

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCHE-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

RaQuN_N – Suchbasierter
N-Wege-Vergleich – ein Verfahren zur
Bestimmung der Ähnlichkeit von
Modellen in der modellbasierten
Softwareentwicklung

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

eingereicht von: David Császár
geboren am: 24.11.1975
geboren in: Leisnig
Gutachter: Prof. Dr. Timo Kehrer
Prof. Dr. rer. nat. Lars Grunske

eingereicht am: 28.03.2019

Inhaltsverzeichnis

Kurzfassung	3
Abstract	3
Danksagung	3
1. Einleitung	5
1.1. Motivation	5
1.2. Begriffe	7
2. Herangehen und verwendete Vorarbeit	8
3. Der suchbasierte Vergleich von N Modellen	10
3.1. Indizierung und Suche	10
3.2. Der Merge	14
3.3. Die Reihenfolge bei der Bildung des Modells	17
4. Komplexitätsanalyse	19
4.1. Einlesen der Daten	19
4.2. Indizierung	19
4.3. Suche	19
4.4. Matching und Merge	20
5. Testablauf	22
5.1. Die Messgröße „Weight“	22
5.2. Die Messgröße „Laufzeit“	22
5.3. Vorgefertigte Testdaten	23
5.4. Generierte Testdaten	24
5.5. Der variable Suchradius	26
6. Testergebnisse	27
6.1. Laufzeitfaktor τ	27
6.2. Gewicht und Laufzeit in Abhängigkeit vom Suchradius	28
6.3. Weight und Time im Vergleich zu Rubin NwM	29
6.4. Verteilung der Laufzeit	31
6.5. Weight und Time mit Look-Ahead	32
7. Fazit	35
8. Kritik und Ausblick	36
9. Literaturverzeichnis	38

Anhang	40
A. Software	40
B. Dimensionen für die Indizierung als k-d-Vektoren	41
C. Aus Open-Source-Projekten extrahierte Modelle	43
D. Messergebnisse der Mikro-Benchmarks	46
Selbstständigkeitserklärung	49

Kurzfassung

Diese Arbeit zeigt, wie mehrere Softwaremodelle effizient miteinander verglichen werden können. Dazu wird ein neues Verfahren zum gleichzeitigen Zusammenführen von N Modellen (N-Way-Merge) entwickelt, basierend auf den Ergebnissen einer Umkreissuche (Range-Query): RaQuN_λ.

Um das Verfahren zu bewerten, wird gezeigt, dass die Qualität der Resultate und die benötigte Rechenzeit die Werte eines etablierten Verfahrens verbessern.

Abstract

This thesis shows how to compare multiple software models in an efficient manner. A novel approach to N-way model merging will be demonstrated:

RaQuN_λ fetches model elements from a central search index using a range query. Afterwards, it merges all N models into a single model based on the results of the range query.

To verify the efficiency of this technique, it is shown that the quality of the outcome as well as the measured run time improve upon the results of an established method.

Danksagung

Danke an:

Die HU Berlin
für die angemessene Haltung
von Langzeitstudenten



Timo Kehret
für richtungsweisende Fragen und
die Vermittlung des Kontakts
zu Julia Rubin.



Andromeda Nebel
für kreative Inspiration.

Angelika Teschner
für eine
Arbeitsumgebung
voller Lieblingsplätze.



Die Infopark AG,
insbesondere:

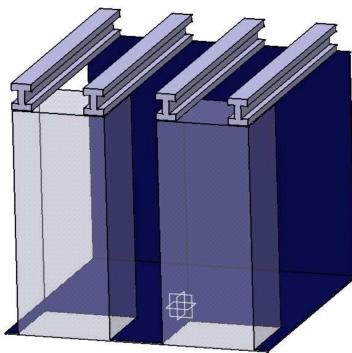
Tilo Pritz,
für ein spannendes Rennen
um das letzte Diplom und

Thomas Witt,
der in keiner Danksagung fehlt darf!

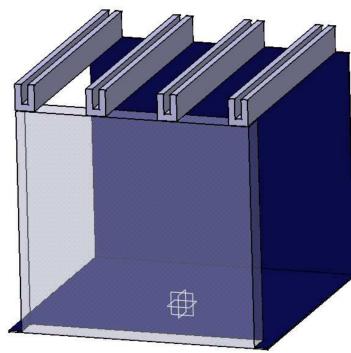
1. Einleitung

1.1. Motivation

In der Softwareentwicklung kann ein Vergleich von bestehenden oder geplanten Software-Anwendungen und -Komponenten objektive Grundlage von strategischen Entscheidungen sein.



Objekt 1



Objekt 2

Abbildung 1. Zwei ähnliche Gestaltungselemente. (aus Champin 2003)

Dabei ist nicht nur der Vergleich von zwei Varianten interessant, der regelmäßig, zum Beispiel beim Review von Änderungen, durchgeführt wird. Auch der Vergleich mehrerer Versionen kann wichtige Erkenntnisse liefern, hier einige Anwendungsgebiete:

Identifikation von Produktlinien und „Clone Unification“: In der Softwareentwicklung kann Interesse daran bestehen, einen Überblick über parallele Softwareentwicklungen in verschiedenen Projekten zu gewinnen. Bei starker Ähnlichkeit von mehreren Anwendungen lässt sich die Architektur anpassen und gemeinsame Teile in mehrfach genutzte Micro-Services oder Module auslagern. Dies reduziert die zu wartende Menge an Quelltext und spart dadurch Kosten.
(nach Rubin 2014:1)

Architekten und Entwickler können durch den Vergleich mehrerer Versionen oder Software-Familien Erkenntnisse für die Weiterentwicklung und Nutzung eines Systems ablesen. Die bei diesen Vergleichen identifizierten Ähnlichkeiten und Varianzen liefern zusätzliche

Informationen, die über eine durch Analyse einzelner Versionen gewonnenen Perspektive hinausgehen. (vergleiche Sabetzadeh 2007:2)

Im Gegensatz zum Vergleich konkreter Implementierungen (z.B. Quelltext) ermöglicht es der Vergleich von Modellen, diese Vorteile auch über Systemgrenzen (z.B. unterschiedliche Programmiersprachen, Datenbanken, oder Frameworks) hinweg zu nutzen. Zudem kann ein noch nicht implementiertes System dadurch bereits in der frühen Modell-Phase kritisch betrachtet und verschiedene Varianten verglichen werden. Diese und andere Vorteile von Modell-Abstraktion machen die Arbeit mit Modellen in Forschung und Praxis zunehmend beliebt. (vergleiche Martinez 2016:396)

Andererseits birgt die Arbeit auf Modellebene im Vergleich zu traditionellen Ansätzen, die sich direkt auf den Quelltext beziehen, zusätzliche Komplexität. So stellt gerade der Fall „Vergleich“ beziehungsweise „Merge“ bei Modellen, aufgrund ihrer graphartigen Struktur, eine besondere Herausforderung dar. (nach Levendovszky 2010:252)

Die tatsächliche Nutzung der oben erwähnten Techniken hängt sehr stark von der Qualität und Effizienz der eingesetzten Werkzeuge ab. Bietet ein Verfahren keinen ausreichend hohen Mehrwert, oder ist der Mehraufwand zu hoch, wird es in der Praxis nicht eingesetzt. (vergleiche Rubin 2014:13)

Prinzipielle Herausforderungen bei Vergleichen, unabhängig vom Grad der Abstraktion und der Anzahl zu vergleichender Dinge, sind: Die Erkennung gemeinsamer Merkmale, die eindeutige Zuordnung von Elementen, und die Auswahl der besten Entsprechung bei mehreren Kandidaten. Zur Veranschaulichung zeigt Abbildung 1 exemplarisch zwei ähnliche Objekte. Die oberen vier Balken sind jeweils bei Objekt 1 und 2 ähnlich, bei der Zuordnung ist aber auch die Position des Balkens entscheidend. Bei der Front kann eine Zuordnung von zwei Elementen aus Objekt 1 zu einem Element von Objekt 2 sinnvoll sein, oder die Entscheidung, diese Elemente nicht als ähnlich anzusehen.

Ziel dieser Arbeit ist, ein Verfahren zu finden, mit dem sich Ähnlichkeiten von N Modellen analysieren lassen. Das Ergebnis soll die Ähnlichkeiten und Unterschiede besser als bestehende Methoden erkennen, und idealerweise auch weniger Aufwand verursachen.

Ein derartiges Werkzeug kann in der Praxis routinemäßig und automatisiert eingesetzt werden. Das könnte die Akzeptanz von Techniken, die auf Vergleichen von mehreren Modellen beruhen, steigern. Dies wiederum führt dazu, dass die damit verbundenen Sparpotenziale öfter genutzt werden.

1.2. Begriffe

Im Verlauf dieser Arbeit werden sowohl die gängigen englischsprachigen, als auch die deutschen Fachbegriffe verwendet. Hier die wichtigsten im Überblick:

- Die Begriffe „Merge“ und „Zusammenführen“ sind Synonyme für die Vereinigung von einzelnen Elementen aus mehreren Softwaremodellen, sowie für die Vereinigung mehrerer vollständiger Softwaremodelle.
- Die Begriffe „Model“ und „Modell“ sind jeweils die Kurzform von „Softwaremodell“.
- „N-Way“ und „N-Wege“ werden austauschbar für Operationen an mehreren (N) Modellen verwendet.
- „Weight“ und „Gewicht“ stehen für die von Rubin beschriebene Messgröße w . (Rubin 2013:303).
- „Time“, „Zeit“ und „Laufzeit“ werden austauschbar für erwartete oder gemessene Rechenzeit verwendet.
- Eine Bereichsabfrage in einem k-dimensionalen Suchindex wird in diesem Dokument „Range-Query“ oder sprechend „Umkreissuche“ genannt. Mögliche Ausprägungen solcher Suchanfragen sind bei Bentley beschrieben. (vergleiche Bentley 1975:512)

2. Herangehen und verwendete Vorarbeit

Dieses Kapitel wirft einen Blick auf die wissenschaftlichen Arbeiten, die als Grundlage dieser Arbeit gedient haben. Die darüber hinausgehenden eigenen Beiträge werden kurz erwähnt. Die ausführliche Beschreibung folgt im nächsten Kapitel.

Treude et al. beschreiben in „Difference computation of large models“ ein suchbasiertes Verfahren zum Finden von Ähnlichkeiten in zwei Modellen. (Treude 2007)

RaQuN_N greift dieses Verfahren auf, wendet es aber auf mehrere Modelle an. Statt nur in zwei, findet der Algorithmus Ähnlichkeiten zwischen beliebig vielen Modellen (Anzahl N).

Rubin und Chechik beschreiben in „N-Way Model Merging“ verschiedene Verfahren, um N Modelle anhand von Ähnlichkeiten und Unterschieden zu einem einzigen Ergebnismodell zusammenzuführen. (siehe Rubin 2013)

RaQuN_N nutzt die bei Rubin beschriebenen Regeln, um anhand der gefundenen Ähnlichkeiten alle betrachteten Modelle zu einem Gesamtmodell zusammenzuführen.

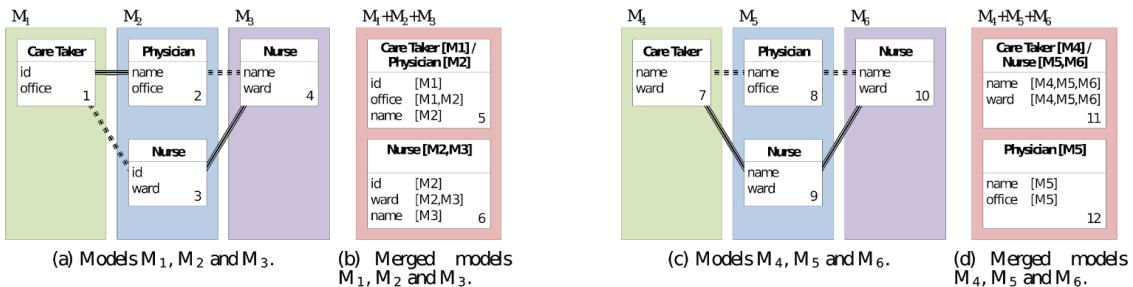


Abbildung 2. Beispiele für UML-Modelle und deren Zusammenführung. (aus Rubin 2013)

Da der Merge mit Hilfe von Vergleichen (compare) durchgeführt wird, und die Verweise auf die verglichenen Eingangs-Modelle erhalten bleiben (match), ist das Endergebnis (compose) des hier beschriebenen Verfahrens geeignet, um die Modelle miteinander zu vergleichen. Das resultierende Gesamtmodell beinhaltet Verweise auf alle Elemente der Quell-Modelle, sowie die ermittelte Zuordnung der Elemente untereinander.

Rubin liefert auch ein Messverfahren, um die Güte des Ergebnisses eines N-Way-Merge zu bewerten. Die von RaQuN[█] berechneten Gesamtmodelle werden der Güteprüfung von Rubin unterzogen.

Die Arbeit wird komplettiert durch Vergleiche mit alternativen Berechnungsmethoden und verschiedenen Eingabeparametern.

3. Der suchbasierte Vergleich von N Modellen

In diesem Kapitel wird das Verfahren RaQuN_λ im Detail beschrieben, gegliedert in die notwendigen Schritte. Die Komplexität dieser Schritte erläutert das nächste Kapitel.

3.1. Indizierung und Suche

Der naheliegende Ansatz beim Vergleich von zwei Modellen ist, dass alle möglichen Paare, die aus je einem Element der beiden zu vergleichenden Modelle bestehen, auf Ähnlichkeit geprüft werden, um die besten Übereinstimmungen zu finden. Alle denkbaren Paare zu untersuchen, hat jedoch eine zu hohe Komplexität: Ist die Anzahl der Elemente eines Modells n , ist der Aufwand $O(n^2)$. (Treude 2007:295)

Treude et al. beschreiben ein Verfahren, mit dem sich mit wenig Aufwand zwei Modelle vergleichen lassen: Um den Prozess zu beschleunigen, werden die Elemente eines Modells zuerst in einem k-dimensionalen Suchbaum indiziert. Als Basis dient hier der LSD-Tree (beschrieben in Henrich 1989), welcher wiederum auf dem k-d-Tree beruht (beschrieben in Bentley 1975).



Abbildung 3. Künstlerische Darstellung eines LSD-Baums (mit Raccoon).

Treude et al. entwickeln einen noch schneller durchsuchbaren Baum (den S³V-Tree), das Prinzip des k-d-Trees behalten sie aber bei:

Elemente werden im Suchraum als Punkte indiziert. Ein Punkt wird durch einen k-dimensionalen Vektor repräsentiert. Die Vektor-Dimensionen beschreiben dabei sowohl

lexikalische (zum Beispiel das Vorkommen eines Namensbestandteils) als auch numerische Eigenschaften der indizierten Elemente (zum Beispiel die Anzahl von Attributen in einer Klasse).

The vectors consist of a metrical and a lexical part. (Treude 2007:300)

Mit einer Umkreissuche (Range-Query) können in der resultierenden Datenstruktur alle hinreichend ähnlichen Paare schnell identifiziert werden. Die Laufzeitkomplexität verringert sich auf $O(n \log n)$. (Treude 2007:296)

Die vorliegende Arbeit greift dieses Verfahren auf, vereinfacht aber einige von Treude et al. implementierten Aspekte:

- Dem RaQuN \bowtie genügt ein k-d-Tree als Suchbaum.
- Auf Hashing als Optimierungsschritt wird verzichtet.
- Die Vektoren sind ungewichtet, es wird eine einfache Normalisierung der Vektorelemente auf den Bereich [0, 1] bevorzugt.
- Statt verschiedener Elementtypen der Modelle separat zu betrachten (z.B. Klasse, Operation, Attribut), werden alle Typen in einen gemeinsamen Baum indiziert. Daraus ergibt sich eine Anpassung:
- Um nicht fälschlich Elemente verschiedenen Typs als zu ähnlich zu behandeln, werden den Vektoren entsprechende Typ-Dimensionen hinzugefügt.

Diese Vereinfachungen sollen die Implementierung und Messung möglichst simpel und allgemeingültig machen. In der Praxis sind die von Treude beschriebenen Optimierungen durchaus sinnvoll. Die hier angestellten Betrachtungen sollen aber keine bereits optimierten Ergebnisse liefern, um Stellen mit Optimierungsbedarf deutlicher erkennen zu können.

In Anlehnung an die von Treude beschriebenen Vektordimensionen nutzt RaQuN \bowtie folgende „*Metrical Indices*“ (Treude 2007:300) für ein Element: Die jeweiligen Kehrwerte ($\frac{1}{x}$, wenn nötig mit Vermeidung einer Division durch Null: $\frac{1}{x+1}$) der Länge des Namens, der Anzahl von Attributen, Methoden und Referenzen. Die Einstufung $x \in \{0, 1\}$ als Element vom Typ Klasse, Attribut, Operation und Referenz.

Weitere Dimensionen ergeben sich durch eine an das Verfahren von Treude angelehnte lexikalische Analyse von Elementnamen beziehungsweise Elementnamensbestandteilen. Zu diesem Zweck werden alle vorkommenden Zeichenketten als „*Lexical Indices*“ (Treude 2007:301) verwendet. RaQuN_✓ berücksichtigt als lexikalische Dimensionen $x \in \{0, 1\}$ alle Namen von Elternelementen, Namen von Elementen, Substrings in Elementnamen (getrennt durch Binnenmajuskel oder Sonderzeichen) und Namen von Kindelementen. Ausführliche Beispiel- und Testdaten finden sich im Anhang B.

Um Paare (bzw. Tupel) aus mehr als zwei Modellen bilden zu können, lässt sich das Verfahren von Treude für N Modelle weiterentwickeln.

Ausgangspunkt: Eine Umkreissuche ist in der Lage, in der Menge aller indizierten Elemente diese zu finden, die einem Punkt im Vektorraum nahe sind. Der Mittelpunkt des Suchumkreis bestimmt sich dabei aus dem Referenzelement, zu dem ein Partner gefunden werden soll. Die Ähnlichkeit (im Sinne der gewählten Vektordimensionen) verhält sich invers zur Distanz zwischen Referenzpunkt und Suchtreffer. Oder: Je näher ein Treffer im Suchraum liegt, desto ähnlicher ist das zugehörige Modellelement.

Werden alle Elemente aller zu vergleichenden Modelle in einen gemeinsamen Index hinterlegt, ist eine Umkreissuche mit ausreichendem Suchradius r in der Lage, alle ähnlichen Elemente aus allen Modellen gleichzeitig zu liefern. Die Ähnlichkeit der resultierenden Elementpaare lässt sich aus den Suchtreffern ableiten. Sie entspricht dem Abstand des Treffers zum Suchmittelpunkt.

Bei je einer Suche für jedes Element ergibt sich die vollständige Menge aller Paare von ähnlichen Elementen.

Hierbei ergeben sich Besonderheiten, die bei der weiteren Betrachtung der Ergebnisse zu berücksichtigen sind:

- Da alle Elemente in einem gemeinsamen Index stehen, wird zu jedem Referenzelement das Element selbst als maximal ähnlich gefunden.
- In den Suchergebnissen können auch weitere Elemente aus dem Modell des Referenzelements vorkommen. Diese sind für das Ziel, Ähnlichkeiten zwischen Modellen zu finden, irrelevant. Ein solcher Treffer muss bei der Zuordnung ignoriert werden. Die Modellzugehörigkeit ist also wichtig und wird von RaQuN_✓ berücksichtigt.

- Ebenso können mehrere ähnliche Elemente aus einem anderen Modell gefunden werden.
Um nicht Ähnlichkeiten innerhalb von Modellen zu betrachten, darf in jeder Menge einander zugeordneter Elemente höchstens ein Element aus jedem Modell vorkommen.
- Die Distanz $d = 0$ markiert den Sonderfall einer maximalen Ähnlichkeit. Bei einer hinreichend gewählten Vektorrepräsentation entspricht dies einer kompletten Übereinstimmung. Die beiden Elemente eines solchen Paares sind gleich, also in beiden beteiligten Modellen unverändert enthalten.

Beim Vergleich von mehr als zwei Modellen ergeben sich nicht nur Paare aus ähnlichen Elementen (Zwei-Tupel), sondern Beziehungen mit mehreren Partnern. Bei N Modellen werden Tupel mit bis zu N Elementen betrachtet.

We [...] consider tuples rather than pairs of elements from multiple distinct input models.
(Rubin 2013:301)

Ein Element kann dabei in vielen Tupeln vorkommen, also mehreren Elementen ähnlich sein. Aufgrund der oben aufgeführten Einschränkungen werden die gefunden Ähnlichkeiten lediglich als Kandidaten betrachtet.

Der prinzipielle Algorithmus für das Finden aller Paare ist einfach. Als Eingabe dienen alle Elemente aller Modelle. Die Ausgabe ist die Menge aller Paare von Elementen, die ähnlich oder identisch sind:

$$E \leftarrow \bigcup_{i \in [0..n]} M_i \quad \text{Alle Elemente aus allen Modellen werden betrachtet.}$$

$$\hat{P} \leftarrow \bigcup_{e \in E} \text{range_query}(e, r) \quad \begin{aligned} \text{Das Ergebnis ist die Menge aller Paare, die von einer} \\ \text{Umkreissuche mit dem durch } e \text{ bestimmten Mittelpunkt} \\ \text{und dem Radius } r \text{ gefunden wird. Jede Umkreissuche ergibt} \\ \text{mindestens } \{(e, e)\}. \text{ Das Ergebnis enthält ausschließlich} \\ 2\text{-Tupel: } \forall p \in \hat{P}, |p| = 2. \end{aligned}$$

Algorithmus 1. Ablauf der Such-Phase bei RaQuN_{elephant}.

Im nächsten Schritt werden geeignete Beziehungen zwischen ähnlichen Elementen aus mehreren Modellen ausgewählt.

3.2. Der Merge

Der Merge-Schritt führt zu einem Gesamtmodell, in dem die Verweise auf die zusammengeführten Elemente hinterlegt sind.

Es gibt verschiedene Arten, N Modelle miteinander zu vereinigen. Naive Ansätze zerteilen das Problem in mehrere paarweise Operationen. Dies ist jedoch aufwändig und liefert stark von der Reihenfolge der einbezogenen Modelle abhängige Ergebnisse. Abbildung 2 (aus Rubin 2013) illustriert dieses „*n-way merging model problem*“ (Rubin 2013:302) anhand eines einfachen Beispiels: Bei paarweiser Zusammenführung von Modellen ist das Endergebnis in Abhängigkeit von der Reihenfolge unerwünscht (b) oder erwünscht (d).

Schon die Auswahl eines bestimmten Modells, mit dem die Vereinigung startet, kann in der Praxis zu Problemen führen, da sich nicht immer entscheiden lässt, welches Modell als Referenz in Frage kommt.

Rubin schlägt einen Ansatz vor, der mehrere Modelle gleichzeitig betrachtet: „*considering multiple models simultaneously*“ (Rubin 2013:302). Das hier vorgestellte neue Verfahren erfüllt diesen Anspruch. Es verzichtet auch auf die Auswahl eines Referenzmodells.

RaQuN[↗] generiert ein neues Modell, in dem alle Elemente aller verglichenen Modelle enthalten sind. Als gleich oder ähnlich identifizierte Elemente werden dabei schrittweise zu Tupeln zusammengefasst. Als Grundlage dienen die von Rubin und Chechik (in Rubin 2013) vorgestellten Verfahren zum N-Way-Model-Merging.

Zuerst übernimmt der Algorithmus von RaQuN[↗] alle Quell-Elemente in das Zielmodell. Damit ist die Anforderung, alle Elemente im Ergebnis zu haben, erfüllt.

Im nächsten Schritt wird eine Merge-Strategie angewandt, die sich an dem von Rubin beschriebenen NwM-Ansatz orientiert:

Für jedes im Suchergebnis als ähnlich bestimmte Paar wird geprüft, ob die beiden Partner im Zielmodell zusammengeführt werden können. Geeignete Paare werden im Zielmodell gemerkt, invalide werden ignoriert. Das Ergebnis ist somit unabhängig von jeglicher Reihenfolge, in

welcher sich die Modelle betrachten lassen. Es besteht jedoch eine Abhängigkeit von der Reihenfolge, in der die gefundenen Ähnlichkeitsbeziehungen zwischen den Elementen einfließen.

Beim paarweisen Merge werden verschiedene Aspekte betrachtet:

Die Gültigkeit der Zusammenführung. Die Tupelgröße muss im Bereich $(1, N)$ liegen, wobei jedes enthaltene Element aus einem anderen Modell stammen muss. Ähnlichkeiten von Elementen innerhalb eines Modells sind in dieser Arbeit nicht von Interesse. Jedes Element wiederum darf nur in genau einem Tupel vorkommen. Diese Gültigkeitsregeln entsprechen den Match-Kriterien „(a) All tuples are valid“ und „(b) All tuples are disjoint“ von Rubin. (Rubin 2013:303)

Die Güte des resultierenden Gesamtmodells. Als Messwert für die Qualität des Merge wird die Weight-Berechnung von Rubin angewendet: Jedes Tupel hat ein Gewicht $w(t)$, welches mit der Ähnlichkeit der enthaltenen Elemente steigt; das Gewicht des Gesamtmodells ist die Summe aller Gewichte $w(\hat{T})$. Je höher das Gewicht, desto besser ist die Güte des Ergebnisses. (vergleiche Rubin 2013:303)

Ein Paar, welches beim Merge das Gesamtgewicht nicht erhöht, wird verworfen.

Die Güte des Paars. Um ein Kriterium für eine geeignete Reihenfolge der Tupelbildung zu finden, wurden drei verschiedene Strategien untersucht, die im Anschluss erläutert werden. Alle Strategien gemeinsam ist die Beobachtung, dass die Güte des Gesamtmodells tendenziell steigt, wenn beim Merge frühzeitig Tupel mit hoher Güte gebildet werden. Diese Strategie wird auch von Rubin propagiert:

The map is ordered by the weight of the matches, from the largest to the smallest, so that the “strongest” matches are retrieved first. (Rubin 2013:307)

Diese Tendenz erklärt sich aus der Annahme, dass mit steigender Anzahl von bereits gruppierten Elementen die Wahrscheinlichkeit für ungültige Matches steigt. Gute Kandidaten sollten daher

frühzeitig betrachtet werden, da die Wahrscheinlichkeit einer Ablehnung zu diesem Zeitpunkt geringer ist.

Hier der Algorithmus in einer Form, die an die von Rubin verwendete Notation angelehnt ist: (siehe Rubin 2013:306)

$\hat{P} \leftarrow \text{search}(M_i, r), i \in [0..n]$	Ausgangspunkt ist die Menge aller von der Suche gefundenen Paare von Elementen \hat{P} . $\forall p \in \hat{P}, p = 2$. \hat{P} kann Tupel mit Duplikaten enthalten.
$\hat{S} \leftarrow T, \forall t \in T, t = 1$	Alle Elemente werden als 1-Tupel dem Ergebnis \hat{S} hinzugefügt.
$\hat{P} \leftarrow \text{rsort}(\hat{P}, w)$	Die Paare werden absteigend nach Rubin-Weight sortiert.
<i>while</i> ($ \hat{P} > 0$) <i>do</i>	Solange noch Paare vorhanden sind,
<i>pick first</i> $p \in \hat{P}$	wähle das nächste Paar,
$\hat{P} \leftarrow \hat{P} - \{p\}$	entferne es aus der Liste der Paare.
$\hat{U} \leftarrow \hat{S}, \forall t \in \hat{S}, p_1 \in t \vee p_2 \in t$	Wähle alle Tupel aus der bisherigen Ergebnismenge aus, die Elemente aus dem betrachteten Paar enthalten.
$u \leftarrow \bigcup \hat{U} \cup \{p\}$	Erzeuge ein neues Tupel, welches die Elemente aller durch die betrachtete Paarbeziehung p verbundenen Tupel zusammenführt. Durch die Mengenoperation werden in diesem Schritt die Elemente des Tupels u unique.
<i>if</i> ($w(u) > w(\hat{U}) \wedge \text{isfit}(u)$) <i>then</i>	Prüfe, ob der Merge valide ist. Die Prüfung $\text{isfit}(u)$ verhindert die Zusammenführung von Elementen aus ein- und demselben Modell: $\forall i \in [0..n] \neg \exists e, e', e \in u \wedge e' \in u \wedge e \in M_i \wedge e' \in M_i \wedge e \neq e'$ Daraus ergibt sich $ u \leq n$.
$\hat{S} \leftarrow (\hat{S} - \hat{U}) \cup \{u\}$	Wenn der Merge valide ist, entferne alle ausgewählten Tupel \hat{U} aus dem Ergebnis und füge stattdessen das vereinigte Tupel hinzu.
<i>end if</i>	
<i>end while</i>	

Algorithmus 2. Ablauf der Merge-Phase bei RaQuN_{NS}.

Dieser iterative Prozess kann als ein Clustering-Vorgang angesehen werden. Nachdem alle Paare betrachtet sind, sind die ursprünglich einzelnen Elemente im Zielmodell zu einer Menge aus gültigen Tupeln zusammengewachsen. Jedes Tupel wiederum repräsentiert je eine Menge von Elementen, die RaQuN_✓ als modellübergreifend ähnlich beziehungsweise identisch zugeordnet hat.

Diese Daten stellen das gewünschte Ergebnis dar. Sie sind geeignet, um als Basis für weitere Auswertungen, visuelle Repräsentationen und Entscheidungshilfen zu dienen.

Da die potentiellen Anwendungen nicht Teil dieser Arbeit sind, wird im Folgenden lediglich die Laufzeit von RaQuN_✓ und das Gewicht des Ergebnis-Modells nach Rubin gemessen, um zu zeigen, dass das beschriebene Verfahren etablierte Gütekriterien erfüllt.

3.3. Die Reihenfolge bei der Bildung des Modells

Um die Komplexität des Verfahrens gering zu halten, ist es wünschenswert, dass die Reihenfolge, in der die ermittelten Paare ins Gesamtmodell integriert werden, einer einfachen Regel folgt.

Im Idealfall sollte die Ausgabe der Ähnlichkeitssuche direkt als Eingabe für den Merge fungieren. Der naivste Ansatz geht von der Tatsache aus, dass eine kurze Distanz im Suchbaum dem Grad der Ähnlichkeit entspricht. Eine plausible Annahme ist, dass dadurch ähnliche Paare bevorzugt (also zuerst) zusammengeführt werden. Dieser einfache Ansatz erhält hier den Namen „Greedy-Distance-Match“. In der Praxis zeigt sich, dass die Ergebnisse subjektiv valide erscheinen. Bei der Messung des Rubin-Weight sind die Ergebnisse aber schlechter als alle von Rubin beschriebenen Verfahren. Deshalb wird dieser Ansatz für diese Arbeit verworfen.

Bei näherer Untersuchung ergibt sich eine ausreichend geeignete Reihenfolge, wenn die gefundenen Paare nach ihrem individuellen Gewicht nach Rubin sortiert werden. Das Verfahren wird im Folgenden „Weight-Optimized-Match“ genannt. Es werden also, statt der Ähnlichkeiten im Sinne der Index-Vektoren, die Paare bevorzugt, die schwergewichtig im Sinne von Rubin sind. Die Erklärung für das bessere Ergebnis ist, dass die Indizierung die Elemente auf eine reduzierte Menge von Eigenschaften, nämlich die gewählten Indexvektor-Dimensionen, abbildet. Bei der

Wichtig nach Rubin jedoch werden die relevanten Eigenschaften der ursprünglichen Daten direkt betrachtet. Dieser Schritt gleicht also nachträglich die bei der Abbildung während der Indizierung eingeführten Ungenauigkeiten aus.

Aus den bisherigen Ausführungen geht hervor, dass kein optimales Ergebnis zu erwarten ist. Um ein Gefühl dafür zu entwickeln, wie viel „Luft nach oben“ das hier entwickelte Verfahren lässt, wird zusätzlich noch ein modifizierter Algorithmus mit erhöhter Komplexität getestet. Der „Weight-Optimized-Lookahead-Match“ wählt aus den verbleibenden vorsortierten Paaren nicht einfach das erste Paar aus, sondern ein Paar, welches das Gesamtgewicht des Modells am stärksten erhöht. Die Tiefe des Look-Ahead, also die Anzahl von Listeneinträgen, die in jedem Schritt erneut betrachtet werden, ist parametrisiert.

Das letztlich als hinreichend gut gewählte Verfahren ist jedoch der „Weight-Optimized-Match“.

4. Komplexitätsanalyse

Bevor im folgenden Kapitel eine Implementation von RaQuN^{elephant} mit konkreten Daten getestet wird, zuerst eine Betrachtung der Komplexität aus theoretischer Sicht:

Wie bereits gezeigt, durchlaufen die Modelldaten mehrere Phasen mit jeweils unterschiedlichen Transformationen. Es ist also sinnvoll, die Komplexität getrennt zu analysieren. Die verwendeten Faktoren sind aber für mehrere Phasen relevant, also definieren wir diese zuerst:

- N - Die Anzahl von verglichenen Modellen
- n - Die Gesamtanzahl von Elementen über alle Modelle.
- k - Die Anzahl von Vektor-Dimensionen im Suchraum. Dieser Wert ergibt sich, aufgrund der lexikalischen Vektorkomponenten, in Abhängigkeit von der Anzahl unterschiedlicher Klassen- und Attributnamen über alle verglichenen Modelle.
- \tilde{r} - Normalisierter Suchumkreis. Teile des Verfahrens sind vom Radius der Umkreissuche abhängig. Zu diesem Zweck steht \tilde{r} für einen Wert im Bereich $[0, 1]$, wobei 0 einem Radius von 0 entspricht, 1 einem idealisierten minimalen Suchradius, der zu jedem Element alle Elemente findet.

4.1. Einlesen der Daten

Der Zugriff auf die Daten, sowie die Analyse der Vektor-Dimensionen, geschieht rein sequentiell und ist mit $O(n)$ optimal.

4.2. Indizierung

Laut Bentley hat eine Einfügeoperation im k-d-Tree eine Komplexität von $O(\log n)$. (Bentley 1975:517)

Da jedes Element in den Suchbaum eingefügt wird, ergibt sich $O(n \log n)$.

4.3. Suche

Für jedes Element wird eine Suche durchgeführt, womit von $O(n \log n)$ auszugehen ist, da die Suche nach einem einzelnen nächsten Nachbarn im k-d-Tree $O(\log n)$ hat. (Bentley 1975:514)

Andere einsetzbare Verfahren zur Umkreissuche bewegen sich in ähnlichem Rahmen und erreichen, über alle Suchen summiert, beispielsweise zwischen $O(kn \log n)$ und $O(kn^2)$. (nach Liu 2006:1139)

Auch nach Treude ergibt sich insgesamt eine Spanne von $O(n \log n)$ bis $O(n^2)$. (Treude 2007:300)

Da die getestete Implementierung nächste Nachbarn sucht, aber eine Umkreissuche mit einem bestimmten Radius gefordert ist, müssen unter Umständen weitere nächste Nachbarn gesucht werden. Bei einer angenommenen Gleichverteilung der Elemente im Suchraum heißt das: Je größer der Radius, desto schneller wächst der Faktor: $O(n^{1+\tilde{r}} \log n)$. Im Worst-Case steigt die Komplexität der implementierten Suche auf $O(n^2 \log n)$.

Die Wahl eines geeigneten kleinen Radius ist also verfahrensbedingt ein kritischer Faktor.

4.4. Matching und Merge

Die Vereinigung der von der Suche gefundenen Paare erfolgt in verschiedenen Schritten, die jeweils ihre eigene Komplexität haben. Der gemeinsame Faktor ist die Anzahl der gefundenen Paare aus der Suchphase $O(n^{1+\tilde{r}})$, im schlimmsten Fall $O(n^2)$.

1. Die Sortierung der gefundenen Paare: $O(n^{1+\tilde{r}} \log n)$.
2. Für die Weight-Optimierung wird für jedes Paar, und anschließend für jedes potenzielle Tupel ein NwM-Weight berechnet: $O(wn^{1+\tilde{r}})$.
3. Die Prüfung, ob sich Tupel valide vereinigen lassen (beziehungsweise die tatsächliche Vereinigung), wird ebenso oft durchgeführt: $O(vn^{1+\tilde{r}})$.

In Summe: $O(n^{1+\tilde{r}} \log n) + O(wn^{1+\tilde{r}}) + O(vn^{1+\tilde{r}})$

Die Faktoren w und v enthalten in der verwendeten Implementierung jeweils Prüfungen auf Uniqueness der betrachteten Elemente, haben also ihrerseits wieder eine Worst-Case-Komplexität von $O(n^{1+\tilde{r}} \log n)$. Der schlimmste Fall beim Merge ist damit theoretisch $O(n^{2+2\tilde{r}})$, beim ungünstigsten Suchradius also $O(n^4)$.

Wie bereits in der Suchphase zeigt sich beim Merge eine enorme Abhangigkeit vom gewahlten Suchradius. Durch die nicht-trivialen Teilschritte fur Weight-Berechnung und Vereinigung tragen die Faktoren w und v hierbei wesentlich zur Steigerung der Laufzeit bei.

5. Testablauf

Dieses Kapitel beschreibt die Grundlagen der Tests, denen RaQuN_{SM} unterzogen wurde. Die Ergebnisse finden sich im darauffolgenden Abschnitt.

5.1. Die Messgröße „Weight“

Bei Rubin wird das Gewicht eines Tupels als Funktion der Properties eines Tupels und der Zahl der Input-Modelle beschrieben. Vereinfacht lässt sich der Wert als Summe der relativen Häufigkeiten aller am Match beteiligten Attribute in allen Elementen des Matches beschreiben. Diese Arbeit folgt der Berechnungsvorschrift von Rubin:

We [...] define the compare function to assign a high weight to tuples with a large number of properties shared by a large number of elements: $w(t)$ (Rubin 2013:303)

Wichtig für die Interpretation der Ergebnisse ist folgendes Detail: In der Erläuterung des Rechenverfahrens (vergleiche Rubin 2013:304) verwendet Rubin sowohl Klassennamen (CareTaker, Physician) als auch Attributnamen (id, office, name) als Property.

In Rubins Messungen an den Testdatensätzen (vergleiche Rubin 2013:308) werden die Klassennamen jedoch *nicht* als Property betrachtet. Diese Tatsache lässt sich anhand des von Rubin zur Verfügung gestellten Quellcodes und eigener Tests nachvollziehen. Eine Gewichtung ohne Klassennamen führt tendenziell zu einem höheren Weight bei den Random-Datensätzen (die unterschiedlichen Random-Klassennamen würden normalerweise das Gewicht verringern) und zu einem geringeren Weight bei den Real-World-Datensätzen (identische Klassennamen bei zusammengeführten Klassen würden normalerweise zu einem höherem Gewicht führen).

Um einen fairen Vergleich zu gewährleisten, wird in dieser Arbeit bei allen Messungen ebenfalls auf die Betrachtung der Klassennamen als Property verzichtet.

5.2. Die Messgröße „Laufzeit“

Rubin beschreibt die Testumgebung als „*Intel Core2Quad CPU 2.33GHz machine using JVM version 7*“. (Rubin 2013:304)

Die hier verwendete Testumgebung ist ein MacBook Pro (13 Zoll, 2016) 3,3 GHz Intel Core i7 mit Java(TM) SE Runtime Environment (build 1.8.0_121-b13).

Um Messungen der Laufzeit nicht optimistisch zu verfälschen, gelten folgende Vorschriften:

- Soweit möglich, wird auf die Nutzung paralleler Algorithmen verzichtet.
- Es wird ein Korrekturfaktor τ eingeführt, der durch drei verschiedene Testläufe der von Rubin gestellten Implementation und den von Rubin gemessenen Werten ermittelt wird. Hierbei sind alle Optimierungen der Laufzeitumgebung erlaubt. Es wird ein möglichst pessimistischer Faktor aus den Vergleichsmessungen ausgewählt.
- Alle Tests an RaQuN^λ müssen für jede Einzelmessung als Mikro-Benchmark laufen, um die Anzahl der Optimierungen der Laufzeitumgebung so gering wie möglich zu halten.

Um kein zu pessimistisches Ergebnis zu erzielen, durchläuft jeder Mikro-Benchmark zu Beginn einen Warm-Up, der dafür sorgt, dass alle benötigten Klassenbibliotheken von der Laufzeitumgebung geladen worden sind.

5.3. Vorgefertigte Testdaten

Rubin und Chechik liefern in ihrer Arbeit Testdaten mit, die in dieser Arbeit ebenfalls verwendet werden.

Dies sind zum Einen die aus der Arbeit von Talaei Rad und Jabbari (siehe Talaei Rad 2012) übernommenen Modelle zu *Warehouses* und *Hospitals*. Diese repräsentieren jeweils eine Softwarefamilie.

Zum Anderen stellt Rubin drei (nach wohldefinierten Kriterien) zufällig generierte Modellfamilien zur Verfügung. Diese sind insbesondere für objektive Messungen geeignet, da sie keine domain-spezifischen Merkmale aufweisen. Die drei Datensätze sind *Random*, welcher empirisch typische Verteilungen von Klassen und deren Attributen in Modellen nachbildet. *Random Tight* mit der Tendenz zu geringer Varianz bei der Modellgröße, weniger Attributen insgesamt und mehr Attributen pro Klasse – wie es sich bei *Hospitals* beobachten lässt. *Random Loose* simuliert die entgegengesetzten Trends aus *Warehouses*: mehr Varianz bei der Klassenanzahl, viele Attribute, aber pro Klasse nur wenige davon.

[T]he Hospital case appears to be more “tight” [...]. The Warehouse case [...] is more “loose”
(Rubin 2013:309)

Die Arbeit mit diesen vorgefertigten Testdaten ermöglicht es, die Ergebnisse mit den Ergebnissen von Rubin zu vergleichen.

Eine Besonderheit der Daten ist, dass die Modelle lediglich die Entitäten „Klassenname“ und „Attributname“ enthalten. Das hier gezeigte Verfahren ist prinzipiell in der Lage, weitere Entitäten (z.B. Operation, Relation) und Eigenschaften (z.B. Attributtyp) zu betrachten. Aufgrund der eingeschränkten Testdaten kommt diese Fähigkeit in den Messungen jedoch nicht zum Tragen.

Die Besonderheit bei der Messung ist, dass jedes der drei Random-Szenarien (bestehend aus jeweils 100 Modellen) in Teilmengen zu zehn Modellen zerlegt wird. Als Ergebnis der Messung wird der Durchschnitt der Einzelmessungen an diesen Teildaten verwendet.

For those cases, we merged each set of 10 models individually and averaged the results for all sets within a case. (Rubin 2013:308)

Alle Messungen an einem der drei Random-Datensätze folgen dieser Berechnungsvorschrift.

Für jeden der fünf Datensätze von Rubin werden Time und Weight des Ergebnisses ermittelt. Das Resultat wird den von Rubin mit dem NwM-Verfahren erreichten Werten gegenübergestellt.

5.4. Generierte Testdaten

Ein potentielles Problem in wissenschaftlicher Arbeit ist, dass sich die für die Arbeit verwendeten Testdaten als nicht generisch genug herausstellen. Dadurch können sich wissenschaftliche Resultate bei der Anwendung auf andere Daten als ungeeignet erweisen.

[W]hen the observed results cannot generalize to other case studies. (Rubin 2013:310)

Zum Beispiel konnte Alexander Pepper zeigen, dass ein wissenschaftlich propagiertes Verfahren zum Repository-Mining bei einem realen Repository nicht die vorhergesagten Resultate liefert. (vergleiche Prechelt 2014:1378)

Um dem entgegenzuwirken, soll das neue Verfahren auch an Modellen getestet werden, welche aus echten Softwareprojekten stammen.

Zu diesem Zweck werden automatisiert Modelle aus Open-Source-Projekten im Bereich JavaScript und ReactJS extrahiert. Als Verfahren genügt hierbei eine Heuristik, die durch Traversieren des Abstract-Syntax-Trees, und Übersetzung ausgewählter Elemente plausible Modelle der betrachteten Software liefert.

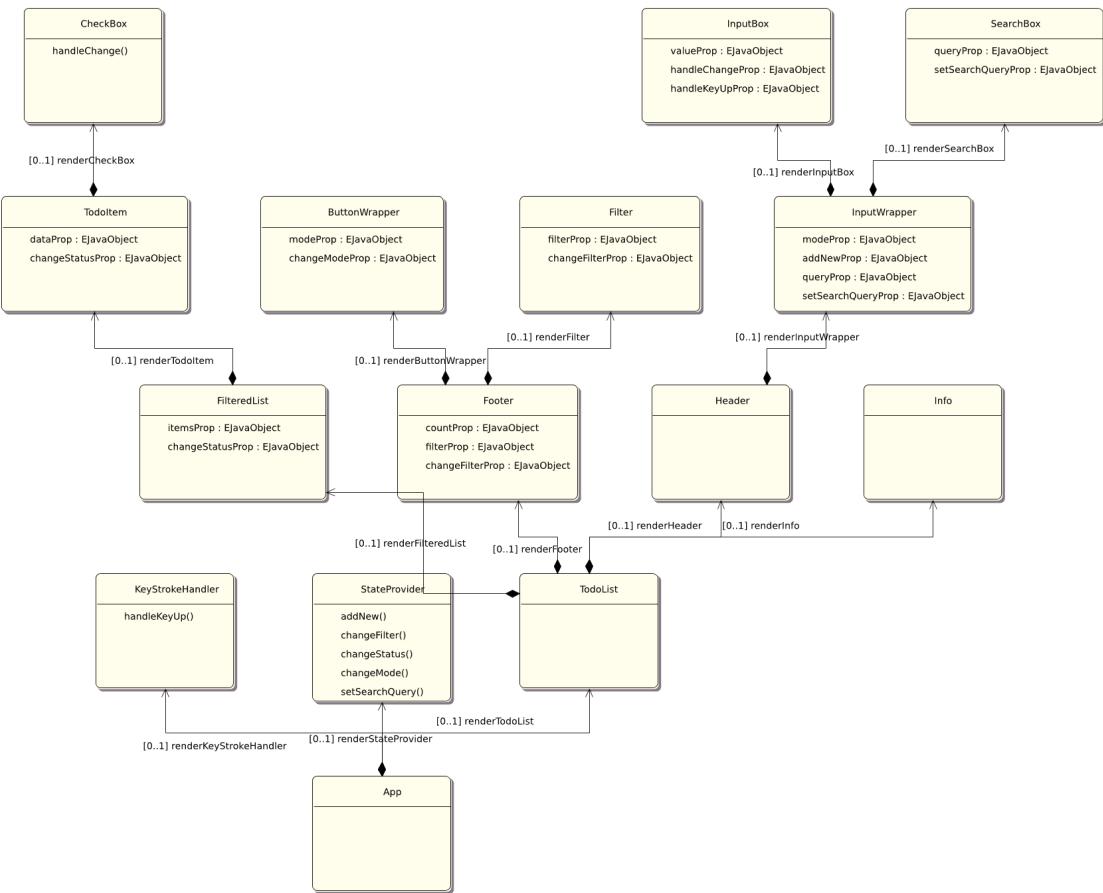


Abbildung 4. Ein automatisch aus „react-todo-app“ extrahiertes Modell.

Das Verfahren durchsucht die Versionsgeschichte eines Projekts auf GitHub, und findet Varianten in Form von Branches, Tags, und Forks. Bei sehr vielen Varianten sind die ausgewählten Versionen durch die Limits der GitHub-API quasi-zufällig.

Jede Variante wird nach einfachen Regeln anhand der vorgefundenen AST-Nodes zu einem Modell transformiert. Ist das Modell leer oder entspricht es einem bereits extrahierten Modell, wird es verworfen. Für eine bessere Vergleichbarkeit mit den Daten von Rubin werden die

Modelle auf Klassen und Attribute reduziert, indem gefundene Relationen und Operationen wie Attribute behandelt werden.

Diese Arbeit trifft keine konkrete Annahme über die Semantik der gefundenen Varianten. Unserer Erfahrung nach stellen Forks in der Open-Source-Community aber praktisch nie Softwarefamilien dar, sondern sind in der Regel Schnapschüsse der History, oder Entwicklungsstände von Features.

Aus verschiedenen extrahierten Repositorys wurden drei ausgewählt, die eine ausreichende Modell-Größenordnung und eine interessante Anzahl von Varianten aufweisen:

Datensatz, GitHub Repository	GitHub-User, Beschreibung, Lizenz	Anzahl Modelle N	$\bar{\Omega}$ Anzahl Klassen pro Modell	$\bar{\Omega}$ Anzahl Attribute pro Klasse
react-shopping-cart	jeffersonRibeiro, Simple ecommerce cart application built with React Redux, MIT	5	10,8	5,037
react-todo-app	kabirbaidhya, a sample react todo app done step-by-step, keine Lizenz	12	7,5	2,756
webamp	captbaritone, A reimplementation of Winamp 2.9 in HTML5 and Javascript, MIT	77	24,7	3,292

Tabelle 1. Übersicht der drei aus Open-Source-Repositorys extrahierten Modellfamilien.

5.5. Der variable Suchradius

Der in der Umkreissuche des hier beschriebenen Verfahrens gewählte Radius hat entscheidenden Einfluss auf die Qualität des Ergebnisses.

In den Messungen werden verschiedene Suchradien getestet, um empirische Erkenntnisse über den Einfluss dieser Größe zu gewinnen. Bei den Vergleichsmessungen mit den Testdaten von Rubin wird ein geeignet großer Suchradius gewählt. Die Veränderung von Gewicht und Laufzeit in Abhängigkeit vom Radius wird durch eine Messreihe sichtbar gemacht.

6. Testergebnisse

Dieses Kapitel stellt die Ergebnisse der Untersuchungen tabellarisch und grafisch dar.

6.1. Laufzeitfaktor τ

Die folgende Tabelle fasst die Messungen der Laufzeit ausgewählter, von Rubin getesteter, Merge-Verfahren auf der ursprünglichen und auf der für diese Arbeit verwendeten Testumgebung zusammen. Daraus lässt sich ein fairer Umrechnungsfaktors für nachfolgende Laufzeitmessungen ablesen.

Testlauf	Time laut Rubin	Time t_2 in s	Time t_3 in s	Faktor τ
Hospital <i>NwM</i>	42.7s	42,7	19,575	2.181354
Warehouse <i>NwM</i>	2.9m	174,0	64,532	2.696337
Warehouse <i>Gr3</i>	45.4s	45,4	16,392	2.769644

Tabelle 2. Gemessene Laufzeiten auf zwei zu vergleichenden Testsystmen.

In Tabelle 2 finden sich die von Rubin in der 2,33-GHz-Umgebung erreichten Laufzeiten wie von den Autoren angegeben (siehe Rubin 2013:308) und in Sekunden (t_2), sowie die auf dem 3,3-GHz-Testsystem gemessenen Werte (t_3). Der Umrechnungsfaktor τ ergibt sich aus $\frac{t_2}{t_3}$.

Es zeigt sich, dass die Geschwindigkeit des hier verwendeten Systems zwei- bis dreimal so hoch wie die des Systems von Rubin ist. Dies entspricht den Erwartungen, die sich aus den technischen Daten der Testumgebung ergeben.

Hervorgehoben ist der größte gemessene Faktor. Die Wahl fällt auf diesen Faktor, um Laufzeiten im Vergleich nicht zu unterschätzen. Alle gemessenen Zeiten werden in nachfolgenden Ergebnissen mit $\tau = \frac{45,4}{16,392}$ multipliziert.

6.2. Gewicht und Laufzeit in Abhängigkeit vom Suchradius

Die den folgenden Diagrammen zugrunde liegenden Messreihen finden sich im Anhang D.

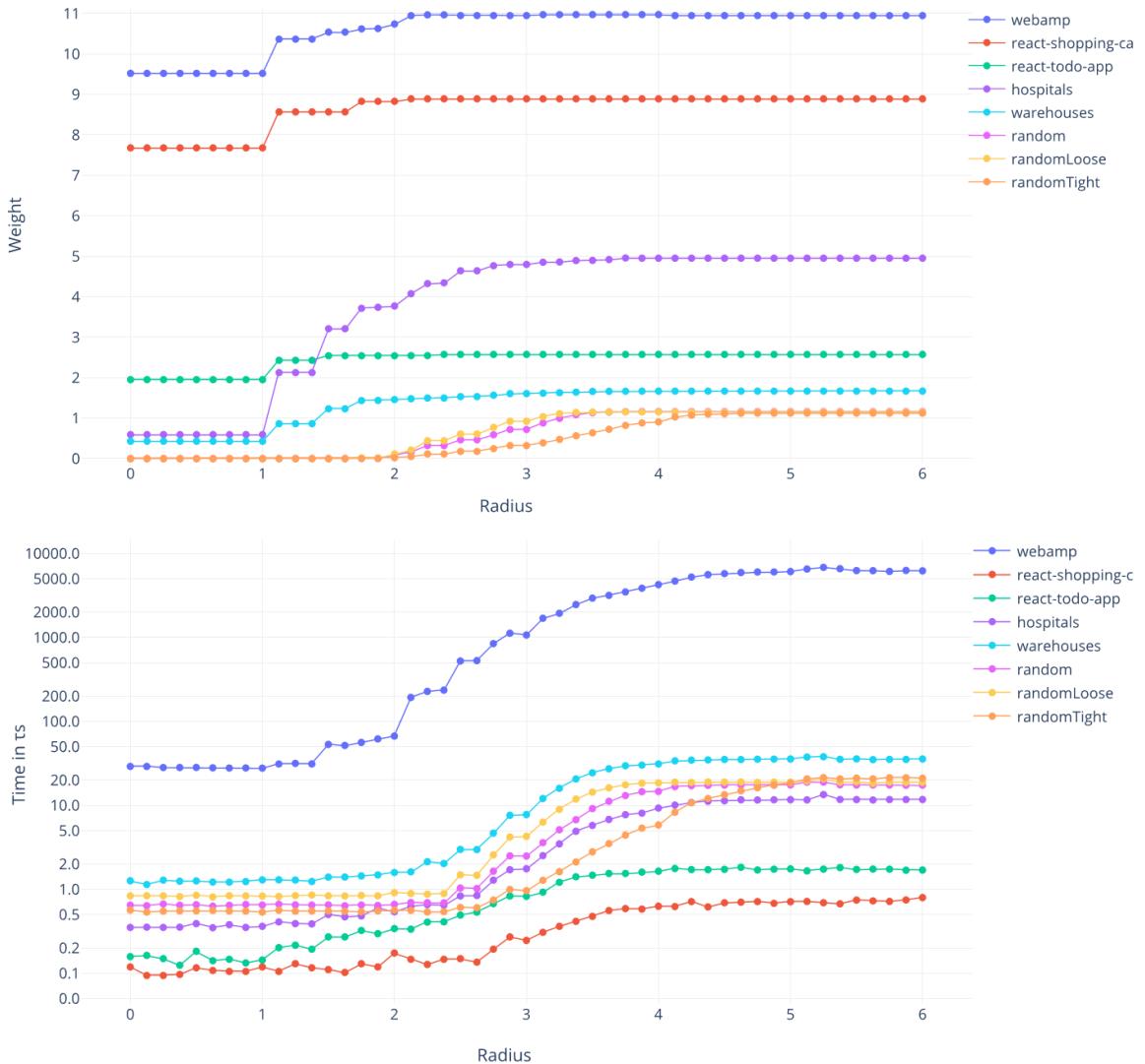


Abbildung 5. Ergebnisse von RaQuN in Abhängigkeit vom Suchradius.

Die beiden Diagramme zeigen die Ergebnisse der Messungen des suchbasierten Verfahrens an den acht Testdatensätzen (mit Weight-Optimized-Match, Suchradius 0 bis 6, in Schritten von 0,125). Die Y-Achse des Laufzeitdiagramms ist logarithmisch, alle anderen Achsen sind linear.

Es wird deutlich, dass der gewählte Suchradius entscheidenden Einfluss auf das von RaQuN erreichte Gewicht und die benötigte Laufzeit hat. Generell kann festgestellt werden:

- Je höher der Radius, desto höher ist das erzielte Weight.
- Je höher der Radius, umso höher die Laufzeit.
- Ab einem gewissen Radius erhöht sich das Gewicht nicht mehr wesentlich. Der Grenzwert hängt vom Input ab. Zum Beispiel: Bei Random und Random (loose) liegt er zwischen 3 und 4, bei Random (tight) zwischen 4 und 5.
- Die Laufzeit steigt teilweise exponentiell mit größerem Radius, strebt bei weiterer Erhöhung des Radius aber ebenfalls einen Grenzwert an. Insbesondere der Datensatz *Webamp* zeigt eine dramatisch schlechte Laufzeit.
- Bei den Real-World-Datensätzen ist der Radius für maximales Weight kleiner als der Radius, bei dem die Laufzeit kritisch ansteigt. Gut sichtbar ist das bei „Hospitals“: Bei Radius 2,5 ist das Weight schon fast maximal, die Laufzeit steigt aber erst bei einem Radius größer als 2,5 deutlich an.
- Im Vergleich zu den Real-World-Daten wird bei den Random-Datensätzen das maximale Gewicht erst bei relativ hohem Suchradius erreicht.
- Für die drei Random-Datensätze liegt der Radius für das maximale Weight etwa im Bereich des Radius, der auch die maximale Laufzeit benötigt.

6.3. Weight und Time im Vergleich zu Rubin NwM

Die folgende Tabelle stellt Gewicht und Laufzeit des RaQuN_λ-Verfahrens (mit Weight-Optimized-Match, Suchradius 5,0) den NwM-Ergebnissen von Rubin gegenüber. (siehe Rubin 2013:308)

	Rubin NwM		RaQuN _λ (Weight-Optimized-Match, Suchradius 5,0)		Verhältnis Q Suchbasiert/NwM in %	
Datensatz	Weight	Time in s	Weight	Time in τs	Q_{Weight}	Q_{Time}
Hospital	4,595	42,7	4,9537	14,01716	107,8	32,8
Warehouse	1,522	174	1,6687	41,33971	109,6	23,8
Random	0,979	96	1,1624	18,63306	118,7	19,4
Random loose	0,980	90	1,1553	19,17092	117,9	21,3
Random tight	0,941	96	1,1227	18,87651	119,3	19,7

Tabelle 3. Weight und Time von NwM und RaQuN_λ im Vergleich.

In Tabelle 3 zeigen die beiden Spalten „Rubin NwM“ die von Rubin gemessenen Werte, wobei die Laufzeit einheitlich in s umgerechnet ist.

Die Spalten „RaQuN[✓]“ zeigen die Messwerte des hier beschriebenen Ansatzes. Die Werte unter Time sind bereits mit dem Umrechnungsfaktor τ multipliziert.

Die weiteren Spalten (Q) zeigen das Verhältnis der gemessenen Werte zu denen von Rubin, wobei 100 % für identisches Gewicht bzw. identische Laufzeit steht. Werte über 100 % repräsentieren einen Zuwachs, Werte unter 100 % eine Reduzierung durch das hier beschriebene Verfahren.

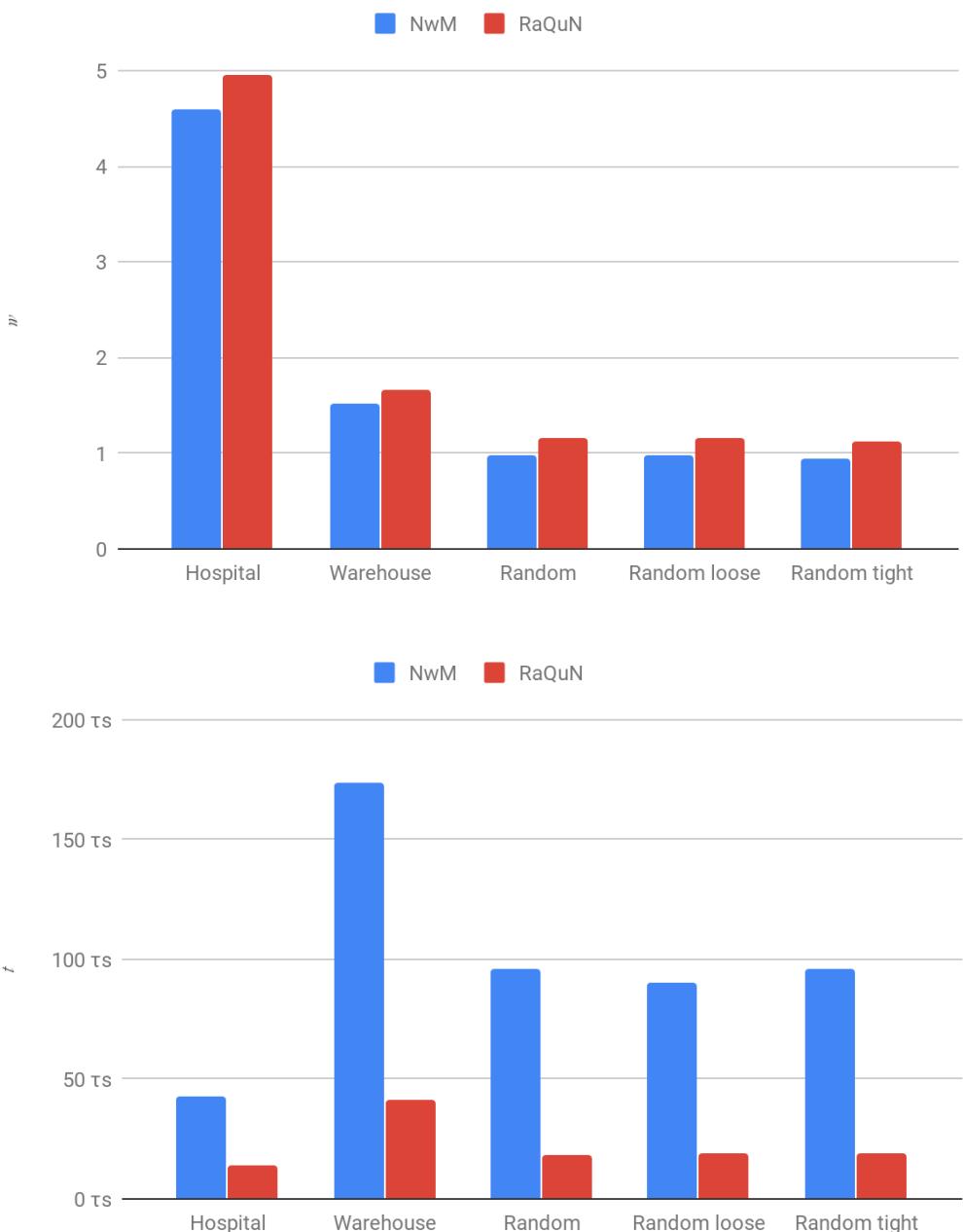


Abbildung 6. Weight und Time von NwM und RaQuN[✓] im Vergleich.

Die Ergebnisse zeigen, dass mit dem gewählten Suchradius von 5,0 das hier beschriebene Verfahren bei allen Datensätzen von Rubin eine Verbesserung der Ergebnisse bewirkt. Dies betrifft sowohl das Weight (Zuwachs zwischen 7 und 19 Prozent), als auch die Laufzeit (Reduktion auf 33 bis 19 Prozent).

Zum Vergleich wurde der Rubin-Algorithmus auf den komplexesten Testdatensatz *Webamp* angewandt. Die Ergebnisse finden sich in nachfolgender Tabelle. Hierbei zeigt sich (hervorgehoben) die Schwachstelle von RaQuN_{moose} bei einem ungünstig gewählten Radius ($r = 5,0$) - eine deutlich längere Laufzeit. Mit dem passenden Suchradius ($r \approx 2,6$) lässt sich aber auch bei diesen Modellen das Gewicht steigern, bei gleichzeitig geringerer Laufzeit.

	Rubin NwM		RaQuN _{moose} (Weight-Optimized-Match)			Verhältnis Q Suchbasiert/NwM in %	
Datensatz	Weight	Time in τs	r	Weight	Time in τs	Q_{Weight}	Q_{Time}
Webamp	8,31819	622	2,625	10,95	531	131,6	85,3
			5,0	10,95	6096	131,6	979,6

Tabelle 4. Vergleich zwischen NwM und RaQuN_{moose} mit dem Datensatz *Webamp*.

6.4. Verteilung der Laufzeit

Neben dem Wert für die Laufzeit t liefert jede Messung drei individuelle Laufzeiten für die Phasen Indizierung, Suche und Merge:

$$t_{index}, t_{query}, t_{merge}, \text{ wobei gilt } t = t_{index} + t_{query} + t_{merge} .$$

Um den Trend der anteilig benötigten Laufzeit zu verdeutlichen, lassen sich für jede Messung die drei Zeiten auf die Gesamtzeit normalisieren, es gilt also: $t_{index} + t_{query} + t_{merge} = 100\%$.

In der folgenden Darstellung sind, gestapelt auf der Y-Achse, die drei Phasen für jeweils alle acht Datensätze abgetragen. Dadurch ergibt sich die Höhe von $8 \times 100\%$.

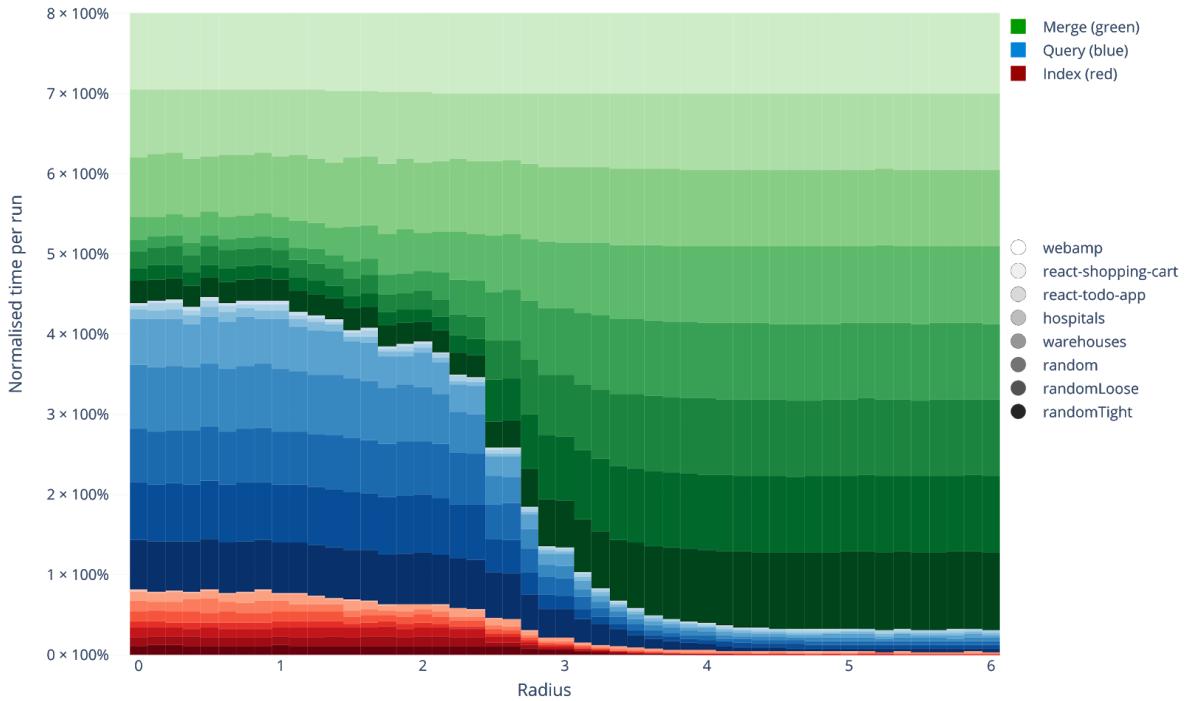


Abbildung 7. Verteilung der für Indizierung, Suche, Merge benötigten Laufzeit.

Es wird deutlich, dass mit steigendem Radius die für den Merge benötigte Zeit (grün eingefärbt, oben) den größten Anteil ausmacht. Bei den drei generierten Datensätzen gilt dies für alle getesteten Radien. Die von Rubin übernommenen Daten benötigen für Indizierung (rot, unten) und Suche (blau, mittig) zusammengenommen bis zu einem Radius von circa $r = 2,5$ mehr Zeit als für den Merge-Schritt.

Für alle Messungen zeigt sich die Tendenz, dass die Indizierung die Phase mit dem geringsten Laufzeitanteil ist.

Vergleicht man diese relativen Trends für die benötigte Laufzeit mit der in Kapitel 4. analysierten Komplexität, passt die Vorhersage zum beobachteten Verhalten: Je größer der gewählte Suchradius, desto kritischer die Laufzeit von Suche und Merge.

6.5. Weight und Time mit Look-Ahead

Die Werte für Gewicht und Laufzeit wurden experimentell auch mit dem „Weight-Optimized-Lookahead-Match“ gemessen. Als Größe des Look-Ahead-Window wurden $L_{10} = 10$ und $L_{100} = 100$ ausgewählt.

Die folgende Abbildung zeigt das Verhältnis der mit dem Weight-Optimized-Lookahead-Match w_L erzielten Werte im Vergleich zum Weight-Optimized-Match w .

Der Quotient ergibt sich aus $Q_w = \frac{w_L}{w}$. Das Verhältnis der Laufzeit ist analog $Q_t = \frac{t_L}{t}$. Der Laufzeit-Quotient Q_t ist als Flächeninhalt der Messpunkte abgetragen. Die kleinste Fläche entspricht $Q_t \approx 1,0$. Der Maximalwert ist jeweils hervorgehoben.

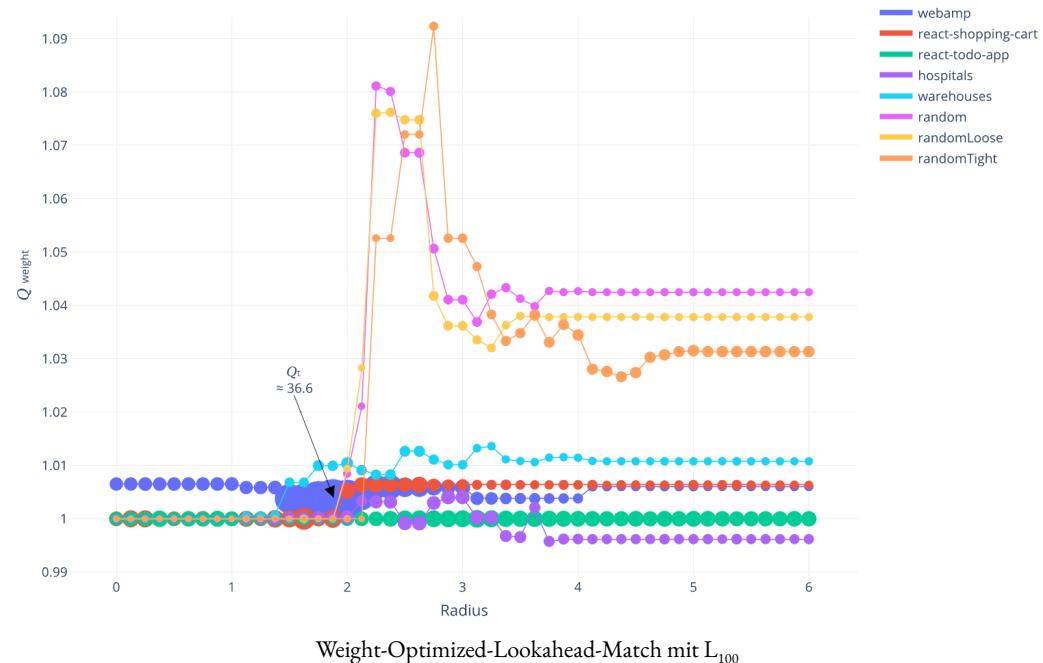
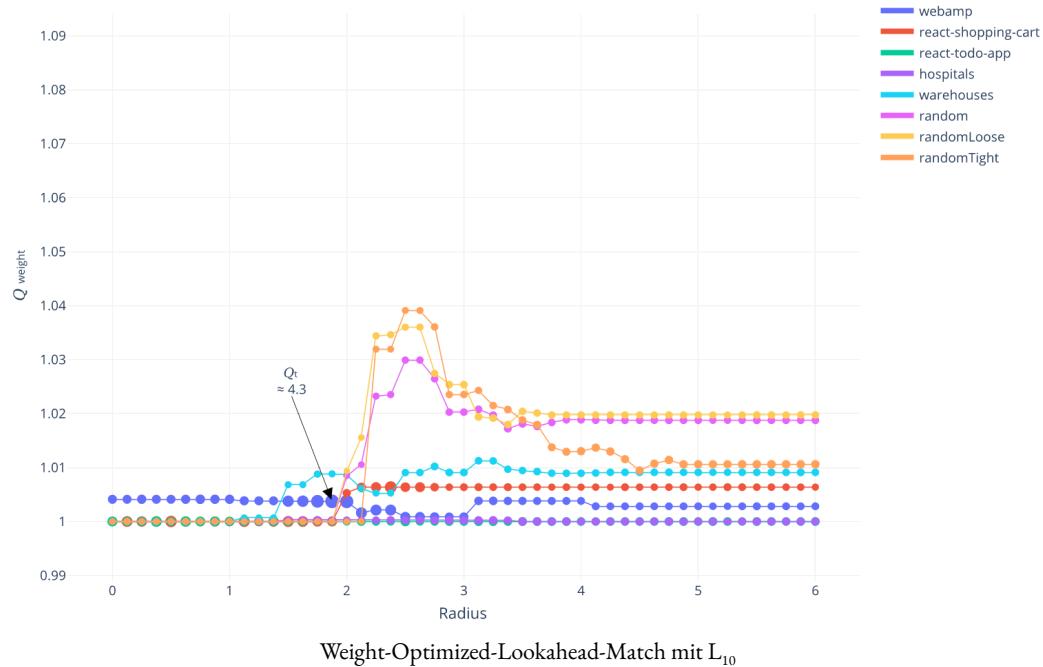


Abbildung 8. Gewicht und Laufzeit mit Look-Ahead im Vergleich.

Es zeigt sich, dass durch Look-Ahead bei den meisten Testdaten eine Verbesserung des Weight erreicht wird. Bei L_{100} steigt das Gewicht im Durchschnitt um 1,24 %. Besonders deutlich ist der Effekt bei den drei Random-Datensätzen.

Es wird aber auch sichtbar, dass die Verbesserung lokale Maxima bei bestimmten Suchradien hat. Dieses Verhalten kann folgendermaßen interpretiert werden: Bei großem Suchradius ist die Anzahl der Suchergebnisse höher, dadurch finden sich im Look-Ahead-Window viele ähnliche Kandidaten. Die Wahrscheinlichkeit, das Gesamtgewicht deutlich zu verbessern, ist deshalb geringer. Bedingt durch die konstante Größe des Fensters nimmt daher mit steigendem Suchradius der potenzielle Nutzen des Look-Ahead ab.

Der Vorteil der konstanten Window-Größe zeigt sich allerdings bei der Laufzeit. Sie steigt bei den gewählten Fenstergrößen kaum, was der Erwartung eines konstanten Komplexitätsfaktors entspricht. Der tatsächliche Faktor variiert jedoch stark, je nach Datensatz und Radius, im Bereich 1 bis 36. Der Durchschnitt bei L_{100} ist $\overline{Q_t} = 3,25$.

7. Fazit

Diese Arbeit hat gezeigt, dass RaQuN⁺ durch die Kombination einer Umkreissuche mit einem angepassten Merge-Verfahren bessere Ergebnisse beim Vergleich von N Modellen erzielt, als ein bestehendes N-Wege-Merge-Verfahren.

Das Verfahren kann deshalb als eine sinnvolle Alternative zu bisherigen Methoden im Bereich N-Wege-Vergleich angesehen werden.

8. Kritik und Ausblick

Eine wichtige Erkenntnis aus den Testreihen mit wachsendem Suchradius ist die Beobachtung, dass das Gewicht des Ergebnisses ab einem bestimmten Wert kaum noch wächst, die Rechenzeit sich jedoch weiter, teilweise massiv, erhöht. Diese Schwelle variiert je nach Datensatz. Um das Verfahren für beliebige Eingaben nutzen zu können, ist es wünschenswert, einen optimalen Suchradius bereits frühzeitig zu ermitteln, zum Beispiel direkt nach der Indizierungsphase. Möglicherweise lässt sich dieser Wert aus der Menge der Vektoren im Suchindex ableiten. Alternativ ist eine progressive Berechnung denkbar, um ein lokales Maximum für den Grenzwert auszuloten.

Alle Messungen mit Hilfe des Quelltextes von Rubin gehen davon aus, dass dieser dem Stand beim Verfassen der Arbeit „N-Way Model Merging“ entspricht. Die Quellen mussten teilweise mit Hilfe von Reverse-Engineering interpretiert und modifiziert werden, um die Zeitmessungen durchzuführen. Die Daten erscheinen im Vergleich zu den berichteten Werten und Erwartungen an die Ergebnisse plausibel. Fehler bei der Ermittlung des Laufzeitfaktors können aber nicht ausgeschlossen werden.

Im Unterschied zum Verfahren von Treude ist der Algorithmus von RaQuN⁺ prinzipiell in der Lage, Modellelemente unterschiedlichen Typs als ähnlich zu identifizieren. Dies erscheint praxisrelevant, da zum Beispiel durch Refactoring Operationen und Attribute durchaus ineinander überführt werden, das gleiche gilt auch für Relationen und Klassen. Konkrete Vor- und Nachteile dieser Fähigkeit untersucht diese Arbeit jedoch nicht.

Die Gültigkeitskriterien schließen eine Zuordnung von mehreren Elementen aus ein- und demselben Modell aus. Wie in Abbildung 1 ersichtlich, sind in der Praxis Fälle denkbar (hier die Frontelemente), bei denen eine derartige Operation erwünscht ist. Dieser Möglichkeit und eventuellen Implikationen geht diese Arbeit nicht nach, sie könnten aber interessant sein.

Um dem Messverfahren nach Rubin zu genügen, betrachten alle vorgenommenen Messungen an Modellen nur die Entitäten „Klassename“ und „Attributname“. Über die Qualität der

Ergebnisse bei Einbeziehung weiterer Modelleigenschaften kann diese Arbeit daher keine objektive Aussage machen.

Die Ergebnisse des „Weight-Optimized-Lookahead-Match“ zeigen, dass durch Anpassungen des Verfahrens noch weitaus bessere Ergebnisse möglich sind. Im Gegensatz zum Brute-Force-Ansatz der getesteten Look-Ahead-Modifikation wäre zu untersuchen, ob es einen effizienten Algorithmus gibt, der die Bestimmung der Reihenfolge bei der Tupelbildung erleichtert. Es ist beispielsweise denkbar, den bereits erstellten Suchindex während der Merge-Phase weiter zu verwenden.

Die Nichtbetrachtung der Klassennamen bei der Berechnung des Weight, die von Rubin übernommen wurde, erscheint fragwürdig. Bei subjektiver Betrachtung sind Klassen mit übereinstimmendem Name ähnlicher, als Klassen mit unterschiedlichen Namen. Wir halten daher eine Wichtung *mit* Klassennamen für aussagekräftiger. Dieses Verfahren wurde aber nicht angewendet, weil es eine vollständige Reproduktion der Testläufe von Rubin erforderlich gemacht hätte. Die Tendenz der vorliegenden Messungen und Vergleiche mit Rubin dürfte jedoch von der gewählten Weight-Variante unabhängig sein. Unabhängig davon gelten auch für diese Arbeit die Bedenken von Rubin, dass andere Berechnungsvorschriften für das Gewicht zu anderen Ergebnissen führen könnten. (vergleiche Rubin 2013:310)

Die getestete Referenzimplementierung von RaQuN_{mouse} ist noch nicht optimal. Die Suche nach nächsten Nachbarn im k-d-Baum kann möglicherweise durch eine echte Bereichssuche oder eine Suche mit k-d-Kugelradius ersetzt werden. Außerdem stehen noch die bewusst nicht gemachten Optimierungen (zum Beispiel Hashing und Nutzung eines S³V-Trees) zur Verfügung.

Im Zuge dieser Arbeit wurde ein Werkzeug geschaffen, welches Modellfamilien aus Open-Source-Repositorys generieren kann. Die hiermit extrahierten Datensätze und andere auf diese Art generierte Daten könnten für weitere Forschung interessant sein.

9. Literaturverzeichnis

- Bentley, Jon Louis (1975): Multidimensional Binary Search Trees Used for Associative Searching. In: Communications of the ACM Volume 18 Issue 9. 509-517.
- Champin, Pierre-Antoine / Solnon, Christine (2003): Measuring the similarity of labeled graphs. In: Proceedings of the 5th international conference on Case-based reasoning; Research and Development. 80-95.
- Henrich, Andreas / Six, Hans-Werner / Widmayer, Peter (1989): The LSD tree: Spatial Access to Multidimensional Point and Nonpoint Objects. In: Proceedings of the Fifteenth International Conference on Very Large Data Bases. 45-53.
- Levendovszky, Tihamer et al. (2010): Model Evolution and Management. In: Model-Based Engineering of Embedded Real-Time Systems. MBEERTS 2007. Lecture Notes in Computer Science, vol 6100. Springer, Berlin, Heidelberg. 241–270.
- Liu, Ting / Moore Andrew W. / Gray, Alexander (2006): New Algorithms for Efficient High-Dimensional Nonparametric Classification. In: The Journal of Machine Learning Research Volume 7. 1135-1158.
- Prechelt, Lutz / Pepper, Alexander (2014): Why software repositories are not used for defect-insertion circumstance analysis more often: A case study. In: Information and Software Technology, Volume 56, Issue 10. Butterworth-Heinemann Newton. 1377-1389.
- Rubin, Julia / Chechik, Marsha (2013): N-Way Model Merging. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM. 301-311.
- Rubin, Julia (2014): Cloned Product Variants: From Ad-hoc to Well-managed Software Reuse. In: Ph.D. dissertation. University of Toronto.
- Sabetzadeh, Mehrdad et al. (2007): A Relationship-Driven Framework for Model Merging. In: Workshop on Modeling in Software Engineering (MiSE'07) at the 29th International Conference on Software Engineering, Minneapolis, USA. IEEE.
- Talaei Rad, Yasaman / Jabbari, Ramtin (2012): Use of Global Consistency Checking for Exploring and Refining Relationships between Distributed Models: A Case Study. Master's thesis, Blekinge Institute of Technology, School of Computing.

- Treude, Christoph et al. (2007): Difference computation of Large Models. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM. 295-304.

Anhang

A. Software

Der in dieser Arbeit entstandene und verwendete Quelltext, einschließlich der CSV-Dateien mit den Testdaten, kann unter folgender Adresse eingesehen werden:

- <https://github.com/dcsaszar/raqun/tree/thesis>

Nachstehende Bibliotheken wurden bei der Implementierung dieser Arbeit verwendet:

- commons-csv-1.6.jar
- javatuples-1.2.jar
- jgrapht-core-1.3.0.jar
- jheaps-0.9.jar
- libssrckdtree-j-1.0.2.jar
- org.apache.commons.io_2.2.0.v201405211200.jar
- org.eclipse.emf.common_2.13.0.v20170609-0707.jar
- org.eclipse.emf.ecore_2.13.0.v20170609-0707.jar
- org.eclipse.emf.ecore.xmi_2.13.0.v20170609-0707.jar
- ujmp-complete-0.3.0.jar

B. Dimensionen für die Indizierung als k-d-Vektoren

D gibt die Anzahl der dem jeweiligen Datensatz und Datentyp zugeordneten Dimensionen an.

Die Daten in den drei Random-Spalten (f, g und h) sind Durchschnittswerte gemäß Beschreibung der Testdaten. Für PARENT_NAME sind bedingt durch die Beschränkung der Testdaten auf Elemente vom Typ Klasse nie mehrere Dimensionen vorhanden.

Typ/Bezeichnung	Be-reich	D ^a	D ^b	D ^c	D ^d	D ^e	D ^f	D ^g	D ^h
INV_LENGTH_OF_NAME	(0, 1]	1	1	1	1	1	1	1	1
INV_NUMBER_OF_ATTRIBUTES	(0, 1]	1	1	1	1	1	1	1	1
INV_NUMBER_OF_METHODS	(0, 1]	1	1	1	1	1	1	1	1
INV_NUMBER_OF_REFERENCES	(0, 1]	1	1	1	1	1	1	1	1
IS_CLASS	{0, 1}	1	1	1	1	1	1	1	1
IS_ATTRIBUTE	{0, 1}	1	1	1	1	1	1	1	1
IS_OPERATION	{0, 1}	1	1	1	1	1	1	1	1
IS_REFERENCE	{0, 1}	1	1	1	1	1	1	1	1
PARENT_NAME	{0, 1}	1	1	1	1	1	1	1	1
NAME	{0, 1}	60	12	14	55	142	269,9	288,8	250,1
NAME_SUBSTRING	{0, 1}	47	5	15	39	97	0	0	0
CHILDREN_NAMES	{0, 1}	196	57	38	162	339	159,9	309,1	120
Dimensionen insgesamt (k)		312	83	76	265	587	438,8	606,9	379,1

a) webamp, b) react-shopping-cart, c) react-todo-app, d) hospitals, e) warehouses, f) random, g) randomLoose, h) randomTight

Beispiele für ausgewählte Dimensionen aus dem Datensatz *Hospitals*:

NAME: Address, AdminAssistant, Bed, Booking Calendar, BookingCalendar, Calendar, Database, Decision, Diagnosis, Display, DisplayScanner, Equipment, GeneralStorage, HealthCard, History, Hospital, IDCard, InPatient, InPatientVisitRecord, IntraWardStay, MedProcDB, MedicalChart, MedicalTeam, Mobile, Nurse, OutPatient, OutPatientVisitRecord, Patient, PatientCheckout, PatientDB, PatientLogger, PatientProfile, PatientTransfer, PatientsList, PersonnelDB, Physician, Prescription, Procedure, Record, RegisteredPatient, Report, Room, Scanner, Schedule, Scheduler, Stationary, Technician, Unit, UnregisteredPatient, Visit, VisitRecord, Ward, WardSpecificMedicationChart, WardStay, WristBand

NAME_SUBSTRING: Admin, Assistant, Band, Booking, Booking , Calendar, Card, Chart, Checkout, DB, Display, General, Health, ID, In, Intra, List, Logger, Med, Medical, Medication, Out, Patient, Patients, Personnel, Proc, Profile, Record, Registered, Scanner, Specific, Stay, Storage, Team, Transfer, Unregistered, Visit, Ward, Wrist

Die vollständigen Daten können mit dem beiliegenden Programm *ShowVectorStatistics.java* ausgegeben werden.

C. Aus Open-Source-Projekten extrahierte Modelle

Die Daten für die Modellfamilie *Webamp* werden aus Platzgründen hier nicht aufgeführt. Der Datensatz (*capitbaritone_webamp.csv*) befindet sich, zusammen mit den hier dargestellten, im Software-Repository.

react-shopping-cart

Model-ID	Element	Attribute
react_shopping_cart_2019020921_master_7db93d71	App	renderGitHubCorner;renderFilter;renderShelf;renderFloatCart
react_shopping_cart_2019020921_master_7db93d71	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2019020921_master_7db93d71	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2019020921_master_7db93d71	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2019020921_master_7db93d71	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2019020921_master_7db93d71	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderPrenderCheckbox
react_shopping_cart_2019020921_master_7db93d71	Shelf	fetchProductsProp;handleFetchProducts;productsProp;filtersProp;sortProp;renderShelfHeader
react_shopping_cart_2019020921_master_7db93d71	ShelfHeader	productsLengthProp;renderSort
react_shopping_cart_2019020921_master_7db93d71	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2019020921_master_7db93d71	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2019020921_master_7db93d71	Root	initialStateProp
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderPrenderCheckbox
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Shelf	fetchProductsProp;addProductProp;handleFilter;handleSort;productsProp;filtersProp;sortProp;renderFilter;renderShelfHeader;renderClearfix
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	ShelfHeader	productsLengthProp;renderSort;renderClearfix
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2018041219_AnyLIUMel_react_shopping_cart_1_documentation_db7143df	Main	renderBanner;renderMain;renderFooter;renderFloatCart
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderCheckbox;renderStarButton
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Shelf	fetchProductsProp;addProductProp;handleFetchProducts;productsProp;filtersProp;sortProp;renderSpinner;renderFile;renderShelfHeader;renderClearfix
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	ShelfHeader	productsLengthProp;renderSort;renderClearfix
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2018120510_AnyLIUMel_react_shopping_cart_1_master_6407a9ab	App	renderCorner;renderShelf;renderFooter;renderFloatCart
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	App	renderGitHubCorner;renderShelf;renderFloatCart
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderCheckbox;renderStarButton
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Shelf	fetchProductsProp;addProductProp;handleFetchProducts;productsProp;filtersProp;sortProp;renderSpinner;renderFile;renderShelfHeader;renderClearfix
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	ShelfHeader	productsLengthProp;renderSort;renderClearfix
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	App	renderCorner;renderShelf;renderFooter;renderFloatCart
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderCheckbox;renderStarButton
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Shelf	fetchProductsProp;addProductProp;handleFetchProducts;productsProp;filtersProp;sortProp;renderSpinner;renderFile;renderShelfHeader;renderClearfix
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	ShelfHeader	productsLengthProp;renderSort;renderClearfix
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2019010922_bheneley_react_shopping_cart_master_00bf8523	App	renderCorner;renderShelf;renderFooter;renderFloatCart
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	App	renderFilter;renderShelf;renderFloatCart
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Checkbox	handleCheckboxChangeProp;toggleCheckboxChange;labelProp
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	CartProduct	removeProductProp;handleMouseOver;handleMouseOut;productProp;renderThumb
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	FloatCart	loadCartProp;updateCartProp;removeProductProp;openFloatCart;closeFloatCart;addProduct;removeProduct;proceedToCheckout;cartProductsProp;newProductProp;productToRemoveProp;renderCartProduct
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Selectbox	handleOnChangeProp;createOptions;onChange;optionsProp;classesProp
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Filter	updateFiltersProp;toggleCheckbox;createCheckbox;createCheckboxes;filtersProp;renderCheckbox;renderStarButton
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Shelf	fetchProductsProp;handleFetchProducts;productsProp;filtersProp;sortProp;renderShelfHeader
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	ShelfHeader	productsLengthProp;renderSort
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Sort	updateSortProp;handleSort;sortProp;renderSelectbox
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	Thumb	altProp;titleProp;classesProp;srcProp
react_shopping_cart_2019011712_react_rookie_react_shopping_cart_master_50c869bb	App	initialStateProp

react-todo-app

Model-ID	Element	Attribute
react_todo_app_2017062714_master_c9ef612a	ButtonWrapper	modeProp;changeModeProp
react_todo_app_2017062714_master_c9ef612a	CheckBox	handleChange
react_todo_app_2017062714_master_c9ef612a	Filter	filterProp;changeFilterProp
react_todo_app_2017062714_master_c9ef612a	FilteredList	itemsProp;changeStatusProp;renderTodoItem
react_todo_app_2017062714_master_c9ef612a	Footer	countProp;filterProp;changeFilterProp;renderButtonWrapper;renderFilter
react_todo_app_2017062714_master_c9ef612a	Header	renderInputWrapper
react_todo_app_2017062714_master_c9ef612a	InputBox	valueProp;handleChangeProp;handleKeyUpProp
react_todo_app_2017062714_master_c9ef612a	InputWrapper	modeProp;addNewProp;queryProp;setSearchQueryProp;renderInputBox;renderSearchBox
react_todo_app_2017062714_master_c9ef612a	SearchBox	queryProp;setSearchQueryProp
react_todo_app_2017062714_master_c9ef612a	TodoItem	dataProp;changeStatusProp;renderCheckBox
react_todo_app_2017062714_master_c9ef612a	TodoList	renderHeader;renderFilteredList;renderFooter;renderInfo
react_todo_app_2017062714_master_c9ef612a	App	renderStateProvider;renderKeyStrokeHandler;renderTodoList
react_todo_app_2017062714_master_c9ef612a	KeyStrokeHandler	handleKeyUp
react_todo_app_2017062714_master_c9ef612a	StateProvider	addNew;changeMode;setSearchQuery
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	App	addNew;changeFilter;renderTodoList
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	Filter	filterProp;changeProp
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	Footer	countProp;filterProp;changeFilterProp;renderFilter
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	Header	titleProp;addNewProp;renderInputBox
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	InputBox	handleChange;clear;handleKeyUp
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	TodoItem	dataProp
react_todo_app_2017021115_Akasky70_react_todo_app_step_10_c15f550b	TodoList	titleProp;itemsProp;addNewProp;filterProp;changeFilterProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	App	addNew;changeFilter;renderTodoList
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	CheckBox	handleChange
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	Filter	filterProp;changeProp
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	Footer	countProp;filterProp;changeFilterProp;renderFilter
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	Header	titleProp;addNewProp;renderInputBox
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	InputBox	handleChange;clear;handleKeyUp
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	TodoItem	dataProp
react_todo_app_2017021315_Akasky70_react_todo_app_step_12_42e0fd2a	TodoList	titleProp;itemsProp;addNewProp;filterProp;changeFilterProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	App	addNew;changeFilter;changeStatus;renderTodoList
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	CheckBox	handleChange
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	Filter	filterProp;changeProp
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	FilteredList	itemsProp;changeStatusProp;renderTodoItem
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	Footer	countProp;filterProp;changeFilterProp;renderFilter
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	Header	titleProp;addNewProp;renderInputBox
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	InputBox	handleChange;clear;handleKeyUp
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	TodoItem	dataProp;changeStatusProp;renderCheckBox
react_todo_app_2017021315_Akasky70_react_todo_app_step_13_301950a3	TodoList	titleProp;itemsProp;addNewProp;filterProp;changeFilterProp;changeStatusProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	App	addNew;changeFilter;changeStatusProp;renderHeader;renderFilteredList;renderFooter
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	CheckBox	handleChange
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	Filter	filterProp;changeProp
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	FilteredList	itemsProp;changeStatusProp;renderTodoItem
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	Footer	countProp;filterProp;changeFilterProp;renderFilter
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	Header	renderInputBox
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	InputBox	handleChange;clear;handleKeyUp
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	StateProvider	addNew;changeFilter;changeStatus
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	TodoItem	dataProp;changeStatusProp;renderCheckBox
react_todo_app_2017021318_Akasky70_react_todo_app_step_14_bdb874fc	TodoList	renderHeader;renderFilteredList;renderFooter
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	ButtonWrapper	modeProp;changeModeProp
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	CheckBox	handleChange
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	Filter	filterProp;changeFilterProp
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	FilteredList	itemsProp;changeStatusProp;renderTodoItem
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	Footer	countProp;filterProp;changeFilterProp;renderFilter
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	Header	renderInputWrapper
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	InputBox	handleChange;handleKeyUp
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	InputWrapper	modeProp;addNewProp;queryProp;setSearchQueryProp;renderInputBox;renderSearchBox
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	SearchBox	queryProp;setSearchQueryProp
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	TodoItem	dataProp;changeStatusProp;renderCheckBox
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	TodoList	renderHeader;renderFilteredList;renderFooter;renderInfo
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	App	renderStateProvider;renderKeyStrokeHandler;renderTodoList
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	KeyStrokeHandler	handleKeyUp
react_todo_app_2017021519_Akasky70_react_todo_app_step_15_4fec6982	StateProvider	addNew;changeFilter;changeStatus;changeMode;setSearchQuery
react_todo_app_2017021113_Akasky70_react_todo_app_step_4_bdecff07	App	renderTodoList
react_todo_app_2017021113_Akasky70_react_todo_app_step_4_bdecff07	Footer	titleProp;itemsProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_5_sb27ba89	Header	renderTodoList
react_todo_app_2017021113_Akasky70_react_todo_app_step_5_sb27ba89	Footer	titleProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_5_sb27ba89	Header	dataProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_5_sb27ba89	Footer	titleProp;itemsProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021113_Akasky70_react_todo_app_step_6_99638261	App	renderTodoList
react_todo_app_2017021113_Akasky70_react_todo_app_step_6_99638261	Footer	countProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_6_99638261	Header	titleProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_6_99638261	Footer	dataProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_6_99638261	Header	titleProp;itemsProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021113_Akasky70_react_todo_app_step_7_4ec94714	App	renderTodoList
react_todo_app_2017021113_Akasky70_react_todo_app_step_7_4ec94714	Footer	countProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_7_4ec94714	Header	titleProp;renderInputBox
react_todo_app_2017021113_Akasky70_react_todo_app_step_7_4ec94714	Footer	dataProp
react_todo_app_2017021113_Akasky70_react_todo_app_step_7_4ec94714	Header	titleProp;itemsProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	App	renderTodoList
react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	Footer	countProp

react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	Header	titleProp;renderInputBox
react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	InputBox	handleChange;handleKeyUp
react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	TodoItem	dataProp
react_todo_app_2017021114_Akasky70_react_todo_app_step_8_99f72a2f	TodoList	titleProp;itemsProp;renderHeader;renderTodoItem;renderFooter
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	App	addNewItem;renderTodoList
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	Footer	countProp
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	Header	titleProp;addNewProp;renderInputBox
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	InputBox	handleChange;clear;handleKeyUp
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	TodoItem	dataProp
react_todo_app_2017021114_Akasky70_react_todo_app_step_9_badd2ca0	TodoList	titleProp;itemsProp;addNewProp;renderHeader;renderTodoItem;renderFooter

D. Messergebnisse der Mikro-Benchmarks

Die Daten ergeben sich aus Einzeltests an acht Testdatensätzen mit 49 Suchradien. Die Daten für die Random-Datensätze sind wie in der Beschreibung der Testdaten angegeben gemittelt.

Radius	Dataset	Chunks	Run ID	Weight-Optimized-Match				Weight-Optimized-Lookahead-Match mit L ₁				Weight-Optimized-Lookahead-Match mit L ₁₀₀								
				Weight	Time in ms	Index in ms	Search in ms	Match in ms	Run ID	Weight	Time in ms	Index in ms	Search in ms	Match in ms	Run ID	Weight	Time in ms	Index in ms	Search in ms	Match in ms
0.000	webamp	1	#0	9.7208	29.12834	0.27419	1.04416	27.80999	#1175	9.5600	55.45260	0.00538	54.23239	#3430	9.5829	122.64536	0.24650	1.07185	121.21301	
0.000	react-shopping-cart	1	#1	7.9510	0.00000	0.00000	0.00000	0.00000	#1176	7.9500	0.24000	0.00531	0.22434	#3431	7.9500	0.00000	0.00000	0.00000	0.15765	
0.000	read-todo-app	1	#2	1.9514	15.1788	0.02219	0.01693	0.11933	#1177	1.9514	0.24000	0.01693	0.01693	#3432	1.9514	0.01693	0.01693	0.01693	0.23771	
0.000	hospitals	1	#3	0.5938	0.35174	0.04708	0.20218	0.10248	#1178	0.5938	0.39683	0.04985	0.19388	#3433	0.5938	0.57054	0.05262	0.23081	0.23081	
0.000	warehouses	1	#4	0.4297	1.26573	0.0947	0.09430	0.17726	#1179	0.4297	1.24653	0.09417	0.91675	#3434	0.4297	1.57870	0.09971	1.00538	0.47361	
0.000	random	10	#14	0.0000	6.64948	0.07340	0.43539	0.14070	#1179	0.0000	6.69700	0.07700	0.45478	0.16563	#3444	0.0000	0.69130	0.07284	0.45672	0.16175
0.000	randomLoose	10	#24	0.0000	0.83422	0.09334	0.06544	0.15343	#1179	0.0000	0.86441	0.09639	0.61071	0.15732	#3454	0.0000	0.85471	0.08724	0.60295	0.16452
0.000	randomTight	10	#34	0.0000	0.56695	0.06259	0.35092	0.15344	#1179	0.0000	0.87569	0.06287	0.34482	0.16840	#3464	0.0000	0.56999	0.06315	0.34731	0.15953
0.125	webamp	1	#35	9.5262	0.00000	0.00000	0.00000	0.00000	#1179	9.5262	0.24000	0.00000	0.00000	0.00000	#3470	9.5262	124.12000	0.00000	0.00000	122.30005
0.125	react-shopping-cart	1	#36	7.9500	0.004917	0.01108	0.00831	0.07478	#1179	7.9500	0.25000	0.004917	0.004917	0.004917	#3471	7.9500	0.004917	0.004917	0.004917	0.004917
0.125	read-todo-app	1	#37	1.9514	1.6341	0.01939	0.01662	0.12740	#1179	1.9514	0.32128	0.01939	0.01662	0.28527	#3472	1.9514	0.78935	0.02216	0.10385	0.75334
0.125	hospitals	1	#38	0.5938	0.35451	0.04708	0.21603	0.09140	#1179	0.5938	0.34066	0.04985	0.17172	0.11909	#3498	0.5938	0.54562	0.05262	0.22434	0.22434
0.125	warehouses	1	#39	0.4297	1.26573	0.0947	0.09430	0.17726	#1179	0.4297	1.24653	0.09417	0.91675	0.23542	#3499	0.4297	1.57870	0.09971	1.00538	0.47361
0.125	random	10	#49	0.0000	0.63840	0.07367	0.42549	0.14014	#1179	0.0000	0.68770	0.07201	0.45602	0.16507	#3499	0.0000	0.68909	0.07672	0.45058	0.15732
0.125	randomLoose	10	#59	0.0000	0.83892	0.06032	0.03438	0.15095	#1179	0.0000	0.87526	0.06039	0.34482	0.16840	#3499	0.0000	0.84114	0.09097	0.59104	0.15953
0.125	randomTight	10	#69	0.0000	0.56695	0.06259	0.35092	0.15344	#1179	0.0000	0.87569	0.06287	0.34482	0.16840	#3499	0.0000	0.85471	0.08724	0.60295	0.16452
0.250	webamp	1	#70	9.5262	28.25591	0.25491	1.30173	0.266937	#1179	9.5262	0.25000	0.04061	0.04061	0.04061	#3499	9.5262	125.34400	0.28312	0.05670	124.05658
0.250	react-shopping-cart	1	#71	7.6800	0.004917	0.01108	0.00831	0.07478	#1179	7.6800	0.23189	0.01385	0.00554	0.21890	#3501	7.6800	0.64810	0.00831	0.10385	0.62594
0.250	read-todo-app	1	#72	1.9514	1.4957	0.01939	0.01662	0.11356	#1179	1.9514	0.30190	0.01939	0.01662	0.26589	#3502	1.9514	0.73119	0.0216	0.0216	0.22612
0.250	hospitals	1	#73	0.5938	0.35174	0.04708	0.21049	0.09417	#1179	0.5938	0.39605	0.04708	0.20218	0.14679	#3503	0.5938	0.58162	0.04708	0.23819	0.29635
0.250	warehouses	1	#74	0.4297	1.28789	0.08586	0.18648	0.18557	#1179	0.4297	1.20728	0.08248	0.96938	0.23542	#3504	0.4297	1.53716	0.09694	0.10193	0.42099
0.250	random	10	#74	0.0000	0.66988	0.07589	0.44785	0.14624	#1179	0.0000	0.67164	0.07312	0.44101	0.15842	#3514	0.0000	0.67496	0.07340	0.43677	0.16480
0.250	randomLoose	10	#74	0.0000	0.84000	0.07580	0.07983	0.13954	#1179	0.0000	0.69000	0.07012	0.45051	0.16507	#3514	0.0000	0.84460	0.07654	0.43649	0.15732
0.250	randomTight	10	#74	0.0000	0.56456	0.06259	0.35092	0.15344	#1179	0.0000	0.87569	0.06287	0.34482	0.16840	#3514	0.0000	0.85510	0.07551	0.43561	0.16136
0.375	webamp	1	#75	9.5262	28.23860	0.25481	1.15494	0.26735	#1179	9.5262	0.25000	0.04061	0.04061	0.04061	#3515	9.5262	121.83663	0.25204	0.10369	120.57090
0.375	react-shopping-cart	1	#76	7.6800	0.00964	0.00831	0.00554	0.08309	#1179	7.6800	0.24373	0.01385	0.00554	0.22434	#3516	7.6800	0.54839	0.02116	0.0216	0.15717
0.375	read-todo-app	1	#77	1.9514	1.4957	0.01939	0.01662	0.11356	#1179	1.9514	0.30190	0.01939	0.01662	0.26589	#3517	1.9514	0.73119	0.0216	0.0216	0.22612
0.375	hospitals	1	#78	0.5938	0.35174	0.04708	0.19884	0.11356	#1179	0.5938	0.39605	0.04708	0.20218	0.14679	#3518	0.5938	0.54562	0.05262	0.22434	0.22434
0.375	warehouses	1	#79	0.4297	1.28789	0.08586	0.14146	0.14291	#1179	0.4297	1.20728	0.08248	0.96938	0.23542	#3519	0.4297	1.53716	0.09694	0.10193	0.42099
0.375	random	10	#79	0.0000	0.65613	0.07146	0.44176	0.14291	#1179	0.0000	0.67025	0.07038	0.43811	0.15676	#3520	0.0000	0.86358	0.08863	0.61597	0.15926
0.375	randomLoose	10	#79	0.0000	0.85028	0.07298	0.61874	0.13876	#1179	0.0000	0.68371	0.07102	0.48891	0.16258	#3520	0.0000	0.86590	0.08000	0.61597	0.15926
0.375	randomTight	10	#79	0.0000	0.81026	0.00895	0.59467	0.14347	#1179	0.0000	0.83587	0.01025	0.62536	0.16737	#3520	0.0000	0.87632	0.00921	0.61737	0.16737
0.500	webamp	1	#80	9.5262	28.22832	0.25489	1.18326	0.26187	#1179	9.5262	0.25000	0.04061	0.04061	0.04061	#3521	9.5262	122.31323	0.24650	0.11069	120.96419
0.500	react-shopping-cart	1	#81	7.6800	0.00964	0.00831	0.00554	0.08309	#1179	7.6800	0.24373	0.01385	0.00554	0.22434	#3522	7.6800	0.54839	0.02116	0.0216	0.15717
0.500	read-todo-app	1	#82	1.9514	1.4957	0.01939	0.01662	0.11356	#1179	1.9514	0.30190	0.01939	0.01662	0.26589	#3523	1.9514	0.73119	0.0216	0.0216	0.22612
0.500	hospitals	1	#83	0.5938	0.35174	0.04708	0.19884	0.11356	#1179	0.5938	0.39605	0.04708	0.20218	0.14679	#3524	0.5938	0.54562	0.05262	0.22434	0.22434
0.500	warehouses	1	#84	0.4297	1.28789	0.08586	0.14146	0.14291	#1179	0.4297	1.20728	0.08248	0.96938	0.23542	#3525	0.4297	1.53716	0.09694	0.10193	0.42099
0.500	random	10	#84	0.0000	0.65613	0.07146	0.44176	0.14291	#1179	0.0000	0.67025	0.07038	0.43811	0.15676	#3525	0.0000	0.86358	0.08863	0.61597	0.15926
0.500	randomLoose	10	#84	0.0000	0.85028	0.07298	0.61874	0.13876	#1179	0.0000	0.68371	0.07102	0.48891	0.16258	#3525	0.0000	0.86590	0.08000	0.61597	0.15926
0.500	randomTight	10	#84	0.0000	0.81026	0.00895	0.59467	0.14347	#1179	0.0000	0.83587	0.01025	0.62536	0.16737	#3525	0.0000	0.87632	0.00921	0.61737	0.16737
0.750	webamp	1	#85	9.5262	27.77122	0.24027	1.14882	0.26379	#1179	9.5262	0.25000	0.04061	0.04061	0.04061	#3526	9.5262	121.84771	0.28312	0.05670	120.49508
0.750	react-shopping-cart	1	#86	7.6800	0.11018	0.00831	0.00554	0.08309	#1179	7.6800	0.24096	0.01385	0.00554	0.22157	#3527	7.6800	0.61022	0.01385	0.01383	0.57886
0.750	read-todo-app	1	#87	1.9514	1.44043	0.01939	0.01662	0.10802	#1179	1.9514	0.30192	0.01939	0.01662	0.26589	#3528	1.9514	0.73119	0.0216	0.0216	0.22612
0.750	hospitals	1	#88	0.5938	0.35174	0.04708	0.19884	0.11356	#1179	0.5938	0.39605	0.04708	0.20218	0.14679	#3529	0.5938	0.54562	0.05262	0.22434	0.22434
0.750	warehouses	1	#89	0.4297	1.28789	0.08586	0.14146	0.14291	#1179	0.429										

2.000	warehouses	1	#564	1.4590	1.58701	0.09417	1.09124	0.40160	#2279	1.4716	1.72549	0.10248	0.95830	0.66747	#3994	1.4742	5.43680	0.12186	1.83590	3.48144	
2.000	random	10	#574	0.0826	0.65333	0.07810	0.43483	0.15039	#2289	0.0833	0.70100	0.07589	0.44037	0.18474	#4004	0.0833	1.09401	0.11660	0.57858	0.39883	
2.000	randomLoose	10	#584	0.1173	0.91904	0.10165	0.66278	0.144651	#2299	0.1184	0.87559	0.09445	0.59104	0.19111	#4014	0.1184	1.37208	0.13460	0.77855	0.45893	
2.000	randomLight	10	#594	0.0504	0.65333	0.07810	0.43483	0.15039	#2299	0.0670	0.70100	0.07589	0.44037	0.18472	#4024	0.0670	1.09401	0.11660	0.57858	0.39883	
2.000	webcamp	1	#595	10.9467	183.0097	0.27945	1.88945	101.0179	#2310	10.9544	50.88572	0.27191	4.04954	50.59995	#4025	10.9899	200.83890	0.34898	2.03370	20.04412	
2.125	react-shopping-cart	1	#596	8.8905	1.44679	0.01385	0.00831	0.12463	#2311	8.9471	0.28528	0.01662	0.01108	0.25758	#4026	8.9471	0.80597	0.01108	0.00564	2.03370	20.04412
2.125	react-todo-app	1	#597	2.5467	0.33513	0.01939	0.02216	0.29358	#2312	2.5467	0.60102	0.02216	0.55670	#4027	2.5467	1.69503	0.02216	0.02216	1.65071		
2.125	hospitals	1	#598	4.0759	0.62317	0.05262	0.24927	0.32128	#2313	4.0770	0.75057	0.04985	0.18280	0.51792	#4028	4.0896	1.96090	0.05262	0.21049	1.69779	
2.125	warehouses	1	#599	1.4814	1.61747	0.07755	0.99430	0.54562	#2314	1.4904	2.14371	0.09417	1.09955	#4029	1.4949	4.23758	0.09971	1.01369	3.12416		
2.125	random	10	#600	0.1612	0.69684	0.07977	0.46502	0.15205	#2324	0.1629	0.72703	0.07755	0.44148	0.20800	#4030	0.1646	0.92644	0.07700	0.43123	0.41822	
2.125	randomLoose	10	#601	0.1159	0.53786	0.05955	0.31215	0.14707	#2329	0.1198	0.61597	0.06453	0.34593	0.20551	#4031	0.1220	0.78270	0.06841	0.34011	0.37418	
2.125	randomTight	10	#602	0.1159	0.53786	0.05955	0.31215	0.14707	#2329	0.1198	0.61597	0.06453	0.34593	0.20551	#4032	0.1220	0.78270	0.06841	0.34011	0.37418	
2.250	webcamp	1	#630	10.9675	227.88352	0.27143	2.01630	225.59879	#2345	10.9908	634.52814	0.27273	2.14370	632.10471	#4060	11.0330	319.18334	0.29359	2.14467	319.74329	
2.250	react-shopping-cart	1	#631	8.8905	0.12741	0.01385	0.00831	0.10525	#2346	8.9471	0.29635	0.01385	0.00831	0.27419	#4061	8.9471	0.74227	0.01108	0.00831	0.72288	
2.250	react-todo-app	1	#632	2.5467	0.33513	0.01939	0.02216	0.29358	#2347	2.5467	0.60865	0.01939	0.27700	0.64256	#4062	2.5467	1.69503	0.02216	0.02216	2.04261	
2.250	hospitals	1	#633	4.3256	0.65917	0.04708	0.20163	0.39606	#2348	4.3266	0.90674	0.04985	0.17449	0.61833	#4063	4.3393	2.68378	0.04985	0.22711	2.40682	
2.250	warehouses	1	#634	1.4978	2.13623	0.08863	0.10955	0.54562	#2349	1.5057	0.55381	0.09694	1.05246	1.40421	#4064	1.5102	3.38419	0.09694	1.06908	4.21817	
2.250	random	10	#644	0.1159	0.53786	0.05955	0.31215	0.14707	#2349	0.1159	0.61597	0.06453	0.34593	0.20551	#4065	0.1200	0.45500	0.06700	0.42266	0.15236	
2.250	randomLoose	10	#645	0.1159	0.53786	0.05955	0.31215	0.14707	#2349	0.1159	0.61597	0.06453	0.34593	0.20551	#4066	0.1200	0.45500	0.06700	0.42266	0.15236	
2.250	randomTight	10	#646	0.1159	0.53786	0.05955	0.31215	0.14707	#2349	0.1159	0.61597	0.06453	0.34593	0.20551	#4067	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	webcamp	1	#666	10.9675	236.04012	0.25204	2.21925	233.57513	#2380	10.9908	654.17013	0.26035	2.15201	652.00737	#4095	11.0330	3220.74669	0.27143	2.12709	319.34817	
2.375	react-shopping-cart	1	#667	8.8905	0.14679	0.01385	0.00831	0.12463	#2381	8.9471	0.47084	0.01385	0.01662	0.44037	#4096	8.9471	0.77550	0.01108	0.01385	0.75057	
2.375	react-todo-app	1	#668	2.5751	0.40911	0.01939	0.02216	0.29358	#2382	2.5751	0.64100	0.02216	0.27700	0.59824	#4097	2.5751	2.35143	0.01938	0.02493	2.30711	
2.375	hospitals	1	#669	4.3445	0.64091	0.04431	0.22434	0.37944	#2383	4.3456	0.90750	0.04985	0.17449	0.61833	#4098	4.3582	2.68378	0.04985	0.22711	2.40682	
2.375	warehouses	1	#670	1.4973	2.13623	0.08863	0.10955	0.54562	#2384	1.5057	0.55381	0.09694	1.05246	1.40421	#4099	1.5102	3.38419	0.09694	1.06908	4.21817	
2.375	random	10	#671	0.1159	0.53786	0.05955	0.31215	0.14707	#2384	0.1159	0.61597	0.06453	0.34593	0.20551	#4100	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomLoose	10	#672	0.1159	0.53786	0.05955	0.31215	0.14707	#2384	0.1159	0.61597	0.06453	0.34593	0.20551	#4101	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomTight	10	#673	0.1159	0.53786	0.05955	0.31215	0.14707	#2384	0.1159	0.61597	0.06453	0.34593	0.20551	#4102	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	webcamp	1	#667	8.8905	0.14679	0.01385	0.00831	0.12463	#2385	8.9471	0.47084	0.01385	0.01662	0.44037	#4097	8.9471	0.7551	0.01108	0.01385	0.75057	
2.375	react-shopping-cart	1	#668	2.5751	0.40911	0.01939	0.02216	0.29358	#2386	2.5751	0.64100	0.02216	0.27700	0.59824	#4098	2.5751	2.35143	0.01938	0.02493	2.30711	
2.375	react-todo-app	1	#669	4.3445	0.64091	0.04431	0.22434	0.37944	#2387	4.3456	0.90750	0.04985	0.17449	0.61833	#4099	4.3582	2.68378	0.04985	0.22711	2.40682	
2.375	hospitals	1	#670	1.4973	2.13623	0.08863	0.10955	0.54562	#2388	1.5057	0.55381	0.09694	1.05246	1.40421	#4100	1.5102	3.38419	0.09694	1.06908	4.21817	
2.375	warehouses	1	#671	0.1159	0.53786	0.05955	0.31215	0.14707	#2388	0.1159	0.61597	0.06453	0.34593	0.20551	#4101	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	random	10	#672	0.1159	0.53786	0.05955	0.31215	0.14707	#2388	0.1159	0.61597	0.06453	0.34593	0.20551	#4102	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomLoose	10	#673	0.1159	0.53786	0.05955	0.31215	0.14707	#2388	0.1159	0.61597	0.06453	0.34593	0.20551	#4103	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomTight	10	#674	0.1159	0.53786	0.05955	0.31215	0.14707	#2388	0.1159	0.61597	0.06453	0.34593	0.20551	#4104	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	webcamp	1	#675	10.9675	227.88352	0.27143	2.01630	225.59879	#2389	10.9908	654.17013	0.26035	2.15201	652.00737	#4095	11.0330	3220.74669	0.27143	2.12709	319.34817	
2.375	react-shopping-cart	1	#676	8.8905	0.14679	0.01385	0.00831	0.12463	#2390	8.9471	0.47084	0.01385	0.01662	0.44037	#4096	8.9471	0.7551	0.01108	0.01385	0.75057	
2.375	react-todo-app	1	#677	2.5751	0.40911	0.01939	0.02216	0.29358	#2391	2.5751	0.64100	0.02216	0.27700	0.59824	#4097	2.5751	2.35143	0.01938	0.02493	2.30711	
2.375	hospitals	1	#678	4.3445	0.64091	0.04431	0.22434	0.37944	#2392	4.3456	0.90750	0.04985	0.17449	0.61833	#4098	4.3582	2.68378	0.04985	0.22711	2.40682	
2.375	warehouses	1	#679	1.4973	2.13623	0.08863	0.10955	0.54562	#2393	1.5057	0.55381	0.09694	1.05246	1.40421	#4099	1.5102	3.38419	0.09694	1.06908	4.21817	
2.375	random	10	#680	0.1159	0.53786	0.05955	0.31215	0.14707	#2393	0.1159	0.61597	0.06453	0.34593	0.20551	#4100	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomLoose	10	#681	0.1159	0.53786	0.05955	0.31215	0.14707	#2393	0.1159	0.61597	0.06453	0.34593	0.20551	#4101	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	randomTight	10	#682	0.1159	0.53786	0.05955	0.31215	0.14707	#2393	0.1159	0.61597	0.06453	0.34593	0.20551	#4102	0.1200	0.45500	0.06700	0.42266	0.15236	
2.375	webcamp	1	#683	10.9675	227.88352	0.27143	2.01630	225.59879	#2394	10.9908	654.17013	0.26035	2.15201	652.00737	#4095	11.0330	3220.74669	0.27143	2.12709	319.34817	
2.375	react-shopping-cart	1	#684	8.8905	0.14679	0.01385	0.00831	0.12463	#2395	8.9471	0.47084	0.01385	0.01662	0.44037	#4096	8.9471	0.7551	0.01108	0.01385	0.75057	
2.																					

4.500	webamp	1	#1261	8.8905	0.69241	0.01108	0.01939	0.66194	#2976	8.9471	0.91953	0.01108	0.02493	0.88352	#4691	8.9471	1.37652	0.01385	0.02493	1.33774
4.500	react-shopping-cart	1	#1262	2.5751	1.73103	0.01939	0.05262	1.65902	#2977	2.5751	2.96629	0.01939	0.06370	2.88320	#4692	2.5751	10.10920	0.02493	0.05262	10.03165
4.500	hospitals	1	#1263	4.9537	11.33615	0.04985	0.45145	10.83485	#2978	4.9537	17.00038	0.04985	0.47084	16.48769	#4693	4.9348	28.08973	0.05262	0.48746	27.54965
4.500	warehouses	1	#1264	1.6885	0.56202	0.01108	1.63302	0.55002	#2979	1.6885	46.17878	0.01108	1.66596	46.25951	#4694	1.6885	59.23893	0.01385	1.70487	57.24265
4.500	random	10	#1265	1.74524	17.45922	0.07284	0.75501	16.62639	#2980	1.74524	29.69294	0.07284	1.76947	22.65399	#4714	1.74524	21.12951	0.07450	1.75197	20.53034
4.500	randomLoose	10	#1266	1.1553	18.81777	0.08863	0.69882	17.23243	#2981	1.1553	24.76560	0.08863	0.98765	23.62029	#4714	1.1553	29.16520	0.08891	0.98793	28.08945
4.500	randomTight	10	#1267	1.1092	13.41671	0.06010	0.53703	12.81957	#3009	1.1197	21.36774	0.06148	0.56362	20.76263	#4724	1.1396	40.85861	0.06675	0.58052	40.21135
4.625	webamp	1	#1295	10.9488	5944.28138	0.2612	20.9108	5923.10745	#3010	10.9797	8372.12061	0.24650	20.99944	8350.87467	#4725	11.0151	12821.29568	0.28250	21.39502	12799.61768
4.625	react-shopping-cart	1	#1296	8.8905	0.70349	0.01108	0.02493	0.66748	#3011	8.9471	0.91953	0.01385	0.03047	0.87521	#4726	8.9471	1.36821	0.00831	0.02770	1.33220
4.625	react-todo-app	1	#1297	2.5751	1.83073	0.01939	0.04431	1.76703	#3012	2.5751	3.17678	0.01939	0.06647	3.09092	#4727	2.5751	9.92640	0.01938	0.06647	9.84054
4.625	hospitals	1	#1298	4.9537	11.33615	0.04708	0.41545	11.20321	#3013	4.9537	17.59001	0.05262	0.47361	17.05378	#4728	4.9348	27.72990	0.04985	0.44314	27.23391
4.625	warehouses	1	#1299	1.1553	38.11768	0.04985	0.41545	33.81115	#3014	1.1553	47.15747	0.04985	0.55717	48.02036	#4729	1.1553	57.59747	0.05020	0.57192	57.47945
4.625	random	10	#1300	1.1624	17.50111	0.07340	0.76470	16.75302	#3024	1.1842	23.49990	0.07256	0.75861	22.65873	#4730	1.1218	27.32558	0.07340	26.47794	26.47794
4.625	randomLoose	10	#1301	1.1553	18.06542	0.08697	0.96079	17.19766	#3034	1.1781	24.58668	0.08697	0.98655	23.51316	#4749	1.1990	28.91093	0.09417	0.99181	27.82495
4.625	randomTight	10	#1302	1.1195	14.93946	0.06148	0.55504	14.32294	#3045	1.0977	8434.05539	0.25870	20.8604	8413.01717	#4760	11.0151	13594.15922	0.27973	25.00157	13568.87792
4.750	webamp	1	#1330	10.9488	6022.96973	0.25841	20.82218	6001.89274	#3046	8.9471	8.97769	0.01385	0.02216	9.94168	#4761	8.9471	1.38206	0.01385	0.02493	1.34328
4.750	react-shopping-cart	1	#1331	8.8905	7.91457	0.01108	0.02493	0.67856	#3047	8.9471	17.30147	0.01385	0.02216	17.30147	#4762	8.9471	25.751	0.01385	0.02493	25.751
4.750	react-todo-app	1	#1332	2.5751	1.83073	0.01939	0.04431	1.76703	#3048	2.5751	3.17678	0.01939	0.06647	3.09092	#4763	2.5751	9.92640	0.01938	0.06647	9.84054
4.750	hospitals	1	#1333	4.9537	11.33615	0.04708	0.41545	11.04084	#3049	4.9537	18.34045	0.05262	0.55684	17.93898	#4764	4.9348	30.97736	0.06709	0.57192	30.26390
4.750	warehouses	1	#1334	1.1624	17.52603	0.07007	0.74116	16.71480	#3050	1.1842	24.42660	0.08115	0.82397	23.52148	#4774	1.2118	28.57497	0.08143	0.80846	27.68508
4.750	randomLoose	10	#1335	1.1553	18.75531	0.08669	0.96661	17.70301	#3059	1.1781	24.96308	0.08905	0.99375	23.81007	#4784	1.1990	31.95477	0.10331	0.10363	30.81783
4.750	randomTight	10	#1336	1.1219	16.21934	0.06038	0.55088	15.60168	#3079	1.1347	26.69050	0.06342	0.59658	26.03050	#4794	1.1564	50.40197	0.09223	0.06670	49.64005
4.875	webamp	1	#1365	10.9488	6034.95675	0.26866	20.97451	6013.71388	#3080	10.9797	8637.76821	0.27419	20.98597	8616.50843	#4795	11.0151	13189.39519	0.32128	23.04637	13166.02770
4.875	react-shopping-cart	1	#1366	8.8905	0.70349	0.01108	0.02493	0.67879	#3081	8.9471	0.91953	0.01385	0.02216	0.94168	#4796	8.9471	1.38206	0.01385	0.02493	1.34328
4.875	react-todo-app	1	#1367	2.5751	1.75087	0.01939	0.05539	1.68179	#3082	2.5751	3.17633	0.01939	0.06624	3.07439	#4797	2.5751	7.74658	0.01399	0.05539	6.71760
4.875	hospitals	1	#1368	4.9537	11.66851	0.04708	0.41822	11.20321	#3083	4.9537	17.57062	0.04708	0.47915	17.40439	#4798	4.9348	28.20882	0.04709	0.48607	27.63637
4.875	warehouses	1	#1369	1.6687	10.58518	0.08669	0.86663	1.67474	#3084	1.6839	48.50774	0.05262	1.71845	48.50774	#4799	1.6867	59.04050	0.09694	1.67010	58.53033
4.875	random	10	#1370	1.1624	17.45873	0.07423	0.73673	16.67478	#3094	1.1842	23.50884	0.07533	0.77107	22.72244	#4809	1.2118	27.28847	0.08087	0.80405	26.40938
4.875	randomLoose	10	#1371	1.1553	18.05031	0.09555	0.96882	17.84033	#3104	1.1781	24.90990	0.09085	0.10098	23.81007	#4819	1.1990	28.72093	0.09306	0.98433	27.64354
4.875	randomTight	10	#1372	1.1226	17.46704	0.06176	0.56667	16.83860	#3114	1.1345	26.69050	0.06342	0.59741	26.05511	#4829	1.1578	50.44712	0.06619	0.59852	49.78241
5.000	webamp	1	#1400	10.9488	6054.44205	0.26866	20.72931	6001.76118	#3115	10.9797	8729.10701	0.20474	21.11656	8707.88522	#4830	11.0151	13814.29562	0.06143	21.39330	13813.27695
5.000	react-shopping-cart	1	#1401	8.8905	0.70349	0.01108	0.02493	0.67879	#3116	8.9471	0.91953	0.01385	0.02216	0.94168	#4831	8.9471	1.38206	0.01385	0.02493	1.34328
5.000	react-todo-app	1	#1402	2.5751	1.75087	0.01939	0.05539	1.67663	#3117	2.5751	3.17633	0.01939	0.06370	2.92751	#4832	2.5751	10.41109	0.02216	0.06647	10.32246
5.000	hospitals	1	#1403	4.9537	11.66848	0.04708	0.46530	11.21755	#3118	4.9537	17.77384	0.04985	0.45422	17.27427	#4833	4.9348	28.91129	0.04985	0.45422	28.40824
5.000	warehouses	1	#1404	1.6687	35.67855	0.08309	0.55931	34.03615	#3119	1.6839	48.45492	0.04917	1.59591	48.45492	#4834	1.6867	58.08970	0.06964	1.65348	58.12928
5.000	randomLoose	10	#1405	1.1624	17.45873	0.07423	0.73673	16.67478	#3120	1.1842	23.50884	0.07533	0.77107	22.72244	#4835	1.2118	27.28847	0.08087	0.80405	26.40938
5.000	randomTight	10	#1406	1.1226	17.46704	0.06176	0.56667	16.83860	#3121	1.1345	26.69050	0.06342	0.59741	26.05511	#4836	1.1578	50.44712	0.06619	0.59852	49.78241
5.375	react-shopping-cart	1	#1471	8.8905	0.70349	0.01108	0.02493	0.67879	#3131	8.9471	0.91953	0.01385	0.02216	0.94168	#4837	8.9471	1.38206	0.01385	0.02493	1.34328
5.375	react-todo-app	1	#1472	2.5751	1.83073	0.01939	0.05539	1.68179	#3132	2.5751	3.17678	0.01939	0.06624	3.07439	#4838	2.5751	10.41109	0.02216	0.06647	10.32246
5.375	hospitals	1	#1473	4.9537	13.44939	0.04708	0.39052	13.01179	#3133	4.9537	17.68418	0.04985	0.49577	17.13856	#4839	4.9348	29.91769	0.06647	0.57678	29.28434
5.375	warehouses	1	#1474	1.6687	37.91782	0.08028	0.56762	36.20618	#3134	1.6839	48.79559	0.05218	0.59447	48.79559	#4840	1.6867	58.16661	0.06647	1.71426	58.16661
5.375	randomLoose	10	#1475	1.1624	17.59018	0.07340	0.73673	16.78999	#3135	1.1842	23.49990	0.07223	0.77156	22.82574	#4841	1.2118	27.42437	0.07700	0.86191	27.48496
5.375	randomTight	10	#1476	1.1553	20.71001	0.09889	0.10109	19.60599	#3136	1.2009	21.17905	0.09887	0.22984	21.17905	#4842	1.2118	30.67952	0.10414	1.05136	29.52142
5.375	webamp	1	#1505	10.9488	6261.93459	0.24373	20.67621	6241.32290	#3137	2.5751	3.17678	0.01385	0.02216	0.94168	#4843	1.2118	27.42437	0.07700	0.86191	27.48496
5.375	react-shopping-cart	1	#1506	8.8905	0.74504	0.01108	0.02493	0.67879	#3138	2.5751	3.17678	0.01385	0.02216	0.94168	#4844	1.2118	27.42437	0.07700	0.86191</	

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 28.03.2019

.....