib92] Wolfgang Bibel. *Deduktion: Automatisierung der Logik*. Oldenbourg, 1992.

- [HKV11] Krystof Hoder, Laura Kovács, and Andrei Voronkov. Invariant Generation in Vampire. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 60–64. Springer, 2011.
- [HV11] Kryštof Hoder and Andrei Voronkov. Sine Qua non for large theory reasoning. In *Proceedings of the 23rd international conference on Automated deduction*, CADE'11, pages 299–314, Berlin, Heidelberg, 2011. Springer-Verlag.
- [Hö09] Steffen Hölldobler. *Logik und Logikprogrammierung, Band 1: Grundlagen*. Kolleg Synchro, 2009.
- [Kor08] K. Korovin. iProver – An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, (IJCAR 2008), volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.
- [OB03] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *Journal of Symbolic Computation*, 36(1-2):139–161, 2003.
- [Ott08] Jens Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions). In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR '08, pages 283–291, Berlin, Heidelberg, 2008. Springer-Verlag.
- [Pud07] Petr Pudlak. Semantic Selection of Premisses for Automated Theorem Proving. In Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors, *ESARLT*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
- [PY03] David A. Plaisted and Adnan H. Yahya. A relevance restriction strategy for automated deduction. *Artif. Intell.*, 144(1-2):59–93, 2003.
- [RPS09] Alex Roederer, Yury Puzis, and Geoff Sutcliffe. Divvy: An ATP Meta-system Based on Axiom Relevance Ordering. In Renate A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 157–162. Springer, 2009.
- [Sch02] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.
- [Sch04] S. Schulz. System Description: E 0.81. In D. Basin and M. Rusinowitch, editors, *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.
- [SP07] Geoff Sutcliffe and Yury Puzis. SRASS - A Semantic Relevance Axiom Selection System. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 2007.
- [Sut09] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

Untersuchung von Beschleunigungsverfahren für den Dijkstra-Algorithmus im Zusammenhang mit multimodalem Routing

David Georg Reichelt
davidgeorg_reichelt@yahoo.de
Universität Hamburg, Master Studiengang Informatik

Zusammenfassung: Vor dem Hintergrund neu aufkommender Verkehrsmittel wie dem Bike- und dem Carsharing stellt sich die Frage der Wegsuche in multimodalen Netzen neu, da herkömmliche Verkehrsmittel wie bspw. der ÖPNV mit neuen Verkehrsmitteln kombiniert werden. Die vorliegende Arbeit untersucht zwei Beschleunigungsverfahren für den Dijkstra-Algorithmus, das ALT-Verfahren und das reichweitenbasierte Pruning. Es werden die Ergebnisse der Untersuchung der Algorithmen anhand eines prototypischen Werkzeugs dargestellt.

1 Einleitung

Durch das Aufkommen großer Bike- und Carsharingangebote verändert sich unsere Mobilität nachhaltig: für Nutzer von öffentlichem Nahverkehr wird es möglich, ihre täglichen Strecken in Kombination mit anderen Verkehrsmitteln zurückzulegen und so ggf. Zeit zu sparen und, im Fall des Bikesharings, Lebensqualität zu gewinnen. Durch große Anbieter wie nextbike, CallABike und StadtRAD ist es bereits heute in vielen deutschen Großstädten möglich, öffentlichen Nahverkehr mit Bikesharing zu kombinieren [vS09]. Dies stellt Anbieter von Verkehrsinformationssystemen vor eine neue Herausforderung: nachdem zuvor nur kürzeste Wege in einem Netz, d.h. dem Netz des öffentlichen Personennahverkehrs ggf. mit vorgelagertem und nachgelagertem Fußwegrouting zu Haltestellen, ermittelt wurden, wird es nun notwendig, verschiedene Netze zu integrieren. Um das Vertrauen der Kunden in die Wegsuche hoch zu halten, ist es sinnvoll, exakte kürzeste Wege zu berechnen, statt sie zu approximieren. Andernfalls würde das Vertrauen in die Wegsuche sinken, sobald Kunden selbst Wege finden, die kürzer als die von der Wegsuche gefundenen Wege sind.

In Zusammenarbeit mit HBT¹, dem Anbieter des Fahrplanauskunftsystems des Hamburger Verkehrsverbunds (HVV), wurde untersucht, welche Lösungsansätze es zur gibt, um kürzeste Wege in multimodalen, integrierten Netzen zu ermitteln. Ziel ist eine spätere Verwendung in einem produktiv einsetzbaren, multimodalen Routingwerkzeug.

Die Integration verschiedener Netze ist bereits durch die DELFI-Studie [GiK97] im Zusammenhang mit der Integration der Netze verschiedener Verkehrsinformationssystemanbieter untersucht wurden. Hier wurde die Lösung gewählt, an vordefinierten Übergangspunkten

¹Hamburger Berater Team, <http://www.hbt.de>

Teilsuchen in den jeweiligen Systemen durchzuführen. Da HBT Zugriff auf Informationen über den ÖPNV in der Metropolregion Hamburg besitzt und Daten über Rad- und Autowege aus OSM² ausgelesen werden können, ist eine Integration aller notwendigen Netzdaten im Rahmen dieser Arbeit möglich. Aus diesem Grund wurde in dieser Arbeit die Lösung gewählt, verschiedene Netze in einem Werkzeug zu integrieren.

Die vorliegende Arbeit behandelt die Untersuchung verschiedener Algorithmen anhand eines Werkzeugs, das verschiedene Netze integriert und auf diesen Netzen kürzeste-Wege-Suchen durchführt. Dabei werden nicht-zeitbehaftete Netze untersucht, d.h. keine Netze mit Abfahrtszeiten wie bspw. das Regionalbahnnetz. Da Netze mit Abfahrtszeiten generell als nicht-zeitbehaftete Netze darstellbar sind [GiK97], kann das erstellte Werkzeug mit einer entsprechenden Netzvorverarbeitung auch kürzeste Wege in einem ÖPNV-Netz finden. Gegenüber Algorithmen für zeitbehaftete Netze ist dieses Verfahren allerdings weniger effizient.

Zuerst werden aktuelle Algorithmen beschrieben. Die Kenntnis grundlegender Algorithmen wie des Dijkstra- und des A*-Algorithmus wird dabei vorausgesetzt. Anschließend werden verwandte Arbeiten beschrieben. Danach folgt eine Darstellung von Laufzeitmessungen, um die vorhandenen Algorithmen zu vergleichen. Hierbei wird auf Rückschlüsse auf die Algorithmen, die bei der Implementierung entstanden, eingegangen. Abschließend wird eine Zusammenfassung und ein Ausblick auf weitere Arbeiten gegeben.

Falls eine formale Beschreibung notwendig ist, soll der kürzeste Weg in einem Graph $G = (V, E)$ gesucht sein. Der Graph bestehe aus der Menge seiner Knoten V und der Menge seiner Kanten E ($E \subseteq V \times V$). Weiterhin soll $l : E \rightarrow \mathbb{R}$ die Gewichtsfunktion des Graphen sein. $d : V \times V \rightarrow \mathbb{R}$ gibt die Länge des kürzesten Wegs zwischen zwei Knoten an. Für jeden Knoten existiere eine Abbildung $v : V \rightarrow V$, die den Knoten auf seinen Vorgänger im aktuellen Pfad abbildet. Es existiere weiterhin eine Funktion $t : V \rightarrow \mathbb{R}$, die jeden Knoten auf seine derzeitige gefundene Distanz zum Startknoten abbildet. Das Relaxieren von k bezeichnet den Schritt, bei dem für jeden Knoten n am Ende einer ausgehenden Kante des Knotens k falls $t(n) > t(k) + l(k, n)$ gilt $t(n) = t(k) + l(k, n)$ und $v(n) = k$ gesetzt wird. Der Dijkstra-Algorithmus bezeichnet den Algorithmus, der eine Menge M besitzt, diese mit $M = s$ (s sei der Startknoten) initialisiert und in jeder Iteration den Knoten k aus M mit minimalem $t(k)$ relaxiert.

2 Algorithmen

In den letzten 15 Jahren sind im wesentlichen fünf Beschleunigungstechniken für kürzeste-Wege-Algorithmen bearbeitet wurden: ALT, Reichweitenbasiertes Pruning, Highway-Hierarchien, Transitknotenrouting und Bogen-Flags. Aus Kapazitätsgründen musste für diese Arbeit eine Auswahl getroffen werden. Dabei sind die drei erstgenannten Techniken zur Bearbeitung ausgewählt wurden. Hier werden aus Platzgründen die ersten beiden Techniken dargestellt. Davor soll eine kurze Begründung gegeben werden, wieso Transitknotenrouting und Bogen-Flags nicht ausgewählt wurden. Außerdem soll die Grundidee

²OpenStreetMap, Projekt bei dem Freiwillige Kartendaten erfassen, <http://www.openstreetmap.org/>

der Highway-Hierarchien dargestellt werden.

Transitknotenrouting (engl. *Transit-Node Routing*) [BFM⁺06] basiert darauf, dass bestimmte Knoten in vielen langen kürzesten Wegen vorkommen. Diese Knoten, sogenannte *Transitknoten*, werden ermittelt, und anschließend werden zwischen allen Transitknoten kürzeste Wege ermittelt. Bei einer kürzesten-Wege-Suche werden in der Umgebung von Start- und Zielpunkt die naheliegenden Transitknoten bestimmt. Anschließend werden die Summen der Wege vom Start- zum Transitknoten, von Transitknoten zu Transitknoten und vom Transitknoten zum Zielknoten berechnet. Wenngleich dieses Verfahren sehr effizient ist, eignet es sich für längere Wege eher als für den Einsatz im dicht besiedelten Stadtverkehr. Aus diesem Grund wird Transitknotenrouting im Folgenden nicht behandelt.

Grundidee der Bogen-Flags (engl. *Arc-Flags*) [HKMS09] ist es, den Graph in Regionen einzuteilen, und für jede Region R zu markieren, ob eine Kante (u, v) auf dem kürzesten Weg von u zu einem Knoten in R liegt. Für jede Suche mit einem Zielknoten in R müssen lediglich die Kanten betrachtet werden, die auf dem Weg nach R liegen. Bogen-Flags sind speicheraufwändig, da für jede Region und jede Kante gespeichert werden muss, ob sie auf einem kürzesten Weg in die betreffende Region ist. Eine höhere Anzahl an Regionen beschleunigt das Verfahren, bringt aber einen mit Kantenzahl proportional steigenden Speicheraufwand mit sich. Da ein sich aus Kantenzahl und Regionenanzahl zusammensetzender Speicheraufwand sehr schnell steigt, wurden Bogen-Flags in der vorliegenden Arbeit nicht behandelt.

Highway-Hierarchien basieren darauf, dass reale Routingnetze hierarchisch gegliedert sind: kurze Wege werden auf Landstraßen zurückgelegt, weitere auf Bundesstraßen und noch weitere auf Autobahnen. Gleiches gilt für das ÖPNV-Netz mit Regionalbussen, Metrobussen und den Schnellbahnen. Um auf einem hierarchischen Netz kürzeste Wege zu finden, durchsucht man einen bestimmten Bereich in der Nachbarschaft von Start- und Zielknoten auf derselben Hierarchieebene und wechselt anschließend in die höhere Hierarchieebene. Dieses Verfahren wiederholt man mit den Eintrittspunkten in höheren Ebenen, bis eine Abbruchbedingung erfüllt ist. Dieses Verfahren wurde in [SS05] formalisiert.

2.1 ALT

ALT ist eine Beschleunigungstechnik für den Dijkstra-Algorithmus, die auf dem A*-Algorithmus, Landmarken und der Dreiecksungleichung (Triangle-Inequality) basiert. Die Grundidee, die von Goldberg et al. in [GH05] beschrieben wurde, ist, dass mithilfe der Dreiecksungleichung und einer Vorverarbeitung in jedem Graphen domänenunabhängige untere Schranken erstellbar sind. Diese können anschließend im A*-Algorithmus angewandt werden. Die Domänenunabhängigkeit ist im Kontext dieser Arbeit besonders relevant, da durch die Kombination von Netzen die einheitliche Domäne verloren geht.

Für ALT wird nach [GH05] eine Menge von Landmarken ausgewählt³. Anschließend werden kürzeste Wege von allen Landmarken zu allen Knoten und von allen Knoten zu allen

³Um Landmarken, die möglichst effiziente kürzeste-Wege-Anfragen ermöglichen, zu generieren, existieren verschiedene Heuristiken. Diese sind in [GH05] dargestellt.

Landmarken bestimmt. Wenn eine Anfrage nach einem kürzesten Weg auszuführen ist, dann wird für jeden Knoten die untere Schranke auf Basis der Abstände zu den Landmarken berechnet. Hierbei nutzt das ALT-Verfahren die Dreiecksungleichung: angenommen, es existieren 3 Knoten A , B und L wie in Abbildung 1 dargestellt. Nach der Dreiecksungleichung, die auch in kürzesten Wegen gilt, ist der Weg von A nach L durch die Summe des Weges von A nach B und von B nach L nach oben beschränkt: $d(A, L) \leq d(A, B) + d(B, L)$. Damit gilt auch $d(A, L) - d(B, L) \leq d(A, B)$. $d(A, L) - d(B, L)$ ist also eine untere Schranke für den kürzesten Pfad von A nach B , also von einem Knoten zum Ziel. Ähnlich lässt sich verfahren, wenn L eine Landmarke ist und die Wege von L zu einem Wegknoten und zum Zielknoten bekannt sind: dann ist nach der Dreiecksungleichung $d(L, B) \leq d(L, A) + d(A, B)$, und damit $d(L, B) - d(L, A) \leq d(A, B)$. $d(L, B) - d(L, A)$ ist also eine untere Schranke für $d(A, B)$. Sind mehrere Landmarken vorhanden, so ist das Maximum über alle so gefundene Werte eine untere Schranke.

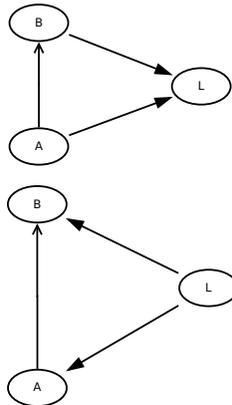


Abbildung 1: Graph mit Ziel-Knoten B , aktuellem Knoten A und Landmarke L . Pfeile repräsentieren hier Pfade.

In [GH05] werden verschiedene Beschleunigungsmöglichkeiten für das ALT-Verfahren vorgeschlagen, die darauf basieren, einen bidirektionalen A*-Algorithmus mit bestimmten Rahmenbedingungen laufen zu lassen. Die probabilistische Analyse des bidirektionalen Dijkstra-Algorithmus zeigte, dass die bidirektionale Version des Dijkstra-Algorithmus schneller als die unidirektionale Version ist [Poh69]. Die folgende Darstellung stützt sich auf die Definition des Dijkstra-Algorithmus aus [GH05] (Seite 3), in der μ die Länge des kürzesten gefundenen Pfads repräsentiert (und in der $\mu = \infty$ solange kein Pfad gefunden wurde). Die bidirektionale Dijkstra-Suche wird dabei abgebrochen, sobald eine Suchrichtung einen Knoten aus M entfernt, der in der anderen Richtung bereits in M eingefügt wurde.

Diese Abbruchbedingung ist auf den A*-Algorithmus nicht übertragbar, wenn man die gemäß ALT berechneten unteren Schranken nutzt, denn dies führt dazu, dass beide Suchen unterschiedliche Längenfunktionen benutzen [GH05]. Um dieses Problem zu umgehen, gibt es nach Goldberg et al. zwei Möglichkeiten: den symmetrischen und den konsistenten

Ansatz. Diese werden im Rest des Kapitels nach [GH05] dargestellt.

Beim symmetrischen A*-Algorithmus wird die Suche abgebrochen, sobald $\mu \leq k(v) = d_s(v) + \pi_t(v)$ für einen gefundenen Knoten gilt⁴. Durch diese Abbruchbedingung kann garantiert werden, dass die Suche läuft, bis ein kürzester Weg gefunden ist. Beim konsistenten Ansatz werden die Funktionen für die untere Schranke so angepasst, dass die Abbruchbedingung des Dijkstra-Algorithmus angewandt werden kann. Hierfür muss für die Funktion der unteren Schranke der Vorwärtsrichtung p_t und die untere Schranke der Rückwärtsrichtung p_s gelten: $p_s = (-1) * p_t$. Die Funktion unterscheidet sich demzufolge von der ermittelten unteren Schranke zum Ziel π_t bzw. zum Start π_s . Goldberg et al. schlagen hier verschiedene Funktionen vor und benutzt letztlich $p_t(v) = \max\{\pi_t(v), \pi_s(t) - \pi_s(v) + \beta\}$. In dieser Arbeit wurde die einfache Maximumsfunktion verwendet.

Ein Beschleunigungsverfahren, das sich ohne Anpassungen mit ALT verbinden lässt, ist das reichweitenbasierte Pruning. Dieses soll im nächsten Kapitel geschildert werden.

2.2 Reichweitenbasiertes Pruning

Die Idee des reichweitenbasierten Prunings (engl. *reach-based pruning*) ist, dass nur ein Teil der Knoten auf langen kürzesten Wegen vorkommt. Knoten, die ausschließlich auf kurzen kürzesten Wegen vorkommen, können so ggf. bei der Suche von der Betrachtung ausgeschlossen werden [Gut04].

Für das reichweitenbasierte Pruning führt [Gut04] zuerst die Reichweite ein: die Reichweite eines Knotens $r_W(k)$ bezüglich eines kürzesten Pfades $W = \langle k_0, k_1, \dots, k, \dots, k_n \rangle$ ist das Minimum aus der Länge von Präfix $d(k_0, k)$ und Suffix $d(k, k_n)$. Das Maximum über die Reichweiten eines Knotens über alle kürzesten Pfade ist die Reichweite des Knotens. Ein Knoten kann von der Suche ausgeschlossen werden, wenn der kürzeste Weg vom Start zum Knoten und vom Knoten zum Ziel größer ist als die Reichweite, d.h. wenn gilt: $d(s, k) > r(k)$ und $d(k, t) > r(k)$. Wenn dabei Knoten nicht geprunt werden, die man hätten geprunt werden können, leidet darunter zwar die Performanz, die Korrektheit des Algorithmus bleibt aber erhalten. Es kann demzufolge eine untere Schranke für $d(s, k)$ und $d(k, t)$ und eine obere Schranke für $r(k)$ genutzt werden.

Problematisch ist die Reichweitenberechnung. Ein einfaches Vorgehen ist es, von jedem Knoten einen kürzesten-Wege-Baum aufzubauen und anschließend für jeden Knoten die Reichweite als Maximum der Reichweiten bzgl. aller Pfade zu berechnen. Dieses Vorgehen würde für einen realen Graphen, bspw. den von Nordamerika mit fast 30 000 000 Knoten, Jahre dauern. Ein vorheriges Abbrechen des Aufbaus der Bäume ist nicht möglich, da man damit untere Schranken erhalten würde, wie oben erläutert aber obere Schranken von Nöten sind.

Goldberg et al. entwerfen in [GKW06] ein iteratives Verfahren zur Berechnung oberer Schranken für die Reichweiten. In jeder Iteration wird von jedem Startknoten s ein kürzester-Wege-Baum erstellt. Dieser wird erstellt, bis alle Knoten vom jeweils nächsten

⁴ $d_s(v)$ sei dabei die gefundene Entfernung des Knoten v vom Startknoten, $\pi_s(v)$ die durch die Metrik gegebene Entfernung des Knoten v von s . Äquivalent $d_t(v)$ und $\pi_t(v)$.

Nachbarknoten von s mehr als 2ϵ entfernt sind. Nach jeder Iteration werden aus den bisherigen Bäumen die bisher erstellbaren Reichweiten berechnet, ϵ wird um den Faktor α erhöht und jeder Knoten k wird aus der weiteren Berechnung ausgeschlossen, für den $r(k) < \epsilon$ gilt. Um obere Schranken zu erhalten, schlägt Goldberg vor, Kantenreichweiten statt Knotenreichweiten zu nutzen und Strafen einzuführen. Die Reichweite einer Kante (u, v) bzgl. eines Pfades $\langle k_0, k_1, \dots, u, v, \dots, k_n \rangle$ ist dabei das Minimum aus Präfix $d(k_0, v)$ und Suffix $d(u, k_n)$, die Reichweite der Kante ist das Maximum der Reichweiten über alle Pfade. Äquivalent werden in Goldbergs Verfahren Kanten statt Knoten von der weiteren Betrachtung ausgeschlossen, wenn ihre Reichweite unter ϵ fällt. Die Ein-Strafe eines Knotens ist dabei nach Goldberg das Maximum über die Reichweiten aller nicht mehr betrachteten eingehenden Kanten. Die Aus-Strafe wird äquivalent definiert. Bei jeder Berechnung einer Reichweite bzgl. eines kürzesten Pfades $\langle k_0, \dots, k_n \rangle$ wird die Ein-Strafe von k_0 auf das Präfix und die Aus-Strafe von k_n auf das Suffix der betrachteten Kante addiert. Nach Ende des Verfahrens können Knotenreichweiten ermittelt werden, da die Knotenreichweite das Maximum der Reichweiten der angrenzenden Kanten sind.

Goldberg et al. schlagen weiterhin vor, Abkürzungskanten einzuführen, um Knoten mit zwei Nachbarn zu überbrücken. Überbrückt werden dabei Linien, d.h. aufeinanderfolgende Knoten, und Sub-Linien, d.h. zusammenhängende Untermengen der Linien. Die Knoten sollen dabei nach dem Überbrücken von der Berechnung ausgeschlossen und durch Strafen repräsentiert werden. Das Abkürzungenstellen wird vor jeder Iteration durchgeführt. In der Wegsuche können anschließend Abkürzungskanten genutzt werden, die später wieder in die alten Kanten umgewandelt werden. Eine weitere Optimierung aus [GKW06] ist, dass am Schluss der Konstruktion eine Verfeinerungsphase eingefügt wird, die die Berechnung der Knoten mit hoher Reichweite neu durchführt. Hierbei müssen Strafen beachtet werden, so dass trotzdem fehlerbehaftete Reichweiten entstehen.

Weiterhin schlägt Goldberg in [GKW07] neben weiteren, eher technischen Optimierungen, vor, reichweitesensitive Landmarken (engl. *reach-aware landmarks*) zu nutzen, die lediglich für Knoten über einer bestimmten Reichweite Landmarken speichern. Auf diesem Weg können mit der gleichen Speichermenge mehr Landmarken gespeichert werden. Da die Knoten mit hoher Reichweite öfter in kürzeste-Wege-Suchen benutzt werden, führt dies zu einer Beschleunigung des Algorithmus.

2.3 Vergleich

Die Ansätze von reichweitembasierten Pruning und ALT unterscheiden sich maßgeblich: Während ALT eine Eigenschaft von kürzesten Wegen in allgemeinen Graphen ausnutzt, nutzt reichweitembasiertes Pruning die Eigenschaften von Verkehrsnetzen, dass manche Knoten auf längeren Wegen vorkommen und andere nicht. Beide sind dadurch ohne Anpassungen kombinierbar, im Gegensatz zu bspw. Highway-Hierarchien und reichweitembasiertem Pruning.

3 Verwandte Arbeiten

Nachdem im letzten Kapitel aktuelle Algorithmen zum Finden von kürzesten Wegen dargestellt wurden, soll sich dieses Kapitel mit Arbeiten beschäftigen, die das Thema der Netzintegration bzw. den Entwurf und Vergleich von Routingalgorithmen behandeln.

Die DELFI-Studie [GiK97] beschäftigt sich mit Netzintegration. Vor dem Hintergrund der Integration lokaler Verkehrsinformationssystemanbieter und der deutschen Bahn als Anbieter von Verkehrsinformationen auf Bundesebene wurde damals untersucht, wie sich Netze integrieren lassen. Hierbei wurden zwei Möglichkeiten vorgeschlagen: Systemverbünde und Datenschnittstellen. Bei Systemverbänden werden über Techniken zur Integration – im Praxiseinsatz über Dienste – die verschiedenen Systeme zusammengeschlossen. Der Vorteil hierbei ist, dass in jedem System netzspezifische Algorithmen eingesetzt werden können. Der Nachteil ist, dass für exakte kürzeste Wege eine große Anzahl von Anfragen notwendig sind. Bei Datenschnittstellen werden alle Daten für die anderen Anbieter verfügbar gemacht, allerdings müssen hierbei lizenzrechtliche Fragen bzgl. der Daten geklärt werden.

Wenn verschiedene Netze integriert werden, stellt sich die Frage, ob es Restriktionen bzgl. der Reihenfolge der Netznutzung geben sollte. Mit diesen Fragen beschäftigt sich [BJM00]. Durch eine Abbildung von den Kanten des dazugehörigen Graphen auf den Netztyp $N = \text{Bus, Bahn, ...}$ $\mathcal{L} : E \rightarrow N$ ist es möglich, das sog. *Label Constraint Shortest Path Problem* zu definieren. Dabei wird eine formale Sprache angegeben, aus der das Wort sein soll, das aus dem ermittelten Pfad mit der Abbildung \mathcal{L} entsteht. Ein Pfad ist nur gültig, wenn das Wort in der definierten Menge ist.

[DPW09] entwirft das Access-Node Routing Verfahren, das an das Transit-Node-Routing angelehnt ist. Hierbei werden Entfernungen von allen Knoten zu allen Knoten in einem zeitbehafteten Netz vorberechnet. Dadurch ist es möglich, kürzeste Wege schneller in Netzen, die aus zeitbehafteten und nicht-zeitbehafteten Teilnetzen bestehen, zu finden, wenn man bestimmte Rahmenbedingungen bezüglich der Abfolge von Netztypen annimmt.

Zu asymptotischen Laufzeiten existieren einige Arbeiten, bspw. die Arbeit von Xu et al. [XLH⁺07], die einen Algorithmus mit verbesserter asymptotischer Laufzeit für die Bestimmung kürzester Pfade in dünn besetzten Graphen angibt. Auch [AFGW10] beschäftigt sich mit asymptotischen Laufzeiten. In dieser Arbeit wird gezeigt, dass auf Basis einer bestimmten Eigenschaft von Netzen, der Highway-Dimension, reichweitenbasiertes Pruning, Highway-Hierarchien u.a. eine asymptotische Laufzeit besitzen, die von verschiedenen Größen, bspw. $\log|V|$ begrenzt ist.

4 Experimente

In den Experimenten, die zur Untersuchung der Algorithmen durchgeführt wurden, wurden der A*-Algorithmus mit unidirektionalem ALT-Verfahren (*ALT*), mit bidirektionalem symmetrischen ALT mit dem von Goldberg vorgeschlagenen Pruning (*BidirektionalSymmetrisch*) und mit konsistentem ALT-Verfahren mit Max-Funktion (*BidirektionalKonsistent*) verglichen (jeweils mit 4 Landmarken). Weiterhin wurde der REAL-Algorithmus, also die

Kombination von reichweitenbasiertem Pruning und unidirektionalem ALT-Algorithmus evaluiert (*REAL*). Dabei wurden 5000 Knotenpaare zufällig aus einem Netz ausgewählt und anschließend die Zeiten und die Anzahlen der gescannten Knoten bei ein kürzesten-Wege-Suche zwischen den Knoten gemessen. Da Anfragen, die beinahe die maximale Länge des Netzes ausnutzen, zu verfälschten Ergebnissen führen (da bspw. der Dijkstra-Algorithmus dann in der 'falschen' Richtung keine Knoten findet), wurde ein Ausschnitt der Daten geplottet. Das Netz bestand hierbei aus einem Netz des erweiterten Hamburger Stadtgebiets. Das Java-basierte Routingwerkzeug wurde auf einem Desktop-Rechner mit 1,8 GHz und 3,6 GB RAM ausgeführt. Die Werte für reichweitenbasiertes Pruning mit ALT liegen unter 200 ms. Eine Verbesserung ist auf einem Server mit mehr Arbeitsspeicher und mehr Landmarken denkbar.

Die Ergebnisse sind in Abbildung 2 dargestellt. Das reichweitenbasierte Pruning ist schneller als alle anderen Algorithmen, in der vorliegenden Version sind sämtliche Versionen des bidirektionalen ALT-Verfahrens langsamer als das unidirektionale ALT-Verfahren aber schneller als der Dijkstra-Algorithmus.

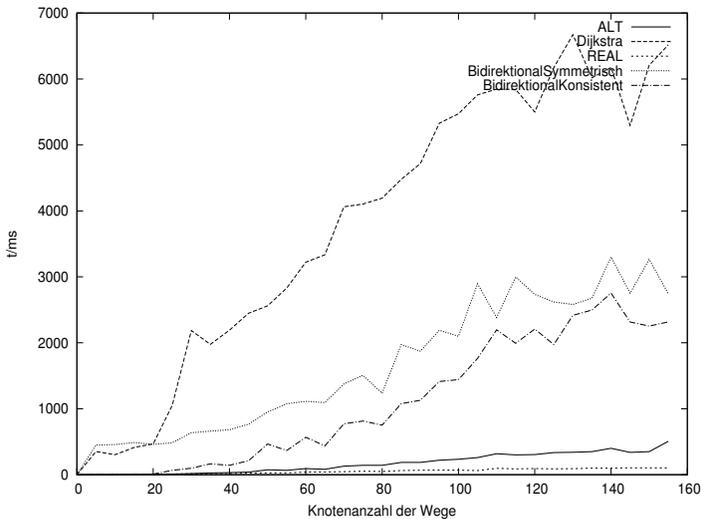


Abbildung 2: Graph der Laufzeiten der Algorithmen.

Bemerkenswert ist, dass die konsistente Ausführung des A*-Algorithmus in Relation zum unidirektionalen A*-Algorithmus derartig hohe Laufzeiten erreicht. Im Folgenden wird reflektiert, wieso die Laufzeit des unidirektionalen ALT-Algorithmus erheblich besser als die Laufzeit des bidirektionalen Algorithmus ist. Anschließend wird eine Betrachtung der Güte von Kanten- und Knotenreichweiten gegeben. Bei der Implementierung des Algorithmus aus [GKW06] zeigte sich außerdem eine Schwäche des Verfahrens zur Berechnung von Abkürzungskanten, dieses soll am Ende des Kapitels dargestellt werden.

Beim ALT-Verfahren ist der symmetrische Ansatz nicht effizient, da die Abbruchbedingung zu restriktiv ist. Für einen Knoten auf dem Weg wird sie erst erfüllt, wenn eine exakte untere Schranke gefunden wird: In diesem Fall wäre μ die Länge des kürzesten Wegs

und die Summe von $d_s(v) + \pi_r(v)$ exakt der kürzeste Weg. Wenn aber $\pi_r(v)$ kleiner als die exakte untere Schranke ist, wäre das Abbruchkriterium nicht erfüllt. Der Algorithmus würde also weiterlaufen, bis eine exakte untere Schranke auf dem Weg gefunden ist. Auf Knoten neben dem Weg würde zwar ggf. $k(v)$ größer sein, bei guten unteren Schranken führt die Suche aber direkt zum Ziel ohne viele entfernte Knoten zu betrachten, so dass die bidirektionale Suche bei guten unteren Schranken zu keiner Beschleunigung führt. Vor diesem Hintergrund sind die Experimentergebnisse zu betrachten.

Während der Durchführung der Experimente wurden Knotenreichweiten und Kantenreichweiten verglichen. Bei der Betrachtung der Kantenreichweite ist es notwendig, die Länge der betrachteten Kante selbst einzubeziehen. Goldberg schreibt, es wäre auch möglich, Kantenreichweiten ohne die Länge der jeweiligen Kante zu berechnen, d.h. die Reichweite einer Kante (u, v) bzgl. eines Pfades als Minimum aus $d(s, u)$ und $d(v, t)$ zu berechnen.

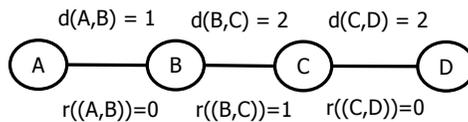


Abbildung 3: Beispielgraph, in dem Kantenreichweiten ohne Einbeziehung der Kantenlänge unkorrekte Knotenreichweiten (unkorrekte Kantenreichweiten sind dargestellt) liefern.

Wie Abbildung 3 zeigt, ist das nicht der Fall: die Reichweite der Kante (B, C) würde 1 betragen, da das Minimum aus $d(A, B)$ und $d(C, D)$ 1 ist, und sämtliche weitere kürzesten Pfade geringere Reichweiten liefern. Bei der Berechnung der Knotenreichweiten aus Kantenreichweiten erhalten B und C die Reichweite 1. C hat allerdings die Reichweite 2: bei dem kürzesten Weg von A nach D hat C ein Präfix von 3 und ein Suffix von 2. Insgesamt hat C also die Reichweite 2. Eine Definition, die Kanten selbst in die Reichweite nicht einschließt, liefert also falsche Knotenreichweiten.

Kantenreichweiten sind nach Goldberg effizienter, da Kantenreichweiten ggf. kleiner sein können als die Reichweiten angrenzender Knoten. Im Folgenden soll gezeigt werden, dass dieses Verfahren schlechtere Knotenreichweiten liefert. Außerdem soll ein effizientes Verfahren zur direkten Berechnung der Knotenreichweiten vorgestellt werden.

Der Grundgedanke lässt sich anhand von Abbildung 4 nachvollziehen: die Kante (A, B) hat lediglich die Reichweite 5^5 , obwohl der angrenzende Knoten A die Reichweite 10 hat⁶. Der Suchalgorithmus ist besonders effizient, wenn möglichst niedrige obere Schranken für die Reichweite gefunden werden, damit möglichst viele Knoten geprunt werden können. Kantenreichweiten liefern also nach diesem Gedankengang niedrigere Reichweiten. Dieser Effekt ist zwar vorhanden, allerdings ist zu beachten, dass in einem gewöhnlichen Routinggraph mehr Kanten als Knoten vorhanden sind, da der Graph stark zusammenhängend ist und Verzweigungen besitzt. Dies führt dazu, dass die Berechnung von Kantenreichweiten aufwändiger ist.

⁵Die Kantenreichweite von (A, B) ist das Maximum aus der Reichweite von (A, B) bzgl. dem Weg $C - A - B$, $\min(d(C, B), d(A, B)) = 5$ und der Reichweite von (A, B) bzgl. dem Weg $D - A - B$, $\min(d(C, B), d(A, B)) = 5$

⁶Der kürzeste Weg $C - A - D$ hat ein Suffix und ein Präfix von 10 und damit hat A die Reichweite 10 bzgl. des Weges.

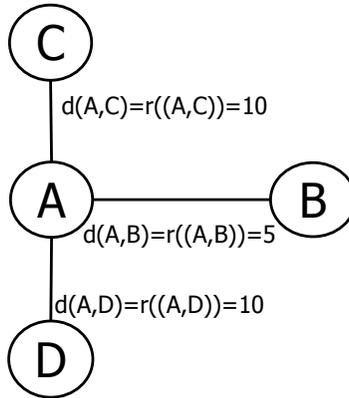


Abbildung 4: Beispielgraph für die Behauptung, dass Kantengewichte niedriger als Knotengewichte sein können.

Knotenanzahl	Knoten	Kante	Kante(Exakt)	Kante
	-Knoten(exakt)	-Knoten(Exakt)	-Knoten(Exakt)	-Kante(exakt)
701	31.350	129.855	89.617	79.737
1398	98.690	245.977	97.982	276.228
2097	166.450	346.431	107.723	457.399
2796	338.521	523.399	116.817	778.853
3340	392.586	554.526	121.044	862.430

Tabelle 1: Stellt den Zusammenhang zwischen Knotenreichweiten, Knotenreichweiten die aus Kantenreichweiten berechnet wurden und deren exakten Pendanten dar

Ein weiterer Effekt hierbei ist die Erhöhung der Knotenreichweiten durch verschiedene Kantenreichweiten. Dieser Effekt ist anhand von Abbildung 4 nachvollziehbar. Die durch Kantenreichweiten errechneten Reichweiten der Knoten wären für A, C und D 10 und für B 5. Die realen Knotenreichweiten betragen jedoch 0 für B, C und D⁷ und 10 für A⁸. Die Knotenreichweiten steigen also, da das Maximum aller angrenzenden Kanten genutzt wird und da für diese Kanten die Kantenlänge einbezogen werden muss.

Dieser Zusammenhang wurde statistisch untersucht. Tabelle 1 zeigt, dass mit exakten Kantenreichweiten die ermittelten Knotenreichweiten von den exakten Knotenreichweiten abweichen, und dass die heuristischen Knotenreichweiten besser sind als die heuristischen Kantenreichweiten. Als Datengrundlage für die Berechnung dienten hierbei Ausschnitte eines dicht besiedelten urbanen Bereichs aus OpenStreetMap, wobei stets zusammenhängende Teilgraphen betrachtet.

Da Kantenreichweiten höhere Abweichungen von den realen Reichweiten liefern als

⁷Da diese stets am Ende bzw. Anfang von Pfaden liegen und damit Präfix bzw. Suffix immer 0 ist und damit das Minimum aus Präfix und Suffix immer 0 bleibt.

⁸Da A auf dem Pfad C – A – D liegt und jeweils ein Suffix und ein Präfix von 10 hat.

Knotenreichweiten, wurden in der vorliegenden Implementierung Knotenreichweiten genutzt. Wie in der ursprünglichen Version werden in jeder Iteration zuerst Abkürzungskanten erstellt und danach Reichweiten und Strafen berechnet. In der Kantenversion ist die Aus-Strafe das Maximum über die Reichweiten aller gelöschten ausgehenden Kanten, weil deren Reichweite ggf. vom Suffix hätte stammen können und damit das Ende des vorherigen Baums war. In der Knotenversion ist die Strafe für Knoten k das Maximum über $\max_n l(k, n) + r(n)$, wobei n alle über ausgehende Kanten verbundene Nachbarknoten sind. Dies gilt, da der längstmögliche Pfad, der sich an einen Nachbar n anschließen kann, der Pfad ist, der die Länge seiner Reichweite plus die Entfernung zu k hat. Gleiches gilt für die eingehenden Kanten und die Ein-Strafen.

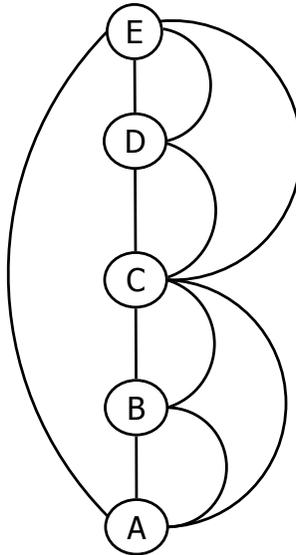


Abbildung 5: Beispielgraph für die Behauptung, dass überspringende Knoten Reichweiten erhalten müssen.

Man könnte annehmen, dass diese Definition der Strafen zu falschen Reichweiten führt, denn in Iteration i mit ϵ_i könnte ein Knoten n ausgeschlossen werden, dessen Suffix s länger ist als sein Präfix p . k sei der Nachbarknoten von n , der die Aus-Strafe erhalten hat. In diesem Fall wäre die Aus-Strafe geringer als das Suffix, wodurch ein Weg $W = \langle k_0, k_1, \dots, k \rangle$ in Iteration $i + 1$ für einen Knoten k_n eine zu geringe Reichweite ergeben könnte, da k eine zu geringe Aus-Strafe erhalten hat. Diese Annahme ist jedoch falsch: es gilt $p < \epsilon_i$ und $s < \epsilon_i$, da n in Iteration i ausgeschlossen wurde. Da der Algorithmus eine Kürzeste-Baum-Erstellung beendet, sobald alle Blätter mehr als $2 * \epsilon_i$ vom ihnen nächsten Nachbarknoten des jeweiligen Startknotens entfernt sind, gibt es einen Knoten k_m , von dem aus das Ende des Suffix s gefunden wird. Es kann keinen Knoten k_{m-1} geben, der in W vor k_m vorkommt, da ansonsten dieser Knoten das Präfix von n verlängert hätte. Es kann also kein Weg W existieren, auf dem ein Knoten eine falsche Reichweite aufgrund der Aus-Strafe von k erhält ist.

Eine dritte Schwachstelle des in [GKW06] beschriebenen Verfahrens betrifft die Erstellung von Abkürzungskanten: nach dem Verfahren würden alle Kanten einer Linie von der weiteren Berechnung ausgeschlossen werden, und damit eine Reichweite von 0 erhalten. Abbildung 5 zeigt, dass das nicht korrekt ist: es existieren kürzeste Wege, mit auf denen C vorkommt und die ein Suffix und Präfix haben das größer als 0 ist (bspw. $\langle B - C - D \rangle$). Da sämtliche Wege, die die gesamte Linie enthalten, über die Abkürzung von A nach E passierbar sind, kann es sich nur um Wege innerhalb der Abkürzung handeln. Eine korrekte Lösung ist es also, das Minimum aus $l(A, C)$ und $l(C, E)$ als Reichweite von C zu nutzen (da kein Pfad, der innerhalb der Linie ist, längere Teilpfade enthalten kann). In der vorliegenden Implementierung wurde deshalb die Reichweite überbrückter Knoten am Ende von Sublinien dementsprechend definiert.

5 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurde das Problem des Findens kürzester Wege bei der Nutzung verschiedener Verkehrsmittel behandelt. Für dieses Problem muss derzeit eine effiziente Lösung gefunden werden, da vor dem Hintergrund des Aufkommens von Bike- und Car-sharing Nutzer dieser Verkehrsmittel kürzeste Wege in den kombinierten Netzen benötigen. Hierfür wurden verschiedene Algorithmen anhand eines Werkzeugs evaluiert. Diese Algorithmen und die dazugehörigen Publikationen wurden beschrieben und kritisch hinterfragt. Der REAL-Algorithmus, die Kombination aus ALT-Verfahren und reichweitenbasiertem Pruning, war am lauffzeiteffizientesten.

Ein Ansatzpunkt weiterer Forschung ist das Implementieren eines effizienten Algorithmus für zeitbehaftete Netze bzw. für integrierte zeitbehaftete und nicht-zeitbehaftete Netze. In [Gei11] sind einige auf Knotenkontraktion, eine Weiterentwicklung der Highway-Hierarchien, basierende Algorithmen für kürzeste Wege in zeitbehafteten Netzen dargestellt, die Laufzeiten von 200-500 μs auf 2,7 GHz, 46 GB RAM und einer C++-Implementierung erreichen. Weiterhin beschäftigt sich diese Arbeit mit einer Beschleunigung von Vorverarbeitungen, um auf Netzänderungen wie bspw. Unfälle und Baustellen schneller reagieren zu können. Eine mögliche Erweiterung der vorliegenden Arbeit wäre die Kombination mit einem Algorithmus für zeitbehaftete Netze. Denkbare wäre hier neben einer Implementierung des Algorithmus aus [Gei11] eine auf dem REAL-Algorithmus basierende Implementierung des Transitknotenverfahrens als Grundlage zu nutzen, wie in [GKW07] vorgeschlagen.

Danksagung

Ich möchte mich herzlich bei Andre Hegerath von HBT und Lars Braubach von der Arbeitsgruppe für Verteilte System an der Universität Hamburg für die Betreuung der Arbeit und bei meinem Komilitonen Carsten Rehder für das Korrekturlesen der Arbeit bedanken.

Literatur

- [AFGW10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2010.
- [BFM⁺06] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortestpath queries in road networks. In *Proc. 9th International Workshop on Algorithm Engineering and Experiments*, pages 46 – 59. SIAM, 2006.
- [BJM00] C. Barrett, R. Jacob, and M.V. Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.
- [DPW09] Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. In *European Symposium on Algorithms*, pages 587 – 598, 2009.
- [Gei11] R. Geisberger. *Advanced Route Planning in Transportation Networks*. PhD thesis, Karlsruher Institut für Technologie, 2011.
- [GH05] A.V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, 2005.
- [GiK97] CSC Ploenzke FAW GiKOM. DELFI: Deutschlandweite Elektronische Fahrplan-Information. Technical report, CSC Ploenzke FAW GiKOM, 1997.
- [GKW06] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In *SIAM Workshop on Algorithms Engineering and Experimentation*, 2006.
- [GKW07] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Better Landmarks within Reach. In *Proceedings of the 6th international conference on Experimental algorithms*, 2007.
- [Gut04] R. Gutman. Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *6. International Workshop on Algorithm Engineering and Experiments*, pages 100–111, 2004.
- [HKMS09] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast Point-to-Point Shortest Path Computations with Arc-Flags. In *Shortest Paths: Ninth DIMACS Implementation Challenge*. American Mathematical Society, 2009.
- [Poh69] Ira Pohl. *Bi-directional and heuristic search in path problems*. PhD thesis, Universität Stanford, Mai 1969.
- [SS05] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *13th European Symposium on Algorithms (ESA)*, page 568–579, 2005.
- [vS09] W. von Sassen. Öffentliche Fahrradverleihsysteme im Vergleich. Analyse, Bewertung und Entwicklungsperspektiven. Master’s thesis, Universität Trier, Fachbereich VI - Angewandte Geographie, Januar 2009.
- [XLH⁺07] M. H. Xu, Y. Q. Liu, Q. L. Huang, Y. X. Zhang, and G. F. Luan. An improved Dijkstra’s shortest path algorithm for sparse network. *Applied Mathematics and Computation*, 185:247 – 254, 2007.

Künstliche Intelligenz auf Basis von „Baby Machines“: Geschichte und Architektur BMA1

Florian Golemo

fgolemo@gmail.com

Universität Leipzig, Bachelor Studiengang Informatik

Zusammenfassung: Das Wissenschaftsgebiet Künstliche Intelligenz (KI) hatte einst das Ziel, menschenähnliche Intelligenz zu kreieren. Bereits Alan Turing hat vorhergesagt, dass dies durch Analyse der kognitiven Funktionen von Kindern möglich sein wird. Diesen Ansatz nennt man „Baby Machines“ (BM). Seine Vergangenheit und mögliche Zukunft wird in diesem Artikel beleuchtet. Im Kapitel Motivation wird der Grundgedanke vorgestellt und im geschichtlichen Hintergrund seine frühe Entwicklung. Das dritte Kapitel zeigt einige einflussreiche Kognitive Architekturen seit den 70er Jahren auf. Ebenso werden aktuelle Forschungsergebnisse behandelt, die maßgeblich zur Schaffung einer BM beitragen könnten. Auf Basis dieser Vorarbeiten wird im vierten Kapitel ein Konzept einer möglichen BM-Architektur vorgestellt. Dieses befindet sich noch in der Entwicklung, vereint jedoch bereits zahlreiche der bisherigen Ansätze. Der vorliegende Artikel stellt eine erste Niederschrift des Lösungsvorschlags dar. Er soll als Grundlage gesehen werden, in Zukunft weitere Konkretisierungen vorzunehmen und nach Möglichkeit eine Implementierung zu erstellen.

1 Motivation

Das einst gesteckte Ziel der KI-Pioniere war es, einen Computer mit einem physischen Körper zu schaffen, der sich menschengleich verhält. Intelligenz als abstraktes Konzept steht nicht im Vordergrund, jedoch allgemeine Anpassungs- und Lernfähigkeit, Kommunikation und Kreativität. Der Baby-Machine-Ansatz¹ zielt nicht darauf ab, der Maschine bestehendes Wissen der Menschheit oder hohe formal-logische Intelligenz einzuprogrammieren. Stattdessen möchte der Ansatz die Lern- und Handlungsfähigkeit von Kleinkindern analysieren und im Computer reproduzieren. Dann, so hofft man, kann man diesen Computer wie ein Kind erziehen. D. h. heißt, mit ihm lernen oder ihn anleiten, sich Informationen selbst spielerisch zu erschließen. Das „Baby“ in „Baby Machine“ (BM) steht nicht dafür, dass man unbedingt das Hirn auf dem Entwicklungsstand von Kleinstkindern modellieren muss. Es geht darum, zu verstehen, wie der Mensch lernt beziehungsweise handelt und dann ein Modell zu kreieren, was mit möglichst wenig

¹Zu Baby Machines wurde in der Literatur keine trennscharfe Definition gefunden. Folgende Kurzfassung wird vorgeschlagen: „Eine BM ist eine Maschine, die das Ziel hat, größtmögliche Ähnlichkeit mit der kognitiven Leistungsfähigkeit des Menschen zu erlangen. Angestrebt wird dies durch Lernen und Interaktion mit einer Welt. Realisiert wird eine BM durch Analyse und Nachbildung von menschlichen Fähigkeiten. Welche Fähigkeiten mindestens notwendig sind, um menschliches Potential zu erlangen, muss ermittelt werden. BM zählen zu Systemen der Starken KI nach [Kur05].“

vordefiniertem Wissen funktioniert und selbstständig lernen kann. Wenn dies gelingt, erhält man ein menschenähnliches Lernmodell, mit dem man in Experimenten beliebige Lernsituationen erproben könnte. Pädagogen beispielsweise hätten so bessere Chancen, neue Techniken zu evaluieren. Sollte die BM zur Leistungsfähigkeit von Erwachsenen heranreifen, dann bieten sich einem weitere Möglichkeiten: Intelligente Computer könnten die Funktion von persönlichen Lernassistenten einnehmen. Computer können heutzutage bereits große Informationsmengen aufnehmen und speichern. Eine Maschine, die diese Informationen auch versteht, könnte genutzt werden, um Menschen beim Lösen von Problemen oder Erreichen von Zielen behilflich zu sein. Wenn die BM auch über ausreichend soziale und emotionale Intelligenz verfügt, wäre sie fähig, sich in beliebige Gesprächspartner zu versetzen und ihre Kommunikation an die Bedürfnisse des jeweiligen Partners anzupassen. Das ermöglichte eine völlig neue Mensch-Maschine-Interaktion [MSS04]. Ziel dieses Artikels ist es, auf einen neuen Ansatz zur Lösung des BM-Problems hinzuwirken. Dies erfolgt durch die Vorstellung des geschichtlichen Hintergrundes und bisheriger Ansätze; gefolgt von deren Zusammenführung zum Konzept einer neuen Architektur.

2 Hintergrund

Alan Turing hat während des zweiten Weltkrieges im Bletchley Park an der Entschlüsselung der Enigma gearbeitet. Er war umgeben von Cryptoanalysten, die aufgrund besonderer Erfahrungen ausgewählt wurden. Dazu zählten z. B. Linguisten, Schach-Experten und einflussreiche Mathematiker [Bbc12]. Kurze Zeit nach dem Krieg entwarf er einen eigenen Programmspeicher-Computer. Umgeben von Genies sammelte Turing Ideen für das Potential der neuartigen Rechenmaschinen. Er entwickelte das geistige Fundament der modernen KI-Forschung und fasste seine Gedanken 1950 in dem einflussreichen Artikel „Computing Machinery And Intelligence“ zusammen [Tur50].

Turing formuliert die Frage „*Can machines think?*“ und erklärte die Frage für alsbald hinfällig: Ende des 20. Jahrhunderts würde es, seiner Meinung nach, eine Selbstverständlichkeit sein, von denkenden Maschinen zu sprechen. Er untermauerte seine Überzeugung, indem er zahlreiche Argumente gegen seine Hypothese auflistete und diese durch schlüssige Argumentation entkräftete. Eines dieser Gegenargumente ist der Einwand, 1842 von Ada Lovelace geäußert, dass Maschinen nur „denken“ können, was ihnen konkret einprogrammiert wurde. Turing begegnet dem Argument mit der Möglichkeit, dass man eine Maschine programmieren könnte, zu lernen. Gemeint ist, man betrachte das Prinzip „Eingabe, Verarbeitung, Ausgabe“. Man kann einen Computer programmieren, anhand von Eingaben das eigene Verarbeitungs-Regelwerk verändern zu lassen oder zu erweitern. Derartige Lernen hat bereits weite Verbreitung, z. B. in E-Mail-Spamfiltern. Diese erkennen aufgrund von vordefinierten Regeln Spam und sondern diesen aus. Wenn sich nun jedoch ein Nutzer entscheidet, dass eine bestimmte Mail kein Spam war, werden die eigenen Regeln geändert. Das nächste mal wenn eine derartige Mail eingeht, wird diese nicht mehr als Spam markiert. Alan Turing schlägt zu Lösung des Problems folgendes vor, was eines der bedeutendsten Zitate der Baby-Machine-Idee wurde: „*Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one*

which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain.“ [Tur50] Marvin Minsky sollte diesen Ansatz, einen Computer von menschenähnlichem Verstand zu schaffen, später mit dem Begriff „Baby Machine“ versehen [Min12]. Um den heutigen Zustand des Forschungsbereiches der menschenähnlichen KI verstehen zu können, benötigt man einen Überblick über die frühe Entwicklung und das Auseinanderbrechen des Faches KI. Dieser soll im Folgenden gegeben werden. Im Sommer des Jahres 1956 fand sich ein kleiner Kreis von Forscher der USA in Dartmouth zusammen. Die zweimonatige Konferenz wird als Geburtsstunde des Fachs KI angesehen und auch wenn Turing zu diesem Zeitpunkt bereits tot war, sorgte sein Gedankengut für eine „Alles ist möglich“-Stimmung [RN04]. Dieser Enthusiasmus hielt rund 20 Jahre an und Herbert Simon hat ihn 1958 wie folgt beschrieben: *„It is not my aim to surprise or shock you [...] but the simplest way I can summarize the situation is to say that there are now in the world machines that think, that learn, and that create. Moreover, their ability to do these things is going to increase rapidly until in a visible future - the range of problems they can handle will be coextensive with the range to which the human mind has been applied.“* [SN58] Leider waren die Erwartungen zu hoch angesetzt. Viele Projekte, die große Hoffnungen geweckt hatten, scheiterten und es folgte der „KI-Winter“.² In dessen Verlauf teilte sich das nunmehr umfangreiche Fach KI in verschiedene Fachbereiche wie Expertensysteme, Maschinenlernen oder Robotik. Dies hatte anfangs den Zweck, kleinere, greifbarere Ziele zu setzen und damit von allen Seiten das „Hauptproblem“ der KI anzugehen [RN04]. Im Laufe der Zeit verselbstständigten sich die Fachbereiche und erzielten eindrucksvolle Ergebnisse. Dazu zählen unter anderem der Sieg des Schachcomputers Deep Blue über Garri Kasparov 1997 oder Googles selbstfahrendes Auto, welches bereits im Straßenverkehr unterwegs ist [Mar12]. Man erfreute sich an den Resultaten in der jeweiligen hermetisch abgegrenzten Domäne und verlor das eigentliche Ziel der menschenähnlichen Intelligenz aus den Augen [Bra06]. Außerdem mutmaßte man, dass es schwer werden würde, erneut Gelder für derartig große Projekte zu erhalten, denn das Wissen über das „Scheitern“ der KI³ in der Vergangenheit bleibt in den Köpfen bestehen.

Es gibt jedoch noch einige Wissenschaftler, die Turings Idee und seine Begeisterung teilen, darunter der frühere Kreis der Dartmouth-Besucher. Die Forschung wurde in kleineren Maßstäben fortgesetzt und im Laufe der 70er und 80er-Jahre entwickelten sich kognitive Modelle beziehungsweise kognitive Architekturen.

²Ein prominentes Beispiel ist das Scheitern der Übersetzungsprogramme '66: Die US-Regierung beschloss 1957 immense Unterstützung der Forschung an maschineller Übersetzung von russischen Texten. Man zielte im Kalten Krieg darauf ab, wissenschaftliche russische Texte schneller zu übersetzen. Nach zahlreichen Tests stellt ein Bericht von 1966 vernichtend fest, dass dies derzeit unmöglich sei und in absehbarer Zeit nicht möglich werden wird. Gründe für das Scheitern war die zu niedrige Einschätzung der Komplexität des Problems. Bis heute gibt es keine perfekte automatische Übersetzung [RN04].

³„Scheitern“ in Anführungszeichen, denn unter anderem nach [Kur05]: KI ist allgegenwärtig. Von der täglichen Suche im Internet bis zur Errechnung von Flugrouten. Überall kommen KI-Algorithmen zum Einsatz.

3 Kognitive Architekturen und aktuelle Entwicklung

Eine Kognitive Architektur (KA) zielt darauf ab, bestimmte kognitive Funktionen nachzubilden. Kognitive Funktionen sind z. B. das allgemeine Lernvermögen, Selbstreflexion oder die Wahl der nächsten Handlung. KA basieren meist auf psychologischen und/oder neurologischen Erkenntnissen und abstrahieren von den unterliegenden biochemischen Prozessen im Gehirn. Die Architekturen finden nicht nur in der KI-Forschung Verwendung, sondern z. B. auch in der Analyse menschlichen Verhaltens in bestimmten Situationen. Die Wirksamkeit eines solchen Modells kann nur durch tatsächliche Implementierung und Tests in psychologischen Experimenten gezeigt werden [Lan04]. Dennoch ist zuerst eine Modellierung des Gesamtsystems notwendig, bei der die Form der Implementierung zweitrangig ist. Nur so kann man die jeweilige Architektur in Gänze verstehen. Beim Hausbau wird auch zuerst das Haus als Ganzes entworfen, dann die Räume, dann werden die zu verwendenden Materialien gewählt.⁴ Leider können die meisten KA nicht direkt als BM eingesetzt werden, da jeweils nur ein Teil der menschlichen Intelligenz modelliert wird. Jedoch liefern die KA wertvolle Einsichten, wie eine BM modelliert und ausgestaltet werden kann. Nachfolgend sollen einige einflussreiche KA und Entwicklungen in dieser Hinsicht aufgezeigt werden.

1976 legte John Anderson als Ergebnis seines Lebenswerkes erstmals ein Computer-Modell für menschliche Kognition vor. Dieses Modell namens „ACT“ (und später „ACT-R“) wurde 1983 erfolgreich in Gänze implementiert und erzielt schnell Erfolge. Es verfolgt einen Ansatz, der „Konnektionismus“ bezeichnet wird. Dem System wird ein Ziel vorgegeben, also ein Problem wie z. B. die Steuerung eines Modellhelikopters über einen Parkour. Man pflegt ACT-R Vorwissen ein, dass man über das Problem besitzt, eine Art Weltbild, das programmiert werden muss. Mit dem Problem und dem Vorwissen kann man den ACT-R-Interpreter starten. Im Vorgang der Verarbeitung werden aus Vorwissen, Wahrnehmung und Zielstellung Informations-Netzwerke gebildet. Dies haben alle KA gemeinsam. Der Ausdruck Konnektionismus bedeutet, dass im Laufe der Verarbeitung weniger das Netzwerk umstrukturiert wird, sondern es werden Wahrscheinlichkeiten und Wichtigkeiten von Elementen geändert. Dies stellt die Art und Weise nach, wie im Hirn häufig benutzte Synapsen gestärkt und wenig genutzte aufgebrochen werden [Lan04]. Die KA kommt bereits erfolgreich zum Einsatz, um neuronale Muster in menschlichen Hirnen bei Problemen vorauszusagen, die hierdurch modellierbar sind [AFQS08]. Die Möglichkeit der Nutzung als persönlicher Lernassistent, z. B. bei der Vermittlung von Lernmaterial an Studenten, wurde ebenfalls bereits bedacht und untersucht [AG01].

Das Modell verfügt über keinen anderen Eigenantrieb, als den, den man ihm vorgibt und für den Problemlöseprozess kann nur Wissen verwendet werden, dass sich aus dem vorgegebenen Wissen erschließen lässt. Es lässt sich aber einsetzen, um nach dem konnektionistischen Ansatz Lernen, Wahrnehmung und Problemlösung zu modellieren. Außerdem ist in der KA das „Chunking“ für eine potentielle BM von Interesse. Chunking ist das Gruppieren von ähnlichen Informationen in eine Einheit des Arbeitsgedächtnisses (AG). Dies ist nützlich, da das AG nur sehr begrenzte Kapazitäten besitzt. Chunking ist

⁴Hier gibt es jedoch einige Ausnahmen, da in einigen Szenarien die Implementierung und die Architektur sich gegenseitig bedingen. Man kann auch, mangels Materialien, (noch) keine schwebenden Häuser bauen.

beim Lernen notwendig im Rahmen von Abstraktion. In ACT-R bedeutet das, dass neue Wahrnehmungen angepasst und eingefügt werden in das, was durch das Vorwissen bekannt ist [Lan04]. Angenommen es sei z. B. das Konzept „Katze“ bekannt, mit einer bestimmten durchschnittlichen Größe und einer bestimmten Gangart. Wenn eine neue Katze wahrgenommen wird, muss diese nun nicht mehr in allen Details gespeichert werden, sondern nur, wo sie von der „Muster-Katze“ abweicht.

Ein ähnliches Ziel wie Anderson mit ACT-R hat auch Allen Newell mit seiner Architektur SOAR verfolgt. In seinem Buch „Unified Theories of Cognition“ forderte er nach einer zusammenhängenden Erklärung für Kognition. Als eigene Lösung entwarf er die SOAR-Architektur und stellte sie 1986 in einem Artikel vor. Diese KA arbeitet auf der Basis des „Symbolismus“. Während beim Lernvorgang von ACT-R Wahrscheinlichkeiten und Gewichtungen von Informationen geändert werden, arbeitet der Symbolismus durch die Anwendung von formalen Regeln. Zu diesem Zweck simuliert SOAR Langzeit- und Arbeitsgedächtnis. Situationsrelevante Informationen werden, wie beim Mensch, aus dem Langzeitgedächtnis in das Arbeitsgedächtnis geholt und dort je nach Problem verarbeitet. Die Kernfunktion ist also die Gleiche wie bei ACT-R: Selbstständiges Problemlösen unter gegebenem Vorwissen. Dieses Modell hat auch aktuell eine große Verbreitung zur Simulation menschlichen Verhaltens in bestimmten Situationen [LLR06]. Die KA teilt sich auch die Schwachstelle von ACT-R, dass man ein sehr breites Spektrum an domänenbezogenem Wissen in die Architektur speisen muss. Jedoch ist auch dieses Modell wichtiger Ideengeber für eine potentielle BM-Architektur, da verdeutlicht wird, dass man menschliches situatives Verhalten auch mit dem symbolischen Ansatz umsetzen kann.

Neben diesen beiden KA, die jeweils gegensätzliche Ansätze für gleiche Probleme entwarfen, verfolgte Marvin Minsky mit seinem, 1988 erschienenen, Buch „The Society of Mind“ eine neue Richtung. Sein Werk stellt einen Vorschlag für eine KA dar, die zahlreiche Aspekte vereint. Das Fundament bilden eine Kombination aus Konnektionismus und Symbolismus. Minsky zeichnet das Hirn als „Gemeinschaft von Agenten“. Ein „Agent“ steht für jeweils eine Aufgabe, eine geistige Kompetenz. Als Beispiel wird häufig die hypothetische „Builder-Society“ genannt, die beim Kleinkind für das Stapeln von Bauklötzen zuständig sein soll. Da dieses Problem zu komplex ist, um mit einem großen statischen Programm gelöst zu werden, muss diese Aufgabe von kleineren, spezifischeren Agenten gelöst werden. Das sind konkret „Anfangen“, „Klotz hinzufügen“, „Beenden“, wobei z. B. „Klotz hinzufügen“ wieder unterteilt ist in „Finden“, „Aufnehmen“, „Oben auf den Turm legen“. „Aufnehmen“ besteht wiederum aus „Greifen“ und „Bewegen“, usw.. Dabei können Agenten, wie z. B. „Greifen“ auch von mehreren anderen Agenten genutzt werden. Damit teilt sich das Problem in viele kleine Unterprobleme, die jeweils keine Intelligenz erfordern oder die zumindest einfach algorithmisch umsetzbar sind. Eine weitere Besonderheit von Minskys Theorie ist auch, dass sie darauf abzielt, ein breites Spektrum an Denken und Handeln zu erklären. Es wird vom Kind ausgegangen und es werden komplexere Zusammenhänge wie Kreativität und Emotionen erklärt [Min86].

Das Problem an Minskys Arbeit ist der hohe Grad an Abstraktheit. Das Buch beinhaltet zwar Vorgaben an die Umsetzung, bisher ist jedoch keine vollständige Implementierung der Ideen gelungen. Die beschriebene KA ähnelt in einigen Punkten der SOAR-Architektur, unterscheidet sich jedoch vollkommen in der grundlegenden Ar-

beitsweise [Sin03]. Trotz der geringen Gegenständlichkeit stellt Minskys Werk eine eindrucksvolle Inspiration für den Baby-Machine-Ansatz dar und es liefert zusammenhängende Lösungsvorschläge für zahlreiche Probleme der Kognition. Auf Grundlage eigener Untersuchungen hat der Neurologe David Eagleman diese Theorie von Minsky ergänzt zur „Democracy of Mind“. Der Autor meint, dass für jede Aufgabe nicht nur jeweils ein Agent Lösungen erarbeiten, sondern stets mehrere. Übergeordnete Agenten stimmen dann ab, welche Lösung am erfolgversprechendsten aussieht und wählen diese [Eag11].

Angenommen man geht von Minskys Society of Mind aus und gliedert jede kognitive Funktion in zahlreiche, einfachere Unterfunktionen auf. Wenn man dieses Vorgehen oft genug wiederholt, so meint Jeff Hawkins, gelangt man schließlich zur Grundlage jeder Intelligenz. Hawkins veröffentlichte 2004 mit Sandra Blakeslee „On Intelligence“. Dieses Buch basiert auf aktuellen Forschungsergebnissen der Neurobiologie. Es geht von einem Algorithmus aus, der überall im Hirn anzutreffen ist, unabhängig von der Funktion des Hirnareals. Das ganze Werk ist getrieben von der Hypothese, dass dieser eine unterliegende Algorithmus die Grundlage aller Intelligenz ist. Hawkins identifiziert Kernfunktionen, die diesen Grundmechanismus kennzeichnen und erklärt mit diesen Funktionen tägliche Handlungsweisen wie auch kreative Prozesse. Wenn z. B. die ersten Takte eines Lieblingsliedes anspielen, erkennt man schnell, um welches Lied es sich handelt. Man kann es sogar, ohne es zu hören, nachsummen oder den Text nachsprechen. Wird man jedoch vor die Aufgabe gestellt, das Lied rückwärts nachzusummen, bereitet dies den meisten Menschen Schwierigkeiten. Dies deutet auf die ersten beiden Kernfunktionen hin: Erinnerungen werden in „Mustern“ gespeichert und wenn man ein Muster aktiviert, an einer beliebigen Stelle, wird der Rest des Musters auch aktiv. Mit „Musterbind allgemein und uneingeschränkt Prozessabläufe gemeint. Sei es ein Lied, die Geschichte eines Buches, Schnürsenkelzubinden, das Kaffeholen im Büro oder andere.[HB04]

Hawkins fasst dies zusammen als „Memory-prediction framework“, eine Kognitive Architektur, die dem Aufbau des Neocortex⁵ nachempfunden ist. Diese besteht aus mehreren Ebenen, wovon jede Ebene eine Ebene der Wahrnehmung wiedergibt. Jede Ebene ist mit den darüber- und darunterliegenden verbunden und führt Mustererkennung⁶ auf unterliegenden Ebenen durch. Die unterste Ebene ist die sinnliche Wahrnehmung. Bereits erkannte Muster werden in Erwartungen umformuliert und an die niedrigeren Ebenen propagiert. So erwartet man z. B. beim Betreten der eigenen Wohnung eine ganz bestimmte Türklinke. Wenn sich die Klinke ein klein wenig schwergängiger anfühlt oder irgendwie verformt ist, bemerkt man dies sehr schnell (als Abweichung vom Muster). Hawkins' einflussreiche Arbeit hat die Schwachstelle, dass es ihr in entscheidenden Bereichen an Konkretisierung fehlt. Das Framework ist auch dahingehend unvollständig, dass es Handlung auf Basis von Motivation und Emotion nicht erklärt. Dennoch sind daraus sog. „Hawkins Machines“ hervorgegangen, die, laut Minsky, erst sehr schnell Lernerfolge zeigten, dann langsamer wurden und schließlich zu keinen neuen Erkenntnissen mehr fähig waren [Min12].

In seinem, 2007 erschienenen, Buch „The Emotion Machine“[Min07] und in [MSS04]

⁵Der Neocortex ist die Oberfläche des Haupthirnes.

⁶Als Muster sind Signal-Abfolgen gemeint, die bestimmte Ähnlichkeiten aufweisen zu anderen Signalfolgen. Z. B. teilen sich die Signalfolgen „1,3,5,7“ und „5,7,9“ das Muster „aufsteigende ungerade Zahlen“.

setzt sich Minsky explizit mit dem Thema „Baby Machines“ auseinander. Er berichtet, dass es bereits Baby-Machine-Ansätze und auch funktionierende Maschinen gab. Diese waren jedoch nach kurzer intensiver Lernzeit nicht mehr in der Lage, neues Wissen aufzunehmen oder bestehendes Wissen umzuorganisieren. Minsky führt dies, je nach verwendetem Ansatz auf verschiedene Aspekte zurück. Das wesentliche Problem hat jedoch alle Kritik gemein: Der Mensch wird mit der Fähigkeit geboren, neue meta-kognitive Funktionen zu erlernen. Das sind Lern- und Gedächtnisstrategien, die dazu dienen, das eigene Wissen besser zu organisieren oder effektiver zu lernen. Keine bisherige Architektur verfügt über derartige Funktionen. Außerdem müsste man einen neuen Weg finden, Wissen im Computer zu repräsentieren (gegenüber Konnektionismus und Symbolismus). Dennoch hält Minsky den Baby-Machine-Ansatz prinzipiell für korrekt [Min07].

4 Neue Architektur BMA1

Es gibt also zahlreiche Ansätze und Ideen zur Umsetzung von Baby Machines. Dennoch ist bisher jede Form der Intelligenz, die in Computern vorkommt, an jeweils einen bestimmten Bereich gebunden. Turing äußerte dazu bereits „*We cannot expect to find a good child machine at the first attempt*“ [Tur50]. Bisher versuchen die Architekturen bestehende psychologische und informatische Erkenntnisse zusammenzubringen. Dagegen versucht der folgende Ansatz, bestehende Kognitive Architekturen zusammenzubringen, um damit Schwierigkeiten mit einzelnen Architekturen zu überbrücken. Genau genommen soll die Lösung aus bestehenden Architekturen und aktuellen kognitiven Erkenntnissen zusammengefügt werden. Der vorliegende Artikel kann keinesfalls eine fertige Lösung für eine BM anbieten. Er stellt eine erste Niederschrift eines neuen Ansatzes dar und umfasst (nur) das äußere Modell. Mit Wurzeln in zahlreichen wissenschaftlichen Arbeiten, soll es in Zukunft als Grundlage für weitere Forschung und erste Testimplementierungen dienen.

Die Problemstellung lautet: Eine Architektur für eine BM schaffen, die über grundlegende Lernmethoden sowie Motivationen und simulierte Emotionen verfügt. Aus diesen sollen selbstständig Handlungsmöglichkeiten erschlossen und wahrgenommen werden. Außerdem soll es möglich sein, die Architektur mit weitgehend beliebigen Aktuatoren⁷ und Sensoren auszustatten, deren Bedienung sich die angestrebte BM selbst erschließt. Zur einfachen Referenzierung wird die Architektur BMA1 genannt, für „Baby-Machine-Architecture 1“. Die „1“ steht für die erste Version des Modells. Ein Teil der bisherigen Ergebnisse sind in Abbildung 1 dargestellt. Im Folgenden werden die Komponenten kurz beschrieben, um deren Zusammenspiel zu erläutern. Deren konkrete interne Funktion ist zum einen zu umfangreich für den Artikel, zum anderen ist die konkrete Ausgestaltung der Komponenten noch nicht abgeschlossen. Jedoch wird die Funktion jeder Komponente und ihr Zusammenspiel mit anderen Komponenten umrissen. Auf das Lernzentrum wird detaillierter eingegangen, da darin der intelligente Kern der Architektur liegt. Und die Sensorischen Einheiten sind für das Verständnis des Lernzentrums nötig.

⁷Aktuatoren sind Geräte, die von einem Computer gesteuert werden können, um mit der Realität zu interagieren. Das können z. B. Motoren, Lautsprecher, Thermostate oder auch digitale Anzeigen sein.

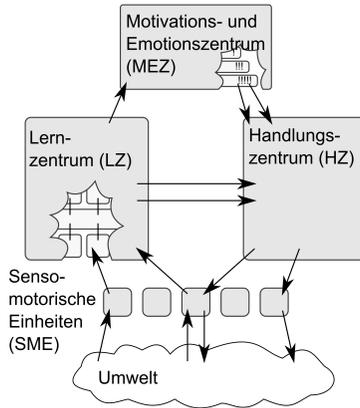


Abbildung 1: Die Abbildung enthält die Komponenten der äußeren Architektur.

4.1 Sensomotorische Einheiten

Die Sensomotorischen Einheiten (SME) sind Schnittstellen der Architektur zur Umwelt. Eine Einheit besteht jeweils aus Sensoren und/oder Aktuatoren für eine bestimmte Aufgabe. Dazu gehören Treiber für die jeweilige Hardware und jeweils ein Vorverarbeitungsprogramm, das die sensorischen Daten auf ein einheitliches Format bringt. Ein Beispiel für die akustische Wahrnehmung und Handlung wäre eine Modul aus Mikrofon und Lautsprecher, welche vom Computer angesprochen werden können. Dazu würden Audiotreiber für diese Geräte gehören; außerdem ein Programm, welches die eingehenden Signale in ein bestimmtes Format für die Architektur bringt. Ausgehende Signale der Architektur müsste das Programm ebenfalls in das jeweilige Format für die Hardware bringt.

Jede Wahrnehmung fließt als ein „Kanal“ in das Lernzentrum (LZ). Wenn ein Signal mehrere unabhängige Eigenschaften aufweist, werden diese Eigenschaften in getrennten Kanälen weitergeleitet. Z. B. lassen sich akustische Signale aufteilen in Frequenz und Amplitude, die unabhängig voneinander sind. Die meisten Signale lassen sich in reelle Zahlen umwandeln oder als solche darstellen. Ein Kanal ist also ein Strom aus reellen Zahlen.

Um maximales Potential der Hardware zu ermöglichen, werden durch die SME keine Handlungsoptionen vorgegeben, die mit der jeweiligen Hardware möglich sind. Die Maschine muss sich die Möglichkeiten der Hardware selbst erschließen. Praktisch bedeutet das, dass das Handlungszentrum beliebige Signale an einen Aktuatoren-Kanal senden kann. Reglementiert wird dies nur von simulierten „Schmerzgrenzen“. Das sind Grenzwerte, die vor dem Einsatz in der jeweiligen SME für einen Kanal definiert werden. Deren Überschreitung führt zu Warnimpulsen im LZ. Kinder kommen auch ohne klare Vorstellung auf die Welt, wie z. B. die Stimme zu benutzen ist. Sie lernen die Nutzung sowie die Grenzen durch ausprobieren.

4.2 Lernzentrum

Das Lernzentrum ist das Kernstück der Architektur, da hier Abstraktion, Analogienbildung und Kreativität liegen. Im Lernzentrum gehen Kanäle von SME und vom Handlungszentrum (HZ) ein. Die Kanäle aus dem HZ sind besonders gekennzeichnet, da diese die eigenen Handlungen widerspiegeln. Sie gehen jedoch nicht direkt ein, sondern über die SME.

Wenn man ein Buch liest, und man erfasst langsam die Bedeutung eines neuen Satzes, dann laufen zahlreiche Prozesse im Hintergrund ab, die betrachtet werden müssen. Was die Augen wahrnehmen, ist ein weißer Hintergrund, auf dem sich schwarze Linien befinden. Diese Linien werden automatisch als wiederkehrende, bekannte Einheiten aufgefasst, als Buchstaben. Mit jedem dieser Buchstaben können verschiedene Dinge assoziiert werden, z. B. deren Klang bei der Aussprache. Wenn einem jeweils eine Anordnungen von Buchstaben vertraut ist, erkennt man sie als Wörter wieder. Zu jedem Wort hat man verknüpfte Erinnerungen, nämlich die Bedeutungen der Wörter. Manchmal kommen einem zu einem Wort mehrere Bedeutungen in den Sinn, aber aus dem Umfeld ergibt sich meist die intendierte. Ein Satz fügt nun diese Reihe von Bedeutungen aneinander und es erschließt sich einem der Sinn des Satzes. Außerdem bringt man den Satz in Kontext zu vorangegangenen Sätzen des Buchkapitels und zur eigenen Erwartungshaltung gegenüber dem Buch. Die Wahrnehmung beziehungsweise das Lernen lässt sich also scheinbar als mehrere Ebenen oder Schichten beschreiben, ganz nach Minsky und Hawkins. Jede Schicht gleicht einer Abstraktionsebene. Das bedeutet, die unterste Schicht arbeitet direkt auf den eingehenden Signalen, höhere Schichten sind für die Bedeutung der Signale zuständig und die jeweils obersten Schichten verwalten ganze Realitäten. Im Buchbeispiel nimmt die unterste Ebene schwarze Linien wahr, die nächste Buchstaben, die nächste Wörter, usw.. Die höchste Ebene speichert, dass man sich gerade in der Realität befindet, nicht in einem Traum oder Videospiel. Von dieser hängt alles Darunterliegende ab, denn man geht in einen Traum oder in ein Videospiel zwar mit Erwartungen an die Physik und an bestimmte Abläufe, aber jedes System, jede Realität kann ihre eigenen Gesetze und Muster besitzen.

Der Aufbau des LZ ist hierarchisch und besteht in jeder Ebene aus Agenten, die Mustererkennung durchführen, auf Signalen aus tieferen Ebenen. Muster werden mit Wahrscheinlichkeiten ausgestattet und häufig vorkommende Muster werden zuerst überprüft.⁸ An einem Mustererkennungsprozess eines Signals sind jeweils mehrere Agenten beteiligt, die unterschiedliche Muster vorschlagen (z. B. bei Wörtern mehrere Bedeutungen). Nach der Theorie von Eagleman wählt die höhere Ebene aus den Vorschlägen aus und versucht ihrerseits die gewonnenen Muster zu ordnen. Was auf einer niedrigeren Ebene ein Muster aus Signalen ist, ist auf der nächsthöheren Ebene ein einziges Signal. Man spricht dann von einem „Symbol“. In der Wort-Ebene ist nicht von Belang, wie die Schriftart des Textes war. Auf der Ebene werden nur Strukturen und Abfolgen von Wörtern betrachtet.

Die Anzahl der Ebenen ist nicht fest vorgegeben. Wenn sich ein Signal im gleichen Kontext anders verhält als erwartet, wird ein neuer Kontext (also ein neues Symbol auf der nächsten Ebene) geschaffen. Ist diese Ebene noch nicht vorhanden, wird sie angelegt.

⁸Douglas Hofstadter hat zu diesem Thema und zur Intuition eindrucksvolle Arbeiten geleistet. [Hof95]

Menschliche Wahrnehmung funktioniert ähnlich. Die Hirnoberfläche, der Neocortex, besteht größtenteils aus 6 Schichten, von denen ebenfalls jede abstrahiert und Muster erkennt [GCP02]. Jedoch ist im Mensch die sechste Schicht nicht gleichbedeutend die abstrakteste Wahrnehmung. Signale können aus der sechsten Schicht von einem Bereich in die erste Schicht eines anderen Bereiches fließen. Damit kann die Anzahl der Ebenen auch nach Bedarf erhöht werden.

Dieses Modell weißt also Merkmale des Symbolismus auf, da in jeder Schicht mit Symbolen gearbeitet wird. Ebenso enthält es Eigenschaften des Konnektionismus, da die Wahrscheinlichkeiten beziehungsweise Gewichte der Muster gelernt werden. Die Kreativität des Systems äußert sich unter anderem, wenn Muster erkannt werden, die untypisch für den jeweiligen Bereich sind. Mit anderen Worten, wenn ein Muster in einem Bereich gelernt und auf einen anderen Bereich übertragen wird, so ist dies ein kreativer Vorgang [Hof95]. Angenommen man nimmt mit den Augen eine Katze wahr, dann ist damit z. B. gemeint, dass man dieses innere Muster der Katze in Muskelbewegung überführen kann. Dies kann sich äußern, indem man mit der Hand ein Bild einer Katze zeichnet oder die Bewegung einer Katze nachahmt. Hier spielt das Handlungszentrum ebenfalls eine große Rolle; dieses wird als nächstes beleuchtet.

4.3 Handlungszentrum

Das HZ extrahiert aus dem Lernzentrum Muster und speichert diese. Es werden solche ausgewählt, die zur Änderung der Umwelt oder der Befriedigung von Bedürfnissen beitragen (dazu mehr im nächsten Abschnitt). Außerdem verwaltet es die Ausführung von Aktionen. Dazu ist es in der Lage, beliebige Signale im standardisierten Format an das SME zu senden. Jede Signalisierung von Aktionen wird auch an das Lernzentrum gemeldet. Das LZ kann so Verbindungen herstellen zwischen Signalimpulsen und deren Wirkung auf die eigenen Aktuatoren. Weiterhin ist es fähig, Experimente durchzuführen, indem von gelernten Handlungsmuster bestimmte Abläufe variiert werden. Das Zentrum ist vergleichsweise jung in der Modellierung und muss weiterhin ausgebaut und detaillierter ausgestaltet werden.

4.4 Motivations- und Emotionszentrum

Alle Handlungen haben in diesem Bereich ihre Rechtfertigung. Dieses Zentrum ist verantwortlich für die Festlegung von Zielen, sowie die Wahl des Weges zur Zielerreichung. Auf Grundlage von Maslows Bedürfnishierarchie⁹ werden der Maschine verschiedene Bedürfnisse einprogrammiert. Die Maschine soll, wie der Mensch, stets versuchen, ihre Bedürfnisse zu befriedigen. Wenn jeweils eine Bedürfnisseebene ausreichend erfüllt ist,

⁹In Kurzform: pyramidenförmiges Modell; ganz unten Grundversorgung, Überleben; darüber Sicherheit und Ordnung; darüber Zugehörigkeit, Zuneigung; darüber Selbstwert und Erfolg; darüber Selbstverwirklichung. Nach Maslow werden Bedürfnisse der nächsthöheren Ebene aktiviert, wenn die unterliegenden Bedürfnisse ausreichend gedeckt sind [Mas70].

wächst der Drang nach Befriedigung von höheren Bedürfnissen [Mas70]. Über die tatsächliche Ausgestaltung der Bedürfnispyramide der BM muss weitere wissenschaftliche Arbeit erfolgen. Hier spielen Fragen der Ethik eine Rolle, ebenso wie soziale und psychologische. Man kann sich aber vorstellen, dass die tiefste Ebene stets für die Gewährleistung der einwandfreien Funktion des Programms zuständig ist. Anders ausgedrückt müsste das Programm mit Computersensoren¹⁰ ausgestattet werden und würde so stets dafür Sorge tragen, keine Operationen durchzuführen, die die Hardware überfordern würde. Höhere Ebenen würden sich dann je nach Anspruch an die BM variieren lassen.

Weiterhin prüft das Motivations- und Emotionszentrum, ob im Handlungszentrum Muster hinterlegt sind, die erlauben, ein jeweiliges Bedürfnis zu befriedigen. Sollten keine Benötigten vorhanden sein, wird versucht, durch Experimente und Interaktion, neue Muster zu erschließen. Zur weiteren Entwicklung werden hierfür sozialpsychologische Betrachtungen notwendig sein. Wenn jedoch Muster vorhanden sind, die in einer jeweiligen Situation ein bestimmtes Bedürfnis befriedigen können, so werden diese Muster als Ziele gespeichert. Sobald es möglich wird, werden die zielgerichteten Muster ausgeführt. Sollten Unstimmigkeiten auftreten, erfolgt eine Neuberechnung von geeigneten Vorgehen. Es wird Journal geführt über die vergangenen Ziele und gewählten Muster, um zu vermeiden, dass Endlosschleifen entstehen. Möglicherweise wäre es sinnvoll, die gewählten Ziele und Muster auch an das Lernzentrum zu melden, damit dies in gleicher Weise lernen kann, welche Situationen welche Maßnahmen bedingen.

5 Fazit

Die umrissene Architektur BMA1 kann genutzt werden, um zu lernen, sich eigene Ziele zu stecken und zu handeln. Darüber hinaus ist sie in der Lage, beliebige Geräte zur Interaktion mit der Realität zu nutzen, sofern sich diese zu Sensomotorischen Einheiten umfunktionieren lassen. Sie verfügt über eine geringe Menge an Vorwissen und kann sich mithilfe menschenähnlicher kognitiver Prozesse selbstständig neues Wissen aneignen. Das gilt natürlich unter der Prämisse, dass die Architektur umgesetzt werden kann und daran wird derzeit gearbeitet¹¹. Damit wird der Einblick in den Architekturvorschlag abgeschlossen. Es gibt zahlreiche Baustellen und das Vorhaben wird noch einige Zeit in Anspruch nehmen. Zum Abschluss wird darauf aufmerksam gemacht, dass es nach wie vor Akzeptanzprobleme mit dem Gebiet der KI-Forschung gibt. Geprägt durch die Vorurteile aus dem „KI-Winter“ ist es schwer, von öffentlicher Seite Gehör zu finden. Und selbst wenn Menschen an die Realisierbarkeit des Vorhabens glauben, beschwören viele von ihnen Horrorszenerien herbei. Dies könnte unter anderem von Filmen wie Matrix und Terminator herrühren, die einen glauben machen wollen, dass Maschinenintelligenz automatisch eine Unterwerfung der Menschheit bedeutet. Die Debatte, ob intelligente Computer Fluch oder Segen werden, ist verdammt, ohne Lösung zu bleiben, sofern man sich mit Vorurteilen vor der Idee verschließt und sich weigert, das Gebiet wissenschaftlich anzugehen.

¹⁰z. B. CPU- und RAM-Sensor, Anzahl der Schreib-/Lesevorgänge auf der Festplatte

¹¹...wie ebenso am Ausbau der Architektur

Literatur

- [AFQS08] John R Anderson, Jon M Fincham, Yulin Qin, and Andrea Stocco. A central circuit of the mind. *Trends in Cognitive Sciences*, 12(4):136–143, 2008.
- [AG01] John R Anderson and Kevin A Gluck. What role do cognitive architectures play in intelligent tutoring systems. *Cognition Instruction Twentyfive years of progress*, 79(March):227–262, 2001.
- [Bbc12] Bbc.co.uk. http://news.bbc.co.uk/2/hi/uk_news/358913.stm, abgerufen 25.07.2012.
- [Bra06] Ron Brachman. Getting Back to The Very Idea. *AI magazine*, 26(4):48–50, 2006.
- [Eag11] David M Eagleman. *Incognito*. Canongate Books Ltd, Edinburgh, 2011.
- [GCP02] Geoffrey J Goodhill and Miguel A Carreira-Perpinan. Cortical Columns. In *Encyclopedia of Cognitive Science*. Macmillan Publishers Ltd., 2002.
- [HB04] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Holt Paperbacks, New York, 2004.
- [Hof95] Douglas R Hofstadter. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, volume 13. Basic Books, 1995.
- [Kur05] Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*, volume 2011 of *A Penguin Book: Science*. Viking, 2005.
- [Lan04] Alex de Landgraaf. Implementing the Mind - Cognitive architectures. *FAAI*, 2004.
- [LLR06] Jill Fain Lehman, John E Laird, and Paul S Rosenbloom. A Gentle Introduction to Soar, an Architecture for Human Cognition: 2006 Update. *Science*, 4(0413013):1–37, 2006.
- [Mar12] John Markoff. <http://www.nytimes.com/2010/10/10/science/10google.html>, abgerufen 25.07.2012.
- [Mas70] Abraham H Maslow. *Motivation and Personality*, volume 2. Harper & Row, 1970.
- [Min86] Marvin Minsky. *Society of Mind*, volume 83. Simon and Schuster, 1986.
- [Min07] Marvin Minsky. *The emotion machine : commonsense thinking, artificial intelligence, and the future of the human mind*. Simon & Schuster, New York, 2007.
- [Min12] Marvin Minsky. <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-868j-the-society-of-mind-spring-2007/>, abgerufen 26.07.2012.
- [MSS04] Marvin Minsky, Push Singh, and Aaron Sloman. The St. Thomas Common Sense Symposium: Designing Architectures for Human-Level Intelligence. *AI Magazine*, 25(2):113–124, 2004.
- [RN04] Stuart Russel and Peter Norvig. *Künstliche Intelligenz*. Pearson Studium, 2 edition, 2004.
- [Sin03] Push Singh. Examining the society of mind. *Computing and Informatics*, pages 1001–1023, 2003.
- [SN58] Herbert A. Simon and Allen Newell. Heuristic problem solving: The next advance in operations research. *Operations research*, 6(1):1–10, 1958.
- [Tur50] Alan M Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

Softwareentwicklung ohne Implementierung von Entwurfsmustern

Nils Rixin

nrexin@imn.htwk-leipzig.de

Hochschule für Technik, Wirtschaft und Kultur Leipzig

Master Studiengang Informatik

Entwurfsmuster sind aus der Softwareentwicklung nicht mehr wegzudenken, seitdem Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides 1992 Entwurfsmuster in der Softwaretechnik beschrieben haben. Deshalb wird die These untersucht, dass Entwurfsmuster in zukünftigen Programmiersprachen nicht explizit implementiert werden müssen, sondern deren Einsatz durch sprachspezifische Features möglich ist. Eine weitere These beschreibt, dass Entwurfsprobleme in funktionalen Programmiersprachen eine klarere Lösung besitzen als in objektorientierten Programmiersprachen. Anhand eines Beispiels werden beide Thesen nachvollzogen. Bei diesem Beispiel wird eine objektorientierte Programmiersprache mit einer funktionalen Programmiersprache verglichen.

Angezapft – Technische Möglichkeiten einer heimlichen Online-Durchsuchung und der Versuch ihrer rechtlichen Bändigung

Rainer Rehak

rehak@informatik.hu-berlin.de

Humboldt-Universität zu Berlin, Diplom Studiengang Informatik

Die dem Poster zugrunde liegende Arbeit hat die heimliche Online-Durchsuchung zum Thema, eine verdeckte staatliche Maßnahme zur Informationsgewinnung mittels Infiltration informationstechnischer Systeme. Sie wurde und wird aktuell von deutschen Behörden verwendet, wobei ihr Einsatz hoch umstritten ist ([BB09, Seite 9] und [Bra11, Seite 686]).

Methodisch werden aus Aussagen von Akteuren der Politik und den gesellschaftlich-rechtlichen Rahmenbedingungen die konzeptionellen Anforderungen, Funktionen und Eigenschaften der Software einer solchen Maßnahme entwickelt. Aus dieser konzeptionellen Beschreibung können innerhalb der aktuellen Computersystemarchitektur (z.B. das Rechtekonzept [Tan08, Seite 2] oder das Datei-/Metadatenkonzept [Tan08, Seite 257 ff.]) konkrete technische Eigenschaften abgeleitet werden.

An der so konstruierten prototypischen Software können dann die resultierenden technischen Eigenschaften und Fähigkeiten analysiert werden: Um effektiv und effizient heimlich [Bun07, Antwort zu Frage 5] im zu infiltrierenden System suchen und insgesamt agieren zu können, muss die Software tief darin eingreifen (z.B. für den verlangten direkten RAM-Zugriff [Bun07, Antwort zu Frage 13]) und erlangt so allumfassende Kontrollmöglichkeiten auf Betriebssystemebene. Folglich kann z.B. keine zuverlässige Kryptographie mehr auf dem kompromittierten System betrieben werden [And08, Seite 147 ff.].

Zusammen mit der Anerkennung, dass private Lebenswelt und Alltag des normalen Bürgers immer mehr zu Prozessen informationstechnischer Verarbeitung werden [Kur09, Seite 2] und der Bezugnahme auf das Verfassungsgerichtsurteil zur Online-Durchsuchung aus dem Jahre 2008 und den dort geforderten technisch-rechtlichen Beschränkungen entstehen folgende Ergebnisse:

1. Eine Funktionsbeschränkung der Software, z.B. als Quellen-Telekommunikationsüberwachung, kann weder sichergestellt noch protokolliert (belegt) werden, daher muss immer die maximale rechtliche Eingriffshürde zur Anwendung kommen.
2. Erlangte Daten haben grundsätzlich keinen Beweiswert.
3. Der Kernbereich privater Lebensgestaltung ist praktisch immer betroffen, technischer (syntaktischer) Kernbereichsschutz ist prinzipiell nicht möglich.

4. Eine Trennung von Telekommunikations- und Nichttelekommunikationsdaten ist technisch-konzeptionell nicht hinreichend lösbar.

Diese Resultate widersprechen der aktuellen politischen Praxis und implizieren, dass Exekutive und Legislative ihr Verständnis der Online-Durchsuchung konsequent korrigieren müssen.

Literatur

- [And08] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed System*. John Wiley & Sons, New York, 2 edition, 2008.
- [BB09] Ulf Buermeyer and Matthias Bäcker. Zur Rechtswidrigkeit der Quellen-Telekommunikationsüberwachung auf Grundlage des §100a StPO. *Onlinezeitschrift für Höchstgerichtliche Rechtsprechung zum Strafrecht (HRRS)*, (10):433–441, 2009.
- [Bra11] Frank Braun. Ozapftis – (Un)Zulässigkeit von „Staatstrojanern“. *Kommunikation & Recht*, (11):681–686, 2011.
- [Bun07] Bundesregierung der 16. Wahlperiode. Drucksache 16/4795 – Antwort der Bundesregierung auf die Kleine Anfrage der Abgeordneten Gisela Piltz, Sabine Leutheusser-Schnarrenberger, Jörg van Essen, weiterer Abgeordneter und der Fraktion der FDP, 10.04.2007 2007. <http://dipbt.bundestag.de/dip21/btd/16/049/1604997.pdf>.
- [Kur09] Constanze Kurz. Kernbereichsschutz im digitalen Zeitalter. *Betrifft Justiz*, (100):164–168, December 2009.
- [Tan08] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3 edition, 2008.

Verbesserung der Performanz der SPARQL-Komponente vom DL-Learner

Didier Cherix

didier.cherix@gmail.com

Universität Leipzig, Bachelor Studiengang Informatik

Der DL-Learner ist ein Framework, um Konzepte in Beschreibungslogiken zu lernen. [Leh09] Er nutzt Machine Learning Algorithmen, um Lernprobleme in OWL zu lösen. Um die Flexibilität zu erhöhen ist der DL-Learner aus Komponenten gebaut. Es existieren 4 verschiedene Typen von Komponenten: **Knowledge Sources** werden benutzt, um Hintergrundwissen zu importieren. Nicht nur lokales Wissen in Form von RDF/XML oder Turtle sind unterstützt, sondern auch die Verwendung eines SPARQL-Endpunktes ist mittels der SPARQL-Komponente möglich. **Reasoner components** liefern Schnittstellen zu zahlreiche Reasoner. Der DL-Learner kann existierende standards Reasoner oder ein eigenes verwenden. **Learning Problems** beschreiben die zu lösende Lernprobleme. Es gibt 3 verschiedene Lernproblemtypen im DL-Learner: 1. Lernen von positiven Beispielen; 2. Lernen von positiven und negativen Beispielen; 3. Lernen von Class-Axiome.

Die SPARQL-Komponente wurde überarbeitet, um ihre Performanz zu erhöhen. Um die Effizienz zu erhöhen wurden Schemaanfragen (T-Box) von den Anfragen zu den tatsächlichen Instanzen (A-Box) getrennt. Aus A-Box und T-Box setzt sich die Ontologie zusammen [BHS08]. Diese Überarbeitung setzt sich aus einer Indexierung der T-Box, der Reduzierung der Anzahl der SPARQL-Anfragen und einer Typisierung der Ontologie. Die Indexierung der T-Box speichert für jede Klasse der Ontologie alle Ihre SuperKlassen bis owl:Thing. Die SPARQL-Anfragen wurden so verändert, dass die A-Box Anfragen getrennt von einer T-Box Anfrage erfolgen. Jede Anfrage holt alle zu den bis jetzt bekannten Instanzen verbunden Instanzen. Es erfolgt in der neuen Version eine A-Box Anfrage pro Iteration bis zur maximalen Tiefe. Die alte Version der Komponenten hat eine Anfrage pro Beispiel genutzt. Im nächsten Schritt wird für jede der im vorherigen Schritt hinzugefügten Instanzen eine Anfrage zum Endpunkt geschickt. Die T-Box Anfrage ist zuständig, um jede Instanz ihre Klasse zuzuweisen. Schliesslich muss die Ontologie Typisiert werden, da die Ressourcen in den SPARQL-Ergebnisse nicht typisiert sind, das heisst eine Klasse ist nicht als owl:Class deklariert. In der alten Version werden die T-Box Informationen gleichzeitig mit den A-Box Ressourcen geholt.

Um die Effizienz der Überarbeitung zu testen haben wir einen Test entwickelt. Dieser Test nutzt die DBpedia [ABK⁺07]. Für jede Klasse der DBpedia¹ wurden zufällig Beispiele unter deren Instanzen ausgewählt. Diese Beispiele wurden im Lernproblem als positive Beispiel verwendet. Die negativen Beispiel wurden unter den Schwesternklassen ausgewählt. Eine Schwesterklasse zu einer Klasse teilt eine ihrer Superklassen mit der Ur-

¹http://downloads.dbpedia.org/3.6/dbpedia_3.6.owl

sprungsklasse ohne diese zu sein. Von allen Schwesternklassen wurde eine zufällig ausgewählt und unter ihren Instanzen die negativen Beispiele. Dieser Test wurde mit der alten SPARQL-Komponente und mit ihrer überarbeitete Version durchgeführt.

Als erster grosse Unterschied kam heraus, dass die überarbeitete Komponente mehr Ergebnisse bekam als die alte Version. Dies ist zu erklären durch eine robustere Implementierung. Die überarbeitete Komponente ist weniger empfindlich zu den Serialisierung Fehlern des SPARQL-Endpunktes. Alle bis zum Fehler gefundene Instanzen können weiter verarbeitet werden und den Reasoner übergeben werden, bei der alten Komponente ist der OWL-Fragment nicht weiter nutzbar und es gibt kein Ergebnis. Es gaben insgesamt 61 JavaExceptions² bei der Durchführung der neuen Komponente, davon haben nur 3 ein Abbruch des Algorithmus erzwingt. Bei der alten Version waren es 93 Exceptions, die alle zu einem Abbruch des Algorithmus führten.

Die Performanz zeigt eine etwa vierfache Verbesserung der Geschwindigkeit. Die Durchschnittszeit der SPARQL-komponent beträgt vor der Überarbeitung 16817 ms und danach 3923 ms. Die Indexierung der T-Box bei der neuen Version braucht nur circa 2s. Die Indexierung erfolgt aber nur einmal am Anfang des Experiments. Die neue Version besitzt keine Cache-Möglichkeit. Die Ergebnisse zeigen jedoch, dass dies kein Nachteil gegenüber der alten Version, die ein Cache besitzt, ist.

Die Korrektheit wurde überprüft mittels der F-Werte. Jede Klassendefinition, die der DL-Learner ausgibt wird auf ihre Genauigkeit mit Hilfe der F-Wert gemessen. Als Referenz dienen die Instanzen die zu dieser Klasse gehören. Die Verteilung der F-Wert zeigt, dass die neue Komponente etwas bessere Ergebnisse liefert, beziehungsweise mehr Ergebnisse hat, die einen besseren F-Wert haben, liefert. Dies ist zum Teil durch den höheren Anzahl an Ergebnisse zu erklären. Ausserdem hat die Typisierung der Ontologie einen Einfluss auf die Ergebnisse, da mit ihrer Hilfe mehr Informationen vom Reasoner abgearbeitet werden können.

Literatur

- [ABK⁺07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, pages 722–735, 2007.
- [BHS08] F. Baader, I. Horrocks, and U. Sattler. Description logics. *Foundations of Artificial Intelligence*, 3:135–179, 2008.
- [Leh09] J. Lehmann. DL-Learner: learning concepts in description logics. *The Journal of Machine Learning Research*, 10:2639–2642, 2009.

²<http://docs.oracle.com/javase/6/docs/api/java/lang/Exception.html>

Autorenverzeichnis

Björn Buchwald

Software-Engineering, Datenbanken, Semantic Web, Geoinformationssysteme

Hochschule: Universität Leipzig, Bachelor Studiengang Informatik

E-Mail: mam09bhi@studserv.uni-leipzig.de

Beitrag: Integration von Geodaten in ein Planungssystem

Victor Christen

Software-Engineering, Semantic Web

Hochschule: Universität Leipzig, Master Studiengang Informatik

E-Mail: mam08bfa@studserv.uni-leipzig.de

Beitrag: REX – eine Webapplikation zur Visualisierung der Evolution von Ontologien in den Lebenswissenschaften

Didier Cherix

Semantic Web, Künstliche Intelligenz, Datenbanken

Hochschule: Universität Leipzig, Bachelor Informatik Studiengang

E-Mail: didier.cherix@gmail.com

Poster: Verbesserung der Performanz der SPARQL-Komponente vom DL-Learner

Mario Frank

Theoretische Informatik

Hochschule: Universität Potsdam, Diplomstudiengang Informatik

E-Mail: Mario.Frank@Uni-Potsdam.de

Beitrag: Relevanzbasiertes Preprocessing für automatische Theorembeweiser

Florian Golemo

Künstliche Intelligenz, Software-Engineering

Hochschule: Universität Leipzig, Bachelor Studiengang Informatik

E-Mail: fgolemo@gmail.com

Beitrag: Künstliche Intelligenz auf Basis von „Baby Machines“: Geschichte und Architektur BMA1

Jörg Hartmann

E-Business, Service-Engineering

Hochschule: Universität Leipzig, Studiengang Diplom Informatik

E-Mail: jhartmann@informatik.uni-leipzig.de

Beitrag: Generierung von Serviceverträgen auf Basis objektorientierter ereignisgesteuerter Prozessketten

Maciej Janicki

Computerlinguistik, Maschinelles Lernen, Theoretische Informatik

Hochschule: Universität Leipzig, Master Studiengang Informatik

E-Mail: macjan@o2.pl

Beitrag: A Lexeme-Clustering Algorithm for Unsupervised Learning of Morphology

Nick Robert Kempka

Theoretische Informatik, Social Web

Hochschule: Universität Leipzig, Bachelor Studiengang Informatik

E-Mail: n.kempka@studserv.uni-leipzig.de

Beitrag: Ein Algorithmus für die Erzeugung Facebook-ähnlicher Netzwerkstrukturen basierend auf dem Barabási-Albert-Modell

Max Kießling

RubyOnRails, Performanceoptimierung, Treemap

Hochschule: Fraunhofer MOEZ, Bachelor Studiengang Informatik

E-Mail: max.kiessling@moez.fraunhofer.de

Beitrag: Visualisierung einer Taxonomie als Treemap und Optimierung der Generierung in einer Web-Applikation

Marcus Nitzschke

Semantic Web

Hochschule: Universität Leipzig, Master Studiengang Informatik

E-Mail: mam08crr@studserv.uni-leipzig.de

Beitrag: Systematic review about the academic impact of DBpedia

Rainer Rehak

Computer Sicherheit

Hochschule: Humboldt-Universität zu Berlin, Diplom Studiengang Informatik

E-Mail: rehak@informatik.hu-berlin.de

Poster: Angezapft – Technische Möglichkeiten einer heimlichen Online-Durchsuchung und der Versuch ihrer rechtlichen Bändigung

David Georg Reichelt

Theoretische Informatik

Hochschule: Universität Hamburg, Master Studiengang Informatik

E-Mail: davidgeorg_reichelt@yahoo.de

Beitrag: Untersuchung von Beschleunigungsverfahren für den Dijkstra-Algorithmus im Zusammenhang mit multimodalem Routing

Nils Rexin

Software-Engineering

Hochschule: Hochschule für Technik, Wirtschaft und Kultur Leipzig, Master Studiengang Informatik Informatik

E-Mail: nrexin@imm.htwk-leipzig.de

Poster: Softwareentwicklung ohne Implementierung von Entwurfsmustern

Michael Schmeißer

Software-Engineering, Betriebssysteme

Hochschule: Hochschule für Technik, Wirtschaft und Kultur Leipzig, Master Studiengang Informatik

E-Mail: michael@skamandros.de

Beitrag: Metriken und optimale Einsatzszenarien für Garbage Collectoren der Java HotSpot Virtual Machine

Annett Schumann

E-Commerce, Social-Commerce, Soziale Netzwerke

Hochschule: Hochschule für Wirtschaft, Technik und Kultur Leipzig, Master Studiengang General Management

E-Mail: annett.schumann@stud.htwk-leipzig.de

Beitrag: Ein Algorithmus für die Erzeugung Facebook-ähnlicher Netzwerkstrukturen basierend auf dem Barabási-Albert-Modell

In der Reihe „Leipziger Beiträge zur Informatik“ sind erschienen:

Band I FÄHNRICH, K.-P.; HERRE, H. S. (Hrsg.): *Content- und Wissensmanagement. Arbeiten aus dem Forschungsvorhaben PreBIS und Beiträge auf den Leipziger Informatik-Tagen 2003*. Leipziger Beiträge zur Informatik: Band I. Leipzig, 2003. – ISBN 3-934178-26-X

Band II FÄHNRICH, K.-P.; MEIREN, T. (Hrsg.): *Computer Aided Engineering. Arbeiten aus dem Forschungsvorhaben ServCase*. Leipziger Beiträge zur Informatik: Band II. Leipzig, 2004. – ISBN 3-934178-39-1

Band III FÄHNRICH, K.-P.; THRÄNERT, M.; WETZEL, P. (Hrsg.): *Umsetzung von kooperativen Geschäftsprozessen auf eine internetbasierte IT-Struktur*. Arbeiten aus dem Forschungsvorhaben Integration Engineering. Leipziger Beiträge zur Informatik: Band III. Leipzig, 2005. – ISBN 3-934178-52-9

Band IV FÄHNRICH, K.-P.; KÜHNE, S.; SPECK, A.; WAGNER, J. (Hrsg.): *Integration betrieblicher Informationssysteme – Problemanalysen und Lösungsansätze des Model-Driven Integration Engineering*. Leipziger Beiträge zur Informatik: Band IV. Leipzig, 2006. – ISBN: 978-3-934178-66-3

Band V FÄHNRICH, K.-P.; HÄRTWIG, J.; KIEHNE, D.-O.; WEISBECKER, A. (Hrsg.): *Technologien und Werkzeuge für ein rollen- und aufgabenbasiertes Wissensmanagement. Zusammenfassender Bericht aus dem Forschungsprojekt PreBIS*. Leipziger Beiträge zur Informatik: Band V. Leipzig, 2007. – ISBN: 978-3-934178-76-2

Band VI FÄHNRICH, K.-P.; THRÄNERT, M.; WETZEL, P. (Hrsg.): *Integration Engineering. Motivation – Begriffe – Methoden – Anwendungsfälle*. Leipziger Beiträge zur Informatik: Band VI. Leipzig, 2007. – ISBN: 978-3-934178-78-6

Band VII AUER, S.: *Towards Agile Knowledge Engineering: Methodology, Concepts and Applications*. Dissertation an der Fakultät für Mathematik und Informatik der Universität Leipzig. Leipziger Beiträge zur Informatik: Band VII. Leipzig, 2007. – ISBN: 978-3-934178-73-1

Band VIII FÄHNRICH, K.-P.; HEYER, G. (Hrsg.): *Games Summer Camp 2007. Interdisziplinäres Blockseminar zum Thema Digitale Spiele*. Eine Dokumentation. Leipziger Beiträge zur Informatik: Band VIII. Leipzig, 2007. – ISBN: 978-3-934178-77-9

Band IX ASLAM, M. A.: *Towards Integration of Business Processes and Semantic Web Services*. Leipziger Beiträge zur Informatik: Band IX. Leipzig, 2008. – ISBN: 978-3-934178-89-2

Band X FÄHNRICH, K.-P.; MÜLLER, R.; MEYER, K.; FREITAG, M. (Hrsg.): *Entwicklung internationaler produktbezogener Dienstleistungen – Ein Handlungsleitfaden für kleine und mittlere Unternehmen*. Leipziger Beiträge zur Informatik: Band X. Leipzig, 2008. – ISBN: 978-3-934178-98-4

Band XI FÄHNRICH, K.-P.; KÜHNE, S.; THRÄNERT, M. (Hrsg.): *Model-Driven Integration Engineering. Modellierung, Validierung und Transformation zur Integration betrieblicher Anwendungssysteme*. Leipziger Beiträge zur Informatik: Band XI. Leipzig, 2008. – ISBN: 978-3-941152-02-1

Band XII MAICHER, L.; GARSHOL, L. M. (Hrsg.): *Subject-centric Computing. Fourth International Conference on Topic Maps Research and Applications, TMRA 2008*. Leipziger Beiträge zur Informatik: Band XII. Leipzig, 2008. – ISBN: 978-3-941152-05-2

Band XIII FÄHNRICH, K.-P.; SCHUMACHER, F. (Hrsg.): *Digitale Spiele in Forschung und Lehre. Beiträge zum Games Summer Camp 2008*. Leipziger Beiträge zur Informatik: Band XIII. Leipzig, 2009. – ISBN: 978-3-941608-00-9

Band XIV HEYER, G. (Ed.): *Text Mining Services – Building and applying text mining based service infrastructures in research and industry. Proceedings of the conference on Text Mining Services – TMS 2009 at Leipzig University*. Leipziger Beiträge zur Informatik: Band XIV. Leipzig, 2009. – ISBN: 978-3-941608-01-6

Band XV THRÄNERT, M.: *Integration-Engineering – Grundlagen, Vorgehen und Fallstudien*. Leipziger Beiträge zur Informatik: Band XV. Leipzig, 2009. – ISBN: 978-3-941608-02-3

Band XVI FÄHNRICH, K.-P.; ALT, R.; FRANZCYK, B. (Eds.): *Practitioner Track – International Symposium on Services Science (ISSS'09)*. Leipziger Beiträge zur Informatik: Band XVI. Leipzig, 2009. – ISBN: 978-3-941608-03-0

Band XVII MEYER, K.: *Software – Service – Co-Design: Eine Methodik für die Entwicklung komponentenorientierter IT-basierter Dienstleistungen*. Leipziger Beiträge zur Informatik: Band XVII. Leipzig, 2009. – ISBN: 978-3-941608-04-7

Band XVIII AUER, S.; LAUENROTH, K.; LOHMANN, S.; RIECHERT, T. (Hrsg.): *Agiles Requirements Engineering für Softwareprojekte mit einer großen Anzahl verteilter Stakeholder*. Leipziger Beiträge zur Informatik: Band XVIII. Leipzig, 2009. – ISBN: 978-3-941608-05-4

Band XIX MAICHER, L.; GARSHOL, L. M. (Eds.): *Linked Topic Maps. Fifth International Conference on Topic Maps Research and Applications, TMRA 2009*. Leipziger Beiträge zur Informatik: Band XIX. Leipzig, 2009. – ISBN: 978-3-941608-06-1

Band XX HÄRTWIG, J.: *Konzept, Realisierung und Evaluation des semantischen Informationsraums. Dissertation*. Leipziger Beiträge zur Informatik: Band XX. Leipzig, 2010. – ISBN: 978-3-941608-07-8

Band XXI MORGENSTERN, U.; RIECHERT, T. (Hrsg.): *Catalogus Professorum Lipsiensis. Konzeption, technische Umsetzung und Anwendungen für Professorenkataloge im Semantic Web*. Leipziger Beiträge zur Informatik: Band XXI. Leipzig, 2010. – ISBN: 978-3-941608-08-5

Band XXII LEHMANN, J.: *Learning OWL Class Expressions*. Leipziger Beiträge zur Informatik: Band XXII. Leipzig, 2010. – ISBN: 978-3-941608-09-2

Band XXIII MEYER, K.; THIEME, M. (Hrsg.): *Vom Bedarf zum Innovationserfolg – In 6 Schritten gemeinsam Potentiale aktivieren*. Leipziger Beiträge zur Informatik: Band XXIII. Leipzig, 2010. – ISBN: 978-3-941608-10-8

Band XXIV MAICHER, L.; GARSHOL, L. M. (Eds.): *Information Wants to be a Topic Map. Sixth International Conference on Topic Maps Research and Applications, TMRA 2010*. Leipziger Beiträge zur Informatik: Band XXIV. Leipzig, 2010. – ISBN: 978-3-941608-11-5

Band XXV HEYER, G.; LUY, J.-F.; JAHN, A. (Hrsg.): *Text- und Data Mining für die Qualitätsanalyse in der Automobilindustrie*. Leipziger Beiträge zur Informatik: Band XXV. Leipzig, 2010. – ISBN: 978-3-941608-12-2

Band XXVI FÄHNRICH, K.-P.; SCHUMACHER, F.; THIEME, M.; GROSS, J. (Hrsg.): *(Über-)Leben in der Kreativwirtschaft - Beiträge zum Creative Summer Camp 2011*. Leipziger Beiträge zur Informatik: Band XXVI. Leipzig, 2011. – ISBN: 978-3-941608-13-9

Band XXVII AUER, S.; RIECHERT, T.; SCHMIDT, J. (Hrsg.): *Studentenkonferenz Informatik Leipzig 2011*. Leipziger Beiträge zur Informatik: Band XXVII. Leipzig, 2011. – ISBN: 978-3-941608-14-6

Band XXVIII GEBAUER, M.; STEFAN, F.: *Systemintegration - Eine qualitative Erhebung aus der Sicht von Integrationsdienstleistern*. Leipziger Beiträge zur Informatik: Band XXVIII. Leipzig, 2011. – ISBN: 978-3-941608-15-3

Band XXIX MEYER, K.; BÖTTCHER, M. (Hrsg.): *Entwicklungspfad Service Engineering 2.0 – Neue Perspektiven für die Dienstleistungsentwicklung*. Leipziger Beiträge zur Informatik: Band XXIX. Leipzig, 2011. – ISBN: 978-3-941608-16-0

Band XXX KERN, H.; KÜHNE, S. (Hrsg.): *Integration Betrieblicher Informationssysteme und modellgetriebene Entwicklung*. Leipziger Beiträge zur Informatik: Band XXX. Leipzig, 2012. – ISBN: 978-3-941608-17-7

Band XXXI BÖTTCHER, M.; KLINGNER, S.; BECKER, M.; SCHUMANN, K. (Hrsg.): *Produktivität von Dienstleistungssystemen – Ergebnisse eines Arbeitskreises*. Leipziger Beiträge zur Informatik: Band XXXI. Leipzig, 2012. – ISBN: 978-3-941608-18-4

Band XXXII GRÄBE, H.-G.; GROEPLER-ROESSER, I. (Hrsg.): *MINT – Zukunft schaffen – Innovation und Arbeit in der modernen Gesellschaft*. Leipziger Beiträge zur Informatik: Band XXXII. Leipzig, 2012. – ISBN: 978-3-941608-19-1

Band XXXIII RIECHERT, T.: *Eine Methodologie für agiles und kollaboratives Requirements-Engineering (Dissertation)*. Leipziger Beiträge zur Informatik: Band XXXIII. Leipzig, 2012. – ISBN: 978-3-941608-20-7

Band XXXIV SCHMIDT, J.; RIECHERT, T.; AUER, S.: *Dritte Studentenkonzferenz Informatik Leipzig 2012*. Leipziger Beiträge zur Informatik: Band XXXIV. Leipzig, 2012. – ISBN: 978-3-941608-21-4

Weitere Informationen und Bestellungen über:
<http://www.infai.org/>