JOHANN WOLFGANG GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Masterthesis

Modellierung von Unsicherheiten in Daten:

Benchmarktests verschiedener Ansätze

zur Erlangung des akademischen Grades

Master of Science

vorgelegt von

Jan Luca Pöpperl

am 20.07.2023

Gutachter:

Dr. Karsten Tolle

Johann Wolfgang Goethe-Universität Frankfurt am Main

Fachbereich 12 - Informatik und Mathematik

Institut für Informatik

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

gemäß § 34, Abs. 16 der Ordnung für den Masterstudiengang Informatik vom 17. Juni 2019

Hiermit erkläre ich

_____

(Nachname, Vorname)

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass die von mir eingereichten schriftlichen gebundenen Versionen meiner Masterarbeit mit der eingereichten elektronischen Version meiner Masterarbeit übereinstimmen.

Frankfurt am Main, den

_____

Unterschrift der/des Studierenden

Zusammenfassung

Das Resource Description Framework (RDF) gilt als De-facto-Standardmodell für Wissensgraphen im Semantischen Web [Abu+22]. Für die Darstellung von Unsicherheiten in RDF-Graphen gibt es bisher keinen allgemein anerkannten Standard, obwohl bereits verschiedene Ansätze vorgeschlagen wurden. Diese Arbeit befasst sich mit verschiedenen Modellierungen zur Darstellung von Unsicherheiten auf der Grundlage von Ontologien und vergleicht diese mit Hilfe von Benchmarktests. Unter anderem werden Modellierungen mit der Erweiterung RDF-star getestet, welche mit Einführung von verschachtelten Tripeln eine Erweiterung von RDF darstellt. Die ausgeführten Benchmarktests behandeln sowohl reale, als auch künstlich erzeugte Datensätze und betrachten den Einfluss der Anzahl an Unsicherheiten und Alternativen pro Unsicherheit. Aus den daraus gewonnen Erkenntnissen lassen sich vorteilhafte und unvorteilhafte Modellierungen identifizieren, sowie einflussreiche Eigenschaften aufdecken.

Inhaltsverzeichnis

1 Einleitung

Daten zählen in vielen Bereichen als wichtiges Gut. Nicht ohne Grund gibt es Unternehmen wie PAYBACK, dessen Kernaktivität es ist, Kundenkaufdaten auszuwerten, diese für Marketingzwecken zu nutzen und für Partner bereitzustellen [Gmb23]. Üblicherweise handelt es sich dabei stets um sichere Fakten, die in Datensätze aufgenommen werden, da Unsicherheiten in der Praxis selten Verwendung finden. In der Forschung sind unsichere Werte jedoch für die Beantwortung mancher Forschungsfragen erforderlich [TW14].

Für das Vermerken einer Unsicherheit bieten sich verschiedene Möglichkeiten an z.B. durch ein einfaches Fragezeichen am Ende des Werts. Spätestens bei komplexeren Formen von Unsicherheiten, wie der Angabe von Wahrscheinlichkeiten oder alternativen Werten, stößt dieses Modell jedoch an seine Grenzen. Die verschiedenen Ansprüche an eine solche Angabe und die seltenen Einsatzmöglichkeiten von unsicheren Werten, hat zur Folge, dass sich bisher noch kein allgemein anerkannter Standard für die Modellierung von Unsicherheiten etabliert hat. Für ein isoliertes System führt das Fehlen eines solchen Standards zu keinen größeren Komplikationen, solang eine einheitliche Angabe verwendet wird. Werden die Daten jedoch mit externen Parteien ausgetauscht, ist eine eindeutige Interpretierbarkeit zwingend notwendig.

Für eine eindeutige Interpretierbarkeit bietet sich das *Resource Description Framework* (*RDF*) an. Dieses stellt eine formale Sprache dar durch die Daten unter Anwendungen ausgetauscht werden können, ohne ihre ursprüngliche Bedeutung zu verlieren [Hit+08]. Damit RDF für einen problemlosen Austausch von unsicheren Daten genutzt werden kann, braucht es hierbei einen allgemein genutzten Standard, welcher bisher fehlt. Zwar existieren bereits mehrere Ansätze zur Modellierung von Unsicherheiten für RDF, jedoch fehlt eine umfangreiche Leistungsanalyse, um so eine passende Modellierung auszuwählen. Ziel dieser Arbeit ist es daher, einen Benchmark zum Vergleich der Modellierungen zu entwickeln und durchzuführen, um dadurch vorteilhafte (oder unvorteilhafte) Modellierungen aufzudecken.

Um dieses Ziel zu erreichen, wird zunächst in Abschnitt 2 eine Einführung in RDF und die damit verbundenen Thematiken gegeben. Darauffolgend werden in Abschnitt 3

Richtlinien für die Durchführung und Dokumentation von Benchmarks gegeben. Diese enthalten Punkte, welche erfüllt werden sollten, um eine hohe Aussagekraft des Benchmarks zu gewährleisten. Im Anschluss folgt in Abschnitt 4 der in dieser Arbeit ausgeführte Benchmark. Dafür wird zunächst in Abschnitt 4.1 alle dafür notwendigen Ressourcen beschrieben, um dann in Abschnitt 4.2 die Ergebnisse des Benchmarks zu präsentieren. In Abschnitt 4.3 wird eine abschließende Diskussion der Ergebnisse geführt und die daraus entstehende Empfehlungen abgeleitet. Enden wird die vorliegende Arbeit mit einem Ausblick möglicher Erweiterungen und offenen Forschungsfragen in Abschnitt 5. Alle in dieser Arbeit enthaltenen Web-Ressourcen sind zum Zeitpunkt der Abgabe abrufbar.

2 Resource Description Framework (RDF)

Um die Syntax und die Funktionsweise von RDF-Graphen und den untersuchten Modellierungen genauer zu beleuchten, werden in diesem Kapitel zunächst die Grundlagen von RDF und die wichtigsten Begriffe erläutert. Für einen tieferen Einblick wird auf die Quellen [Du 13; Gei09; Hit+08; EE04] verwiesen, die für dieses Kapitel als Grundlage dienen.

2.1 Das Semantische Web

Das *World Wide Web* (kurz *WWW*) hat sich zu einer unverzichtbaren und fundamentalen Komponente unseres täglichen Lebens entwickelt, da es die Art und Weise, wie wir kommunizieren, Geschäfte tätigen und uns miteinander vernetzen, an vielen Stellen revolutioniert hat [Cas22; Kol20]. Zudem ermöglicht es eine noch nie da gewesene Bereitstellung großer Datenmengen für die breite Bevölkerung und das in vernachlässigbarer Zeit[Hit+08].

Obwohl das Web mit einer unüberschaubar großen Menge an Daten gefüllt ist, lassen sich durch die Nutzung von Suchmaschinen wie Google oder Bing, Internet-Ressourcen eines gewünschten Themas abfragen. Bei komplexeren Eingaben zeigt sich jedoch eine Schwäche des Webs. Wird Google beispielsweise nach Artikeln mit den Stichworten „Politikerin arbeitet in Landtag, verheiratet mit Bundeskanzler“ gefragt, so erhält man ausschließlich drei Artikel[1] über Politiker in Österreich, die nie mit einem Bundeskanzler verheiratet waren. Während Google die Eingabe augenscheinlich nicht korrekt interpretiert hat, können menschliche Leser hingegen verschiedene Beziehungen zwischen Informationen ableiten. In dem genannten Beispiel wird nach einer weiblichen Politikerin gesucht, welche in einem Landtag arbeitet und mit einem Bundeskanzler verheiratet ist, was bspw. auf Britta Ernst und Doris Schröder-Köpf zutrifft.

Ein Ansatz, um solche Probleme zu vermeiden, bietet das Modell des *Semantischen Webs* (*Semantic Web*). Die von Tim Berners-Lee stammende Idee, verfolgt das Ziel, das WWW so zu erweitern, dass Maschinen die Daten nicht nur lesen, sondern auch semantisch „verstehen“ können. Im Gegensatz zum WWW verspricht das Semantische

[1] Die Google Suche wurde am 11.04.2023 auf dem „News“ Bereich von Google durchgeführt.

Web, dass auch Programme Beziehungen zwischen Informationen auslesen können. Somit lassen sich auch Aussagen wie „Politikerin arbeitet in Landtag“ von Maschinen interpretieren. In dem *World Wide Web Consortium*[2] (kurz *W3C*) arbeiten Mitgliedsorganisationen, Festangestellte und die Öffentlichkeit zusammen an der Entwicklung von Webstandards, die auch für das Semantische Web gelten [Gei09]. Einer dieser *W3C-Standards* ist das Resource Description Framework, auf das im nächsten Kapitel besonders eingegangen wird.

2.2 Grundlagen von RDF

Um die Ansprüche des Semantischen Webs realisieren zu können, verfolgt Berners-Lee die Idee von verlinkten Daten [Du 13] einer Datenstruktur, bei der Daten untereinander verknüpft werden können. Damit Daten des Semantischen Webs nicht in verschiedenen Formaten auftreten, wurde daraufhin Ende des letzten Jahrhunderts das *Resource Description Framework* (kurz *RDF*) eingeführt. Mit RDF war es nun möglich, die zugrundeliegenden Daten in Form von Graphen zu beschreiben, um so Beziehungen zwischen Daten zu speichern.

Im Ansatz gleichen sich RDF-Graphen und gerichtete Graphen aus der Graphentheorie, mit der Nutzung von Knoten und Kanten. Knoten und Kanten eines RDF-Graphen werden als *Ressourcen* bezeichnet und besitzen dabei einen eindeutigen Bezeichner in Form von *URI*s (*Uniform Resource Identifiers*). Diese werden vom WWW dazu verwendet, um Webseiten oder andere Dateien zu bezeichnen [Hit+08]. Die wahrscheinlich bekannteste Gruppe von URIs, sind die *URLs* (*Uniform Resource Locators*), welche dazu verwendet werden Webseiten zu lokalisieren und abzurufen. So kann beispielsweise die URI *http://beispiel.com/buch1* ein Bezeichner eines Buchs mit ID 1 sein. Neben URIs können auch *Literale* als Ressource zur Repräsentation von Datenwerten genutzt werden. Um auf diese verweisen zu können, wird intern eine URI für das Literal erzeugt [Tol06].

Ein RDF-Graph besteht aus einer Menge von *RDF-Aussagen* der Form *Subjekt Prädikat Objekt* (kurz *spo*), welche jeweils eine Kante des Graphen beschreiben. Somit

[2]Das World Wide Web Consortium wurde 1994 von Berners-Lee gegründet und ist online erreichbar unter: `https://www.w3.org/`

wird die Größe eines RDF-Graphen hauptsächlich durch die Anzahl seiner Kanten beeinflusst. Von einer RDF-Aussage ist das Subjekt dabei das Ding der Aussage, das Prädikat eine Eigenschaft und das Objekt der Eigenschaftswert des Prädikats. Während ein Objekt eine beliebige Ressource repräsentieren kann, müssen Subjekt und Prädikat durch eine URI referenziert werden. Somit lassen sich Aussagen wie zum Beispiel „Buch 1 hat den Titel Harry Potter und der Stein der Weisen“ mit dem Tripel (*http://beispiel.com/buch1, http://beispiel.com/hatTitel, Harry Potter und der Stein der Weisen*) beschreiben, woraus der in Abbildung 2.1 gezeigte RDF-Graph entsteht. Üblicherweise werden Ressourcen mit bekannter URI durch ein Oval dargestellt und Literale durch Rechtecke. Zudem wird jede gerichtete Kante mit der Eigenschaft (bzw. Prädikat) des repräsentierten Tripels betitelt.

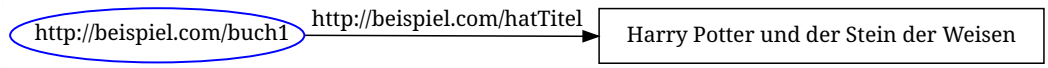


Abbildung 2.1: Einfaches Beispiel eines RDF-Graphen.

Häufig werden keine einzelnen Ressourcen im Internet hinterlegt, sondern eine Sammlung von Ressourcen in sogenannten *Namensräumen*. Es entsteht dadurch eine Vielzahl von URIs, welche sich nur im Suffix unterscheiden, wie in Abbildung 2.1 bei den URIs *http://beispiel.com/buch1* und *http://beispiel.com/hatTitel* zu sehen. Um die Bezeichnung von Ressourcen aus dem gleichen Namensraum zu vereinfachen, können den Namensräumen Präfixe zugeordnet werden, sodass Ressourcen nach dem Schema *Präfix:Rest-URI* angegeben werden können. Für den Namensraum *http://beispiel.com/* bietet sich der Präfix *bsp* an, wodurch die genannten Ressourcen vereinfacht durch *bsp:buch1* und *bsp:hatTitel* bezeichnet werden können. In späteren Beispielen wird der Namensraum ebenfalls mit Hilfe des Präfixes abgekürzt, dieser ist jedoch kein realer Namensraum und dient nur als repräsentatives Beispiel. Eine Liste mit den in dieser Arbeit verwendeten Namensräumen, befindet sich in Anhang A.

Literale können in RDF-Graphen in verschiedenen Formen auftreten. In vielen Fällen besitzt der Datenwert eines Literals einen konkreten Datentyp, wie z.B. Ganzzahlen

oder Wahrheitswerte, welche ebenfalls in RDF durch URIs zugeordnet werden können. Anstelle eines Datentyps kann ein Literal auch einer Sprache zugeordnet werden, um so zu kennzeichnen, in welcher Sprache die enthaltene Zeichenkette verfasst wurde. In Abbildung 2.2 wurde das oben genannte Beispiel um passende Datentypen, Sprachen und Namensräume erweitert. So wurde dem Buch mit ID 1 ein weiterer Titel in einer anderen Sprache zugeordnet und dem Erscheinungsjahr ein dazu passender Datentyp. Literale ohne Datentyp und Sprache werden als einfache Zeichenketten interpretiert.

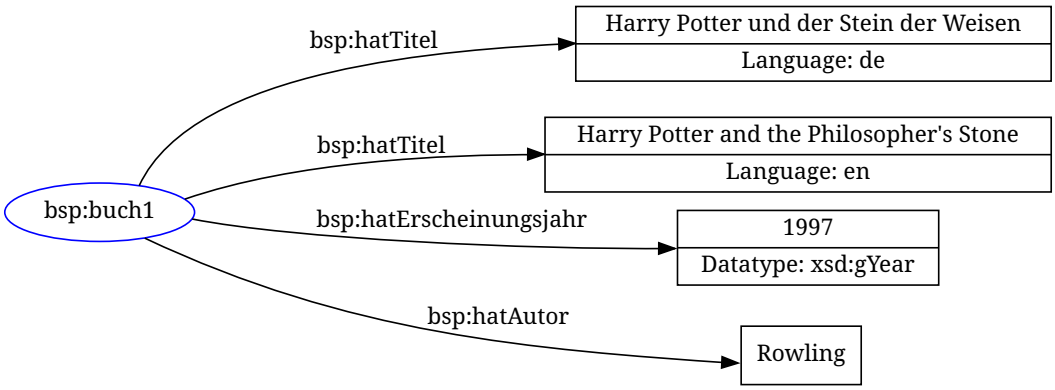


Abbildung 2.2: Erweiterter RDF-Graph aus Abbildung 2.1 mit Nutzung von Namensräumen, Datentypen und Sprachen.

Neben Ressourcen in Form von URIs oder Literalen, gibt es noch ein weiterer häufig genutzter Typ, die *leeren Knoten* (*blank nodes*). Dabei handelt es sich um eine Ressource, die als Knoten in Form eines Subjekts oder Objekts auftreten kann. Diese werden als „leer“ bezeichnet, da für diese weder URI, noch Datenwert angegeben werden kann. Ähnlich zu den Literalen wird auch für leere Knoten intern eine URI erzeugt. Üblicherweise kommen leere Knoten zum Einsatz, wenn eine Ressource keine bekannte URI besitzt und auch keine solche erstellt werden soll. Ein Beispiel dafür ist ein Warenkorb in einem Onlinehandel, bei dem es wahrscheinlich ist, dass dieser nur von der internen Anwendung des Online-Handels genutzt wird und somit außerhalb des RDF-Graphen keine Verwendung findet. Abbildung 2.3 zeigt eine Erweiterung des oben genannten Beispiels mit einem solchen Warenkorb. Der leere Knoten wurde hier mit einem unbeschrifteten Kreis dargestellt.

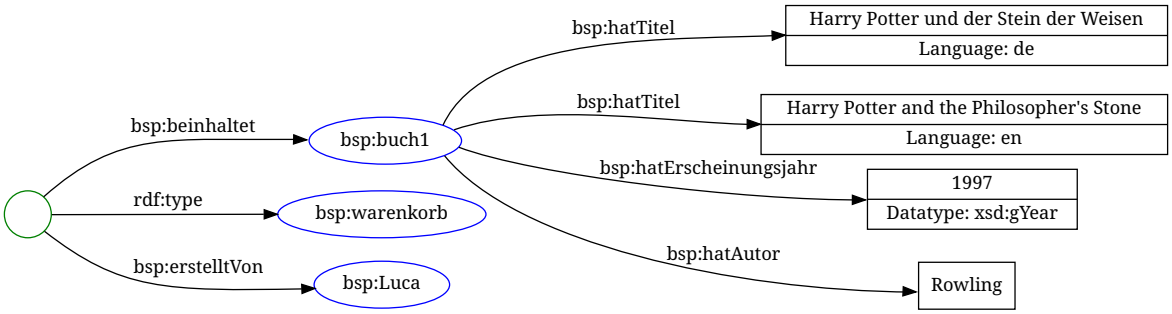


Abbildung 2.3: Erweiterter RDF-Graph aus Abbildung 2.2 mit Nutzung eines leeren Knotens.

2.3 Die RDF-Formate XML und Turtle

Bisher wurde betrachtet, welche Informationen ein RDF-Graph enthält. Wichtig ist darüber hinaus in welchem Format diese in einem RDF-Dokument hinterlegt werden. Bereits zu Beginn hat sich vorallem das *XML*-Format[3] (*Extensible Markup Language*) als W3C-Standard durchgesetzt, mit dem sich hierarchisch strukturierte Daten darstellen lassen, die leicht zwischen Computern und Programmen ausgetauscht werden können [Gei09]. Somit prinzipiell das, was durch RDF-Graphen erreicht werden soll. Ein weiterer Vorteil von XML ist ebenfalls, dass sich das Format zu einem gängigen Standard entwickelt hat und es somit von den meisten Software-Bibliotheken genutzt werden kann. In Abbildung 2.4 ist das Beispiel aus Abbildung 2.3 im XML-Format zu sehen. Eine genaue Beschreibung, welche Möglichkeiten XML besitzt, um Daten zu speichern, würde den Rahmen dieser Arbeit sprengen. Aus diesem Grund wird im Folgenden nur die grundlegende Syntax beschrieben. Für einen tieferen Einblick in XML, ist unter anderem [EE04] zu empfehlen.

[3] Aktuelle Fassung von XML erreichbar unter: `https://www.w3.org/TR/xml/`. XML wurde 1998 erstmals als W3C-Standard festgehalten: `https://www.w3.org/TR/1998/REC-xml-19980210`.

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
 xmlns:bsp="http://beispiel.com/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
 <bsp:warenkorb rdf:nodeID="vWK1r0c0">
  <bsp:erstelltVon rdf:resource="http://beispiel.com/Luca"/>
  <bsp:beinhaltet>
   <rdf:Description rdf:about="http://beispiel.com/buch1">
    <bsp:hatAutor>Rowling</bsp:hatAutor>
    <bsp:hatErscheinungsjahr rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">1997</bsp:hatErscheinungsjahr>
    <bsp:hatTitel xml:lang="de">Harry Potter und der Stein der Weisen</bsp:hatTitel>
    <bsp:hatTitel xml:lang="en">Harry Potter and the Philosopher's Stone</bsp:hatTitel>
   </rdf:Description>
  </bsp:beinhaltet>
 </bsp:warenkorb>
</rdf:RDF>
```

Abbildung 2.4: RDF-Graph aus Abbildung 2.3 im XML-Format.

Ein XML-Dokument beginnt in der Regel mit einem XML-Prolog, welcher die grundlegenden Eigenschaften des Dokuments angibt. Der Rest des Dokuments beschreibt eine Baumstruktur, in der die Daten abgelegt werden. Im Falle von RDF ist die Wurzel des Baums das Element *rdf:RDF*. Die Kinder der Wurzel sind im Block <rdf:RDF> ... </rdf:RDF> enthalten und sind im Fall von RDF die Tripel bzw. Kanten des Graphen. Zudem können in der Wurzel die verwendeten Präfixe und ihre Namensräume definiert werden.

Während die Darstellung mit XML für die meisten Programme verständlich bzw. lesbar ist, können selbst kleine Graphen für den Menschen unüberschaubar komplex werden. Aus diesem Grund lohnt sich ein Blick auf das alternative Format *Turtle*, welches etwa 16 Jahre nach XML ebenfalls als ein W3C-Standard festgehalten wurde[4]. Das oben betrachtete Beispiel ist in Abbildung 2.5 mit der Turtle-Syntax dargestellt. Auf den ersten Blick zeigt das Turtle-Format eine deutlich intuitivere und kompaktere Darstellung. In den ersten beiden Zeilen werden benötigte Namensräume nach dem Schema „@prefix *präfix*: <*uri*>“ festgelegt. Der Rest des Dokuments enthält die Kanten des RDF-Graphen in Form von Tripeln. Enden Tripel in einem Turtle-Dokument mit einem Semikolon, wird in der darauf folgenden Zeile ein weiteres Tripel mit dem gleichen Subjekt beschrieben. Endet ein Tripel hingegen mit einem Komma, wird in der nächsten Zeile ein Tripel mit dem gleichen Subjekt-Prädikat-Paar beschrieben. Ansonsten endet ein Tripel mit einem Punkt. Aufgrund dieser kompakteren Syntax,

[4]Aktuelle Fassung von Turtle erreichbar unter: `https://www.w3.org/TR/turtle/`.

```
@prefix bsp: <http://beispiel.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] a bsp:warenkorb ;
    bsp:beinhaltet bsp:buch1 ;
    bsp:erstelltVon bsp:Luca .

bsp:buch1 bsp:hatAutor "Rowling" ;
    bsp:hatErscheinungsjahr "1997"^^xsd:gYear ;
    bsp:hatTitel "Harry Potter und der Stein der Weisen"@de,
        "Harry Potter and the Philosopher's Stone"@en .

```

Abbildung 2.5: RDF-Graph der Aussage aus Abbildung 2.3 im Turtle-Format.

werden die in dieser Arbeit gezeigten Beispiele ebenfalls mittels Turtle dargestellt.

Aus dem Beispiel in Abbildung 2.5 lassen sich zudem weitere Darstellungsformen herauslesen. Mit eckigen Klammern lassen sich beispielsweise leere Knoten erzeugen. Zudem lassen sich nach Literalen mit dem Marker „^^“ ein Datentyp zuordnen und mit dem Marker „@“ eine Sprache.

2.4 Ontologien und Nomisma.org

Einer der wichtigsten Gründe für die Nutzung von URIs, ist die Vermeidung von zweideutigen Angaben und der Nutzung eindeutiger Bezeichner. Beispielsweise kann in einem Navigationsprogramm der Kontinent des Eiffelturms als „Europa“ eingespeichert sein und in einem anderen als „Europe“, während ein drittes Programm für Historiker den Namen „Europa“ als den Namen einer phönizischen Prinzessin interpretiert[5]. Doch auch hier können einfache URIs keine eindeutige Bezeichnung gewährleisten. So lässt sich aus der URI *http://geographic.com/europa* für den Menschen vermuten, dass es sich um den Kontinent handelt, während die URI *http://greek-mythology.com/europa* vermutlich eher die Figur Europa meint. Bei Computern und Programmen stoßen wir hier jedoch wieder auf die gleiche Schwäche wie schon im WWW - die der nicht maschinenlesbaren Informationen. Ebenso können für diese beiden Beispiele weitere URIs existieren, sodass weder ein eindeutiger Bezeichner, noch eine eindeutige Interpretation

[5]Die aus der griechischen Mythologie stammende Prinzessin trägt den Namen Europa und gilt als Namensgeberin des gleichnamigen Kontinents[Obe19].

der URI gewährleistet ist.

Einen Lösungsansatz schaffen hier sogenannte Ontologien. Diese sind Vokabularien, welche einer Menge von URIs (meist in Form von Namensräumen) jeweils eine Beschreibung (bzw. Semantik) zuordnen, sowie Regeln für die Verwendung dieser URIs bestimmen [Gei09]. Einfach gesagt, geben Ontologien eine Art Anleitung an, welche URIs in konkreten Fällen genutzt werden sollen und auf was sich diese URIs beziehen. Stellt sich eine Ontologie somit als Standard für einen Anwendungsfall dar, ergibt sich dadurch eine eindeutigere Nutzung der enthaltenen URIs.

Beispiele für solche Standards sind die vom W3C definierten Namensräume *rdf* und *rdfs*[6]. Der Namensraum *rdf* beinhaltet grundlegende Eigenschaften für RDF-Graphen, wie die bereits beim Warenkorb genutzte Eigenschaft *rdf:type*, um einem Knoten eine Klasse zuzuordnen. Der Namensraum *rdfs* (auch RDF Schema) hingegen, beinhaltet Ressourcen, um Hintergrundwissen über die in einem Vokabular verwendeten Begriffe anzugeben [Hit+08]. Dazu gehört zum Beispiel *rdfs:Class* zum definieren neuer Klassen.

Ein weiteres Beispiel für eine Ontologie, welche später in der Arbeit von größerer Relevanz sein wird, ist die Nomisma Ontologie[7]. Diese stellt Konzepte für den Bereich der Numismatik (Münzkunde) bereit. Dort wird beschrieben, wie typische Aussagen über Münzen und deren Funde in RDF-Graphen modelliert werden. Die Aussage „Münze 1 wurde in Athen geprägt“ mit Ressource *bsp:coin_1*, wird dadurch in RDF-Graphen der Nomisma Ontologie mit dem Tripel *bsp:coin_1 nmo:hasMint nm:athens* dargestellt. Abbildung 2.6 zeigt eine Münze mit einigen Attributen aus der AFE Datenbank für Münzfunde aus Europa, auf die später genauer eingegangen wird.

[6] *rdf* und *rdfs* sind die üblich verwendeten Präfixe für die beiden Namensräume `http://www.w3.org/1999/02/22-rdf-syntax-ns#` und `http://www.w3.org/2000/01/rdf-schema#`.

[7] Die offizielle Seite der Nomisma Ontologie ist erreichbar unter `https://nomisma.org/ontology`.

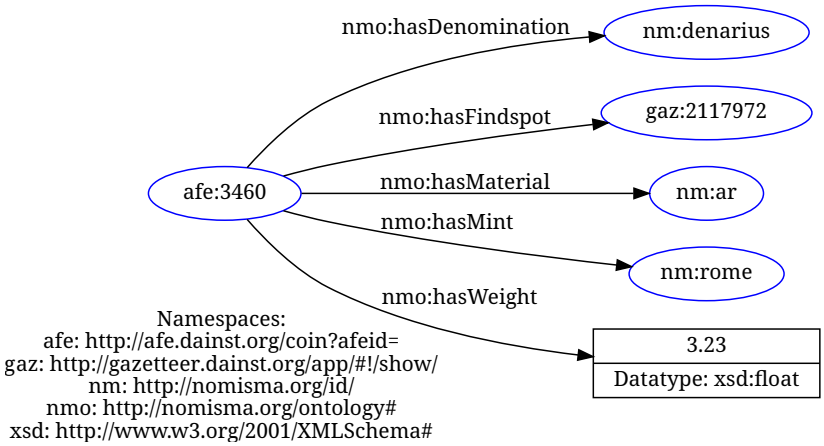


Abbildung 2.6: RDF-Graph einer Münze der AFE Datenbank und deren Attribute.

2.5 SPARQL

Bisher wurde betrachtet, wie RDF-Graphen aufgebaut und beschrieben werden. Im Folgenden wird erläutert, wie auf solche Daten zugegriffen, bzw. wie der RDF-Graph nach bestimmten Daten durchsucht werden kann. Dazu wurden mehrere Anfragesprachen für RDF vorgeschlagen, wobei sich die 2004 vorgeschlagene Sprache *SPARQL* (*SPARQL Protocol and RDF Query Language*) als W3C Standard etabliert hat. Eine SPARQL-Anfrage wie in Abbildung 2.7 besteht aus den drei Teilen PREFIX, SELECT und WHERE [Gei09]. Durch das Schlüsselwort „PREFIX“ kann ähnlich wie in Turtle ein Namensraum mit einem Präfix abgekürzt werden. Dazu wird für jeden Namensraum am Anfang der Anfrage eine Zeile „PREFIX *Präfix*: <*Namensraum*>“ eingefügt. Danach folgen, ähnlich zu SQL, ein SELECT-Block und der optionale WHERE-Block. Im SELECT-Block wird durch Variablen der Form *?Bezeichnung* beschrieben, welche Daten ausgegeben werden sollen. Diese Variablen können danach im WHERE-Block genauer durch sogenannte Graph-Pattern im Turtle-Format spezifiziert werden. Der Graph wird bei Ausführung der SPARQL-Anfrage nach dem Graph-Pattern durchsucht und bei einem erfolgreichen Fund werden die im SELECT-Block aufgelisteten Variablen der Ausgabe hinzugefügt. Bei der Ausführung der SPARQL-Anfrage aus Abbildung 2.7 auf einen RDF-Graphen der AFE Datenbank, wird somit nach Münzen gesucht, die in Rom geprägt wurden und ein eingetragenes Material besitzen. Von diesen wird jeweils nur das Material der Ausgabe hinzugefügt, sodass beim Ausführen der

```
PREFIX nm: <http://nomisma.org/id/>
PREFIX nmo: <http://nomisma.org/ontology#>

SELECT ?object
WHERE {
    ?subject nmo:hasMint nm:rome;
        nmo:hasMaterial ?object.
}
```

Abbildung 2.7: Beispiel einer SPARQL-Anfrage.

SPARQL-Anfrage auf den in Abbildung 2.6 enthaltenen Graph die Ausgabe *nm:au* zu erwarten ist.

SPARQL bietet eine Reihe weiterer Schlüsselwörter an, um auch komplexere Anfragen beschreiben zu können. Im späteren Verlauf der Arbeit werden bei SPARQL-Anfragen unter anderem das Schlüsselwort MINUS verwendet, um zu beschreiben, welche Graph-Pattern eine Ausgabe nicht besitzen darf. Weitere fortgeschrittenere Konzepte von SPARQL sind für die späteren Benchmarktests nur bedingt relevant, weshalb an dieser Stelle nicht weiter auf diese Methoden eingegangen wird. Für einen tieferen Einblick in die Anfragesprache SPARQL empfiehlt sich beispielsweise [Du 13].

2.6 Reification und RDF-star

Sobald für einen Sachverhalt ein Bezeichner in Form einer URI existiert, lassen sich durch RDF-Graphen viele Aussagen dazu darstellen. Jedoch gibt es auch einfache Fälle, in denen RDF an seine Grenzen stößt, wie im folgenden Beispiel: „Buch 1 erschien 1997. Das steht auf der Seite `http://de.wikipedia.org/wiki/Harry_Potter_und_der_Stein_der_Weisen`“[8]. Ein Teil der Aussage lässt sich durch das Tripel *bsp:buch1 bsp:hatErscheinungsjahr 1997* darstellen. Im zweiten Satz stellt sich jedoch zwangsläufig die Frage, welche Ressource als Subjekt genutzt werden soll, um die Aussage anzugeben. Anstatt eine der drei Ressourcen als Subjekt zu nutzen, sollte sich das Tripel auf die gesamte Aussage beziehen.

Eine mögliche Lösung für diesen Anwendungsfalls bietet die *RDF-Reification*, ein

[8]Die Gültigkeit aller in dieser Arbeit referenzierten Webseiten wurden am 17.07.2023 überprüft.

Konzept, um eine beliebige Aussage zu „verdinglichen“ (reifizieren). Dazu wird ein Hilfsknoten erzeugt, welcher die Aussage repräsentiert und mit Hilfe des *rdf*-Namensraums werden die dazu gehörigen Attribute zugeordnet. Für das oben genannte Beispiel entsteht so der in Abbildung 2.8 gezeigte RDF-Graph. Da der leere Knoten die Verdinglichung der Aussage darstellt, kann dieser nun als Subjekt für die Quellenangabe dienen.

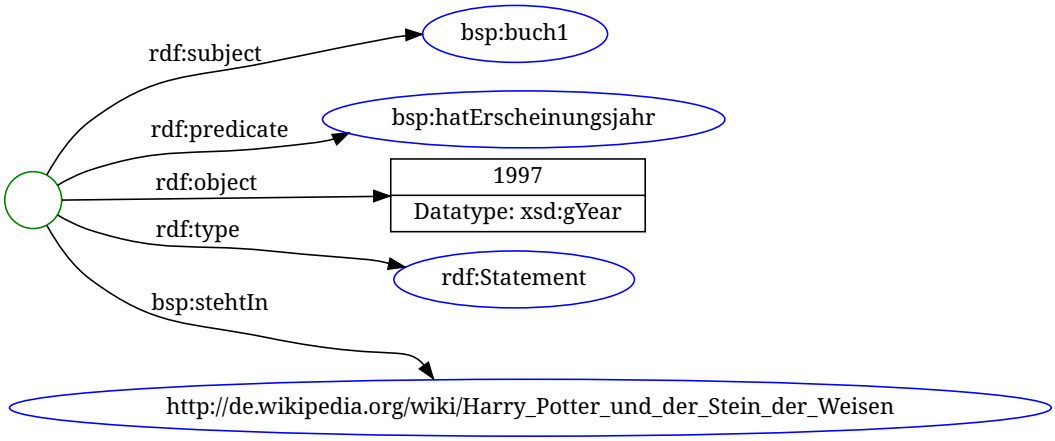


Abbildung 2.8: Standard RDF-Reification zur Angabe einer Quelle.

Ein Nachteil dieser Reification zeigt sich in der Größe des daraus entstandenen RDF-Graphen. Um mit Hilfe der RDF-Reification eine Aussage über eine Aussage zu formulieren, werden fünf zusätzliche Kanten, also auch fünf zusätzliche Tripel im RDF-Dokument benötigt. Ein weiterer Lösungsansatz, welcher weniger Tripel benötigt, ist *RDF-star* (auch *RDF**) eine im W3C vorgeschlagene Erweiterung von RDF. RDF-star verfolgt dabei den Ansatz von *verschachtelten Tripeln*, indem auch ganze Tripel als Subjekt oder Objekt eines weiteren Tripels enthalten sein können. Zwar besitzt RDF-star nicht den Status eines W3C-Standards, jedoch wird diese Erweiterung bereits in den Datenbanksystemen Blazegraph und AnzoGraph und von dem Open-Source-Framework Apache Jena[9] unterstützt [Har19]. Das eben gezeigte Beispiel lässt sich dadurch wie in Abbildung 2.9 darstellen. Die Aussage über die Quellenangabe bezieht sich hier nicht mehr auf einen leeren Knoten, sondern auf die gesamte Aussage „Buch 1 erschien 1997“.

[9] Apache Jena Fuseki wird in den späteren Benchmarktests als SPARQL-Server genutzt.

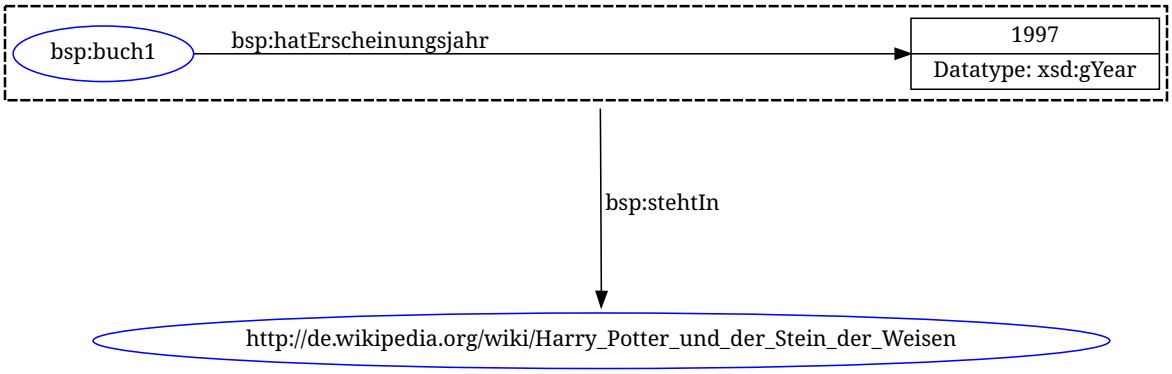


Abbildung 2.9: Verwendung von RDF-star zur Angabe einer Quelle.

Wie genau ein verschachteltes Tripel im Turtle-Format aussieht, ist in Abbildung 2.10 zu erkennen. Dazu wurde das Tripel in zweifachen spitzen Klammern anstelle eines Subjekts eingefügt. Der verbleibende Teil ist äquivalent zur normalen Turtle-Syntax. Die erweiterte Syntax wurde passend zur Erweiterung, *Turtle-star* (auch *Turtle**) genannt. Da Verschachtelungen rekursiv angewandt werden können, lassen sich auch mit spitzen Klammern eingebundene Tripel weiter verschachteln. Zu beachten ist jedoch, dass Tripel in spitzen Klammern jeweils keine Kanten des daraus entstehenden RDF-Graphen darstellen. Somit sind Tripel in spitzen Klammern als ein eigenständiger Graph zu betrachten, welcher als Ressource für den eigentlich beschriebenen Graphen dient. Für das Beispiel bedeutet das, dass der RDF-Graph nur eine Kante besitzt.

```
1 @prefix bsp: <http://beispiel.com/> .
2
3  <<bsp:buch1 bsp:hatErscheinungsjahr "1997">>  bsp:stehtIn
     <http://de.wikipedia.org/wiki/Harry_Potter_und_der_Stein_der_Weisen> .
```

Abbildung 2.10: Beispiel eines RDF-star-Graphen im Turtle-star-Format.

Neben Turtle-star braucht es zusätzlich noch eine erweiterte Abfragesprache, um die so modellierten Daten abfragen zu können. Dazu wurde für die bereits bekannte Abfragesprache SPARQL auch eine Erweiterung *SPARQL-star* (auch *SPARQL**) vorgeschlagen, welche die neu hinzugekommenen Methoden von RDF-star interpretieren kann. Eine SPARQL-star-Anfrage, welche alle genutzten Quellen anfragt, ist in Abbildung 2.11 zu sehen. Ausgeführt auf dem RDF-star-Graph in Abbildung 2.9 ist somit

```
PREFIX bsp: <http://beispiel.com/>

SELECT ?object
WHERE {
    << ?subject bsp:hatErscheinungsjahr ?object >> bsp:stehtIn
        <http://de.wikipedia.org/wiki/Harry_Potter_und_der_Stein_der_Weisen> .
}
```

Abbildung 2.11: Beispiel einer SPARQL-star Anfrage mit verschachteltem Tripel.

die Wikipedia-URL als Ergebnis zu erwarten.

Im Vergleich zur RDF-Reification, benutzt RDF-star somit weniger Kanten, um eine Aussage über eine Aussage zu formulieren. Während RDF-Reification fünf Tripel benötigt, kommt RDF-star mit einer Kante im RDF-Graph und einer weiteren in einem „internen Graph“ aus. Ob sich durch die Nutzung weniger Kanten auch eine effizientere Abfrage ergibt, ist damit jedoch noch nicht geklärt. Die später in dieser Arbeit durchgeführten Benchmarktests geben dazu einen ersten Einblick in die Leistungsunterschiede. Für einen genaueren Blick auf RDF-star, Turtle-star und SPARQL-star sind zum Beispiel der offizielle Abschlussbericht [Har+21] oder auch [HT14] und [Har19] zu empfehlen.

2.7 Unsicherheiten in RDF-Graphen

In den bisher betrachteten Graphen, wurden Aussagen entweder als sichere Fakten behandelt, oder es war nicht ersichtlich, bei welchen es Zweifel gibt. Unglücklicherweise enthalten Datenbestände in vielen Fällen eine gewisse Unsicherheit, wie z.B. bei Ergebnissen fehleranfälliger Messungen. Während solche Unsicherheiten oftmals unbewusst in Daten enthalten sind und eine Angabe der Unsicherheit somit nur eingeschränkt möglich ist, treten in manchen Anwendungsfällen auch „bewusste Unsicherheiten“ auf. So sollte ein Arzt beispielsweise bei nicht eindeutigen Diagnosen diese Unsicherheit mit angeben, damit in Zukunft weitere Untersuchungen angestellt werden können.

Wie solche Unsicherheiten angegeben werden, ist dabei sehr unterschiedlich. Zum einen hat sich bisher noch kein allgemein anerkannter Standard durchgesetzt, was vermutlich dadurch begünstigt wird, dass die wenigsten Anwendungsfälle unsichere

Informationen verwerten. Zum anderen werden auch verschieden mächtige Angaben benötigt. Sollte sich der oben erwähnte Arzt nicht sicher sein, ob er einen Tumor oder eine einfache Entzündung entdeckt hat, sollte die Angabe der Unsicherheit mehrere Alternativen aufnehmen können. Bei bewussten Störungen in Messungen, lassen sich möglicherweise auch Wahrscheinlichkeiten über die Unsicherheit angeben. Neben diesen beiden Beispielen gibt es zudem viele weitere Möglichkeiten wie die Angabe von geordneten Alternativen oder die Angabe von unmöglichen Werten. Das lässt vermuten, dass ein möglichst mächtiger Standard für die Angabe von unsicheren Daten nicht ohne komplexes Regelwerk aufgestellt werden kann.

Auch in den Empfehlungen des W3C, fehlt bislang ein festgelegter Standard für die Darstellung von Unsicherheiten in RDF-Graphen. Erste Ansätze bieten manche Ontologien, welche auf das Themenfeld der unsicheren Daten gestoßen sind wie z.B. *crm*, *nm* und *edtfo* (siehe Anhang A). In der Arbeit [SS22] wurden zu diesem Zweck acht verschiedene Ansätze für die Modellierung von Unsicherheiten in RDF-Graphen festgehalten. Um diese Modellierungen genauer miteinander zu vergleichen, werden diese in Abschnitt 4 mit Hilfe eines Benchmarks analysiert.

3 Richtlinien für Benchmarktests

“Alle Wege führen nach Rom„. Nicht umsonst ist dieser Satz ein übliches Sprichwort dafür, dass es mehrere Möglichkeiten gibt ein Ziel zu erreichen. Auch in der Informatik gibt es häufig nicht die eine richtige Lösung für eine Fragestellung, weshalb es umso wichtiger ist, Verfahren zum Vergleichen und Testen verschiedener Lösungsansätze zur Verfügung zu haben. Im Bereich von Software und Hardware ist hier die Rede von *Benchmarktests* bzw. *Benchmarks*, welche verschiedene Systeme in Bezug auf Leistung und Funktionalität miteinander vergleichen [Saw93].

Während der Recherche dieser Arbeit konnte keine einheitliche Definition von Benchmarks aufgefunden werden. Im Rahmen dieser Arbeit wird unter einem Benchmarktest eine Analyseeinheit verstanden, in der verschiedene Systeme unter einem bestimmten Kontext verglichen werden. Ein Benchmark umfasst einen oder mehrere Benchmarktests. Allerdings sollten Benchmarks sorgfältig konzipiert und durchgeführt werden, um genaue und unvoreingenommene Ergebnisse liefern zu können [WSC+19]. In den Arbeiten [WSC+19; Hup09] werden dazu wichtige Richtlinien definiert, welche gute Benchmarks erfüllen sollten, von denen die grundlegendsten im Folgenden vorgestellt werden.

3.1 Definition von Zweck, Umfang und Methoden

Benchmarks können aus verschiedenen Gründen durchgeführt werden und je nachdem, welchem Zweck diese dienen, sollte auch der Benchmark danach ausgerichtet werden. Häufig werden solche von Methodenentwickler*innen erstellt, um eine neue Methode mit bestehenden zu vergleichen oder sie werden von neutralen Forschenden durchgeführt, um einen möglichst unparteiischen Vergleich der Methoden durchzuführen [WSC+19]. In beiden Fällen sollte jedoch sowohl der Zweck, als auch der Umfang anfangs transparent definiert werden, um so die Relevanz und das Ziel des Benchmarks auszudrücken. Je nachdem zu welchem Zweck der Benchmark durchgeführt wird, resultieren daraus auch verschiedene Aspekte, die den Umfang und die Methoden betreffen.

Methodenentwickler*innen sollten ihre neuen Ansätze vor allem mit solchen vergleichen, die dem aktuellen Stand der Technik entsprechen und die in der Praxis häufig

eingesetzt werden. Zwar kann der Umfang des Benchmarks dabei so angepasst werden, dass dieser möglichst viele Vorteile der neuen Methode aufdeckt, jedoch darf dabei keine Benachteiligung der anderen Methoden zustande kommen. Neutrale Benchmarks sollten hingegen so konzipiert sein, dass der*die Forschende mit den ausgewählten Methoden gleich gut vertraut ist, um so eine Voreingenommenheit zu minimieren. Ebenso sollte der Umfang des Benchmarks möglichst breit gewählt werden, um so einen Gesamtüberblick über alle Vor- und Nachteile der Methoden aufzudecken.

3.2 Definition der Testumgebung

Um die erzeugten Ergebnisse einordnen zu können, ist eine Transparenz über die genutzten Ressourcen notwendig. Dazu gehören vor allem die genutzte Hardware, Softwareversionen und Betriebssysteme. Durch die Nutzung der aktuellsten Versionen lassen sich die derzeit optimalen Einzelkomponenten nutzen [WSC+19], trotzdem sollte die genutzte Testumgebung, wenn möglich, ähnlich zu der Umgebung eines Nutzers passen, um so realistische Ergebnisse zu gewährleisten [Hup09].

Auch die Wahl der genutzten Parameter haben einen Einfluss auf die Ergebnisse des Benchmarks. Manche Methoden haben verschiedene Parameter, welche je nach Festlegung, unterschiedliche Leistungen aufzeigen. Während Methodenentwickler*innen sich auf die für ihre Methode optimalen Parameter festlegen können, sollten neutrale Benchmarks eine möglichst umfassende Reihe von Parametern nutzen. In beiden Fällen sollte die Wahl der Parameter jedoch transparent vermittelt werden.

Einflussreich ist ebenso die Wahl der genutzten Testdatensätze. Künstlich erzeugte Datensätze sind vor allem dann nützlich, wenn nur wenige oder keine Datensätze öffentlich zugänglich sind, welche für den Benchmark in Frage kommen. Neben vollständig künstlichen Datensätzen, gibt es auch teilweise künstliche bzw. manipulierte Datensätze, mit denen sich die Methoden leicht auf bestimmte Eigenschaften testen lassen. Unter diesen sollte jedoch mindestens ein realer Datensatz enthalten sein, da diese häufig eine andere Struktur besitzen und dadurch praxisorientierte Ergebnisse liefern.

3.3 Festhalten der Bewertungskriterien

Die Bewertung der Methoden erfolgt durch Leistungskennzahlen für zum Beispiel Laufzeit, Speicherbedarf, Stromverbrauch oder Skalierbarkeit. Auch hier können Methodenentwickler*innen die zu ihrer Methode passenden Leistungskennzahlen auswählen, während neutrale Benchmarks eine möglichst umfangreiche Betrachtung der Leistung enthalten sollte [WSC+19]. Für die Messung sollten zusätzlich verschiedene Messverfahren ausgetestet und das dazu passendste ausgewählt werden. Mögliche Messverfahren sind in Laufzeitmessung bspw. die Ermittlung der Mediane oder Mittelwerte. Neben der Leistung sollte auch die Funktionalität der Methoden einbezogen werden, worunter beispielsweise die Multifunktionalität und die Nutzer- und Entwicklerfreundlichkeit fallen. Bei durchweg schlechten Methoden, sollten neutrale Benchmarks diese ebenfalls benennen, damit diese in Zukunft vermieden werden können.

Durch Metriken können Ergebnisse weiter verarbeitet und aufbereitet werden. Beispiele für Metriken sind etwa der mittlere quadratische Fehler, Abstandsmaße oder Kreuzentropie [WSC+19]. Die Wahl der Metrik hat dabei einen großen Einfluss auf die Interpretation der Ergebnisse, weshalb eine möglichst Methoden-neutrale und repräsentative Metrik ausgewählt werden sollte.

3.4 Interpretation der Ergebnisse

Je umfangreicher ein Benchmark definiert wurde, desto wichtiger ist eine saubere Darstellung der Ergebnisse, um dem*der Leser*in die Relevanz des Benchmarks zu verdeutlichen. Bei der Bewertung verschiedener Kennzahlen entstehen höchst wahrscheinlich auch verschiedene Rangreihenfolgen. Daher kann ein Gesamtranking oftmals sinnvoll eingesetzt werden, um einen Überblick der Ergebnisse zu zeigen.

Ein weiterer wichtiger Aspekt ist die Behandlung verschiedener Sichtweisen oder Anwendungsfälle. Während Kennzahlen wie Laufzeiten vor allem die Nutzer*innen der Methoden interessiert, sind Entwickler*innen zudem auch an der Komplexität der Implementation interessiert. Ergebnisse sollten optimalerweise alle möglichen Anwendungsfälle beleuchten und mit Hilfe von Empfehlungen festgehalten werden. Beschränkungen des Benchmarks sollten ebenfalls transparent dargelegt werden [Hup09],

beispielsweise bei Festlegungen von Parametern oder die Eingrenzung bestimmter Bereiche [HB15].

3.5 Verfügbarkeit und Erweiterbarkeit des Benchmarks

Die Ergebnisse des Benchmarks gewinnen stark an Aussagekraft, wenn diese auch reproduzierbar sind. Dafür sollte der verwendete Code und die verwendeten Daten verfügbar gemacht werden, damit die Reproduzierbarkeit der Ergebnisse gewährleistet wird. Zudem kann dadurch der Benchmark auch auf anderen Systemen (andere Hardware oder Betriebssystemen) durchgeführt werden, falls diese dem Forschenden nicht zur Verfügung stehen. Idealerweise wurde der Benchmark zudem so konstruiert, dass er mit anderen Methoden erweitert werden kann, sodass zukünftige Durchführungen nicht auf die bereits verwendeten Methoden beschränkt sind und auch später entwickelte Methoden hinzugenommen werden können. So ist es Anwender*innen möglich den Benchmark mit eigenen Daten, Parametern und Methoden zu wiederholen, um Ergebnisse zu erzeugen, die zu ihrem Anwendungsbereich passen.

4 Beschreibung und Durchführung des Benchmarks

Die in dieser Arbeit durchgeführten Benchmarktests haben zum Ziel, gute und schlechte Ansätze für die Modellierung von Unsicherheiten zu identifizieren. Durch die Vielzahl verschiedener Ansprüche, die an solche Modellierungen gestellt werden können, ist es unwahrscheinlich, wenn nicht sogar unmöglich, eine perfekte Lösung zu finden. Jedoch lassen sich durch einen Vergleich für einen gegebenen Anwendungsfall die dazu passenden Modellierungen auswählen oder weitere effiziente Lösungen finden.

4.1 Setup des Benchmarks

Zunächst wird das Setup des Benchmarks definiert. Unter „Setup“ werden in diesem Zusammenhang die Bestandteile und Methoden verstanden, die zur Erfüllung der in Abschnitt 3 enthaltenen Richtlinien benötigt werden. Die wichtigsten Fragen, die dieses Unterkapitel klären soll sind:

1. Welche Modellierungen werden miteinander verglichen?
2. Wie lassen sich die Modellierungen vergleichen?
3. Nach welchen Aspekten werden die Modellierungen verglichen?
4. Welche Ressourcen wurden für den Benchmark genutzt?

4.1.1 Modellierungen

Auch wenn bisher noch kein durch den W3C festgelegter Standard zur Angabe von Unsicherheiten in RDF-Graphen existiert, bieten manche Ontologien bereits Ressourcen für solche Angaben. Die Arbeit [SS22] befasst sich mit solchen Ontologien und zeigt acht verschiedene Modellierungen, die in RDF-Graphen genutzt werden können. Die Arbeit bezieht sich dabei auf ungewichtete und gewichtete Unsicherheiten, sowie auf die Angabe von alternativen Werten. Da die dort durchgeführte Analyse einen geringen Umfang besitzt, werden für diese Arbeit die Ansätze aus [SS22] genutzt, um einen umfangreicheren Benchmark durchzuführen. Zudem wird der Benchmark um eine RDF-star Modellierung erweitert, sodass das Potential der Erweiterung ebenfalls untersucht wird.

Um einen Überblick über die genutzten Modellierung zu bekommen, werden diese im Folgenden kurz vorgestellt. Alle Modellierungen beziehen sich auf Ressourcen bereits bestehender Namensräume, sodass keine neuen URIs angelegt werden müssen. Darunter sind bspw. Namensräume von CIDOC CRM[10] für die Modellierungen 3, 4 und 5. Diese wurden in [SS22] erstmals festgehalten und unter Absprache mit Dr. Martin Doerr (erster Vorsitzender der CRM Special Interest Group) erstellt. Eine genauere Beschreibung der ersten acht Modellierungen inklusive deren Entstehung, ist in [SS22] zu finden[11]. Zudem wurden die gezeigten Graphen im elektronischen Anhang unter *unco/data/thesis_data/code_snippets/models* im Turtle-Format hinterlegt. Jede Modellierung wird zunächst anhand von Beispiel 4.1.1 beschrieben und auf wichtige Eigenschaften der Modellierungen hingewiesen. Wenn nicht anders angegeben, lassen sich alternative Werte nach dem gleichen Verfahren angeben, wie die dort gezeigte Unsicherheit. Der Präfix *bsp* bezieht sich wie bereits in Abschnitt 2 auf den fiktiven Namensraum *http://beispiel.com/*.

Beispiel 4.1.1. *bsp:coin_0* wurde in Rom geprägt und für *bsp:coin_1* ist es nicht sicher, ob diese in Rom geprägt wurde.

Modellierung 1: Umformung der RDF-Reification

Die erste Modellierung basiert auf der in Abschnitt 2.6 beschriebenen Methode der RDF-Reification und stammt aus einer Idee des CIDOC CRM in Verbindung mit den Projekten [Ale12; Ale+14; Mus14]([TW14]). Während die Reification dazu genutzt wird eine Aussage zu verdinglichen, wird hier die gleiche Idee genutzt, um die Aussage als unsicher zu markieren. In Abbildung 4.1 ist diese Modellierung anhand von Beispiel 4.1.1 dargestellt.

Auffällig ist hierbei, dass die Angabe eines unsicheren Wertes nur durch die Nutzung von sechs Tripeln möglich ist. Bei Datensätzen mit vielen unsicheren Daten, kann dadurch der RDF-Graph rasant an Größe gewinnen. Auch bietet diese Modellierung lediglich die Möglichkeit, Daten als unsicher zu markieren und bietet keine Lösung

[10] CIDOC CRM ist nach eigenen Angaben ein theoretisches und praktisches Instrument für die Informationsintegrität im Bereich des kulturellen Erbes und wurde von der CRM Special Interest Group entwickelt [Gro06].

[11] Die Nummerierung der Modellierungen wurde beibehalten, sodass die hier vorgestellte Modellierung $X < 9$ in [SS22] unter Solution X zu finden ist.

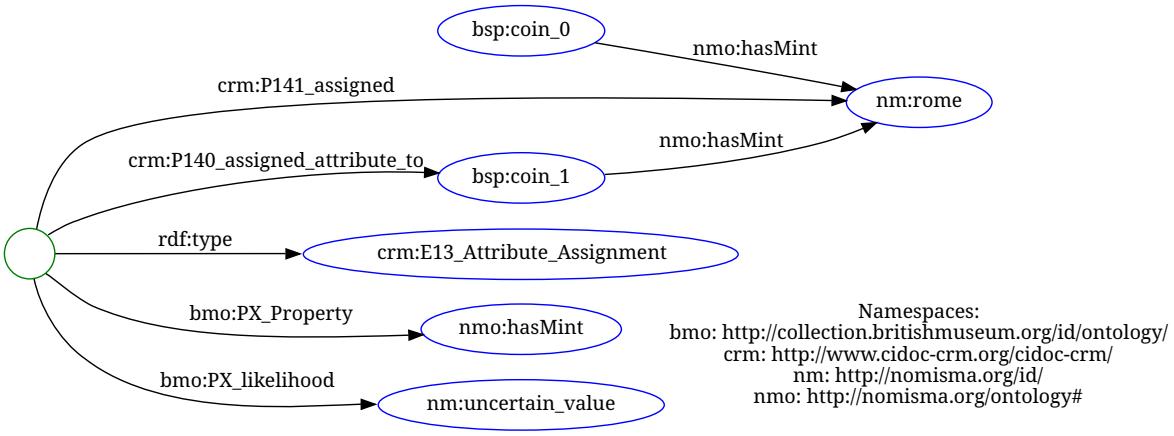


Abbildung 4.1: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 1.

für komplexere Anforderungen, wie gewichtete Unsicherheiten. Bei der Nutzung von SPARQL-Anfragen muss außerdem auf Besonderheiten geachtet werden. Beispielsweise bei der Anfrage aller Münzen mit ihren Prägeorten in Abbildung 4.2.

Als Ergebnis sind hier nämlich sowohl die sicheren, als auch die unsicheren Münz-Prägeort-Paare zu erwarten. Um Anfragen zu erzeugen, die ausschließlich sichere bzw. unsichere Paare ausgeben, sind komplexere Bedingungen im WHERE-Block notwendig.

```
1 PREFIX nmo: <http://nomisma.org/ontology#>
2
3 SELECT ?subject, ?object
4 WHERE {
5     ?subject nmo:hasMint ?object .
6 }
```

Abbildung 4.2: Beispiel SPARQL-Anfrage aller Münzen mit ihren Prägeorten.

Modellierung 2: Integrierung der Unsicherheit in RDF-Aussage

Die zweite Modellierung wurde in [TW14] vorgeschlagen und unterscheidet sich zur ersten vor allem darin, dass die Beschreibung der Unsicherheit, in das Statement „eingearbeitet“ wird. So ist die Aussage *bsp:coin_1 nmo:hasMint nm:rome* nicht selbst als Tripel im RDF-Graph enthalten. Stattdessen wird ein leerer Knoten ähnlich zu einem Zwischenschritt hinzugefügt. Abbildung 4.3 zeigt das oben genannte Beispiel durch

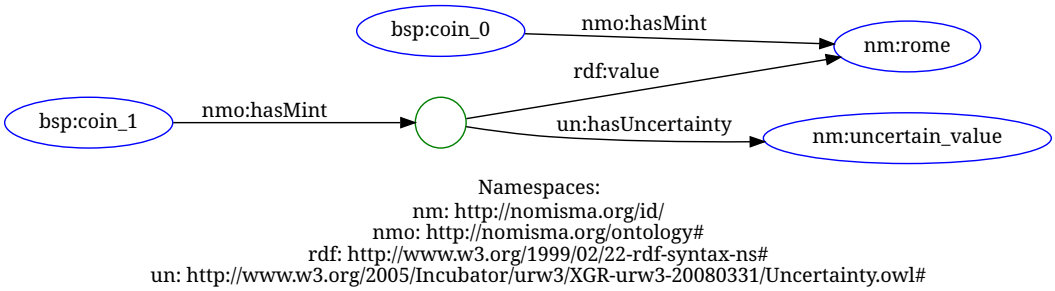


Abbildung 4.3: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 2.

diese Modellierung.

Auch mit dieser Modellierung lassen sich RDF-Aussagen nur als unsicher markieren, ohne komplexere Darstellungen mit einzubeziehen. Es ist jedoch leicht erkennbar, dass für diese Modellierung weitaus weniger Tripel benötigt werden, um eine Aussage als unsicher zu markieren. Während bei Modellierung 1 für eine unsichere Aussage sechs Tripel benötigt wurden, braucht es bei der zweiten ausschließlich drei Tripel.

Modellierung 3: Zuweisung von Zuverlässigkeiten aus CRM

Die dritte Modellierung basiert auf Ideen aus [NH16] und wurde in [SS22] unter Absprache mit Dr. Doerr konkretisiert. Für die Angabe einer Unsicherheit, werden hier die Eigenschaften aus dem CIDOC CRM Namensraum *crm* genutzt, mit denen eine Zuverlässigkeit (*reliability*) angegeben werden kann. Wie Abbildung 4.4 zeigt, werden hier im Gegensatz zu den bisherigen Modellierungen gewichtete Unsicherheiten angegeben. Gewichte können dabei beispielsweise als Wahrscheinlichkeiten interpretiert werden. Angemerkt sei hier, dass in der Arbeit und dem entwickelten Modul von [SS22] die Ressource *crm:E13* genutzt wurde, welche so nicht im Namensraum *crm* existiert. Diese wurde im Folgenden mit *crm:E13_Attribute_Assignment* ausgetauscht, auf die sich ursprünglich bezogen wurde.

Der mit einem Stern markierte leere Knoten wird dabei für alle unsicheren Aussagen (inklusive Alternativen) genutzt, sodass an diesem weitere Teilgraphen mit *crm:T1_assessed_the_reliability_of* angefügt werden können. Mit neun Tripeln pro unsicherer Aussage, erzeugt diese Modellierung den größten RDF-Graphen (die Typzuweisung von *crm:R1_Reliability_Assessment* ausgeschlossen). Jedoch können andere Eigenschaften der Modellierung von Vorteil sein. So z.B. dass vom markierten Knoten

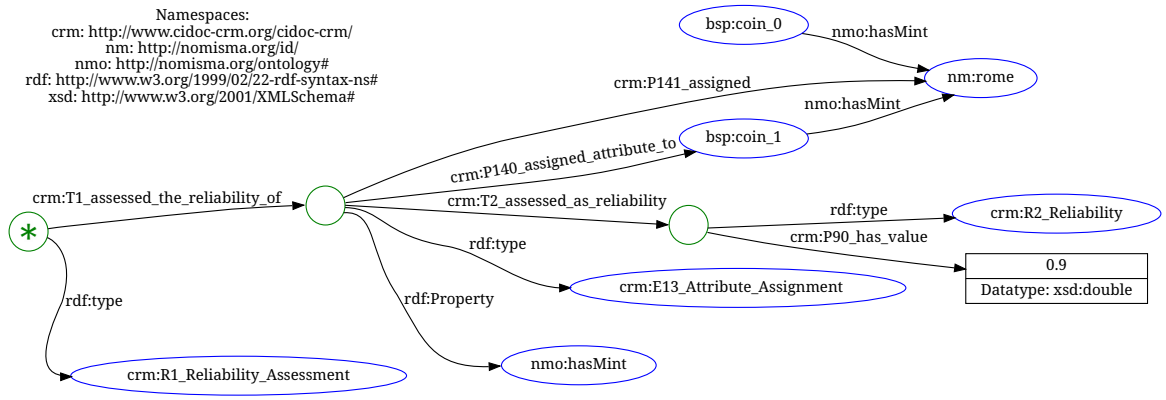


Abbildung 4.4: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 3.

aus alle unsicheren Aussagen erreicht werden können. Ob diese Eigenschaft von Fuseki ausgenutzt werden kann, um somit effizient alle unsicheren Aussagen zu erhalten, kann durch den Benchmark untersucht werden.

Modellierung 4: Zuweisung von Glaubenswerten aus CRMinf

Die vierte Modellierung wurde ebenfalls von [SS22] in Zusammenarbeit mit Dr. Doerr vorgeschlagen und nutzt den Namensraum *crminf*, eine Erweiterung von *crm* zur Unterstützung von Argumentationen [DS+14]. Anstelle der Zuweisung von Unsicherheiten oder Zuverlässigkeiten, werden hier die in *crminf* enthaltenen Eigenschaften zur Zuweisung von Glaubenswerten (*belief values*) genutzt. Abbildung 4.5 zeigt, wie das oben genannte Beispiel hiermit angegeben wird.

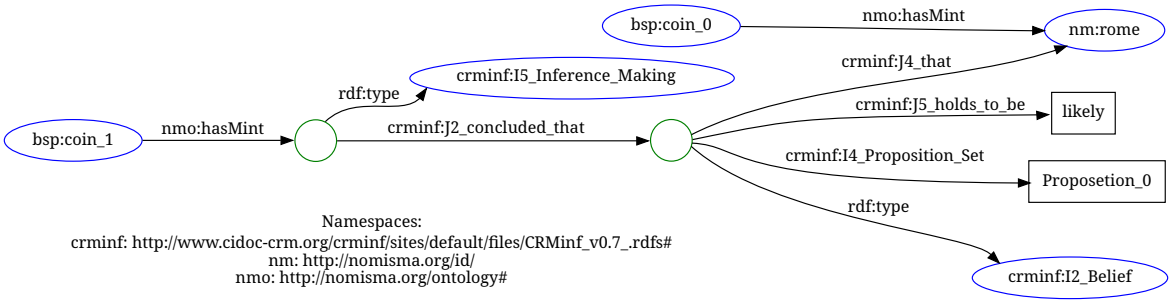


Abbildung 4.5: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 4.

Ähnlich wie in Modellierung zwei, wird hier das Tripel der Hauptaussage nicht in den Graphen übernommen, sondern ein leerer Knoten „eingeschoben“, der es ermöglicht, weitere Aussagen über die Hauptaussage hinzuzufügen. In diesem Fall wird hier der Typ *crminf:I2_Belief* zugewiesen und zusätzlich eine genauere Beschreibung über

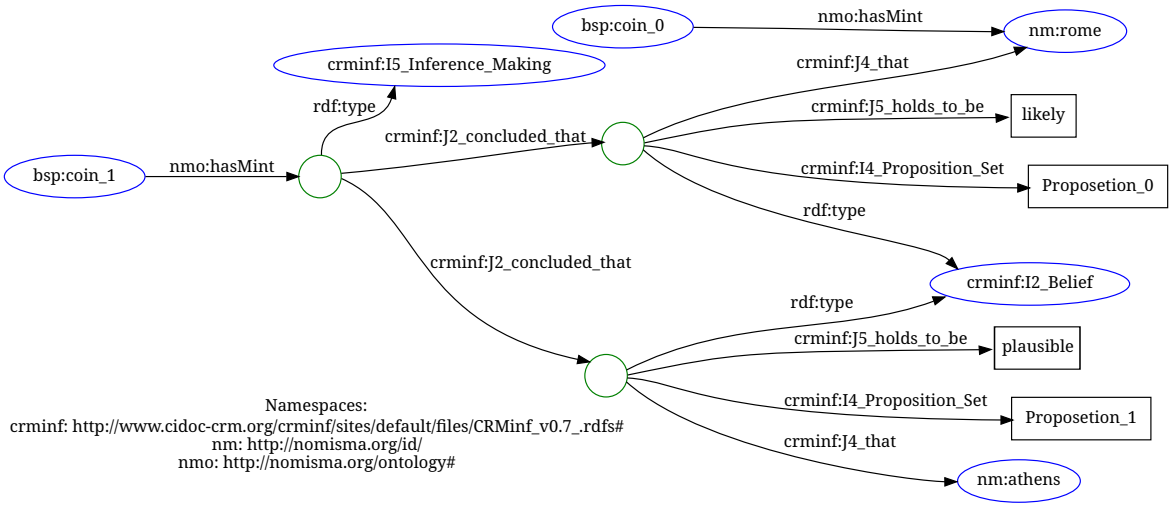


Abbildung 4.6: Erweiterter RDF-Graph von Modellierung 4.

den Typ des Glaubens. Für diese Arbeit wurden die vier Beschreibungen *uncertain*, *plausible*, *likely* und *very likely* festgelegt, um so eine bessere Interpretation der Werte zu gewährleisten. Damit die Modellierung mit den restlichen vergleichbar ist, werden die Beschreibungen als Gewichte durch die folgende Verteilung interpretiert:

Beschreibung	Gewichtsbereich
uncertain	$[0, \frac{1}{4})$
plausible	$[\frac{1}{4}, \frac{1}{2})$
likely	$[\frac{1}{2}, \frac{3}{4})$
very likely	$[\frac{3}{4}, 1)$

Tabelle 4.1: Übersicht durch welche Beschreibungen in Modellierung 4 Gewichte dargestellt werden.

Wie Abbildung 4.6 zeigt, lassen sich Alternativen durch diese Modellierung vereinfacht hinzufügen. Anstatt vom Subjekt *bsp:coin_1* aus eine neue Aussage hinzuzufügen, kann eine weitere Alternative an einen bereits bestehenden leeren Knoten angefügt werden.

Eine einfache unsichere Aussage lässt sich durch sieben Tripel darstellen, weshalb die Modellierung ebenfalls verhältnismäßig große Graphen erzeugt. Dadurch, dass für jede Alternative jedoch nur fünf weitere Tripel benötigt werden, kann sich diese Modellierung vorallem bei Graphen mit durchschnittlich vielen Alternativen als effizient

herausstellen. Zudem werden den Alternativen auch eine Ordnung (*Proposetion_X*) zugeteilt, was unter den verglichenen Modellierungen ein Alleinstellungsmerkmal ist.

Modellierung 5: Zuweisung von .2-Properties aus CRM

Die fünfte Modellierung wurde von Doerr vorgeschlagen [SS22] und nutzt, wie die beiden Modellierungen davor, Eigenschaften von CIDOC CRM. Genauer verwendet die Modellierung hierfür *.2-Properties*, welche als Eigenschaften von Eigenschaften verstanden werden können. Die Idee hierbei ist es, einer Eigenschaft wie *nmo:hasMaterial* eine weitere Eigenschaft *crm:P137.2_uncertain_material* zuzuordnen, um anzugeben, dass es sich um ein unsicheres Material handelt. Mit *amt:weight* lassen sich unsicheren Aussagen zusätzlich noch eine Gewichtung zwischen 0 und 1 zuordnen. In Abbildung 4.7 ist eine Darstellung dieser Modellierung zu sehen.

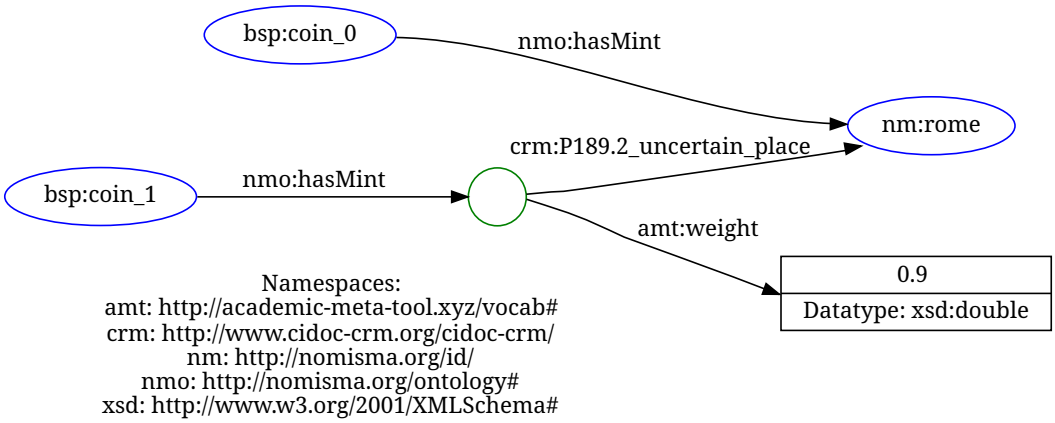


Abbildung 4.7: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 5.

Augenscheinlich besitzt diese Modellierung eine große Ähnlichkeit zur zweiten Modellierung. Beide benutzen nur drei Tripel, um ein unsicheres Statement zu beschreiben und beide enthalten nicht das eigentliche Tripel des unsicheren Statements. Vorteilhaft gegenüber der zweiten Modellierung ist hier jedoch die zusätzliche Angabe eines Gewichts.

Auffällig ist hierbei, dass die .2-Property *crm:P189.2_uncertain_place* (übersetzt: unsicherer Ort) von der Eigenschaft der Aussage abhängig ist. Somit müssen bei anderen Eigenschaften evt. auch andere .2-Properties zur Markierung der Unsicherheit verwendet werden, was nicht nur die Erstellung und das Abfragen des RDF-Graphen

komplexer macht, sondern auch, mit Blick auf das Semantische Web, das Verstehen der Aussage aus Sicht eines Computers. Für die Nomisma Ontologie werden so 14 verschiedene .2-Properties genutzt. Eine Tabelle von Eigenschaften aus der Nomisma Ontologie und den dazu gehörigen .2-Properties, ist in Anhang B zu finden.

Modellierung 6: Anpassung von RDF-Reification mit EDTFO

Die sechste Modellierung nutzt die in Abschnitt 2.6 beschriebene RDF-Reification und wurde von [SS22] vorgeschlagen. Anstelle der Nutzung von *rdf:Statement*, wird hier jedoch *edtfo:UncertainStatement* genutzt, um zu verdeutlichen, dass es sich um eine unsichere Aussage handelt. Abbildung 4.8 zeigt das oben beschrieben Beispiel mit dieser Modellierung.

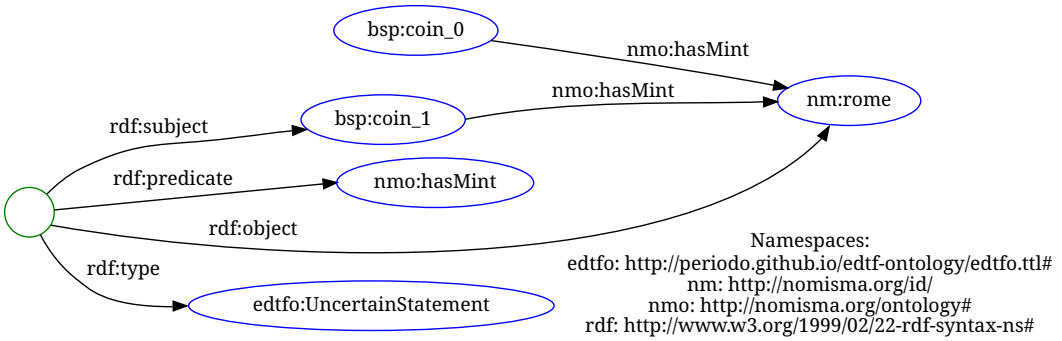


Abbildung 4.8: Darstellung von Beispiel 4.1.1 durch Modellierung 6.

Diese Modellierung besitzt einen ähnlichen Aufbau wie bereits die erste, welche sich ebenfalls an der RDF-Reification orientiert. Unterschiede besitzen die beiden Modellierungen in der Wahl der Namensräume und den daraus gewählten Eigenschaften. Zudem benötigt die sechste Modellierung ein Tripel weniger.

Modellierung 7: Approximative Aussage durch EDTFO

Auch die siebte Modellierung wurde von [SS22] vorgeschlagen und basiert auf der zweiten Modellierung. Die Idee ist die Nutzung von *edtfo:ApproximateStatement*, um eine approximative Aussage darzustellen. Eine approximative Aussage ist in diesem Kontext eine Aussage, die „höchst wahrscheinlich„ zutrifft (orig. „*possibly correct, or close to being correct*“). Dafür wird auch hier ein leerer Knoten genutzt, um auf diese Eigenschaft zu verweisen, wie in Abbildung 4.9 zu sehen.

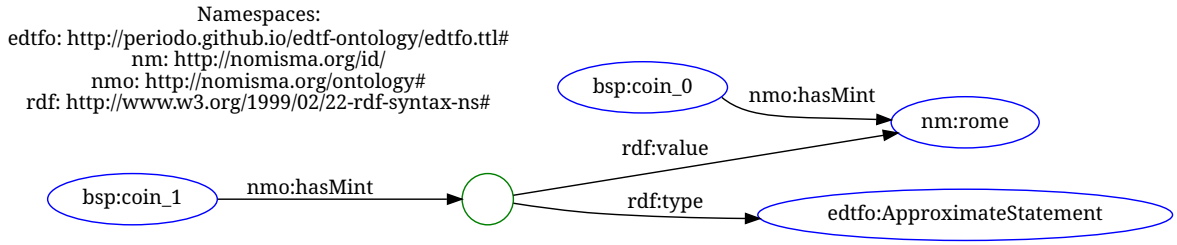


Abbildung 4.9: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 7.

Diese Modellierung unterscheidet sich von der zweiten nur in dem Tripel *b rdf:type edtfo:ApproximateStatement*, welches das Tripel *b un:hasUncertainty nm:uncertain_value* ersetzt hat. Durch die ähnliche Struktur der beiden Modellierungen sind hier auch ähnliche Laufzeitverhalten zu erwarten. Ein Vergleich der beiden kann jedoch Aufschluss über den Einfluss der Namensräume und Eigenschaften geben. Zudem wäre eine Kombination der beiden Modellierungen denkbar, bei der Aussagen sowohl als unsicher (Modellierung 2) oder als wahrscheinlich zutreffend (Modellierung 7) dargestellt werden können, ohne eine andere Graph-Struktur zu nutzen.

Modellierung 8: Zuweisen von gebündelten Unsicherheiten

Auch die achte Modellierung wurde von [SS22] vorgeschlagen und verfolgt einen neuen Ansatz mit der Bündelung von Unsicherheiten. Anstatt die einzelnen Aussagen als unsicher zu „markieren“, wie es die bisherigen Modellierungen zeigen, werden hier alle unsicheren Statements eines Subjekts gebündelt zugewiesen. Abbildung 4.10 zeigt, wie das oben genannte Beispiel mit dieser Modellierung aussieht.

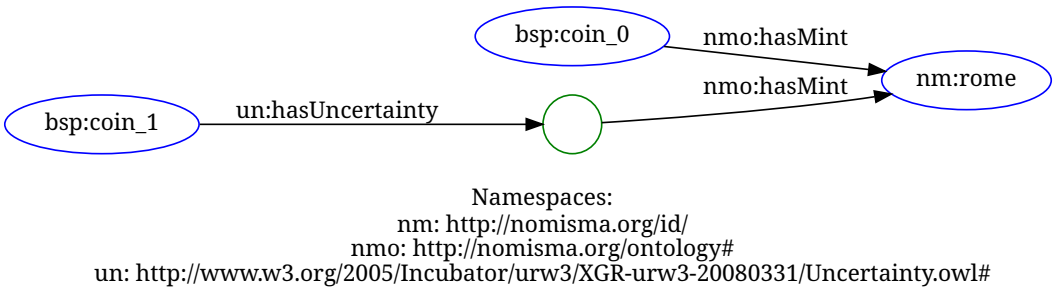


Abbildung 4.10: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 8.

Mit einer einzigen unsicheren Aussage lässt sich nicht darstellen, was unter gebündelten Unsicherheiten gemeint ist. Daher lohnt sich ein Blick auf Abbildung 4.11, bei

der eine Alternative und eine neue unsichere Aussage hinzugefügt wurden. Alternativen werden hier gleich behandelt wie einzelne Unsicherheiten, indem das alternative Objekt durch die gleiche Eigenschaft an den gleichen leeren Knoten angehängt wird. Dadurch werden alle Unsicherheiten eines Subjekts unter dem gleichen leeren Knoten zusammengefasst[12].

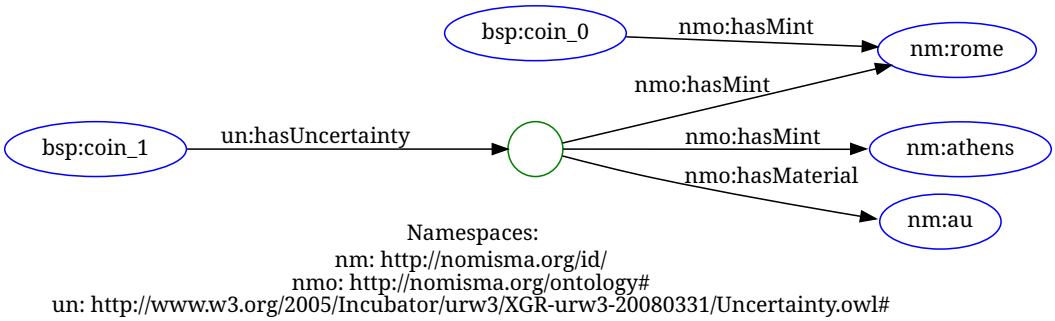


Abbildung 4.11: Angabe von Alternativen durch Modellierung 8.

Für die Angabe einer einzelnen Unsicherheit, benötigt die achte Modellierung insgesamt zwei Tripel. Jede zusätzliche Unsicherheit des gleichen Subjekts und jede Alternative kann jedoch durch ausschließlich einem Tripel hinzugefügt werden. Dadurch ist die Modellierung besonders für Datensätze vielversprechend, bei dem Subjekte mehrere unsichere Attribute besitzen.

Modellierung 9: RDF-star zur Zuweisung unsicherer Aussagen

Die letzte Modellierung ist die einzige, welche nicht aus dem Vergleich in [SS22] stammt und wurde für diesen Benchmark erstellt, um die Effizienz von RDF-star im Vergleich zur RDF-Reification zu testen. Dafür bieten sich gleich zwei verschiedene Ansätze an, welche hier als Modellierung 9a und 9b kurz beschrieben werden.

Wie in Abschnitt 2.6 bereits beschrieben, sind sowohl RDF-star, als auch RDF-Reification dazu geeignet, Aussagen über Aussagen zu tätigen. Im Bezug zu unsicheren Aussagen ist es somit naheliegend diese beiden Ansätze zu nutzen, um eine Aussage als unsicher zu beschreiben. Während die sechste Modellierung bereits auf der RDF-Reification basiert, nutzt die Modellierung 9a in Abbildung 4.12 nun die Konzepte von RDF-star, um die gleiche Aussage zu tätigen.

[12]Man beachte, dass Solution 8 in [SS22] in manchen Punkten inkonsistent ist und die Modellierung aus diesem Grund angepasst wurde.

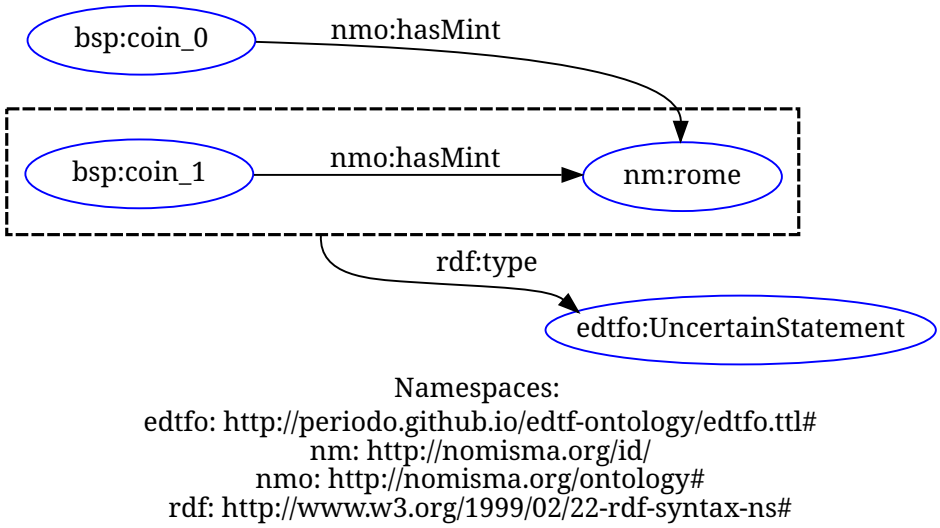


Abbildung 4.12: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 9a, durch Nutzung von RDF-star. Das Rechteck mit gestrichelten Linien stellt die Verschachtlung des darin befindenden Tripels dar.

Anstatt einen leeren Knoten zu erzeugen, um die Aussage als Ressource darzustellen, wird hier die Aussage selbst als Ressource, in Form eines verschachtelten Tripels genutzt. Dafür benötigt die Modellierung nur das verschachtelte Tripel (welches sich nicht im eigentlichen Graph befindet) und ein weiteres Tripel zur Zuweisung der Klasse *edtfo:UncertainStatement*.

Der zweite Ansatz für eine RDF-star Modellierung für Unsicherheiten, entstammt aus einem Blog-Eintrag von Olaf Hartig[13], welcher zusammen mit Bryan Thompson die RDF-star Erweiterung 2014 vorgeschlagen hat [Cha22]. Das Beispiel in dem Blog Eintrag ist der RDF-star-Graph

```
<< :bob foaf:age 23 >> ex:certainty 0.9 .
```

Somit wird der Aussage „Bob ist 23 Jahre alt“ eine Sicherheit von 0.9 zugewiesen und weißt somit im Gegensatz zu Modellierung 9a gewichtete Unsicherheiten zu. Da die Ressource *ex:certainty* jedoch nicht wirklich existiert, wird für Vorschlag 9b die Eigenschaft *un:hasUncertainty* genutzt und somit keine Sicherheit sondern Unsicherheit zugewiesen. Die endgültige Version der Modellierung ist in Abbildung 4.13 anhand von Beispiel 4.1.1 dargestellt.

[13]Der Blog Eintrag ist erreichbar unter: `https://blog.liu.se/olafhartig/2019/01/10/position-statement-rdf-star-and-sparql-star/`

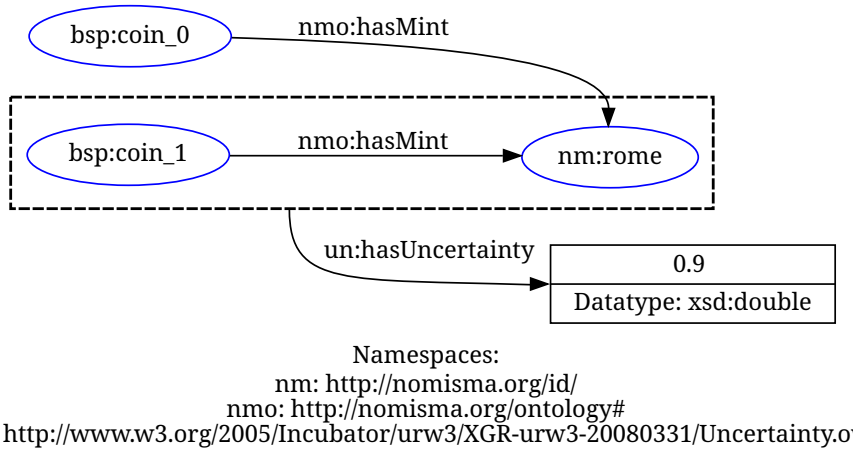


Abbildung 4.13: Darstellung von Beispiel 4.1.1 mit Hilfe der Modellierung 9b, durch Nutzung von RDF-star. Das Rechteck mit gestrichelten Linien stellt die Verschachtlung des darin befindenden Tripels dar.

Die beiden Modellierungen besitzen somit die gleiche Struktur und unterscheiden sich nur in der Verwendung der Ressourcen und Namensräume. Die geringe Anzahl genutzter Kanten und die meist einfache Struktur der SPARQL-Anfragen, machen die Modellierungen vielversprechend für effiziente Angaben von Unsicherheiten. Nachteilhaft sind bei beiden jedoch, dass die RDF-star Erweiterungen aktuell von vielen RDF Systemen nicht unterstützt wird.

4.1.2 Datensätze

Alle durchgeführten Analysen müssen auf zuvor festgelegten Datensätzen ausgeführt werden. Für diesen Benchmark wurden reale, teilweise künstliche und vollständig künstliche Datensätze genutzt. Durch die verschiedenen Arten der Datensätze, kann sowohl der praxisnahe Bezug der Ergebnisse gezeigt, als auch verschiedene Einflüsse einzelner Eigenschaften der Daten überprüft werden.

Der reale Datensatz muss gewisse Anforderungen erfüllen. Zunächst muss dieser für das Resource Description Framework „geeignet“ sein, was in diesem Kontext bedeutet, dass die Daten Aussagen der Form „Subjekt Prädikat Objekt“ darstellen und es zu den meisten Nicht-Literalen jeweils eine URI gibt. Zusätzlich muss der Datensatz bereits unsichere Aussagen enthalten, um die Modellierungen miteinander vergleichen zu können. Vor allem die letzte Anforderung trifft bisher auf die wenigsten Datensätze zu, was vermutlich daran liegt, dass sich die meisten Anwendungen auf sichere Fak-

ten beziehen und Unsicherheiten dort keine Verwendung finden. Ein weiterer, nicht zu vernachlässigender Aspekt, ist aber auch hier, dass das Fehlen eines allgemein anerkannten Standards die Angabe von Unsicherheiten verkomplizieren.

Als realer Datensatz wurde die Datenbank *Antike Fundmünzen in Europa*[14] (kurz AFE) ausgewählt, welche von der Römisch-Germanischen Kommission in Zusammenarbeit mit dem Frankfurt Big Data Lab der Goethe Universität Frankfurt am Main entwickelt wurde. Wie der Name bereits vermuten lässt, wird die Datenbank zur Erfassung antiker Fundmünzen genutzt. Da für Archäologen in vielen Fällen die Erfassung von unsicheren Daten zwingend erforderlich ist [TW14], besitzt AFE für einige Attribute bereits unsichere Angaben und eignet sich somit gut für den Benchmark. Da diese jedoch stets ungewichtet sind, wird für die Modellierung gewichteter Unsicherheiten jeweils ein pseudo zufälliges Gewicht zwischen 0 und 1 zugewiesen. Um einen kurzen Einblick in die Datenstruktur von AFE zu bekommen, ist in Tabelle 4.2 ein Überblick der für den Benchmark wichtigsten Attribute gezeigt.

	Coin	**Material**	**Mint**	**Denomination**	**Weight**
Ressourcentyp	URI	URI	URI	URI	Literal
#Werte	16554	15981	10341	13727	8237
#unterschiedliche	16554	6	48	33	1408
#Unsicherheiten	0	39	128	219	13

Tabelle 4.2: Überblick über den AFE Datensatz.

Der verwendete Datensatz von AFE beschreibt Münzen und die dazugehörigen Attribute, die mit Hilfe der Nomisma Ontologie beschrieben werden. Da der Datensatz nur teilweise öffentlich ist, kann dieser nicht im vollem Umfang zur Verfügung gestellt werden. Neben Material, Prägeort, Nominal, Gewicht und Prägestempel, werden noch weitere Attribute aufgenommen, sodass eine Tabelle mit insgesamt 33 Spalten und 16554 Zeilen entsteht. Die in Tabelle 4.2 gezeigten Attribute sind die mit den meisten eingetragenen Unsicherheiten. Die vollständige Übersicht, sowie der bereits öffentliche AFE Datensatz sind im elektronischen Anhang unter *unco/data/thesis_data/afe* zu finden.

[14] Erreichbar unter: `http://afe.dainst.org/`

Neben der AFE Datenbank, werden zusätzlich noch teilweise künstliche Datensätze genutzt. Diese basieren auf AFE, jedoch wurden einzelne Eigenschaften der Daten manipuliert, um deren Auswirkungen auf die Laufzeit der SPARQL-Abfragen zu testen. Weiterhin wurden Analysen auch auf vollständig künstlich erstellte Datensätze ausgeführt. Mit diesen lassen sich auch Vergleiche mit dichteren RDF-Graphen ausführen.

4.1.3 SPARQL-Queries

Da die Modellierungen per se keine Leistungskennzahlen besitzen, müssen für den Benchmark vergleichbare Einheiten definiert werden. Für RDF-Graphen bieten sich hierfür die Laufzeiten von SPARQL-Anfragen (im Folgenden auch *Queries*) an, da diese eine übliche Schnittstelle zum RDF-Graphen darstellen. Hierfür existieren eine Vielzahl von möglichen Anwendungsfällen und dazu gehörige Anfragen, wie z.B. die Anfrage einzelner unsicherer Werte. Zudem müssen die Queries stets immer an die jeweilige Modellierung angepasst werden, um die richtigen Ergebnisse ausgeben zu können. Damit die Modellierungen dennoch vergleichbare Queries erhalten, werden in Tabelle 4.3 sechs verschiedene Anfragen beschrieben, die für den Benchmark genutzt werden.

	Anfrage
Query 1:	Alle (m, p), bei dem Münze m den sicheren Prägeort p hat.
Query 2:	Alle (m, p), bei dem Münze m den unsicheren Prägeort p hat.
Query 3:	Alle (m, p, c), bei dem Münze m mit Sicherheit c in p geprägt wurde.
Query 4:	Alle $(s,\ p,\ o)$, sodass Aussage $s\ p\ o$ sicher ist.
Query 5:	Alle $(s,\ p,\ o)$, sodass Aussage $s\ p\ o$ unsicher ist.
Query 6:	Alle $(s,\ p,\ o,\ w)$, sodass Aussage $s\ p\ o$ sicher oder unsicher mit Gewicht $w \geq 0.5$ ist.

Tabelle 4.3: Definition der sechs Queries zum Vergleich der Modellierungen.

Wie oben bereits erwähnt, sind die Queries, die aus diesen Anfragen entstehen, von Modellierung zu Modellierung verschieden. Problematisch ist zudem, dass unter-

```
PREFIX nm: <http://nomisma.org/id/>
PREFIX nmo: <http://nomisma.org/ontology#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX un: <http://www.w3.org/2005/Incubator/urw3/XGR-urw3-20080331/Uncertainty.owl#>

SELECT ?subject ?object
WHERE {
    {
        ?subject nmo:hasMint ?object.
    }
    MINUS {
        ?object un:hasUncertainty nm:uncertain_value ;
            rdf:value ?b .
    }
}
```

Abbildung 4.14: SPARQL-Query 1 von Modellierung 2.

schiedliche Queries die gleiche Semantik haben können und somit das gleiche Ergebnis ausgeben. Ein Beispiel dafür ist die in Abbildung 4.14 gezeigte Query für Modellierung 2. Diese fragt nach allen *?subject ?object* Werten, sodass *?subject nmo:hasMint ?object* ein Tripel ist, jedoch ohne die Ergebnisse, bei denen das Objekt der leere Knoten eines unsicheren Verweises ist. Somit handelt es sich um die erste Query der Modellierung 2, welche jedoch auch mit einem *FILTER NOT EXISTS* anstelle des *MINUS* dargestellt werden kann. Obwohl durch eine solche Veränderung die Semantik der Query nicht verändert wird, zeigte sich eine schlechtere Leistung bei solchen mit *FILTER NOT EXISTS*. Um solche Unterschiede unter den Modellierungen zu vermeiden, wurde auf einen einheitlichen Aufbau und einer gleichen Nutzung der SPARQL-Methoden geachtet und können im elektronischen Anhang unter *unco/src/benchmark/queries* genauer betrachtet werden.

4.1.4 Fragestellung

Die Modellierungen können auf verschiedene Aspekte hin verglichen werden. Durch die Festlegung der SPARQL-Anfragen und deren Laufzeit, haben die Modellierungen eine vergleichbare Leistungskennzahl. Wichtig ist zudem eine Definition, in welchem Umfang die Tests durchgeführt werden. Dazu wurden in Tabelle 4.4 eine Fragestellung festgehalten, auf die sich dieser Benchmark beschränkt.

Wie verhalten sich die Query-Laufzeiten bei:
... dem AFE Datensatz?
... steigender Anzahl Unsicherheiten?
... steigender Anzahl Alternativen?
... künstlich erzeugten Datensätzen?

Tabelle 4.4: Übersicht der genutzten Hardware.

Mit Hilfe der Fragestellung können Merkmale und Eigenschaften identifiziert werden, die einen Einfluss auf die Laufzeit der Queries haben. Auch wenn die Ergebnisse des Benchmarks keine Modellierung als beste identifizieren kann, so lassen sich aus diesen möglicherweise Merkmale ableiten, die eine optimale Modellierung besitzen muss.

4.1.5 Hardware

Die für den Benchmark ausgeführten Programme wurden auf einem Linux Rechner durchgeführt. Dieser Rechner wurde mit 16 GB Arbeitsspeicher und einer 125 GB großen SSD Festplatte ausgestattet. Die genauen Angaben des Systems sind in Tabelle 4.5 aufgelistet. Das gesamte System belegte 32,3 GB der Festplatte (inklusive Linux Distribution, Programme und Datensätze). Für eine einwandfreie Durchführung der Tests wäre ein Arbeitsspeicher von doppelter Größe zu empfehlen, da manche der hier durchgeführten Tests nur in zwei Teilen ausgeführt werden konnten. Wann und warum eine solche Aufteilung notwendig war, wird im späteren Verlauf genauer beschrieben.

Betriebssystem	Prozessor	Arbeitsspeicher	Festplatte
Ubuntu 22.04.2 LTS	Intel Core i5-6500	16 GM RAM	125 GB SSD

Tabelle 4.5: Übersicht der genutzten Hardware.

Neben der Nutzung des Linux Rechners, wurde der Benchmark testweise auch auf einem Windows 11 Rechner ausgeführt. Die Ergebnisse zeigten dabei keine signifikanten Unterschiede zu denen vom Linux System, weshalb die Modellierungen sich als betriebssystemunabhängig zeigen.

4.1.6 Software

Für die Durchführung des Benchmarks wurde das Tool *UnCo* (*Uncertainty Comparator*) entwickelt, welches für einen eingegebenen Datensatz den dazugehörigen RDF-Graphen erzeugt (siehe Abbildung 4.15). Darüber hinaus enthält UnCo alle weiteren Methoden, die für die Durchführung des Benchmarks notwendig sind. Entwickelt und getestet wurde das Tool mit einem Python Interpreter der Version 3.10.6 und lässt sich über

`https://github.com/UncertaintyC/unco`

oder dem beigefügten elektronischen Anhang beziehen. Zum Zeitpunkt der Abgabe wurde die Version 1.0.0 von UnCo veröffentlicht. Eine Liste mit allen genutzten Python Bibliotheken und anderer Software inklusive Versionen findet sich in Anhang C [15]. Eine ausführliche Dokumentation über das Eingabeformat, Anleitungen und den Klassen von UnCo ist auf deutsch und englisch verfügbar und befindet sich im Projektordner unter *docu/dokumentation_de.html* bzw. *docu/documentation_en.html*.

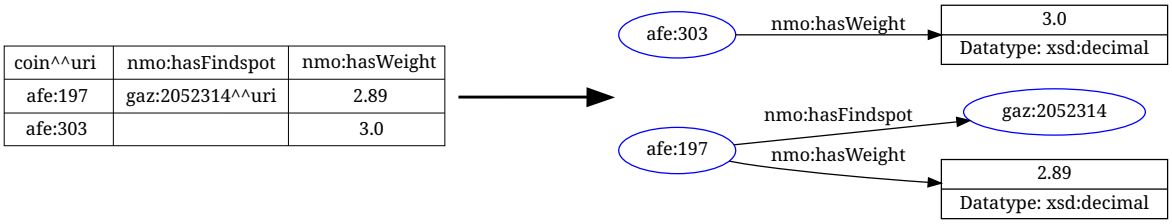


Abbildung 4.15: Beispiel einer RDF-Graph Erzeugung durch UnCo.

Zu den Aufgaben von UnCo gehören unter anderem Funktionen wie Datenaufbereitung, Erzeugung von RDF-Graphen, Hinzufügen von pseudo zufälligen Unsicherheiten und eine Schnittstelle zur Abfrage von Queries. Mit welchen Komponenten UnCo diese Aufgaben erfüllt, wird im Folgenden kurz anhand seiner Klassen beschrieben. Abbildung 4.16 gibt einen ersten Überblick über den Aufbau des Tools.

[15] Die genutzten Python Bibliotheken, waren zum Zeitpunkt vom 28.06.2023 die derzeit aktuellsten Versionen. Eine Liste mit Python Bibliotheken ist auch im UnCo Projekt als *requirements.txt* gespeichert.

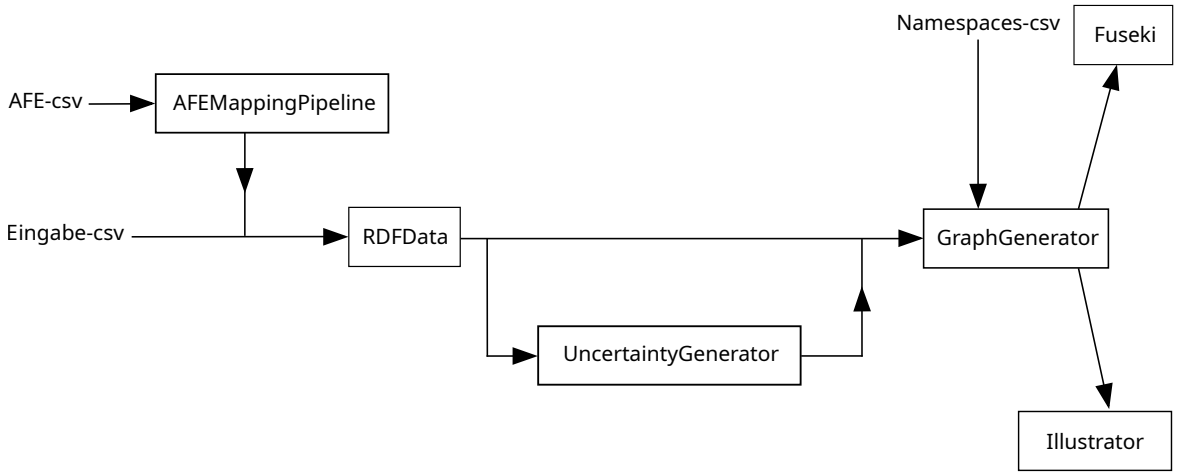


Abbildung 4.16: Komponenten Diagramm des Uncertainty Comparator.

RDFData:

Für die Eingabe von UnCo kamen anfangs mehrere Eingabeformate in Frage. Da das Tool Datensätze einlesen und aus diesen RDF Graphen erzeugt, sollte ein dazu passendes Eingabeformat gewählt werden. Final wurde hierfür das csv-Format ausgewählt, da dieses bereits häufig für tabellarische Daten genutzt wird und somit von den meisten Bibliotheken als Eingabe akzeptiert wird. Damit sich aus csv-Tabellen auch komplexere RDF-Graphen erzeugen lassen, musste zudem ein Regelwerk festgelegt werden, mit der die Daten innerhalb der Tabelle dargestellt werden können. Dadurch lassen sich auch Datentypen, Sprachen, Unsicherheiten oder verkettete Aussagen beschreiben.

Das verwendete Regelwerk für das Datenmapping orientiert sich an Methoden des Turtle-Formats. So lassen sich beispielsweise Sprachen mit dem Marker „@“ und Datentypen mit dem Marker „^^“ angeben. Eine vollständige Beschreibung des Datenmappings befindet sich ebenfalls in der Dokumentation des Projekts.

Die RDFData Klasse erhält als Eingabe eine csv-Datei die dem beschriebenen Regelwerk entspricht und extrahiert alle wichtigen Informationen. Zusätzlich werden hier kleinere Bearbeitungen durchgeführt, wie zum Beispiel die Ersetzung von nicht lesbaren Sonderzeichen (wie griechische Buchstaben) oder auch das korrekte Setzen von nicht angegebenen Datentypen bei Ganzzahlen, Gleitzahlen und Wahrheitswerten. Dadurch sind die Daten für spätere Komponenten aufbereitet und stehen diesen zur Verfügung.

AFE-Mapping Pipeline:

In Abschnitt 4.1.2 wurde bereits ein kleiner Einblick in die AFE Datenbank geben. Damit diese jedoch als Eingabe für die RDFData Klasse geeignet ist, müssen zuvor einige Umformungen getätigt werden. Da AFE als relationale Datenbank vorliegt, wurde zunächst mit Hilfe einer SQL Abfrage eine CSV-Tabelle aller wichtigen Einträge extrahiert. Dazu gehören vor allem alle Aussagen, die sich mit der Nomisma Ontologie beschreiben lassen, sowie die dafür eingetragenen Unsicherheiten. Diese Tabelle dient als Eingabe für die AFE-Mapping Pipeline, welche weitere notwendige Umformungsschritte tätigt, um die Tabelle an das Eingabeformat für RDFData anzupassen. Darunter zählen beispielsweise die Angabe der URIs, die Zusammenfassung von Mehrfacheinträgen und die korrekte Angabe der Unsicherheiten. Die Ausgabe umfasst mehrere csv-Tabellen, welche dem oben genannten Eingabeformat entsprechen und somit als Eingabe für RDFData genutzt werden können.

UncertaintyGenerator:

Die UncertaintyGenerator Klasse hat zur Aufgabe, einem RDFData-Objekt Unsicherheiten und/oder alternative Werte hinzuzufügen. Zur Verfügung stehen zum einen das Umwandeln von sicheren Aussagen in unsichere Aussagen durch pseudo zufälliges Auswählen und zum anderen das Hinzufügen von Alternativen zu unsicheren Aussagen. Durch diese Komponente lässt sich später der Einfluss der beiden Eigenschaften ermitteln.

GraphGenerator:

In der GraphGenerator Klasse werden RDF-Graphen erzeugt. Als Eingabe erhält der GraphGenerator ein Objekt der RDFData Klasse und erzeugt den dort beschriebenen RDF-Graphen. Wichtige Parameter sind die ID einer Modellierung[16] aus Abschnitt 4.1.1 und das Ausgabeformat, welches entweder auf XML oder Turtle gesetzt werden kann. Wurden in der csv-Tabelle Präfixe genutzt um Namensräume abzukürzen, kann zudem eine eigene csv-Tabelle mit allen Präfixen und den dazugehörigen

[16]Modellierung 9a und 9b besitzen im Programm die ID 9 und 10.

Namensräumen eingegeben werden[17].

Fuseki:

Für die Ausführung der SPARQL-Anfragen wird ein lokaler *Apache Jena Fuseki* (kurz *Fuseki*) Server der Version 4.8.0 genutzt. Fuseki ist eine häufig verwendete Anfrageschnittstelle, wodurch die Ergebnisse praxisnaher eingeordnet werden können. Da Fuseki auf der Programmiersprache Java basiert, ist eine einfache Einbindung in UnCo nicht möglich. Daher muss für die Nutzung ein eigenständiger Fuseki Server heruntergeladen und der Pfad dieses Servers der Python-Klasse übergeben werden. Der Fuseki Server wurde durch Java der Version 11.0.19 ausgeführt.

RDFier und Illustrator:

Der *RDFier* ist zwar keine Komponente von UnCo, jedoch eine daraus entstandene Webanwendung, welche zusätzlich im Rahmen dieser Arbeit entwickelt wurde. Der RDFier soll die Erzeugung und Darstellung von RDF-Graphen erleichtern und erlaubt dabei die Auswahl der gewünschten Modellierung für Unsicherheiten. Für die Darstellung der erzeugten RDF-Graphen wird die Illustrator Klasse genutzt. Diese kommuniziert mit Hilfe des HTTP-Protokolls mit der Webanwendung *RDF Grapher*[18], die Visualisierungen der zugesendeten RDF-Graphen erzeugt.

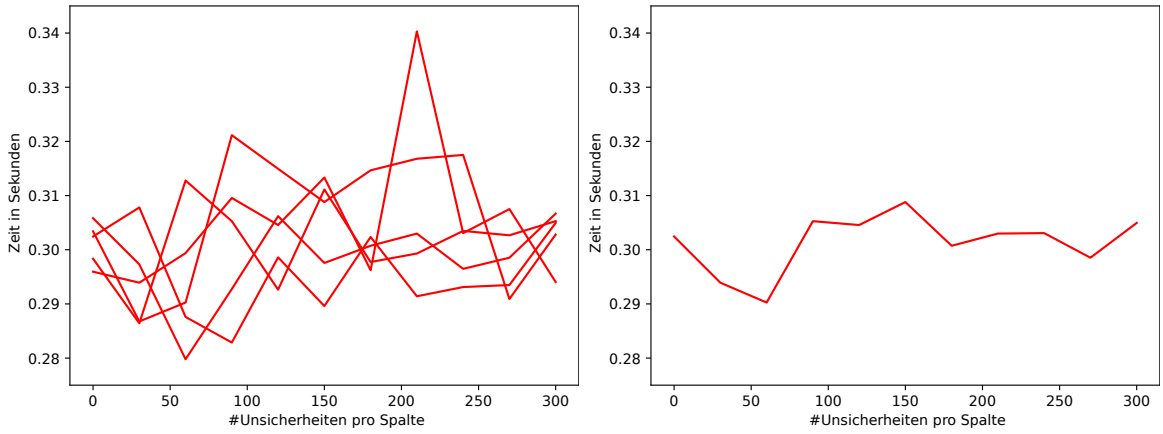
4.1.7 Messverfahren und Metrik

In Abschnitt 4.1.3 wurden bereits die Laufzeit der Queries als Leistungskennzahlen festgehalten. Ein weiterer wichtiger Aspekt umfasst das Messverfahren dieser Leistungskennzahlen, da zu erwarten ist, dass Laufzeitmessungen nicht komplett störungsfrei ablaufen. Diese Vermutung bestätigte sich auch bei der Durchführung des Benchmarks, weshalb verschiedene Verfahren ausgetestet wurden, um robuste Ergebnisse zu erhalten. Das final genutzte Messverfahren wird in diesem Abschnitt vorgestellt.

[17] Eine solche Tabelle ist im Projektordner unter *data/input/namespaces.csv* zu finden. Standardmäßig werden Präfixe von häufig verwendeten Namensräumen erkannt. Eine Liste welche Namensräume verfügbar sind ist in der Dokumentation unter „Eingabeformat“ zu finden.

[18] Der RDF-Grapher ist erreichbar unter: `https://www.ldf.fi/service/rdf-grapher`

Um einen ersten Eindruck über das Störungsverhalten der Laufzeitmessungen zu erhalten, wurden zunächst beispielhaft die erste Modellierung und die vierte Query ausgewählt. Der AFE Datensatz wurde dann mit der ersten Modellierung in einen RDF-Graph überführt, auf den lokalen Fuseki Server geladen und mit Hilfe dessen die vierte Query abgefragt. Dieser Versuchsaufbau wurde daraufhin schrittweise mit mehr Unsicherheiten ausgeführt und insgesamt fünf-mal wiederholt, um zu überprüfen, ob in allen fünf Fällen die gleichen Ergebnisse erzeugt werden. Die Laufzeitmessungen sind in der Grafik (a) von Abbildung 4.17 dargestellt, bei dem eine Linie jeweils eine Durchführung des eben beschriebenen Versuchs darstellt. Es zeigt sich ein deutliches Rauschen in den Messergebnissen mit großen Ausreißern, obwohl diese die gleichen Versuche ausführen. Um solche Schwankungen im Benchmark zu vermeiden, werden alle Durchführungen fünf-mal wiederholt und aus diesen nur die Medianwerte genutzt, da der Median vor allem zum Filtern von Ausreißern geeignet ist. Grafik (b) von Abbildung 4.17 zeigt den aus (a) entstandenen Median der fünf Durchführungen.



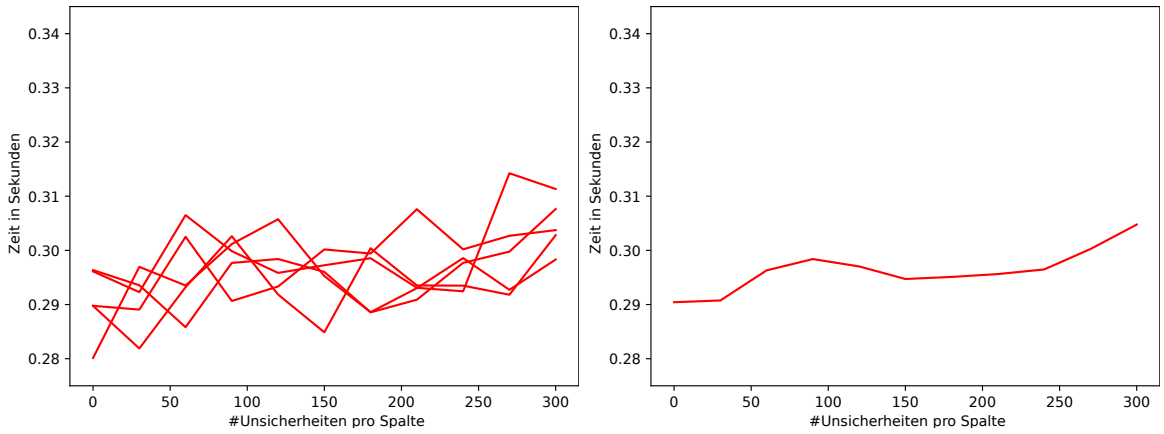
(a) Fünf gleiche Durchführungen

(b) Median der fünf Durchführungen

Abbildung 4.17: Laufzeitverhalten von Fuseki bei der Abfrage von Query 4. Es wurde der AFE-Datensatz als Grundlage genutzt, schrittweise 30 Aussagen zusätzlich als unsicher markiert und jeweils mit Modellierung 1 der RDF-Graph gebaut. Grafik (a) zeigt das Laufzeitverhalten von fünf solcher Durchführungen und Grafik (b) den daraus errechneten Median.

Der Median zeigt sich deutlich störungsfreier, als die einfach durchgeführten Versuche. Trotzdem zeigten die Benchmarktests mit diesem Messverfahren weiterhin ein größeres Rauschen in den Ergebnissen. Ein Vergleich darüber zeigt Grafik (a) von Ab-

bildung 4.18, in der fünf der Mediane mit den gleichen Durchführungen abgebildet wurden. Es zeigt sich, dass die Mediane weniger Ausreißer besitzen, als der vorherige Vergleich. Um das enthaltene Rauschen weiter zu verringern wird aus den fünf Medianen der Mittelwert berechnet, welcher in Grafik (b) der gleichen Abbildung zu sehen ist. Der Mittelwert eignet sich hier besser als der Median, da weniger starke Ausreißer in den Ergebnissen enthalten sind und der Mittelwert den Einfluss aller Mediane enthält.



(a) Fünf Mediane

(b) Mittelwert der fünf Mediane

Abbildung 4.18: Gleicher Versuchsaufbau wie in Abbildung 4.17, jedoch zeigt Grafik (a) fünf simultan errechnete Mediane und Grafik (b) den aus diesen entstandenen Mittelwert.

Für den Benchmark wurde somit der sogenannte *Mean-of-Medians* genutzt, welcher sich auch in anderen Arbeiten als robustes Verfahren zeigte [Zho+21]. Dadurch dass für jeden Datenpunkt 25 Ausführungen getätigt werden müssen, zeigt sich jedoch ein hoher Bedarf des Arbeitsspeichers. Zu diesem Zweck wurde das Programm weiter optimiert, sodass ungenutzte Variablen und Datensätze mit einem hohen Arbeitsspeicherbedarf gelöscht, oder in eine optimierte Datenstruktur überführt werden. Trotz der implementierten Optimierungen, ließen sich manche Tests aus Abschnitt 4.2 nur in getrennt ausgeführten Teilen durchführen. Für eine Wiederholung des Benchmarks wird ein Arbeitsspeicher von 32 GB empfohlen, bei denen die Anzahl errechneter Mediane und Mittelwerte evt. auch erhöht werden können. Eine Anleitung wie die in diesem Kapitel gezeigten Abbildungen erzeugt werden können, befindet sich in der Dokumentation von UnCo. Dort wird zudem beschrieben, wie die später beschriebenen

Benchmarktests ausgeführt werden können und in welchen Teilen die Benchmarktests durchgeführt wurden.

Neben einem zuverlässigen Verfahren zur Messung der Leistungskennzahlen ist auch die Interpretation dieser ein wichtiger Aspekt. Zu diesem Benchmark bietet sich die Ergebnissicherung in tabellarischer Form an, um für jede Modellierung anzugeben, welche Laufzeit die dazugehörige Query besaß. Dadurch ist eine Interpretation der Ergebnisse jedoch erschwert, da eine Laufzeit t erst mit den anderen Laufzeiten der Query verglichen werden muss, um so eine Aussage über die Güte des Wertes herleiten zu können. Daher sollten neben den Laufzeiten auch Werte angegeben werden, die den relativen Vergleich zu den anderen Modellierungen bereits beinhalten. Dafür bieten sich unterschiedliche Metriken an, wie bspw. die prozentuale Angabe der Güte (die beste Modellierung hat 100% Güte, die schlechteste 0% und die restlichen erhalten Werte relativ zu ihren Laufzeiten). Die prozentuale Angabe der Güte hat zum Nachteil, dass diese nur im Vergleich zur besten und schlechtesten Modellierung angegeben werden und der relative Vergleich zu den anderen Modellierungen fehlt.

Daher wird für die Ergebnisse des Benchmarks Metriken in Form von Ranglisten genutzt. Diese werden nach dem gleichen Schema erstellt wie in den Olympischen Spielen, bei denen der beste Sportler den Rang 1, der zweitbeste den Rang 2 usw. erhält. Bei zwei gleichen Ergebnissen, erhalten beide den besseren Rang, sodass es bspw. zwei Sportler mit Rang 1 und der nächst bessere mit Rang 3 geben kann. Für den Benchmark wird das gleiche Verfahren für die Ergebnisse jeder Query angewandt, sodass jede Query eine eigene Rangliste erhält. In Tabelle 4.6 ist eine solche Rangliste für Query 1 gezeigt. Während bei Ergebnissen von Sportlern nur bei genauer Gleichheit auch ein gleicher Rang vergeben wird, sollten „fast gleiche“ Ergebnissen des Benchmarks trotzdem einen gleichen Rang erhalten, da Störungen in den Ergebnissen nicht auszuschließen sind. Daher wird ein Toleranzbereich von 5% des Laufzeitbereichs angewandt, sodass Laufzeitergebnisse die sich in einem 5% Bereich befinden zu einem gleichen Rang führen[19].

Die hier genutzten Ranglisten sollen einen Überblick über die Güte der Modellie-

[19]Eine Anleitung, wie der Toleranzbereich in UnCo festgelegt werden kann befindet sich in der Dokumentation unter „Anleitungen“.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
Laufzeiten:	0.68	0.31	1.43	0.88	1.32	0.59	0.31	0.23	0.16	0.17
Rangliste:	7	4	10	8	9	6	4	3	1	1

Tabelle 4.6: Beispiel zur Erstellung einer Rangliste. Es werden die Laufzeiten in Sekunden von Query 4 des ersten Benchmarktests gezeigt.

rungen geben, ohne dabei auf konkrete Anwendungsfälle zu achten. Daher ist es dem Leser überlassen, die Laufzeitergebnisse mit einer zu ihrem Anwendungsfall passenden Metrik weiter umzuformen.

4.2 Ergebnisse und Auswertung

In diesem Kapitel werden nun die Ergebnisse der einzelnen Benchmarktests gezeigt. Für einen Überblick über die verglichenen Modellierungen sind in Tabelle 4.7 wichtige Eigenschaften zusammengefasst.

	#Kanten pro Unsicherheit	**Gewichtete Unsicherheit?**	**Namensräume**
Modellierung 1	6	Nein	*bmo, crm, nm, rdf*
Modellierung 2	3	Nein	*nm, rdf, un*
Modellierung 3	9	Ja	*crm, rdf*
Modellierung 4	7 (-2)	Ja	*crminf, rdf*
Modellierung 5	3	Ja	*amt, crm*
Modellierung 6	5	Nein	*edtfo, rdf*
Modellierung 7	3	Nein	*edtfo, rdf*
Modellierung 8	2 (-1)	Nein	*un*
Modellierung 9a	2	Nein	*edtfo, rdf*
Modellierung 9b	2	Ja	*un*

Tabelle 4.7: Übersicht der verglichenen Modellierungen.

4.2.1 Vergleich bezüglich AFE

Für den ersten Benchmarktest wurde zunächst der AFE Datensatz mit jeder Modellierung in den jeweiligen RDF-Graphen überführt. Diese wurden nacheinander auf einem lokalen Fuseki Server geladen, welcher anschließend die SPARQL-Anfragen aus Abschnitt 4.1.3 ausführte. Um störungsfreie Ergebnisse zu erhalten, wurden sowohl hier als auch in den anderen Benchmarktests der Mean-of-Medians als Messverfahren genutzt. Die daraus entstandenen Laufzeiten sind in Tabelle 4.8 zu sehen. Im Projektordner sind die ungekürzten Werte in *data/results/afe.txt* hinterlegt.

Query	**Modellierung**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	0.076	0.077	0.079	0.085	0.076	0.075	0.069	0.066	0.065	0.065
2	0.020	0.020	0.023	0.026	0.042	0.019	0.020	0.019	0.016	0.016
3	0.067	0.064	0.068	0.068	0.067	0.061	0.069	0.060	0.057	0.059
4	0.300	0.287	0.318	0.295	0.305	0.300	0.286	0.300	0.282	0.288
5	0.013	0.016	0.021	0.020	0.022	0.017	0.015	0.015	0.012	0.015
6	X	X	0.380	0.357	0.378	X	X	X	X	0.346

Tabelle 4.8: Laufzeiten in Sekunden für jedes Modellierung-Query-Paar. Jeder Wert entspricht dem Mean-of-Medians von fünf Medianen, die wiederum aus 5 Ausführungen errechnet wurden.

Die Ergebnisse zeigen erste Eindrücke, welche Modellierungen sich als effektiv für AFE zeigen. Auch wenn die Laufzeiten sich oft nur um Bruchteile einer Sekunde unterscheiden, können diese Unterschiede in anderen Umgebungen von größerer Relevanz sein, wenn diese z.B. bei Servern mit mehreren hundert Zugriffen gleichzeitig eingesetzt werden. Um diese und die späteren Ergebnisse besser interpretieren zu können, werden, wie schon in Abschnitt 4.1.7 beschrieben, jeweils für jede Query eine Rangliste erstellt. Das Resultat lässt sich in Tabelle 4.9 sehen.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	5	7	9	10	7	5	4	1	1	1
2	3	3	8	9	10	3	3	3	1	1
3	6	5	9	7	7	4	10	3	1	2
4	6	2	10	5	9	6	2	6	1	2
5	2	6	9	8	10	7	3	3	1	3
6	X	X	4	2	3	X	X	X	X	1

Tabelle 4.9: Ranglisten der Queries über allen verglichenen Modellierungen.

Auf den ersten Blick zeigt sich, dass die RDF-star Modellierung 9a stets den ersten Rang (mit-)besetzt, mit Ausnahme der letzten Query, da diese nur für gewichtete Unsicherheiten ausführbar ist. Auch die Modellierung 9b belegt in allen Queries einen der besten drei Rängen. Ein Grund für diese Vorteile könnte hier zum einen eine geringe Anzahl Kanten pro unsicherer Aussage sein, zum anderen auch die Komplexität der SPARQL-Anfrage, welche bei beiden Modellierungen verhältnismäßig gering ausfällt.

Neben diesen effektiven finden sich auch unvorteilhafte Modellierungen in den Ranglisten. So ist beispielsweise Modellierung 3 in allen Queries unter den drei schlechtesten Modellierungen, was sich ebenfalls durch die Anzahl der Tripel und die Komplexität der SPARQL-Anfrage erklären lassen kann, da hier nach Teilgraphen bestehend aus 10 Kanten gesucht werden muss. Auch die Modellierungen 4 und 5 erhalten in den meisten Fällen einen vergleichsweise schlechten Rang. Um eine endgültige Rangliste für den ersten Benchmarktest zu erhalten, werden in Tabelle 4.10 für jede Modellierung der durchschnittliche Rang festgehalten. Dabei haben Modellierungen mit gewichteten Unsicherheiten den Vorteil, dass die sechste Query nur Ränge von 1 bis 4 besitzen, und somit der schlechteste Rang höchstens 4 sein kann. Da diese durch gewichtete Unsicherheiten jedoch vielseitiger sind, fließt diese positive Eigenschaft auch in den Rang ein.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
Mittelwerte	4.4	4.8	8.2	6.8	7.7	5	4.4	3.2	1	1.7

Tabelle 4.10: Mittelwerte aller Ränge pro Modellierung aus Tabelle 4.9.

Durch den errechneten durchschnittlichen Rang jeder Modellierung, bestätigt sich das positive Verhalten von Modellierung 9a und 9b, sowie das schlechtere Verhalten von Modellierung 3, 4 und 5. Für die Angabe von gewichteten Unsicherheiten ist für die AFE Datenbank somit die Modellierung 9b zu empfehlen, da diese einen deutlich besseren Rang besitzt, als die alternativen Modellierungen 3, 4 und 5. Für Angaben ohne Gewicht lässt sich ebenfalls eine Empfehlung für AFE aussprechen, da Modellierung 9a den bestmöglichen Rang erhalten hat. Modellierung 8 ist mit Rang 3.2 hingegen zu empfehlen, wenn auf RDF-star verzichtet werden möchte. Die restlichen Modellierungen lassen sich mit einem Rangunterschied von höchstens 0.6 nicht eindeutig unterscheiden.

Der gleiche Benchmarktest wurde ebenso mit der Python Bibliothek *rdflib* als Anfrageschnittstelle durchgeführt[20]. Da *rdflib* bisher jedoch keine Unterstützung der Erweiterung RDF-star anbietet, konnten die Modellierungen 9a und 9b nicht mit einbezogen werden. Trotzdem zeigen die Ergebnisse eine ähnliche Reihenfolge der ersten acht Modellierungen wie schon bei Fuseki. Bei dieser erhielt jedoch Modellierung 8 und 6 die beiden besten, und 3 und 4 die beiden schlechtesten Ränge. Da die Laufzeiten von *rdflib* im Vergleich zu Fuseki deutlich höher sind (teilweise über 100 Sekunden für die Ausführung einer SPARQL-Anfrage) und die Unterstützung der Modellierungen 9a und 9b fehlt, wird im weiteren Verlauf auf die Ausführung mit *rdflib* verzichtet. Fuseki ist eine häufig verwendete Anfrageschnittstelle, weshalb diese Ergebnisse ohne hin einen größeren Bezug zur praktischen Anwendung besitzen.

[20]Die Ergebnisse des Benchmarktests mit *rdflib* als Anfrageschnittstelle sind ebenso im Projekt unter *data/results/afe.txt* zu finden.

4.2.2 Vergleich bei steigender Anzahl Unsicherheiten

Bei einem allgemein anerkannten Standard für Unsicherheiten in (RDF-) Daten, werden diese nicht nur in mehr Datensätzen enthalten sein, auch die Anzahl der unsicheren Aussagen wird in RDF-Graphen höher sein, da bereits von Anfang an bei der Erstellung der Datenbank auf die Erfassung von Unsicherheiten geachtet wird. Daher ist es sinnvoll zu überprüfen, wie sich das Laufzeitverhalten der Modellierungen bei steigender Anzahl unsicherer Aussagen verhält.

Für diesen Benchmarktest wird der AFE Datensatz ohne die dort eingetragenen Unsicherheiten als Grundlage genutzt. Schrittweise wird nun die im vorherigen Abschnitt ausgeführte Analyse auf dem Datensatz ausgeführt, wobei in jedem Schritt 1000 pseudozufällig ausgewählte sichere Aussagen pro Spalte als unsicher markiert werden. Dabei erhält nicht jede Spalte zufällige Unsicherheiten, sondern nur diese, für die bereits Unsicherheiten eingetragen wurden. Im Fall von AFE trifft das auf neun Spalten zu, von denen die vier in Tabelle 4.2 über 8000 Einträge enthalten und die restlichen unter 500. Wie sich die Laufzeit von Query 4 bei steigender Anzahl unsicherer Werte verhält, zeigt Abbildung 4.19. Die Abbildungen der anderen Queries sind in Anhang D zu sehen. Zudem sind die ungekürzten Ergebnisse im Projektordner unter *data/results/increasing_uncertainties.txt* hinterlegt.

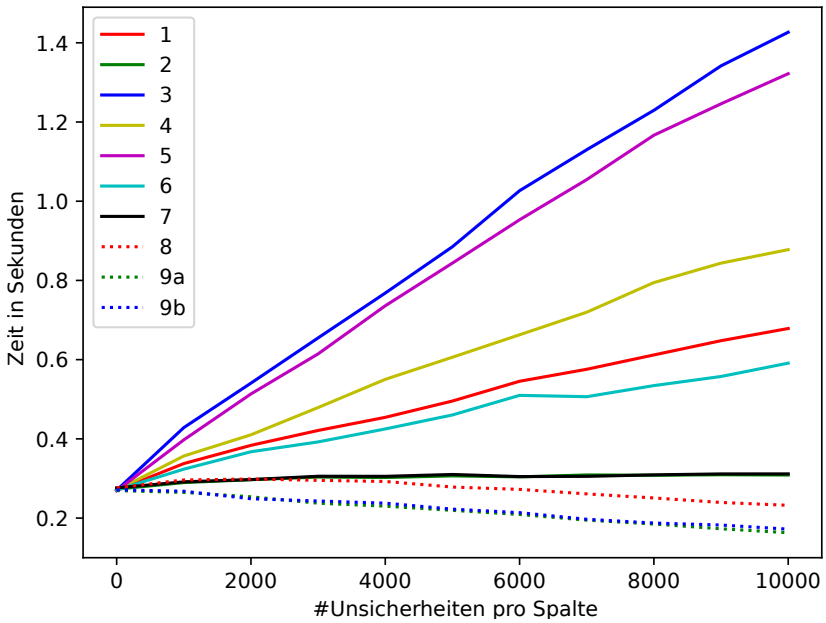


Abbildung 4.19: Laufzeitverhalten von Fuseki bei Ausführung der Query 4. Es wurden schrittweise bis zu 1000 sichere Aussagen pro Spalte durch unsichere ausgetauscht.

Es zeigt sich ein nahezu linearer Verlauf der einzelnen Modellierungen. Im Bereich von 0 bis 1000 besitzen einige Kurven die größte Steigung, was sich damit erklären lässt, dass dort noch alle neun Spalten mehr Unsicherheiten erhielten. Danach besitzen bereits nur noch fünf von diesen unsichere Werte. In der Grafik lassen sich verschiedene Modellierungen in gleiche Gruppen aufteilen. So zeigt sich beispielsweise, dass die beiden RDF-star Modellierungen 9a und 9b, sowie die Modellierungen 2 und 7 zu jedem Zeitpunkt eine fast identische Laufzeit besitzen. Das hat zur Folge, dass diese auch in einer späteren Rangliste jeweils den gleichen Rang erhalten.

Auffällig ist zudem, dass manche Modellierungen mit steigender Anzahl Unsicherheiten auch ein steigendes Laufzeitverhalten zeigen, während andere eine immer kürzere Laufzeit aufweisen. Das sinkende Laufzeitverhalten lässt sich durch die Ausgabegröße erklären, da durch Query 4 alle sicheren Aussagen ausgegeben werden und somit die Ausgabegröße schrittweise kleiner wird. Somit besteht vermutlich ein Großteil der Laufzeiten von Modellierung 8, 9a und 9b aus der Zeit, die Fuseki für die Ausgabe des Ergebnisses braucht. Die Query-Laufzeiten der anderen Modellierungen scheinen hingegen von der Durchsuchung des Graphen dominiert zu sein.

Für die anderen Queries sehen diese Verläufe ähnlich aus. Auffällig ist hierbei jedoch, dass die ersten drei Queries deutliche Ausreißer in den Ergebnissen aufweisen, weshalb in Abbildung 4.20 ein genauerer Blick auf die Ergebnisse der ersten Query geworfen wird.

Die Laufzeitergebnisse der ersten Query sind nicht so deutlich wie zuvor bei der vierten, da viele Modellierungen ein ähnliches Laufzeitverhalten haben und diese ein stärkeres Rauschen besitzen. Das lässt sich mit der geringeren Größe der Ausgabe[21] erklären. Dadurch sind auch die Laufzeiten deutlich kleiner, als in den Queries 4 bis 6, sodass dort auch kleinere Abweichungen für große Unterschiede sorgen. Als Gesamtergebnis wird zunächst in Tabelle 4.11 die Laufzeiten bei 10.000 Unsicherheiten pro Spalte festgehalten. Der Datensatz zu diesem Zeitpunkt wird im Weiteren auch *afe-10kU* genannt.

[21] Query 1 gibt anfangs eine Ausgabe mit 10.450 Zeilen und am Ende eine Ausgabe mit 343 Zeilen aus. Query 4 gibt anfangs eine Ausgabe mit 59.888 Zeilen und am Ende eine Ausgabe mit 19.786 Zeilen aus.

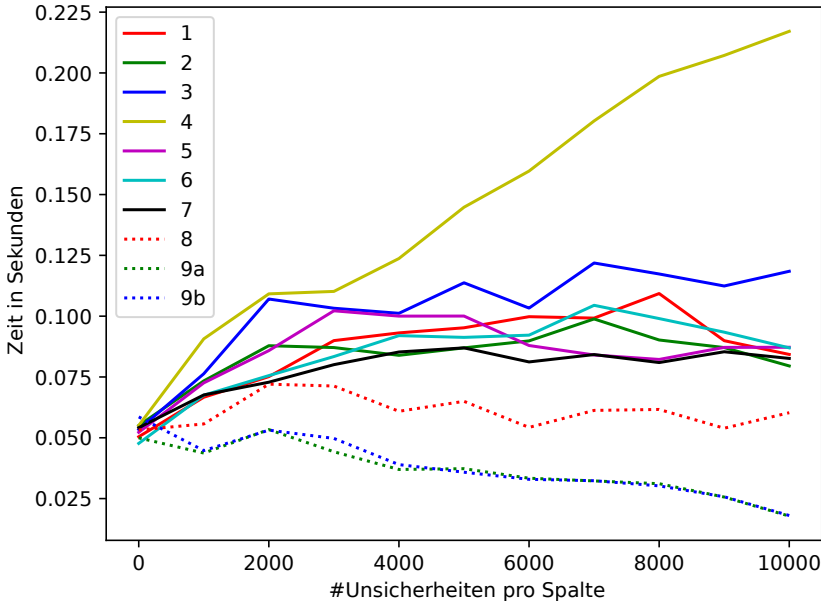


Abbildung 4.20: Laufzeitverhalten von Fuseki bei Ausführung der Query 1. Es wurden schrittweise bis zu 1000 sichere Aussagen pro Spalte durch unsichere ausgetauscht.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	0.084	0.080	0.118	0.217	0.087	0.087	0.083	0.060	0.018	0.018
2	0.091	0.102	0.127	0.260	0.058	0.077	0.101	0.062	0.049	0.050
3	0.148	0.174	0.230	0.474	0.125	0.125	0.170	0.086	0.056	0.063
4	0.679	0.309	1.427	0.878	1.322	0.591	0.312	0.232	0.163	0.172
5	0.360	0.283	0.493	0.449	0.724	0.329	0.277	0.200	0.160	0.161
6	X	X	1.918	1.118	2.730	X	X	X	X	0.309

Tabelle 4.11: Ergebnisse der Laufzeiten in Sekunden für jedes Modellierung-Query-Paar. RDF-Graphen wurden aus dem Datensatz *afe-10kU* erzeugt.

Die Laufzeiten dieses Benchmarktests sind deutlich größer als noch in Abschnitt 4.2.1. Zu diesen Laufzeiten lassen sich nun, ähnlich wie im vorherigen Benchmarktest, Ranglisten erstellen, um so die Werte besser interpretieren zu können. Aus diesen Laufzeiten ergeben sich die in Tabelle 4.12 enthaltenen Ränge für jede Query.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	4	4	9	10	4	4	4	3	1	1
2	6	8	9	10	1	5	6	4	1	1
3	6	7	9	10	4	4	7	3	1	1
4	7	4	10	8	9	6	4	3	1	1
5	7	4	9	8	10	6	4	3	1	1
6	X	X	3	2	4	X	X	X	X	1

Tabelle 4.12: Ranglisten der Queries über allen verglichenen Modellierungen, ausgehend der Laufzeiten von *afe-10kU*.

In den hier erzeugten Ranglisten zeigen sich ähnliche Vor- bzw. Nachteile wie schon zuvor in Abschnitt 4.2.1. Besonders auffällig sind die Modellierungen 9a und 9b, welche ausnahmslos den ersten Rang belegen, während der letzte Rang unter den Modellierungen 3, 4 und 5 vergeben wurde. Die fünfte Modellierung zeigt jedoch unter den ersten drei Queries deutlich bessere Laufzeiten als in den anderen. Das liegt vermutlich an der Komplexität der SPARQL-Abfragen, da diese in den letzten drei Queries nach 14 verschiedenen Eigenschaften überprüfen, während in den ersten drei die Eigenschaft für unsichere Prägeorte bekannt ist. Daher kann die fünfte Modellierungen auch für RDF-Graphen eine Empfehlung sein, bei denen hauptsächlich SPARQL-Anfragen mit bekannten Eigenschaften genutzt werden. Um auch hier eine finale Rangliste zu erhalten, werden in Tabelle 4.13 die durchschnittlichen Ränge jeder Modellierungen festgehalten.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
Mittelwerte	6	5.4	8.2	8	5.3	5	5	3.2	1	1

Tabelle 4.13: Mittelwerte aller Ränge pro Modellierung aus Tabelle 4.12, ausgehend der Laufzeiten von *afe-10kU*.

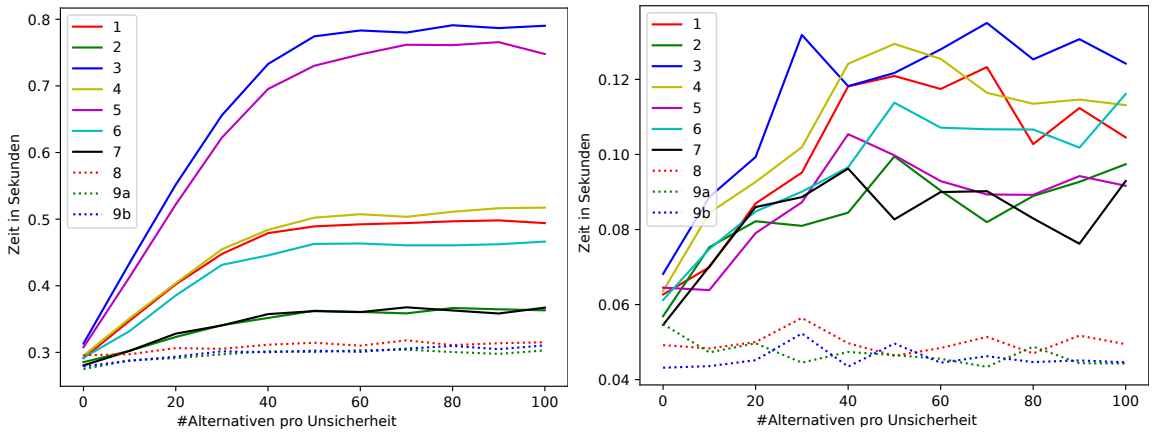
Wie schon in den durchschnittlichen Rängen aus Abschnitt 4.2.1, haben auch hier die RDF-star Modellierungen die beiden höchsten Ränge. Unter denen ohne RDF-

star besitzt, wie schon zuvor, die achte Modellierung den besten Rang. Jedoch zeigen sich auch Unterschiede zwischen den beiden finalen Ranglisten. Es haben sich bei den Modellierungen mit gewichteten Unsicherheiten die Ränge so verändert, dass nun die vierte einen deutlich höheren Rang erhalten hat. Die fünfte Modellierung hingegen, profitiert von dem positiven Laufzeitverhalten der ersten drei Queries und hat damit sogar einen besseren Rang als Modellierung 1 und 2 erhalten.

4.2.3 Vergleich bei steigender Anzahl Alternativen

Während bei den meisten Modellierungen Alternativen nach dem gleichen Schema hinzugefügt werden wie andere Unsicherheiten, gibt es auch solche, die dafür weniger Kanten gebrauchen. So beispielsweise die achte Modellierung, bei der für jede zusätzliche Alternative nur eine Kante zum RDF-Graph hinzugefügt wird. Neben der achten, benötigt auch die vierte Modellierung weniger Kanten, um Alternativen hinzuzufügen.

Um zu Überprüfen, ob Modellierungen bei vielen Alternativen effizientere SPARQL-Anfragen besitzen, wird in diesem Abschnitt der AFE Datensatz mit Unsicherheiten als Grundlage genutzt und schrittweise die Anzahl an Alternativen erhöht. Damit dem AFE Datensatz keine neuen Ressourcen hinzugefügt werden, wird für jede Alternative ein anderer Wert aus der gleichen Spalte hinzugefügt, sodass bspw. für alternative Prägeorte auch tatsächliche Ressourcen für Prägeorte genutzt werden. Unter den Spalten mit Unsicherheiten hat die Spalte der Eigenschaft *nmo:hasWeight* mit 1408 Ressourcen die meisten verschiedenen Einträge, alle anderen besitzen weniger als 50 verschiedene. Je nachdem wie viele verschiedene Einträge die Spalte hat, werden so jeder Unsicherheit schrittweise bis zu zehn Alternativen hinzugefügt. Beispielhaft wird in Abbildung 4.21 das Laufzeitverhalten anhand von Query 4 und 1 gezeigt. Die Abbildungen der anderen Queries sind in Anhang E zu sehen. Zudem sind die ungekürzten Ergebnisse im Projektordner unter *data/results/increasing_alternatives.txt* hinterlegt.



(a) Laufzeitverhalten von Query 4

(b) Laufzeitverhalten von Query 1

Abbildung 4.21: Laufzeitverhalten von Fuseki bei Ausführung der Query 4 und 1. Es wurden schrittweise bis zu 10 Alternativen pro Unsicherheit hinzugefügt.

Auffällig sind auch in diesem Benchmark die unterschiedliche Störungsanfälligkeiten der beiden Queries, was sich ebenfalls mit einer geringen Ausgabe erklären lässt. Zudem zeigt sich vor allem in Query 4, dass die Kurven der Modellierungen ab 50 Alternativen nur eine geringe Steigung besitzen, was sich mit der Tatsache erklären lässt, dass ab diesem Punkt nur die Aussagen mit *nmo:hasWeight* weitere Alternativen erhalten.

Die beiden Modellierungen, welche durch Alternativen profitieren sollten, sind Modellierung 4 und 8. Obwohl die achte Modellierung jeweils nur eine Kante pro Alternative dem Graph hinzufügt, reicht dieser Unterschied nicht aus, um eine geringere Laufzeit als die RDF-star Modellierungen zu bekommen. Auch die vierte Modellierung zeigt sich im Vergleich zum vorherigen Benchmarktest effizienter, besitzt aber auch hier eine höhere Laufzeit als Modellierung 1. In den anderen Queries ist ein ähnliches Verhalten zu sehen. Die Laufzeitergebnisse des letzten Schritts (bis zu 100 pro Spalte) sind in Tabelle 4.14 zu sehen. Der dort genutzte Datensatz wird im Weiteren auch *afe-100A* genannt.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	0.105	0.097	0.124	0.113	0.092	0.116	0.093	0.049	0.044	0.045
2	0.276	0.058	0.084	0.109	0.048	0.261	0.057	0.035	0.035	0.038
3	0.335	0.123	0.174	0.212	0.113	0.317	0.123	0.073	0.068	0.069
4	0.494	0.363	0.790	0.517	0.748	0.466	0.367	0.315	0.303	0.311
5	0.140	0.110	0.194	0.158	0.282	0.128	0.112	0.069	0.067	0.071
6	X	X	0.976	0.573	1.191	X	X	X	X	0.442

Tabelle 4.14: Ergebnisse der Laufzeiten in Sekunden für jedes Modellierung-Query-Paar. RDF-Graphen wurden aus dem Datensatz *afe-100A* erzeugt.

Auch hier wird für eine bessere Interpretierbarkeit der Ergebnisse zunächst die Ranglisten erstellt. Diese werden in Tabelle 4.15 festgehalten.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	7	6	10	8	4	8	4	3	1	1
2	10	4	7	8	4	9	4	1	1	1
3	10	4	7	8	4	9	4	1	1	1
4	7	4	10	7	9	6	4	1	1	1
5	7	4	9	8	10	6	4	1	1	1
6	X	X	3	2	4	X	X	X	X	1

Tabelle 4.15: Ranglisten der Queries über allen verglichenen Modellierungen, ausgehend von Tabelle 4.14.

Im Vergleich zu Tabelle 4.12 scheinen die Modellierungen 4 und 8 tatsächlich von mehr Alternativen anstelle von anderen Unsicherheiten zu profitieren. Während Modellierung 4 bei *afe-10kU* in drei verschiedenen Queries den letzten Rang belegt hat, haben sich diese in *afe-100A* auf 8 verbessert. Die achte Modellierung hat ebenfalls deutlich bessere Ränge, da diese in den meisten Queries mit den RDF-star Modellierungen gleichgesetzt wird. Die durchschnittlichen Ränge der Modellierungen werden

in Tabelle 4.16 festgehalten.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
Mittelwerte	8.2	4.4	7.7	6.8	5.8	7.6	4	1.4	1	1

Tabelle 4.16: Mittelwerte aller Ränge pro Modellierung aus Tabelle 4.15, ausgehend der Laufzeiten von *afe-100A*.

Neben den positiven Einflüssen auf Modellierung 4 und 8, zeigt sich durch die durchschnittlichen Ränge auch eine, welche einen deutlich schlechteren Rang erhalten hat. Die erste Modellierung zeigt sich in diesem Benchmarktest als schlechteste Wahl mit einem durchschnittlichen Rang von 8.2. Somit sollte diese Modellierung nicht für RDF-Graphen mit vielen Alternativen pro Unsicherheit gewählt werden.

4.2.4 Vergleich bei künstlich erzeugten Datensätze

Verschiedene Datensätze besitzen in der Regel auch verschiedene Strukturen der RDF-Graphen. Vor allem die Dichte des Graphen, also die durchschnittliche Anzahl an Kanten pro Knoten, kann zwischen verschiedenen Datensätzen sehr verschieden sein. Während der AFE Datensatz einen relativ lichten RDF-Graph erzeugt, würden Datensätze von z.B. Patienten eines Krankenhauses vermutlich einen dichteren RDF-Graph erzeugen, da diese oftmals nahezu vollständig ausgefüllt sind. Daher werden die Modellierungen im letzten Benchmarktest auch auf vollständigen Datensätzen verglichen.

Der Benchmarktest wird nach dem gleichen Verfahren ausgeführt, wie schon zuvor in Abschnitt 4.2.1. Anstelle des AFE Datensatzes, wird hier ein vollständig künstlich erzeugter Datensatz genutzt. Dazu wurden die Spalten des AFE Datensatzes übernommen und alle Werte (außer die ID der Münzen) durch zufällige Einträge der Spalte ersetzt. Dadurch enthält der synthetische Datensatz die gleiche Anzahl an Spalten und Zeilen, sowie die gleichen Ressourcen wie AFE. Damit die Ergebnisse dieses Benchmarktests auch mit anderen Ergebnissen vergleichbar sind, werden in den Spalten mit Unsicherheiten jeweils 10.000 Aussagen als unsicher markiert. Die Laufzeitergebnisse des Benchmarktests sind in Tabelle 4.17 gezeigt. Ungekürzte Ergebnisse sind im Projektordner unter *data/results/synthetic_afe.txt* hinterlegt.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	0.121	0.206	0.152	0.530	0.178	0.111	0.178	0.087	0.039	0.037
2	0.097	0.193	0.131	0.565	0.061	0.088	0.193	0.103	0.065	0.070
3	0.176	0.363	0.253	1.077	0.203	0.147	0.357	0.163	0.096	0.098
4	2.792	1.878	4.673	3.296	4.539	2.609	1.896	1.741	1.425	1.436
5	0.903	0.656	1.235	1.104	1.813	0.832	0.661	0.477	0.369	0.368
6	X	X	6.191	4.073	8.207	X	X	X	X	1.955

Tabelle 4.17: Ergebnisse der Laufzeiten in Sekunden für jedes Modellierung-Query-Paar. Anstelle des AFE Datensatzes wurde ein künstlich erzeugter und vollständiger Datensatz genutzt.

In diesem Benchmarktest zeigen sich die bisher längsten Laufzeiten pro SPARQL-Anfrage. Bei der Ausführung von Query 6 und der Modellierung 6, zeigt sich eine Laufzeit von über 8 Sekunden, was vermutlich hauptsächlich an der Größe des Graphen liegt, die mehr als das 5-fache größer ist als der Graph des unveränderten AFE Datensatzes. Für eine bessere Interpretation der Werte wird auch hier zunächst für jede Query eine Rangliste erstellt. Tabelle 4.18 zeigt die daraus entstehenden Ränge.

Query	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
1	5	9	6	10	7	3	7	3	1	1
2	4	8	7	10	1	4	8	4	1	1
3	3	8	7	10	6	3	8	3	1	1
4	7	3	9	8	9	6	3	3	1	1
5	6	4	9	8	10	6	4	3	1	1
6	X	X	3	2	4	X	X	X	X	1

Tabelle 4.18: Ranglisten der Queries über allen verglichenen Modellierungen, ausgehend von Tabelle 4.17.

Auch diese Ranglisten zeigen größtenteils die gleichen vorteilhaften bzw. unvorteilhaften Modellierungen. Eine Besonderheit zeigen die Modellierungen 2 und 7, welche

beide in diesen Ergebnissen deutlich schlechtere Ränge besitzen als in den bisherigen Ranglisten. Vor allem zeigt sich dieses Verhalten in den ersten drei Queries, in denen diese mit Modellierung 4 zu den schlechtesten gehören. Auch hier werden die durchschnittlichen Ränge erstellt und in Tabelle 4.19 zusammengetragen.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
Mittelwerte	5	6.4	6.8	8	6.2	4.4	6	3.2	1	1

Tabelle 4.19: Mittelwerte aller Ränge pro Modellierung aus Tabelle 4.18, ausgehend der Laufzeiten des synthetisch erzeugten Datensatzes.

Modellierung 2 und 7 besitzen zwar jeweils ihren bisher höchsten Rang, jedoch sind sie nicht die beiden schlechtesten. Die anderen Ränge zeigen in diesen Ergebnissen keine großen Besonderheiten im Vergleich zu den bisherigen. Nur Modellierung 6 besitzt ihren bisher besten Rang.

4.3 Diskussion und Empfehlungen

Die soeben gezeigten Benchmarktests sollten jeweils einen Teil der zuvor definierten Fragestellung genauer betrachten. Somit wird jeweils eine Frage der Fragestellung durch die Ergebnisse eines Benchmarktests beleuchtet. Dafür wurden dazu passende Datensätze erstellt bzw. modifiziert und die Laufzeiten der Modellierung-Query Paare festgehalten. Welche Schlüsse aus diesen Ergebnissen gezogen werden können, wird im Folgenden anhand von Tabelle 4.20 gezeigt, in der die Ergebnisse der einzelnen Benchmarktests zusammengefasst sind. Durch die Gegenüberstellung der Ränge lassen sich somit zu einem gegebenen Anwendungsfall die passende Modellierung auswählen.

	Modellierung									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9a**	**9b**
AFE	4.4	4.8	8.2	6.8	7.7	5	4.4	3.2	1	1.7
afe-10kU	6	5.4	8.2	8	5.3	5	5	3.2	1	1
afe-100A	8.2	4.4	7.7	6.8	5.8	7.6	4	1.4	1	1
synthetisch	5	6.4	6.8	8	6.2	4.4	6	3.2	1	1

Tabelle 4.20: Durchschnittliche Ränge aller Benchmarktests.

In den durchschnittlichen Rängen gibt es zum Teil größere Unterschiede zwischen den einzelnen Datensätzen, sodass sich keine einheitliche Reihenfolge der Modellierungen erkennen lässt. Werden jedoch nur die besten drei Modellierungen betrachtet, zeigt sich, dass diese in allen vier finalen Ranglisten von den Modellierungen 8, 9a und 9b belegt werden. Mit Blick auf die Übersicht der Modellierungen in Tabelle 4.7 lässt sich erkennen, dass diese auch die wenigsten Kanten pro Unsicherheit hinzufügen, weshalb diese Eigenschaft vermutlich einen positiven Einfluss auf die Query Laufzeiten hat. Auch zeigen sich die SPARQL-Queries der Modellierungen 9a und 9b als unkomplex, da durch RDF-star und SPARQL-star häufig deutlich leichtere Anfragen erzeugt werden können. Der größte Nachteil den diese Modellierungen mit sich bringen, ist die eingeschränkte Nutzbarkeit der RDF-star Erweiterung, was sich im Benchmark ebenfalls gezeigt hat, da die primär genutzte Python Bibliothek *rdflib* bisher noch keine Unterstützung für RDF-star bietet.

Empfehlung 1. Von der eingeschränkten Nutzbarkeit von RDF-star abgesehen, sind die Modellierungen 9a (für ungewichtete Unsicherheiten) und 9b (für gewichtete Unsicherheiten) im Bezug dieses Benchmarks eine klare Empfehlung. Für Systeme ohne RDF-star Unterstützung, ist die Modellierung 8 für ungewichtete Unsicherheiten zu empfehlen, welche zusätzlich in Datensätzen mit vielen Alternativen eine besonders geringe Laufzeit besitzen.

Neben einer alternativen Modellierung für ungewichtete Unsicherheiten, sollte auch eine für gewichtete Unsicherheiten festgehalten werden. Die Ergebnisse des Benchmarks zeigen jedoch, dass keine der Modellierungen für gewichtete Unsicherheiten mit

dem Rang von 9b vergleichbar ist. Sollte jedoch eine Entscheidung zwischen den Modellierungen 3, 4 und 5 getroffen werden, besitzt die fünfte häufig den besseren Rang. Bei genauerer Betrachtung der Ränge von Modellierung 5, scheinen vor allem die Laufzeiten der letzten drei Queries besonders hoch zu sein. Daher sollte bei der Wahl der Modellierung auf den Anwendungsfall bzw. dem Einsatzgebiet des RDF-Graphen geachtet werden.

Empfehlung 2. Für Systeme ohne RDF-star Unterstützung und gewichteten Unsicherheiten, sollte besonders auf die Datenstruktur und dem Einsatzgebiet des RDF-Graphen geachtet und eine dazu passende Modellierung ausgewählt werden. Die errechneten Ränge sind von Modellierung 5 am besten. Jedoch hat diese in den letzten drei Queries oftmals die größte Laufzeit und eine hohe Komplexität für Entwickler*innen, da Unsicherheiten mit 14 verschiedenen Eigenschaften zugewiesen werden können. Modellierung 3 hat in keinem der Datensätze den besten Rang und ist damit nicht zu empfehlen.

Die restlichen Modellierungen lassen sich nicht als eindeutig gut oder schlecht Klassifizieren. Modellierung 1 und 6 sind in ihren Rängen häufig ähnlich und besitzen beide bei vielen Alternativen eine deutlich höhere Laufzeit. Modellierung 2 und 7 unterscheiden sich in ihren durchschnittlichen Rängen jeweils um genau 0.4, obwohl beide eine identischen Aufbau besitzen und sich nur durch die Wahl der Ressourcen unterscheiden. Modellierung 2 nutzt dafür einen Namensraum mehr als 7, was eventuell den Unterschied von 0.4 erklären könnte. Der Einfluss der Anzahl an verwendeten Namensräume sollte jedoch zunächst weiter untersucht werden, da der Unterschied zu gering ist um eine eindeutige Aussage darüber zu tätigen.

5 Schluss und Ausblick

Ziel dieser Arbeit war es, einen umfangreichen Vergleich verschiedener Modellierungen für Unsicherheiten in RDF-Graphen durchzuführen, um so vorteilhafte und unvorteilhafte Möglichkeiten aufzudecken. Dafür wurden verschiedene Modellierungen vorgestellt und eine testbare Umgebung definiert, die ein möglichst breites Anwendungsumfeld abdeckt. Die daraus gewonnenen Ergebnisse zeigten drei Modellierungen die sich gegenüber der anderen durchgesetzt haben.

Die hier verglichenen Modellierungen decken zwar ein breites Spektrum der Möglichkeiten ab, jedoch gibt es auch Konzepte von RDF, welche weitere Ansätze bieten. Darunter RDF-Listen, -Container oder -Collections, welche zu fortgeschritteneren Ausdrucksmittel von RDF gehören und mit denen vereinfachte Modellierungen möglich sind [Hit+08]. Davon bietet sich bspw. der Container *rdf:alt* an, welcher zur Angabe von Alternativen genutzt wird und somit auch für Unsicherheiten mit alternativen Werten zum Einsatz kommen kann. Eine Anleitung, wie UnCo um neue Modellierungen erweitert werden kann, befindet sich in der Dokumentation unter „Anleitungen“.

Auch das gesamte Themenfeld RDF-star bietet weitere Ansätze für zukünftige Forschungen. Die in dieser Arbeit verglichenen RDF-star Modellierungen zeigen die bisher besten Laufzeiten der einzelnen SPARQL-Queries. Diese Ergebnisse sind jedoch nur mit Vorsicht zu genießen. Zum einen, da sich RDF-star noch in der Entwicklung[22] befindet und somit weitere Konzepte oder einflussreiche Veränderungen vorgenommen werden können. Zum anderen hat die geringe Systemunterstützung der Erweiterung auch zur Folge, dass nur eine eingeschränkte Forschung der Modellierungen möglich ist. So könnten die geringen Laufzeiten daher rühren, dass verschachtelte Tripel im RDF-Graphen nur zur Modellierung von Unsicherheiten genutzt werden und Fuseki somit in kurzer Zeit alle unsicheren Aussagen findet. Dies lässt sich mit UnCo in Zukunft genauer erforschen, wenn die Python Bibliothek *rdflib* auch die Funktionen von RDF-star übernimmt. An einer Erweiterung, welche *rdflib* um die Methoden von RDF-star erweitert, arbeitet zur Zeit ein Projekt der Australian National University[23].

[22] Für die Erweiterung der RDF Standards um die von RDF-star eingeführten Funktionen, wurde eine vom W3C erstellte Arbeitsgruppe gegründet [Cha22].

[23] Projekt Webseite erreichbar unter: `https://cecc.anu.edu.au/research/student-research-projects/rdf-star-reference-implementation`.

Literaturverzeichnis

[Abu+22] Ghadeer Abuoda, Daniele Dell'Aglio, Arthur Keen und Katja Hose. „Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches". In: *Proceedings of the QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs co-located with 21st International Semantic Web Conference (ISWC 2022)*. CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3279/paper2.pdf.

[Ale+14] Vladimir Alexiev u. a. *BM Association Mapping v2*. 2014. URL: https://confluence.ontotext.com/display/ResearchSpace/BM+Association+Mapping+v2#BMAssociationMappingv2-ProbablyUnlikelyProducedBy (besucht am 13. 04. 2023).

[Ale12] Vladimir Alexiev. „Types and Annotations for CIDOC CRM Properties". In: *Digital Presentation and Preservation of Cultural and Scientific Heritage (DiPP2012) conference*. 2012.

[Cas22] Manuel Castells. *Die Internet-Galaxie*. Springer, 2022. DOI: https://doi.org/10.1007/978-3-658-35671-2.

[Cha22] Pierre-Antoine Champin. *RDF-star Working Group Charter*. 2022. URL: https://www.w3.org/2022/08/rdf-star-wg-charter/ (besucht am 14. 07. 2023).

[DS+14] Martin Doerr, Stephen Stead u. a. „CRMinf: the Argumentation Model. An Extension of CIDOC-CRM to support argumentation". In: *The CIDOC Conceptual Reference Model* (2014). URL: http://old.cidoc-crm.org/docs/cidoc_crm_sig/CRMinf-0.4.pdf.

[Du 13] Bob Du Charme. *Learning SPARQL : querying and updating with SPARQL 1.1*. O'Reilly Media, 2013. ISBN: 9781449371432.

[EE04] Rainer Eckstein und Silke Eckstein. *XML und Datenmodellierung: XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*. dpunkt.verlag GmbH, 2004. ISBN: 3898642224.

[Gei09] Matthias Geisler. *Semantic Web: schnell + kompakt*. entwickler.press von Software & Support Verlag GmbH, 2009. ISBN: 9783868020281.

[Gmb23] mosaiic GmbH. *Auf Daten fokussiert – das Geschäftsmodell von PAYBACK*. 2023. URL: https://www.mosaiic.de/das-geschaeftsmodell-von-payback/ (besucht am 12. 07. 2023).

[Gro06] CRM Special Interest Group. *Official CIDOC CRM website*. 2006. URL: https://cidoc-crm.org (besucht am 03. 07. 2023).

[Har+21] Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg und Andy Seaborne. *RDF-star and SPARQL-star*. 2021. URL: https://www.w3.org/2021/12/rdf-star.html (besucht am 09. 05. 2023).

[Har19] Olaf Hartig. „Position Statement: The RDF* and SPARQL* Approach to Annotate Statements in RDF and to Reconcile RDF and Property Graphs“. In: *Linköpings Universitet* (2019). URL: https://blog.liu.se/olafhartig/2019/01/10/position-statement-rdf-star-and-sparql-star/.

[HB15] Torsten Hoefler und Roberto Belli. „Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results“. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015. URL: https://doi.org/10.1145/2807591.2807644.

[Hit+08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph und York Sure. *Semantic Web: Grundlagen*. Springer, 2008. ISBN: 9783540339939. DOI: 10.1007/978-3-540-33994-6.

[HT14] Olaf Hartig und Bryan Thompson. „Foundations of an Alternative Approach to Reification in RDF“. In: *CoRR* (2014). DOI: https://doi.org/10.48550/arXiv.1406.3399.

[Hup09] Karl Huppler. „The Art of Building a Good Benchmark“. In: *Performance Evaluation and Benchmarking*. Hrsg. von Raghunath Nambiar und Meikel Poess. Springer, 2009. ISBN: 978-3-642-10424-4.

[Kol20] Tobias Kollmann. *Handbuch Digitale Wirtschaft*. Springer, 2020. DOI: https://doi.org/10.1007/978-3-658-17291-6.

[Mus14] The British Museum. *ResearchSpace - a Digital Wunderkammer for the Cultural Heritage*. 2014. URL: https://researchspace.org (besucht am 13. 04. 2023).

[NH16] Franco Niccolucci und Sorin Hermon. „Expressing reliability with CIDOC CRM“. In: *Int. J. Digit. Libr.* 18.4 (2016), S. 281–287. DOI: 10.1007/s00799-016-0195-1.

[Obe19] Land Oberösterreich. *Woher kommt Europa?* 2019. URL: https://europainfo.at/at25eu/woher-kommt-europa/ (besucht am 04. 05. 2023).

[Saw93] Tom Sawyer. „Doing Your Own Benchmark“. In: *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Hrsg. von Jim Gray. Morgan Kaufmann, 1993. ISBN: 1-55860-292-5.

[SS22] Ram Sabah und Zeena Sabah. „Modeling Nomisma ontology and Comparing Solutions for Uncertainty“. Magisterarb. Johann Wolfgang Goethe Universität Frankfurt am Main, 2022. URL: http://www.bigdata.uni-frankfurt.de/wp-content/uploads/2023/02/Masterarbeit_Ram_and_Zeena-Sabah.pdf.

[Tol06] Karsten Tolle. „Semantisches Web und Kontext: Speicherung von und Anfragen auf RDF-Daten unter Berücksichtigung des Kontextes“. Diss. Goethe University Frankfurt am Main, Germany, 2006. URL: https://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/2492.

[TW14] K Tolle und D Wigg-Wolf. „Uncertainty handling for ancient coinage“. In: *CAA2014. 21st Century Archaeology. Concepts, Methods and Tools. Proceedings of the 42nd Annual Conference on Computer Applications and Quantitative Methods in Archaeology (Oxford 2015)*. 2014, S. 171–178.

[WSC+19] Lukas M. Weber, Wouter Saelens, Robrecht Cannoodt u. a. *Essential guidelines for computational method benchmarking*. 2019. URL: https://doi.org/10.1186/s13059-019-1738-8.

[Zho+21] Han Zhong, Jiayi Huang, Lin Yang und Liwei Wang. „Breaking the Moments Condition Barrier: No-Regret Algorithm for Bandits with Super Heavy-Tailed Payoffs“. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/843a4d7fb5b1641b0bb8e3c2b2e75231-Paper.pdf.

Abbildungsverzeichnis

Tabellenverzeichnis

A RDF Namensräume

Präfix	Namensraum
amt	http://academic-meta-tool.xyz/vocab#
bmo	http://collection.britishmuseum.org/id/ontology/
crm	http://www.cidoc-crm.org/cidoc-crm/
crminf	http://www.cidoc-crm.org/crminf/sites/default/files/CRMinf_v0.7_.rdfs#
edtfo	http://periodo.github.io/edtf-ontology/edtfo.ttl#
nm	http://nomisma.org/id/
nmo	http://nomisma.org/ontology#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#
un	http://www.w3.org/2005/Incubator/urw3/XGR-urw3-20080331/Uncertainty.owl#

Tabelle A.1: Tabelle der genutzten Namensräume.

B .2-Properties von Modellierung 5

.2-Property	Nomisma Eigenschaft
P3_uncertain_value	hasAxis hasDepth hasMaxDepth hasMinDepth hasHeight hasMaxHeight hasMinHeight hasWidth hasMaxWidth hasMinWidth hasWeight hasWeightStandard hasBearsDate hasEndDate hasStartDate hasDenomination
P107_uncertain_member	hasCollection hasTypeSeriesItem
P102_uncertain_name_or_ethnic	hasContemporaryName hasScholarlyName
P16_uncertain_technique_or_object_used_for_creation	hasDie hasProductionObject hasManufacture
P67_uncertain_type	hasObjectType representsObjectType

Tabelle B.1: Erster Teil der Zuordnung der Nomisma Eigenschaften aus *nmo* zu den entsprechenden .2-Properties aus *crm*. Die Tabelle wurde aus [SS22] übernommen.

.2-Property	Nomisma Eigenschaft
P103_uncertain_symbole_or_feature	hasCountermark hasMintmark hasSecondaryTreatment hasPeculiarity hasPeculiarityOfProduction hasCorrosion hasWear
P138_uncertain_authenticity	hasAuthenticity
P14_uncertain_authority_or_issuer	hasAuthority hasIssuer
P189_uncertain_place	hasMint hasFindspot
P137_uncertain_material	hasMaterial
P136_uncertain_context_or_taxonomy	hasContext
P139_uncertain_form	hasAppearance hasShape hasEdge
P19_uncertain_mode	hasFace hasObverse hasReverse
P19_uncertain_depiction	hasPortrait hasIconography hasLegend

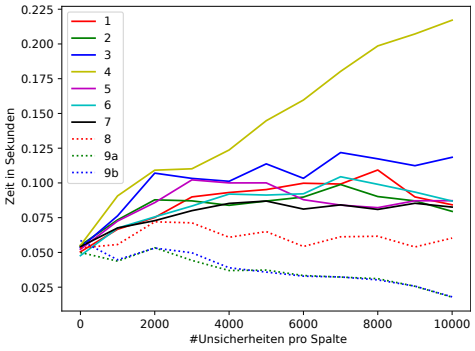
Tabelle B.2: Zweiter Teil der Zuordnung der Nomisma Eigenschaften aus *nmo* zu den entsprechenden .2-Properties aus *crm*. Die Tabelle wurde aus [SS22] übernommen.

C Genutzte Software und Bibliotheken

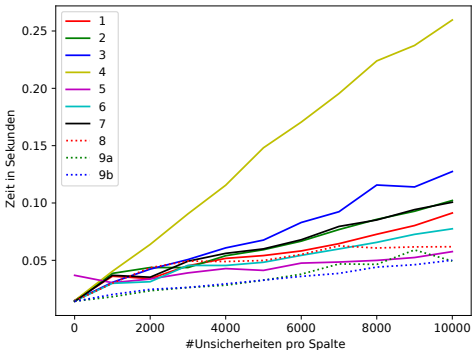
Name der Software	Version
Ubuntu	22.04.2 LTS
Python	3.10.6
Java	11.0.19
Apache Jena Fuseki	4.8.0
matplotlib	3.7.1
numpy	1.25.0
pandas	2.0.2
pathlib	1.0.1
psutil	5.9.5
pydoc-markdown	4.8.2
requests	2.31.0
rdflib	6.3.2
streamlit	1.24.0
tqdm	4.65.0

Tabelle C.1: Tabelle der genutzten Software und Python Bibliotheken inklusive Versionen.

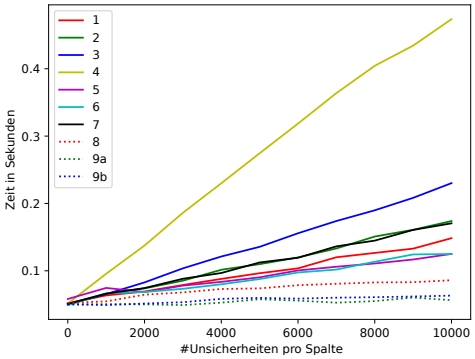
D Ergebnisse bei steigender Anzahl Unsicherheiten



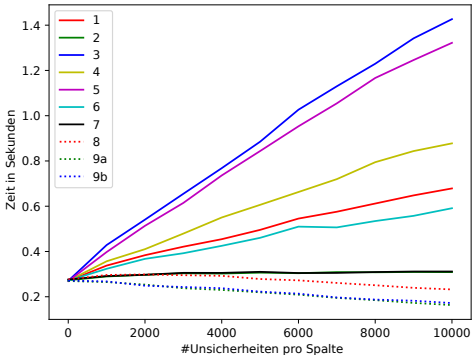
(a) Laufzeitverhalten Query 1



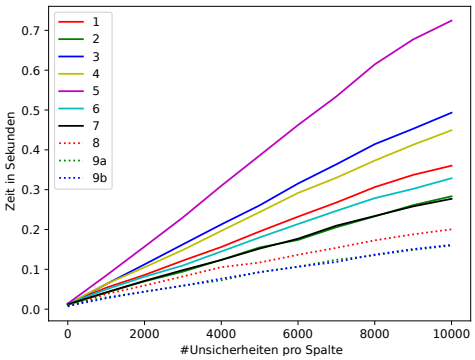
(b) Laufzeitverhalten Query 2



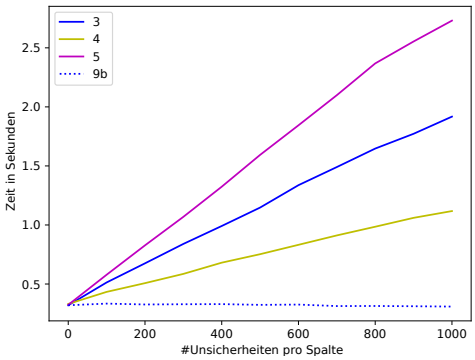
(c) Laufzeitverhalten Query 3



(d) Laufzeitverhalten Query 4



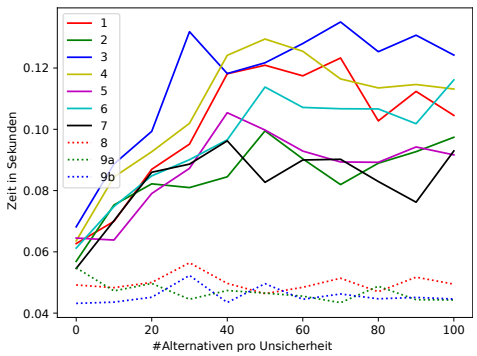
(e) Laufzeitverhalten Query 5



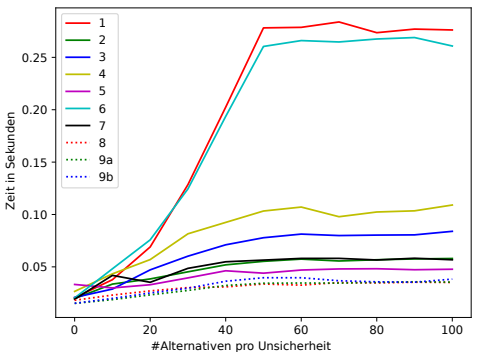
(f) Laufzeitverhalten Query 6

Abbildung D.1: Laufzeitverhalten von Fuseki bei Ausführung der Queries. Schrittweise wurden den Spalten 2, 3, 4, 7, 10, 16, 17, 18 und 19 bis zu 1000 Unsicherheiten zugewiesen.

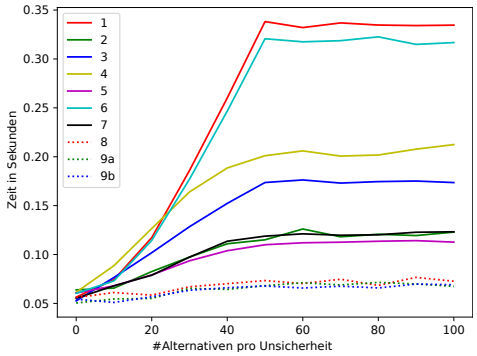
E Ergebnisse bei steigender Anzahl Alternativen



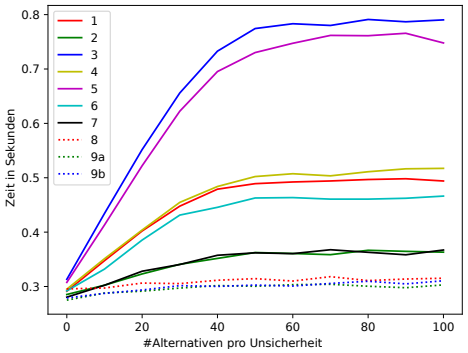
(a) Laufzeitverhalten Query 1



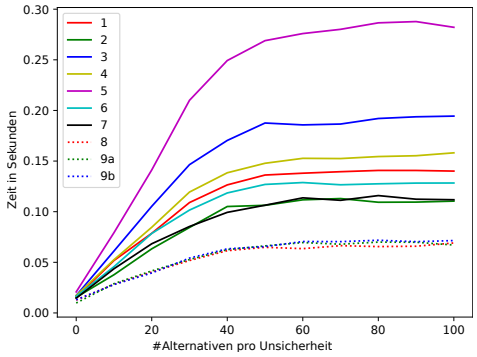
(b) Laufzeitverhalten Query 2



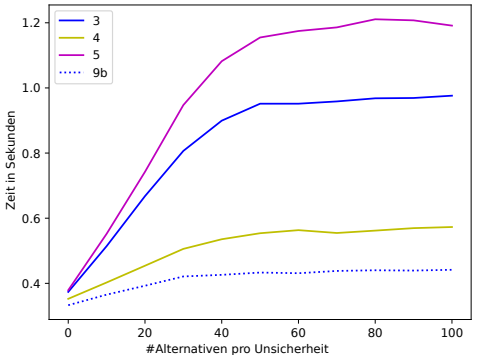
(c) Laufzeitverhalten Query 3



(d) Laufzeitverhalten Query 4



(e) Laufzeitverhalten Query 5



(f) Laufzeitverhalten Query 6

Abbildung E.1: Laufzeitverhalten von Fuseki bei Ausführung der Queries. Schrittweise wurden den unsicheren Aussagen bis zu 100 Alternativen hinzugefügt.