DHBW
Duale Hochschule
Baden-Württemberg

Visualisierung von großen Datasets aus Graph-Datenbanken

Studienarbeit

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Heidenheim

von

Jonas Wiedenmann

September 2024

Bearbeitungszeitraum	22 Wochen
Matrikelnummer, Kurs	Matrikelnummer, INF2021
Gutachter	Franziska Schütz

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *Visualisierung von großen Datasets aus Graph-Datenbanken* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Heidenheim an der Brenz, September 2024

_____

Jonas Wiedenmann

Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
SPARQL	SPARQL Protocol And RDF Query Language
RDF	Resource Description Framework
SPA	Single Page Application
DOM	Document Object Model
JSON	JavaScript Object Notation

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Die Visualisierung von Daten hat sich in den letzten Jahren zu einem bedeutenden Forschungsfeld entwickelt. Sie befasst sich mit der Erstellung verständlicher grafischer Darstellungen aus komplexen und großen Datenmengen. Dadurch wird es möglich, komplexe Informationen auf eine zugängliche Weise darzustellen und tiefere Erkenntnisse aus den Daten zu gewinnen. Grafische Darstellungen können Muster und Merkmale sichtbar machen, die in rohen Statistiken oft übersehen werden (vgl. [Unw20], S. 2).

In den letzten Jahren hat sich die Datenlandschaft jedoch drastisch verändert, insbesondere durch den Aufstieg von Big Data. Verknüpfte Informationen, die die Beziehungen zwischen Entitäten beschreiben, haben enorm an Bedeutung gewonnen. Besonders in Bereichen wie der Biomedizin, sozialen Netzwerken und Kommunikationsnetzwerken fallen große Mengen solcher Daten an (vgl. [Che+19], S. 1 f.). Diese Daten können durch Graphen modelliert werden. Sie sind mathematische Strukturen, die Knoten und Kanten verwenden, um komplexe Netzwerke darzustellen.

Obwohl Graphen mathematisch gut beschrieben werden können, erfordert die visuelle Analyse der Daten eine geeignete grafische Darstellung. Es existieren zahlreiche Visualisierungstechniken, die Graphen darstellen können, darunter Node-Link-Diagramme, Adjazenzmatrizen und Hypergraphen (vgl. [Che+19], S. 2). Jede dieser Methoden hat jedoch ihre spezifischen Stärken und Schwächen. So sind Node-Link-Diagramme beispielsweise leicht verständlich, werden jedoch schnell unübersichtlich, wenn die Anzahl der Knoten oder Kanten zunimmt. Das eigentliche Problem bei der Visualisierung von Graphen liegt nicht im Fehlen geeigneter Techniken, sondern in der Auswahl der richtigen Methode. Diese Auswahl hängt stark vom jeweiligen Anwendungsfall ab, da jede Technik unterschiedliche Aspekte der Daten hervorhebt.

1.1 Zielsetzung

Ziel der Arbeit ist es, zu ermitteln, welche Visualisierungstechnik sich am besten für Knowledge Graphen eignet. Anschließend soll die Visualisierung eines Knowledge Graphen implementiert werden.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier Hauptkapitel, die systematisch aufeinander aufbauen und die verschiedenen Aspekte der Visualisierung von großen Datasets aus Graph-Datenbanken behandeln.

Im ersten Kapitel, der Einleitung, wird das Thema der Arbeit eingeführt und die Zielsetzung klar definiert. Es wird erläutert, welche Relevanz die Visualisierung von Graph-Datenbanken hat, insbesondere im Kontext großer und komplexer Datenmengen. Darüber hinaus wird das Ziel der Arbeit formuliert.

Das zweite Kapitel beschäftigt sich mit den theoretischen Grundlagen der Arbeit. Es erläutert die Definition und Merkmale von Graphen sowie Knowledge Graphen und geht auf deren Unterschiede ein. Weiterhin werden Anwendungsgebiete von Knowledge Graphen beschrieben und verschiedene Visualisierungstools wie Gephi und Cytoscape vorgestellt.

Das dritte Kapitel behandelt die praktische Umsetzung der im theoretischen Teil gewonnenen Erkenntnisse. Es beginnt mit einer detaillierten Anforderungsanalyse, in der sowohl funktionale als auch nicht-funktionale Anforderungen an die entwickelte Lösung definiert werden. Anschließend wird die Auswahl der verwendeten Technologien sowie die konkrete Systemarchitektur beschrieben. Der Hauptteil dieses Kapitels widmet sich der Implementierung.

Im vierten und letzten Kapitel erfolgt eine Zusammenfassung der Ergebnisse der Arbeit. Die wichtigsten Erkenntnisse werden zusammengetragen und kritisch bewertet, wobei sowohl die Stärken als auch die Schwächen der entwickelten Lösung beleuchtet werden. Abschließend gibt das Kapitel einen Ausblick auf mögliche Weiterentwicklungen und Erweiterungen der Visualisierungstechniken sowie die potenziellen Einsatzmöglichkeiten der entwickelten Lösung in weiteren Anwendungsbereichen.

2 Theoretische Grundlagen

Dieses Kapitel bildet die Grundlage für die folgenden Vergleiche und Evaluationen von Graph Visualisierungen. Es werden Graphen und Knowledge Graphen genauer erläutert, ein Datenformat zur Speicherung der Daten und zuletzt die verschiedenen Visualisierungsoptionen.

2.1 Graphen und Knowledge Graphen

Graphen sind eine grundlegende Datenstruktur in der Informatik, die aus Knoten und Kanten besteht und zur Modellierung von Netzwerken verwendet wird. Diese Netzwerke können verschiedenste Formen annehmen, von sozialen Netzwerken über Kommunikationsstrukturen bis hin zu biologischen Netzwerken. Die einfache Darstellung von Beziehungen zwischen Entitäten durch Knoten und Kanten macht Graphen zu einem mächtigen Werkzeug für die Datenanalyse und -visualisierung.

In diesem Kapitel werden zunächst die grundlegenden Merkmale von Graphen erläutert und anschließend die besonderen Eigenschaften von Knowledge Graphen vorgestellt. Dabei wird ein besonderer Fokus auf die Unterschiede zwischen traditionellen Graphen und Knowledge Graphen gelegt sowie auf die verschiedenen Anwendungsbereiche, in denen Knowledge Graphen eine Rolle spielen.

2.1.1 Definition und Merkmale von Graphen

Ein Graph ist eine mathematische Struktur, die aus einer Menge von Knoten (engl. vertices) und einer Menge von Kanten (engl. edges) besteht. Die Kanten haben die Aufgabe, Paare von Knoten zu verbinden. Graphen werden verwendet, um Netzwerke zu modellieren, in denen Objekte und deren Beziehungen zueinander abgebildet werden. Diese Netzwerke können unterschiedlichster Natur sein, wie soziale Netzwerke, Kommunikationsnetzwerke oder biologische Netzwerke. Die grundlegenden Merkmale eines Graphen sind seine Knoten, die Entitäten repräsentieren, und die Kanten, die die Beziehungen zwischen diesen Entitäten darstellen (vgl. [Wil96], S. 1 ff.).

Graphen sind besonders wertvoll in der Datenvisualisierung, da sie es ermöglichen, komplexe Beziehungen und Strukturen auf eine Weise darzustellen, die visuell zugänglich und

verständlich ist. In der Visualisierung werden Knoten typischerweise durch Kreise oder Punkte dargestellt, während Kanten als Linien oder Pfeile visualisiert werden, die die Richtung und Stärke der Beziehungen anzeigen. Die räumliche Anordnung der Knoten und Kanten spielt dabei eine entscheidende Rolle, da sie die Lesbarkeit und Interpretierbarkeit des Graphen maßgeblich beeinflusst. Verschiedene Layout-Algorithmen können verwendet werden, um die Knoten so anzuordnen, dass Muster wie Cluster oder Hierarchien besser erkennbar werden (vgl. [Wil96], S. 1 ff.). Die verschiedenen Möglichkeiten zur Darstellung von Graphen werden im Kapitel 2.3 noch genauer beschrieben.

2.1.2 Unterschied zwischen traditionellen Graphen und Knowledge Graphen

Während traditionelle Graphen primär auf die Darstellung von Entitäten und ihren direkten Beziehungen fokussiert sind, erweitern Knowledge Graphen dieses Modell, indem sie zusätzliche semantische Schichten und Kontextinformationen integrieren. Knowledge Graphen verbinden Entitäten nicht nur durch einfache Beziehungen, sondern auch durch semantische Bedeutungen, die es ermöglichen, komplexe Zusammenhänge und Abhängigkeiten darzustellen. Ein Knowledge Graph ist somit nicht nur eine Struktur von Knoten und Kanten, sondern ein umfassendes Netzwerk, das Entitäten, deren Beziehungen und die zugrundeliegenden Bedeutungen in einem kontextualisierten Modell vereint (vgl. [Cha+22], S. 17 f.).

2.1.3 Anwendungsbereiche von Knowledge Graphen

Knowledge Graphen finden in einer Vielzahl von Anwendungsbereichen Einsatz, insbesondere dort, wo es auf die Darstellung und Analyse komplexer Beziehungen ankommt. Ein prominentes Beispiel ist der Einsatz von Knowledge Graphen in Suchmaschinen wie Google, wo sie verwendet werden, um Suchergebnisse durch die Verknüpfung von Entitäten und deren Beziehungen kontextualisiert und präzisiert darzustellen. Auch in Sprachassistenten wie Siri oder Alexa spielen Knowledge Graphen eine wichtige Rolle, indem sie die Verbindungen zwischen Begriffen und Konzepten erkennen und nutzen, um relevante und sinnvolle Antworten zu generieren (vgl. [Cha+22], S. 18 ff.).

Durch die Visualisierung von Knowledge Graphen werden diese komplexen und vernetzten Datenmengen zugänglicher und verständlicher. Die Visualisierung unterstützt nicht nur die Datenanalyse, sondern erleichtert auch die Kommunikation von Erkenntnissen, indem

sie abstrakte Zusammenhänge in eine greifbare Form überführt. Dies macht Knowledge Graphen zu einem unverzichtbaren Werkzeug in vielen modernen datengetriebenen Anwendungen (vgl. [Cha+22], S. 26).

2.2 Visualisierungstools für Knowledge Graphen

Die Visualisierung von Knowledge Graphen ist entscheidend, um komplexe Beziehungen und Datenstrukturen verständlich darzustellen. Diese Visualisierungen ermöglichen es, die semantischen Verbindungen zwischen Entitäten zu erkennen und tiefere Einblicke in die zugrunde liegenden Daten zu gewinnen. In diesem Kapitel werden die Techniken untersucht, die in den weit verbreiteten Visualisierungstools Gephi und Cytoscape eingesetzt werden. Diese Analyse soll die Grundlage für die spätere Implementierung einer eigenen Software bilden.

2.2.1 Herausforderungen bei der Visualisierung von Knowledge Graphen

Die Visualisierung von Knowledge Graphen bringt spezifische Herausforderungen mit sich, die über die traditionellen Anforderungen an Graphvisualisierungen hinausgehen. Das Paper *Knowledge Graph Visualization: Challenges, Framework, and Implementation* beschreibt verschiedene Herrausforderungen, die entscheidend dafür sind, komplexe Datenmengen in einer Weise zu visualisieren, die sowohl zugänglich als auch verständlich ist. In diesem Abschnitt werden die zentralen Herausforderungen beleuchtet, die es bei der effektiven Visualisierung von Knowledge Graphen zu berücksichtigen gilt.

Eine zentrale Herausforderung ist die Skalierbarkeit, insbesondere in Bezug auf die Performance bei großen Datenmengen. Knowledge Graphen bestehen oft aus einer sehr großen Anzahl von Knoten und Kanten, was die Visualisierung vor erhebliche technische Herausforderungen stellt. Die Visualisierungstools müssen in der Lage sein, diese Datenmengen effizient zu verarbeiten und darzustellen, ohne dass die Systemleistung darunter leidet oder die Benutzererfahrung beeinträchtigt wird (vgl. [NKI20], S. 175). Ein weiterer wichtiger Aspekt ist die Interaktivität. Benutzer müssen die Möglichkeit haben, die Visualisierung dynamisch zu erkunden, um relevante Informationen herauszufiltern und die Daten aus verschiedenen Perspektiven zu betrachten. Dies erfordert Techniken, die eine reibungslose und intuitive Interaktion mit dem Graphen ermöglichen (vgl. [NKI20], S. 175). Die Verständlichkeit der Darstellung ist ebenfalls von großer Bedeutung. Sie bezieht sich darauf, wie gut die Visualisierung die strukturellen Verbindungen und Bedeutungen zwischen

den Entitäten vermittelt. Eine klare und verständliche Darstellung ist notwendig, damit Benutzer die Beziehungen innerhalb des Knowledge Graphen intuitiv erfassen können (vgl. [NKI20], S. 175).

Das Paper gibt neben den gerade vorgestellten Herausforderungen außerdem auch konkrete Empfehlungen zur Entwicklung eines Visualisierungstools. Eine der zentralen Empfehlungen des Papers ist, die Benutzeroberfläche so zu gestalten, dass sie speziell auf Nicht-Experten ausgerichtet ist. Die Visualisierung sollte den Benutzern eine intuitive und leicht verständliche Schnittstelle bieten, um komplexe Knowledge Graphen zu analysieren und zu verstehen (vgl. [NKI20], S. 176). Ein weiterer wichtiger Aspekt ist die Leistungsfähigkeit des Visualizers. Um die Verarbeitung großer Datenmengen zu erleichtern, sollte das Backend-System vom UI getrennt werden. Dies ermöglicht die Verlagerung rechenintensiver Aufgaben auf spezialisierte oder skalierbare Servercluster, was die Reaktionsfähigkeit des Visualizers verbessert und die Benutzerfreundlichkeit erhöht (vgl. [NKI20], S. 176). Zudem empfiehlt das Paper die Implementierung einer modularen Architektur. Ein Knowledge Graph Visualizer sollte erweiterbar und flexibel sein, um neue Technologien und anwendungsspezifische Anforderungen unterstützen zu können. Dies ist entscheidend, um die Anpassungsfähigkeit des Tools zu gewährleisten und es kontinuierlich weiterentwickeln zu können (vgl. [NKI20], S. 176).

2.2.2 Gephi

Gephi ist ein Open-Source-Tool, das sich auf die Visualisierung und Analyse von Netzwerken spezialisiert hat. Es ist besonders nützlich für die Verarbeitung und Darstellung großer und komplexer Graphen. Ein wesentlicher Vorteil von Gephi liegt in seiner Fähigkeit, große Mengen an Knoten und Kanten effizient zu verarbeiten und darzustellen (vgl. [BHJ09], S. 1).

Ein zentrales Feature von Gephi ist das Force-Directed Layout (vgl. [BHJ09], S. 1). Diese Technik simuliert physikalische Kräfte, bei denen Knoten als geladene Teilchen und Kanten als elastische Federn modelliert werden. Dadurch werden die Knoten so angeordnet, dass verwandte Entitäten näher beieinander liegen, was es ermöglicht, Cluster innerhalb des Netzwerks zu identifizieren und die Struktur des Netzwerks verständlicher darzustellen (vgl. [yWob]). Force-Directed Layouts werden in Gephi häufig eingesetzt, um komplexe Netzwerke intuitiv zu visualisieren.

Darüber hinaus verfügt Gephi über Techniken für Clustering und Community Ermittlung, die es ermöglichen, Gruppen verwandter Knoten in einem Netzwerk zu identifizieren (vgl. [BHJ09], S. 1 f.). Diese Funktionen sind besonders nützlich für die Analyse semantischer Beziehungen in großen Netzwerken. Durch die Anwendung von Algorithmen können

Benutzer Untergruppen innerhalb des Netzwerks erkennen und analysieren, was eine detailliertere Untersuchung der Netzwerkstruktur ermöglicht (vgl. [yWoa]).

Ein weiteres nützliches Feature von Gephi ist die Unterstützung für dynamische Graphvisualisierungen. Diese Funktion ermöglicht es, zeitliche Veränderungen in Netzwerken darzustellen, was besonders hilfreich ist, um Entwicklungen über Zeit zu analysieren (vgl. [BHJ09], S. 2).

Eine der technischen Stärken von Gephi liegt in seiner Unterstützung für GPU-Rendering, das die Verarbeitungsgeschwindigkeit bei der Visualisierung großer Netzwerke erheblich steigert. Durch die Auslagerung rechenintensiver Aufgaben auf die Grafikkarte wird die CPU entlastet, was eine flüssigere und schnellere Darstellung von Graphen ermöglicht. Zusätzlich ist Gephi für Multitasking ausgelegt, was es ermöglicht, mehrere Prozesse gleichzeitig auszuführen, ohne die Gesamtleistung der Software erheblich zu beeinträchtigen. Diese Funktionen sind besonders vorteilhaft bei der Verarbeitung großer Datenmengen und tragen dazu bei, dass Gephi auch bei komplexen Visualisierungsaufgaben effizient arbeitet (vgl. [BHJ09], S. 1).

Gephi bietet damit eine robuste Plattform für die Visualisierung und Analyse von Netzwerken, wobei der Fokus auf der effizienten Verarbeitung großer Datenmengen und der Unterstützung interaktiver und dynamischer Visualisierungen liegt.

2.2.3 Cytoscape

Cytoscape ist ein Open-Source-Tool, das ursprünglich für die Visualisierung und Analyse von biomolekularen Netzwerken entwickelt wurde. Heute wird es jedoch in verschiedenen Bereichen zur Visualisierung komplexer Netzwerke verwendet, einschließlich Knowledge Graphen (vgl. [Sha+03], S. 1).

Eine wichtige Visualisierungstechnik, die Cytoscape bietet, ist das hierarchische Layout. Dieses Layout ordnet Knoten in einer hierarchischen Struktur an, wodurch Beziehungen zwischen verschiedenen Ebenen eines Netzwerks verdeutlicht werden können. Diese Methode ist besonders nützlich für Netzwerke, die eine natürliche Hierarchie aufweisen, da sie es ermöglicht, die Struktur des Netzwerks klar und strukturiert darzustellen (vgl. [Sha+03], S. 3).

Cytoscape nutzt Edge-Bundling, um visuelle Unordnung in dichten Netzwerken zu reduzieren. Dabei werden Kanten, die in ähnliche Richtungen verlaufen, gebündelt, was die Darstellung von Verbindungen zwischen Knoten deutlich klarer und übersichtlicher macht (vgl. [Thea]).

Darüber hinaus ermöglicht Cytoscape, die Layouts basierend auf Knotenattributen individuell anzupassen. Diese Funktion erlaubt es den Nutzern, die Visualisierung gezielt nach den relevanten Merkmalen der Knoten auszurichten. Dies ist besonders hilfreich, wenn spezifische Attribute im Fokus der Analyse stehen, da die Darstellung so optimal auf diese abgestimmt werden kann und eine präzisere Untersuchung der Daten ermöglicht wird (vgl. [Sha+03], S. 3 f.).

Ein weiterer wesentlicher Aspekt von Cytoscape ist seine Erweiterbarkeit. Das Tool unterstützt eine Vielzahl von Plugins, die es ermöglichen, die Funktionalität von Cytoscape zu erweitern und an spezifische Anforderungen anzupassen. Diese Erweiterungsfähigkeit macht Cytoscape zu einem flexiblen Werkzeug, das in vielen verschiedenen Anwendungsbereichen eingesetzt werden kann (vgl. [Sha+03], S. 6).

Insgesamt stellt Cytoscape eine robuste Lösung für die Visualisierung von Netzwerken dar, die besonders durch ihre Flexibilität und Erweiterbarkeit überzeugt. Die unterstützten Visualisierungstechniken wie hierarchische Layouts, Edge-Bundling und attributbasierte Layouts ermöglichen es, komplexe Netzwerke strukturiert und übersichtlich darzustellen, was es zu einem wertvollen Werkzeug in der Netzwerkvisualisierung macht.

2.3 Visualisierungstechniken

Im vorherigen Kapitel wurden die Visualisierungstools Gephi und Cytoscape detailliert analysiert. Dabei wurden die wichtigsten Funktionen und Techniken hervorgehoben, die es den Tools ermöglichen, komplexe Netzwerke zu visualisieren und Verbindungen zwischen Entitäten verständlich darzustellen. Während diese Tools leistungsstarke Frameworks zur Visualisierung bieten, liegt der Fokus dieses Kapitels auf den spezifischen Visualisierungstechniken, die zur Darstellung von Knowledge Graphen verwendet werden können.

Die Wahl der richtigen Visualisierungstechnik ist entscheidend für die effektive Darstellung und Analyse von Knowledge Graphen. Unterschiedliche Techniken haben ihre eigenen Stärken und Schwächen, die je nach Anwendungsfall variieren. In diesem Kapitel werden die gängigen Diagrammarten und Visualisierungstechniken untersucht, die für Knowledge Graphen relevant sind. Es wird aufgezeigt, wie diese Techniken die Darstellung großer und komplexer Datenmengen beeinflussen und welche Aspekte bei der Auswahl der geeigneten Methode berücksichtigt werden sollten.

2.3.1 Diagrammarten

Die Visualisierung spielt eine zentrale Rolle bei der Analyse von Knowledge Graphen, da sie das Verständnis der komplexen Beziehungen zwischen den Entitäten und der Datenstruktur erheblich erleichtert. In diesem Kapitel werden verschiedene Diagrammtypen vorgestellt, die häufig für die Visualisierung von Knowledge Graphen verwendet werden. Dabei werden die jeweiligen Eigenschaften dieser Diagramme hervorgehoben und ihre Vor- und Nachteile erläutert, um eine fundierte Grundlage für die Wahl der geeigneten Visualisierungsmethode zu bieten.

Node-Link Diagramme

Node-Link Diagramme sind die intuitivste Art der Graphdaten-Visualisierung. Sie bieten eine zugängliche und visuell ansprechende Möglichkeit, die Vielschichtigkeit von Netzwerken darzustellen. In diesen Diagrammen werden Entitäten als Knoten abgebildet, die meistens durch Kreise dargestellt werden. Um unterschiedliche Typen von Entitäten oder deren Zustände zu kennzeichnen können allerdings auch andere Formen verwendet werden. Die Beziehungen zwischen diesen Entitäten werden durch Linien oder Pfeile dargestellt, wobei häufig verschiedene Stile oder Farben genutzt werden, um die Beschaffenheit oder Intensität dieser Verbindungen zu illustrieren. Die räumliche Positionierung der Knoten und Verbindungen in Node-Link Diagrammen kann die Lesbarkeit sowie die aus der Visualisierung gewonnenen Einsichten stark beeinflussen. Eine sinnvolle Anordnung der Knoten erlaubt es, die spezifischen Charakteristiken des Netzwerks, wie beispielsweise Cluster zu betonen und diese erkennbar zu machen (vgl. [Theb]).

Der größte Vorteil von Node-Link Diagrammen besteht in ihrer Fähigkeit, abstrakte relationale Daten begreifbar zu machen. Sie ermöglichen es dem Betrachter, Muster zu entdecken, Unregelmäßigkeiten zu erkennen und die Struktur des Netzwerks durch visuelle Untersuchung zu begreifen. Das macht sie unverzichtbar für jeden Einsatz, bei dem die Beziehungsdynamiken und die Struktur der Daten von Belang sind. Ein Node-Link Diagramm kann beispielsweise in der Analyse sozialer Netzwerke auf einen Blick die Influencer aufzeigen, indem es die Knoten mit den meisten Verbindungen hervorhebt (vgl. [Theb]).

Allerdings nimmt die Effektivität von Node-Link Diagrammen mit der Größe und Komplexität des Graphen ab. In dicht vernetzten oder sehr großen Graphen kann die Übersichtlichkeit schnell abnehmen. Überlappende Verbindungen und Knoten führen oft zu einem Durcheinander, das die Visualisierung unklar und verwirrend macht. Dieses Problem kann durch interaktive Visualisierungstools reduziert werden. Diese Werkzeuge erlauben es Nutzern, den Graphen dynamisch zu vergrößern, zu filtern und zu durchsuchen. Dennoch bleibt

die Herausforderung der Verständlichkeit bei großen Graphen bestehen (vgl. [BBC+21], S. 3). Dies erfordert eine sorgfältige Planung des Diagrammdesigns und den Einsatz von Vereinfachungstechniken wie kraftbasierte Layouts und Kantenverschmelzung, um die Lesbarkeit zu verbessern.

Adjazenzmatrizen

Einen gänzlich anderen Ansatz zur Visualisierung von Graphdaten bilden die Adjazenzmatrizen. Sie bieten eine strukturierte und analytische Perspektive auf die Daten. Adjazenzmatrizen verwenden ein rasterbasiertes Layout und eignen sich daher ausgezeichnet für die Abbildung komplexer und großflächiger Netzwerke. Bei einer Adjazenzmatrix werden die Entitäten des Graphen sowohl auf der horizontalen als auch auf der vertikalen Achse einer quadratischen Matrix aufgeführt. Die Zellen innerhalb der Matrix werden gefüllt, um die Existenz einer Beziehung zwischen den entsprechenden Entitäten zu signalisieren. Diese Herangehensweise verwandelt die Vernetzung des Graphen in ein visuelles Muster aus besetzten und unbesetzten Zellen. Eine besetzte Zelle am Kreuzungspunkt der Zeile ii und Spalte jj signalisiert eine direkte Verbindung von Entität ii zu Entität jj (vgl. [SW]). Ein Beispiel für die Darstellung eines Node-Link Diagramms als Adjazenzmatrix Diagramm ist in Abbildung 2.1 zu sehen.

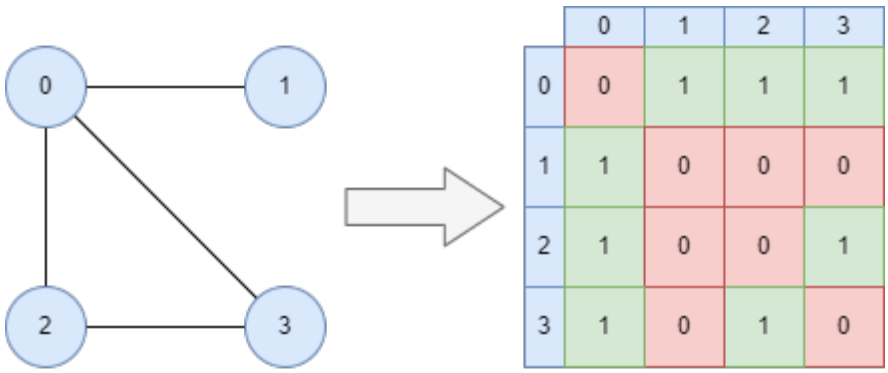

Abbildung 2.1: Darstellung eines Node-Link Diagramms und dessen Repräsenation als Adjazenzmatrix Diagramm.

Die Stärke von Adjazenzmatrizen liegt in ihrer kompakten Darstellung großer und dichter Graphen. In solchen Fällen kann ein Node-Link Diagramm, das jeden Knoten und jede Kante visuell darstellt, schnell überladen und unlesbar werden. Im Gegensatz dazu nutzt eine Adjazenzmatrix den Raum optimal. Sie stellt die An- oder Abwesenheit von Kanten zwischen allen Knotenpaaren in einer Rasterstruktur dar. Diese Darstellung hat die Eigenschaft, dass sie keine Überlappungen oder Kreuzungen von Kanten erzeugt, was die Visualisierung deutlich übersichtlicher macht (vgl. [OJK19], S. 2941). Da sowohl vorhandene als auch nicht vorhandene Verbindungen in der Matrix dargestellt werden,

hängt die Größe der Matrix direkt mit der Anzahl der Knoten zusammen. Das bedeutet, dass unabhängig davon, wie dicht das Netzwerk wird, die Adjazenzmatrix eine konstante Größe beibehält (vgl. [OJK19], S. 2945). Diese Eigenschaft macht sie besonders für große Netzwerke geeignet, bei denen ein Node-Link Diagramm zu unübersichtlich wäre.

Die Darstellung der Verbindungen in einer Matrix vereinfacht außerdem das Erkennen von Mustern. Sie ermöglicht eine klare und unmittelbare Visualisierung der Beziehungen zwischen Knoten, wodurch Muster und Cluster leichter erkannt werden können, als dies bei anderen Visualisierungstechniken der Fall ist (vgl. [OJK19], S. 2949). Diese Fähigkeit ist in der Datenanalyse besonders wertvoll, da sie eine effiziente und benutzerfreundliche Methode zur Identifizierung von Gruppenstrukturen in komplexen Netzwerken darstellt.

Die Darstellung in Matrixform hilft nicht nur bei der Visualisierung dichter Netzwerke, sondern erhöht auch die Recheneffizienz der Visualisierung. Bei einem Node-Link Diagramm wird viel Rechenleistung benötigt, um die optimale Positionierung der Knoten zu ermitteln. Das Ziel bei der Positionierung der Knoten in Node-Link Diagrammen ist, ein Diagramm zu erzeugen, dass eine hohe Lesbarkeit hat. Um das zu erreichen, wird versucht, ein Layout zu erzeugen, bei dem die Kanten zwischen den Knoten so wenig wie möglich überlappen. Die Berechnung der optimalen Positionierung ist bei großen Netzwerken sehr rechenintensiv. Die Suche nach der optimalen Positionierung entfällt bei Adjazenzmatrizen gänzlich, da die Positionierung durch die Rasterstruktur vorgegeben wird und hier keine Überlappungen entstehen können (vgl. [OJK19], S. 2947).

Trotz aller Vorteile von Adjazenzmatrizen gibt es auch einen entscheidenden Nachteil. Adjazenzmatrizen sind für Personen, die nicht mit der Matrixdarstellung von Graphen vertraut sind, weniger zugänglich als Node-Link Diagramme. Das intuitive Verständnis, welches durch die räumliche Anordnung von Knoten und Kanten ermöglicht wird, fehlt hier, was es schwieriger macht, auf den ersten Blick ein intuitives Verständnis für die Gesamtstruktur des Graphen zu entwickeln (vgl. [OJK19], S. 2950).

Dennoch bieten Adjazenzmatrizen für bestimmte Aufgaben wie den Vergleich struktureller Ähnlichkeiten zwischen zwei Graphen oder die Durchführung von detaillierten quantitativen Analysen eine unübertroffene Klarheit und Effizienz. Um die Lesbarkeit und analytische Stärke von Adjazenzmatrizen zu verbessern, können verschiedene Techniken angewandt werden, wie etwa das Umordnen der Matrix, um ähnliche Entitäten nebeneinander zu platzieren, oder die Nutzung von Farbverläufen zur Darstellung unterschiedlicher Beziehungsintensitäten. Diese Methoden helfen, die Kluft zwischen struktureller Präsentation und intuitivem Verstehen zu schließen.

Hierarchische Layouts

Hierarchische Layouts stellen eine der klassischen Techniken zur Visualisierung von Netzwerken dar, die eine klare Rangordnung oder eine natürliche Hierarchie aufweisen. Diese Layouts ordnen die Knoten eines Graphen in verschiedenen Ebenen an, wobei jede Ebene eine spezifische Hierarchiestufe repräsentiert. Knoten auf derselben Ebene sind in der Regel horizontal oder vertikal ausgerichtet, während die Kanten zwischen den Knoten die Beziehungen zwischen den unterschiedlichen Hierarchiestufen verdeutlichen (vgl. [Che+19], S. 630).

Der Hauptvorteil hierarchischer Layouts liegt in ihrer Fähigkeit, die übergeordnete Struktur eines Netzwerks auf eine Weise darzustellen, die für den Benutzer leicht verständlich ist. Diese Layouts ermöglichen es, komplexe Strukturen in einer klaren, baumartigen Darstellung abzubilden, was insbesondere bei der Visualisierung von Organisationen, Entscheidungsprozessen, Klassifikationssystemen oder genealogischen Bäumen von Vorteil ist (vgl. [Che+19], S. 629). Durch die visuelle Trennung der Ebenen können Betrachter leicht erkennen, wie sich Informationen oder Autoritäten von einer Hierarchiestufe zur nächsten übertragen.

Ein weiterer Vorteil hierarchischer Layouts ist ihre Effizienz in der Darstellung großer Netzwerke. Indem Knoten in klar definierten Ebenen angeordnet werden, kann visuelles Durcheinander minimiert und eine geordnete Darstellung auch bei umfangreichen Datensätzen gewährleistet werden (vgl. [Che+19], S. 630). Dies macht hierarchische Layouts zu einer bevorzugten Wahl für Netzwerke mit zahlreichen Knoten und Kanten, wo andere Visualisierungstechniken möglicherweise an ihre Grenzen stoßen (vgl. [Che+19], S. 629).

Trotz ihrer Stärken haben hierarchische Layouts auch einige Einschränkungen. Sie sind weniger geeignet für Netzwerke ohne eine eindeutige Hierarchie oder für solche, bei denen zahlreiche Querverbindungen zwischen den Knoten bestehen (vgl. [Che+19], S. 630). In solchen Fällen kann die Darstellung unübersichtlich werden, da die Querverbindungen die klare Struktur des Layouts stören können. Zudem kann es bei sehr komplexen Netzwerken mit vielen Hierarchieebenen schwierig sein, alle Knoten auf begrenztem Raum anzuordnen, ohne dass die Darstellung überladen wirkt.

Ein weiteres Potenzial der hierarchischen Layouts liegt in ihrer Kombination mit interaktiven Visualisierungstechniken. Moderne Tools ermöglichen es dem Benutzer, Teile der Hierarchie einzuklappen oder detaillierter zu untersuchen, was die Benutzerfreundlichkeit und das Verständnis weiter verbessert (vgl. [Che+19], S. 631). Dies ist besonders nützlich, wenn nur bestimmte Teile eines Netzwerks für die aktuelle Analyse relevant sind, während andere Bereiche ausgeblendet werden können, um die Übersichtlichkeit zu wahren.

Zusammenfassend bieten hierarchische Layouts eine robuste Methode zur Visualisierung von Netzwerken, die auf Hierarchien beruhen. Sie eignen sich hervorragend für Szenarien, in denen die klare Darstellung von Über- und Unterordnungen im Vordergrund steht und bieten zugleich die Möglichkeit, komplexe Netzwerke in einer geordneten und verständlichen Weise zu präsentieren (vgl. [Che+19], S. 630).

2.3.2 Vereinfachungs- und Interaktionstechniken

Im vorherigen Kapitel wurden die verschiedenen Diagrammarten, für die Visualisierung von Knowledge Graphen eingehend beleuchtet. Die Auswahl der richtigen Diagrammart ist entscheidend, um komplexe Zusammenhänge verständlich und übersichtlich darzustellen. Doch selbst mit der optimalen Diagrammart kann die Darstellung umfangreicher Knowledge Graphen schnell an Grenzen stoßen. Vor allem, wenn es darum geht, die Fülle an Informationen für den Betrachter zugänglich zu machen. In diesem Kapitel werden daher Techniken vorgestellt, die eingesetzt werden können, um diese Darstellungen zu vereinfachen und für den Betrachter zugänglicher zu machen.

Node Clustering

Eine der effektivsten Methoden zur Vereinfachung komplexer Netzwerke ist das Clustering von Knoten. Beim Node Clustering werden Knoten, die ähnliche Eigenschaften teilen oder eng miteinander verbunden sind, zu sogenannten Clustern zusammengefasst. Diese Technik zielt darauf ab, das visuelle Durcheinander (enlg. visual clutter) eines Graphen zu reduzieren und das Verständnis der Netzwerkstruktur zu erleichtern. Indem verwandte Knoten zu Clustern gruppiert werden, können Benutzer Muster und Beziehungen innerhalb des Netzwerks schneller und klarer erkennen (vgl. [Che+19], S. 634).

Die Erkennung von Clustern in Netzwerken stellt jedoch eine signifikante Herausforderung dar, da es keine universelle Definition dafür gibt, was genau einen Cluster ausmacht. Allgemein wird ein Cluster als eine Gruppe von Knoten verstanden, die untereinander stärker verbunden sind als mit Knoten außerhalb des Clusters (vgl. [JD88], S. 1). Die Schwierigkeit besteht jedoch darin, geeignete Kriterien für die Clusterbildung zu definieren und Algorithmen zu wählen, die diese Kriterien in unterschiedlichen Netzwerkstrukturen effektiv umsetzen können. Die Struktur des Netzwerks, die Dichte der Verbindungen und die Art der Daten spielen dabei eine entscheidende Rolle. Ein weiterer komplexer Aspekt ist die Auswahl des richtigen Algorithmus, da verschiedene Methoden unterschiedliche Arten von Clustern erkennen können, was die Vergleichbarkeit der Ergebnisse erschwert.

Eine weit verbreitete Clustering-Methode, die häufig in der Netzwerkvisualisierung angewendet wird, ist der k-Means-Algorithmus. Er hat seinen Namen von der Eigenschaft, dass er die Knoten eines Netzwerks in eine vordefinierte Anzahl von k Clustern einteilt. Der Algorithmus wählt zunächst zufällig k Zentroiden aus, die als Mittelpunkte der Cluster fungieren. Die Knoten werden dann den nächstgelegenen Zentroiden zugewiesen, wodurch vorläufige Cluster entstehen. Der Algorithmus iteriert diesen Prozess, indem er die Position der Zentroiden basierend auf den zugewiesenen Knoten aktualisiert und die Knoten neu gruppiert, bis die Cluster stabil sind und sich die Zuordnungen nicht mehr signifikant ändern (vgl. [XW05], S. 651 f.).

k-Means ist besonders effektiv bei der Aufteilung großer Netzwerke in eine überschaubare Anzahl von Clustern, was die Analyse erleichtert. Ein wesentlicher Vorteil dieses Algorithmus liegt in seiner Einfachheit und seiner Fähigkeit, Netzwerke schnell zu verarbeiten. Allerdings setzt k-Means voraus, dass die Anzahl der Cluster vorab festgelegt wird, was problematisch sein kann, wenn diese Information nicht bekannt ist. Zudem neigt der Algorithmus dazu, Cluster in kugelförmigen Strukturen zu erkennen, was seine Anwendung in Netzwerken mit unregelmäßigen oder komplexen Clusterformen einschränken kann (vgl. [XW05], S. 652 f.).

Der Louvain-Algorithmus ist eine weitere populäre Methode zur Clustererkennung, die speziell für die Analyse großer Netzwerke entwickelt wurde. Im Gegensatz zu k-Means erfordert der Louvain-Algorithmus keine vorherige Festlegung der Anzahl der Cluster. Stattdessen basiert er auf der Optimierung der Modularität, einem Maß, das die Dichte der Verbindungen innerhalb von Clustern im Verhältnis zu den Verbindungen zwischen verschiedenen Clustern beschreibt (vgl. [Que+15], S. 28).

Der Louvain-Algorithmus arbeitet in zwei Hauptphasen. In der ersten Phase werden alle Knoten zunächst als separate Cluster betrachtet. Der Algorithmus iteriert dann über die Knoten und verschiebt jeden Knoten in den Cluster seines Nachbarn, wenn dies die Modularität erhöht. Diese Phase wird wiederholt, bis keine weitere Optimierung der Modularität möglich ist. In der zweiten Phase werden die neu gebildeten Cluster zu Superknoten zusammengefasst, wodurch ein neues Netzwerk auf einer höheren Abstraktionsebene entsteht. Dieser iterative Prozess setzt sich fort, bis die Modularität nicht weiter gesteigert werden kann. Der Louvain-Algorithmus ist besonders effektiv bei der Erkennung von Clustern in Netzwerken, die keine vorab bekannte Struktur aufweisen. Seine Fähigkeit, hierarchische Clusterstrukturen zu erkennen, macht ihn zu einem vielseitigen Werkzeug für die Netzwerkvisualisierung (vgl. [Que+15], S. 28).

Edge Bundling

Edge Bundling ist eine Visualisierungstechnik, die entwickelt wurde, um die Darstellung von Netzwerken mit hoher Kantendichte zu vereinfachen. In großen Graphen mit vielen Verbindungen zwischen den Knoten kann es schnell zu einer Überlagerung von Kanten kommen, was die Visualisierung unübersichtlich und schwer interpretierbar macht. Edge Bundling bietet eine Lösung für dieses Problem, indem es Kanten, die ähnliche Verläufe haben, in einem Bündel zusammenführt. Dadurch entsteht eine geordnete Struktur, die es ermöglicht, die wesentlichen Verbindungen innerhalb des Netzwerks leichter zu erkennen (vgl. [HW09], S. 1).

Die Technik des Edge Bundling basiert auf der Idee, dass Kanten, die denselben oder einen ähnlichen Verlauf nehmen, in ihrer Darstellung zusammengeführt werden können, ähnlich wie physische Kabel oder Straßen gebündelt werden, um den Verkehr oder Datenfluss effizienter zu organisieren. In der Visualisierung wird dies durch Algorithmen erreicht, die Kanten nach bestimmten Kriterien gruppieren und sie dann gemeinsam in einer visuellen Einheit darstellen. Diese Gruppierung kann auf verschiedenen Attributen basieren, wie z. B. der Nähe der Knoten, den Richtungen der Kanten oder anderen topologischen Merkmalen des Netzwerks (vgl. [HW09], S. 2).

Ein bedeutender Vorteil von Edge Bundling ist die erhöhte Lesbarkeit und Verständlichkeit der Netzwerkgrafiken. Durch die Bündelung von Kanten wird die Menge an Informationen, die gleichzeitig dargestellt wird, reduziert, was es den Benutzern erleichtert, Muster und Verbindungen zu identifizieren. Diese Technik ist besonders nützlich, wenn die Analyse des Netzwerks darauf abzielt, übergeordnete Strukturen zu identifizieren (vgl. [HW09], S. 2).

Jedoch bringt Edge Bundling auch Herausforderungen mit sich. Ein Nachteil dieser Technik ist, dass einzelne Kanten innerhalb eines Bündels möglicherweise weniger gut sichtbar sind, was zu einem Verlust an Detailgenauigkeit führen kann. Dieser Informationsverlust kann problematisch sein, wenn eine detaillierte Analyse der Verbindungen notwendig ist. Außerdem kann die Wahl der Bündelungskriterien die Interpretation der Daten beeinflussen, da unterschiedliche Algorithmen unterschiedliche visuelle Ergebnisse liefern können (vgl. [HW09], S. 2).

Graph Filterung

Die Graph Filterung spielt eine entscheidende Rolle sowohl bei der Vorverarbeitung der Daten (engl. preprocessing step), als auch als interaktive Technik in der Visualisierung und Analyse von Knowledge Graphen. Durch die Anwendung von Filtern können irrelevante

oder weniger wichtige Knoten und Kanten aus dem Graphen entfernt oder ausgeblendet werden, was die Komplexität reduziert und den Fokus auf relevantere Daten ermöglicht. Dies ist besonders wertvoll bei der Arbeit mit großen Netzwerken, in denen die schiere Menge an Daten die Analyse erschweren kann (vgl. [Che+19], S. 634).

Als preprocessing Schritt ermöglicht die Graph Filterung die Reduktion der Datengröße und deren Komplexität, bevor der Graph überhaupt visualisiert wird. In dieser Phase werden Filter angewendet, um nur diejenigen Knoten und Kanten im Graphen zu behalten, die für die spezifische Fragestellung oder Analyse von Bedeutung sind. Durch diese Vorselektion wird sichergestellt, dass die nachfolgende Visualisierung übersichtlicher ist und die Rechenressourcen effizienter genutzt werden. Diese Art der Filterung kann auch dazu beitragen, dass die Visualisierung schneller lädt und reaktionsfähiger ist, da weniger Daten verarbeitet werden müssen (vgl. [Che+19], S. 634).

Interaktive Filterung hingegen bietet den Nutzern die Möglichkeit, den Graphen während der Analyse dynamisch zu verändern und anzupassen. Hierbei können Filter in Echtzeit angewendet werden, um bestimmte Knoten und Kanten basierend auf ausgewählten Attributen, wie etwa Verbindungsdichte, Kategorien oder Zeitstempel, ein- oder auszublenden. Diese interaktive Methode erlaubt es den Nutzern, verschiedene Aspekte des Netzwerks zu erkunden und sich auf relevante Teilmengen des Graphen zu konzentrieren.

Durch die Kombination von Filterung beim preprocessing und der interaktiven Filterung können Benutzer den Graphen nicht nur für eine effizientere Verarbeitung vorbereiten, sondern auch während der Analyse flexibel auf die Daten zugreifen und diese an ihre Bedürfnisse anpassen. Diese duale Anwendung der Graph Filterung verbessert die Benutzererfahrung erheblich, indem sie die Komplexität reduziert und gleichzeitig tiefere Einblicke in die Struktur und die Dynamik des Netzwerks ermöglicht.

Fischaugen-Ansicht

Die Fischaugen-Ansicht, auch bekannt als Focus+Context oder Fish-eye View, ist eine Interaktionstechnik, die in der Visualisierung komplexer Graphen und Netzwerke verwendet wird. Diese Technik ermöglicht es dem Benutzer, einen bestimmten Bereich des Graphen in einer vergrößerten Darstellung zu betrachten, während der Rest des Netzwerks in einer verkleinerten und weniger detaillierten Ansicht sichtbar bleibt. Das Hauptziel der Fischaugen-Ansicht besteht darin, dem Benutzer eine detaillierte Analyse eines spezifischen Teils des Graphen zu ermöglichen, ohne dabei den Überblick über das gesamte Netzwerk zu verlieren (vgl. [SB92], S. 83).

In großen und komplexen Netzwerken kann es schwierig sein, sich auf bestimmte Knoten oder Teilbereiche zu konzentrieren, da die Menge an Informationen und Verbindungen

oft überwältigend ist. Die Fischaugen-Ansicht löst dieses Problem, indem sie den Bereich von Interesse vergrößert darstellt und gleichzeitig den Kontext des gesamten Netzwerks beibehält. Dies ermöglicht es dem Benutzer, sowohl die Details des fokussierten Bereichs zu analysieren als auch dessen Position und Beziehung im größeren Netzwerk zu verstehen (vgl. [SB92], S. 83).

Technisch gesehen basiert die Fischaugen-Ansicht auf der Idee, die Vergrößerung graduell zu reduzieren, je weiter ein Knoten oder eine Kante vom Fokuspunkt entfernt ist. Im Zentrum des Fokusbereichs wird der Graph stark vergrößert, was eine detaillierte Untersuchung ermöglicht. Je weiter man sich vom Zentrum entfernt, desto stärker wird die Verkleinerung angewendet, bis der Graph schließlich in einer weit entfernten, stark verkleinerten Ansicht dargestellt wird. Diese Technik bietet somit eine verzerrte, aber kontextuelle Darstellung des Netzwerks, die sowohl Detailtreue als auch Übersichtlichkeit vereint (vgl. [SB92], S. 83 f.).

Die Fischaugen-Ansicht ist besonders nützlich in Szenarien, in denen Benutzer eine detaillierte Analyse von spezifischen Knoten und deren unmittelbaren Verbindungen vornehmen müssen, während sie gleichzeitig die globalen Muster und Strukturen im Netzwerk im Auge behalten möchten. Ein typisches Anwendungsbeispiel wäre die Untersuchung eines zentralen Knotens in einem sozialen Netzwerk, wo der Benutzer sowohl die direkten Verbindungen dieses Knotens als auch dessen Position und Rolle im gesamten Netzwerk untersuchen möchte. Darüber hinaus kann die Fischaugen-Ansicht auch dynamisch sein, sodass Benutzer interaktiv durch das Netzwerk navigieren können. Der Fokusbereich kann einfach durch Klicken oder Ziehen über den Graphen verschoben werden, wodurch verschiedene Bereiche des Netzwerks schnell und einfach analysiert werden können (vgl. [SB92], S. 83 f.).

Zusammenfassend bietet die Fischaugen-Ansicht eine effektive Möglichkeit, in großen und komplexen Netzwerken sowohl den Überblick zu behalten als auch spezifische Details zu untersuchen. Durch die Kombination von fokussierter Vergrößerung und kontextueller Übersicht ermöglicht diese Technik eine tiefere und gleichzeitig umfassende Analyse von Knowledge Graphen, was sie zu einem wertvollen Werkzeug in der Graphvisualisierung macht.

3 Realisierung

Nachdem im theoretischen Teil die Grundlagen, Herausforderungen und geeigneten Visualisierungstechniken für Knowledge Graphen erarbeitet wurden, widmet sich dieses Kapitel der praktischen Umsetzung der gewonnenen Erkenntnisse. Ziel ist es, ein System zu entwickeln, welches dazu in der Lage ist, einen Knowledge Graphen zu visualisieren und den Anforderungen, die sich aus dem Theorieteil ableiten lassen, gerecht zu werden. Das System wird im folgenden Kapitel als Graph Visualizer bezeichnet.

3.1 Konzept

In diesem Abschnitt wird das Konzept für die Umsetzung des Graph Visualizers entwickelt. Auf Basis des theoretischen Teils werden zunächst die funktionalen und nicht-funktionalen Anforderungen an den Graph Visualizer analysiert. Anschließend werden die geeigneten Diagrammarten ausgewählt und die notwendigen Technologien festgelegt. Das erarbeitete Konzept bildet die Grundlage für die nachfolgende Implementierung und gewährleistet, dass der Graph Visualizer sowohl den technischen als auch den benutzerseitigen Anforderungen entspricht. Dabei wird besonders darauf geachtet, dass die Architektur des Systems flexibel und erweiterbar gestaltet ist, um zukünftige Anpassungen und Erweiterungen zu ermöglichen.

3.1.1 Anforderungsanalyse

In diesem Abschnitt werden die spezifischen Anforderungen an den Graph Visualizer untersucht und definiert. Dabei wird zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden. Die funktionalen Anforderungen beschreiben die konkreten Funktionen, die der Graph Visualizer erfüllen muss, wie etwa die Visualisierung von Knowledge Graphen, die Interaktivität und die Anpassungsfähigkeit der Darstellung. Die nicht-funktionalen Anforderungen hingegen befassen sich mit Aspekten wie Performance, Skalierbarkeit, Benutzerfreundlichkeit und Sicherheit. Die Anforderungen werden aus den Erkenntnissen aus Kapitel 2 abgeleitet. Diese Analyse bildet die Grundlage für die Auswahl der geeigneten Technologien und die Entwicklung eines Systems, das den Erwartungen der Benutzer entspricht und gleichzeitig den technischen Herausforderungen gerecht wird.

Funktionale Anforderungen

Für dieses Projekt wurde festgelegt, dass der Graph Visualizer die Daten des DBpedia Graphen visualisieren soll. DBpedia ist eine umfassende Wissensdatenbank, die auf strukturierten Informationen aus Wikipedia basiert. Diese Datenquelle wurde ausgewählt, da sie einen großen, frei zugänglichen Knowledge Graphen zur Verfügung stellt. Die genauen Daten sind in diesem Projekt zweitrangig. Es geht darum, mit welchen Techniken ein Knowledge Graph dargestellt werden kann und nicht darum, welche Daten dargestellt werden.

Der Graph Visualizer muss in der Lage sein, Knowledge Graphen effektiv darzustellen. Dies umfasst die Fähigkeit, Knoten und Kanten zu rendern und unterschiedliche Visualisierungstechniken anzuwenden, um die Daten strukturiert und verständlich zu präsentieren. Die Visualisierung muss es ermöglichen, komplexe Netzwerke übersichtlich darzustellen und verschiedene Perspektiven auf die Daten anzubieten.

Ein wesentliches Merkmal des Graph Visualizer ist die Interaktivität. Benutzer müssen die Möglichkeit haben, mit dem Graphen zu interagieren, indem sie Knoten und Kanten auswählen, verschieben, vergrößern oder verkleinern. Darüber hinaus soll der Benutzer durch einfache Aktionen wie Klicks oder Drag-and-drop den Graphen dynamisch erkunden und manipulieren können, um tiefere Einblicke in die Daten zu gewinnen.

Der Graph Visualizer soll es ermöglichen, die Darstellung der Graphen anzupassen. Dazu gehört die Fähigkeit, verschiedene Layouts auszuwählen sowie die Darstellung des Graphen anzupassen. Diese Anpassungen sollen helfen, die Visualisierung an den spezifischen Kontext der Analyse anzupassen und die relevanten Informationen hervorzuheben.

Die eben definierten funktionalen Anforderungen sind in der Tabelle 3.1 zusammengefasst.

Tabelle 3.1: Funktionale Anforderungen an den Graph Visualizer

	FR	**Bedeutung**
FR01	Darstellung von Daten aus DBpedia	Das System soll DBpedia als Datenquelle verwenden.
FR02	Visualisierung von Knowledge Graphen	Das System soll dazu in der Lage sein, einen Knowledge Graphen zu visualisieren.

Tabelle 3.1: Funktionale Anforderungen an den Graph Visualizer

	FR	**Bedeutung**
FR03	Interaktivität	Das System muss dem Nutzer verschiedene Möglichkeiten zu Interaktion bieten.
FR04	Anpassbarkeit der Darstellung	Der Nutzer muss die Darstellung anpassen können.

Nicht-funktionale Anforderungen

Der Graph Visualizer muss eine hohe Reaktionsgeschwindigkeit aufweisen, insbesondere bei der Verarbeitung großer Datenmengen. Ladezeiten sollten minimiert und Interaktionen wie das Verschieben, Zoomen oder Auswählen von Knoten sollten ohne Verzögerungen durchgeführt werden können. Das System sollte in der Lage sein, selbst bei Netzwerken mit Tausenden von Knoten und Kanten flüssig zu arbeiten.

Das System muss stabil und zuverlässig funktionieren. Es darf nicht zu Abstürzen kommen, selbst wenn es mit unerwarteten Eingaben oder sehr großen Datenmengen konfrontiert wird.

Der Graph Visualizer sollte auf verschiedenen Plattformen und Betriebssystemen lauffähig sein. Der Nutzer soll den Graph Visualizer von jedem gängigen System aus verwenden können.

Das System sollte leicht wartbar sein. Der Quellcode muss gut strukturiert sein, um zukünftige Anpassungen und Erweiterungen zu erleichtern. Ein modularer Aufbau des Systems ist wünschenswert, um einzelne Komponenten einfach austauschen oder aktualisieren zu können.

Die Benutzeroberfläche sollte intuitiv und leicht verständlich sein. Es sollten umfassende Hilfsmittel wie Tooltips zur Verfügung stehen.

Die eben definierten nicht-funktionalen Anforderungen sind in der Tabelle 3.2 zusammengefasst.

Tabelle 3.2: Nicht-funktionale Anforderungen an den Graph Visualizer

	NFR	Bedeutung
NFR01	Performance	Die Anwendung muss eine hohe Reaktionsgeschwindigkeit aufweisen und selbst bei Graphen mit tausenden Knoten flüssig arbeiten.
NFR02	Zuverlässigkeit	Das System muss stabil und zuverlässig funktionieren.
NFR03	Kompatibilität	Der Graph Visualizer sollte auf verschiedenen Plattformen und Betriebssystemen lauffähig sein.
NFR04	Wartbarkeit	Das System sollte leicht zu warten und erweitern sein.
NFR05	Benutzerfreundlichkeit	Die Benutzeroberfläche sollte intuitiv und leicht verständlich sein.

3.1.2 Vergleiche und Auswahl der Visualisierungstechniken

In diesem Kapitel werden die verschiedenen Visualisierungstechniken, die im theoretischen Teil der Arbeit vorgestellt wurden, analysiert und verglichen. Ziel ist es, die am besten geeigneten Techniken für die Implementierung im Graph Visualizer auszuwählen. Die Auswahl der Techniken erfolgt mit Blick auf die in Kapitel 3.1.1 definierten Anforderungen. Dieses Kapitel ist in zwei Unterkapitel unterteilt. Das erste beschäftigt sich mit der Auswahl geeigneter Diagrammarten, während das zweite die spezifischen Visualisierungstechniken betrachtet, die für die Darstellung der Knowledge Graphen eingesetzt werden sollen.

Diagrammarten

Für die Implementierung des Graph Visualizers werden sowohl Node-Link-Diagramme als auch Adjazenzmatrizen ausgewählt, da beide Diagrammarten unterschiedliche Stärken aufweisen, die den spezifischen Anforderungen des Projekts gerecht werden. Node-Link-Diagramme werden aufgrund ihrer intuitiven Darstellung und Benutzerfreundlichkeit

gewählt, da sie es ermöglichen, die Beziehungen zwischen Knoten direkt und visuell ansprechend zu repräsentieren. Diese Diagrammart unterstützt die Anforderung, ein System zu entwickeln, das auch für Nutzer ohne tiefgehende technische Kenntnisse leicht verständlich und bedienbar ist. Auf der anderen Seite bieten Adjazenzmatrizen eine effiziente Möglichkeit, dichte Netzwerke kompakt und übersichtlich darzustellen, was insbesondere für die Anforderung der Skalierbarkeit entscheidend ist. Adjazenzmatrizen eignen sich hervorragend zur Analyse komplexer und stark verbundener Netzwerke, indem sie Verbindungen in einer klar strukturierten Form präsentieren. Obwohl beide Diagrammarten für die Implementierung ausgewählt wurden, spielen sie nicht zusammen, sondern werden getrennt verwendet, um ihre jeweiligen Vorteile auszunutzen. In einem späteren Kapitel werden sie miteinander verglichen, um zu evaluieren, welche der beiden Methoden für bestimmte Anwendungsfälle besser geeignet ist und wie sie sich in der praktischen Anwendung bewähren. Hierarchische Layouts hingegen werden nicht berücksichtigt, da ein Knowledge Graph typischerweise nicht hierarchisch ist und solche Layouts daher die komplexen, nicht-linearen Strukturen eines Knowledge Graphen nicht adäquat abbilden könnten.

Visualisierungstechniken

Für die Implementierung des Graph Visualizers wurden sorgfältig verschiedene Visualisierungstechniken ausgewählt, die den Anforderungen des Projekts am besten entsprechen und gleichzeitig die Grundlage für eine spätere Erweiterung des Systems bieten. Eine der zentralen Techniken ist die Graph Filterung, die sowohl in der Vorverarbeitung als auch während des Betriebs des Systems zum Einsatz kommt. Diese Technik ermöglicht es, irrelevante oder weniger wichtige Knoten und Kanten aus dem Graphen zu entfernen oder auszublenden, wodurch die Komplexität der Darstellung reduziert und der Fokus auf relevante Daten gelenkt wird. Dies ist besonders wichtig bei großen Netzwerken, da die Analyse solcher Netzwerke oft durch die schiere Menge an Informationen erschwert wird. Die interaktive Filterung, die es den Nutzern ermöglicht, den Graphen dynamisch an ihre Bedürfnisse anzupassen, trägt entscheidend zur Benutzerfreundlichkeit und Flexibilität des Systems bei. Zudem wird ein Force-Directed Layout implementiert, das sich durch seine intuitive Darstellung und die Möglichkeit der Anpassung aller Parameter durch den Nutzer auszeichnet. Diese Technik basiert auf physikalischen Modellen, die dafür sorgen, dass verwandte Knoten näher beieinanderliegen, was es dem Nutzer erleichtert, Cluster und Beziehungen im Netzwerk zu erkennen. Das Force-Directed Layout unterstützt die Anforderung, ein hochgradig interaktives und anpassbares System zu entwickeln, das auch komplexe Netzwerke verständlich darstellt. Ein weiterer wesentlicher Bestandteil des Graph Visualizers ist die Integration des Louvain-Algorithmus zur Clustererkennung. Dieser Algorithmus ist besonders für große Netzwerke geeignet und ermöglicht es, Cluster

ohne vorherige Festlegung ihrer Anzahl zu identifizieren. Dies ist von besonderer Bedeutung für die Visualisierung von Knowledge Graphen, die in der Regel keine vorab bekannte Struktur aufweisen. Durch die Clustererkennung wird das Netzwerk in überschaubare Segmente unterteilt, was die Analyse erleichtert und die Skalierbarkeit des Systems verbessert. Andere im theoretischen Teil vorgestellte Techniken wie das Edge Bundling oder die Fischaugen-Ansicht werden aufgrund zeitlicher Einschränkungen nicht implementiert, obwohl sie ebenfalls nützlich sind. Der Fokus liegt auf den Methoden, die die Anforderungen am besten erfüllen und gleichzeitig die Grundlage für eine spätere Erweiterung des Systems bilden. Diese ausgewählten Techniken werden in späteren Kapiteln evaluiert, um ihre Effektivität und Eignung für die spezifischen Anforderungen des Graph Visualizers zu überprüfen und zu validieren.

3.1.3 Technologieauswahl

In diesem Abschnitt werden die Technologien beschrieben, die für die Implementierung des Graph Visualizers ausgewählt wurden. Die Wahl der Technologien basiert auf den Anforderungen, die in den vorherigen Kapiteln definiert wurden, und zielt darauf ab, ein robustes, flexibles und leistungsfähiges System zu schaffen, das die Visualisierung von Knowledge Graphen effektiv unterstützt.

Frontend-Technologien

Das Frontend des Graph Visualizers wird als Single Page Application (SPA) realisiert. Dafür wurde Svelte als Framework gewählt, da es eine hohe Performance bietet und einfach von Einsteigern erlernt werden kann. Svelte ermöglicht es, reaktive Webanwendungen zu erstellen, bei denen die Benutzeroberfläche auf Benutzerinteraktionen reagiert. Um das Styling der Anwendung zu unterstützen, wird TailwindCSS eingesetzt. Ergänzend dazu wird DaisyUI verwendet, um die Entwicklung zu beschleunigen. DaisyUI ist eine auf TailwindCSS basierende UI-Komponentenbibliothek, die vorgefertigte Komponenten bereitstellt.

Für die Visualisierung der Graphen kommt D3.js zum Einsatz. D3.js ist eine JavaScript-Bibliothek, die für die Erstellung dynamischer und interaktiver Datenvisualisierungen entwickelt wurde. D3.js ermöglicht es, Daten in verschiedensten Formaten zu binden und diese durch Transformation des Document Object Model (DOM) in komplexe Visualisierungen umzusetzen. Diese Flexibilität und dessen Verbreitung machen D3.js besonders geeignet für die Darstellung von Knowledge Graphen (vgl. [Bos]). Es gibt auch Alternativen zu D3.js, wie zum Beispiel Sigma.js, eine Bibliothek, die sich ebenfalls für die Darstellung

von Netzwerken eignet und sich durch eine optimierte Performance für große Netzwerke auszeichnet. Sigma.js ist besonders dann nützlich, wenn der Fokus auf der schnellen und einfachen Darstellung von Netzwerken liegt, ohne die umfassende Anpassungsfähigkeit, die D3.js bietet (vgl. [JPS]). Trotz der Vorteile von Sigma.js wurde sich für D3.js entschieden, da es weiter verbreitet ist und mehr online Ressourcen zur Verfügung stehen.

Backend-Technologien

Das Backend des Graph Visualizers wird mit ASP.NET Core entwickelt. Diese Wahl bietet mehrere Vorteile, darunter eine hohe Performance, Plattformunabhängigkeit und eine gute Unterstützung für moderne Webtechnologien. Darüber hinaus wurde ASP.NET Core aufgrund der bereits vorhandenen Kenntnisse und Erfahrungen mit diesem Framework ausgewählt. ASP.NET Core wird verwendet, um die Webseite auszuliefern. Gleichzeitig dient die Anwendung als Schnittstelle zur Verarbeitung und Bereitstellung der Knowledge Graphen-Daten. Das Backend ist dafür verantwortlich, die Daten aus der zugrunde liegenden Knowledge Graph-Datenquelle zu laden und diese über einen Application Programming Interface (API) Endpunkt zur Verfügung zu stellen, der von der Svelte-Anwendung im Frontend abgerufen wird.

Schnittstellen und Datenformate

Die Kommunikation zwischen dem Frontend und dem Backend des Graph Visualizers erfolgt über eine REST API, die vom Backend bereitgestellt wird. Das Backend, entwickelt mit ASP.NET Core, übernimmt die Aufgabe, die Daten aus einer externen Knowledge Graph-Datenquelle abzurufen. Diese Daten werden über einen SPARQL Protocol And RDF Query Language (SPARQL) Endpunkt im RDF-Format bezogen, einem Standardformat für die Darstellung von Metadaten in semantischen Netzwerken.

Resource Description Framework (RDF) ist ein Datenmodell, das zur Darstellung von Informationen im Web verwendet wird. Es stellt Informationen in Form von Triples dar, die aus Subjekt, Prädikat und Objekt bestehen. Ein Triple beschreibt eine Relation zwischen zwei Entitäten, wobei das Subjekt für die Entität, das Prädikat für die Beziehung und das Objekt für den Wert oder eine weitere Entität steht. RDF wird häufig in semantischen Netzwerken und Knowledge Graphen verwendet, um komplexe Beziehungen zwischen Daten zu modellieren und miteinander zu verknüpfen (vgl. [w3o]).

Die ASP.NET Core Anwendung verarbeitet diese RDF-Daten und bringt sie in eine für das Frontend verständliche Form. Die Svelte-Anwendung im Frontend erhält die aufbereiteten

Daten dann im JavaScript Object Notation (JSON)-Format. JSON bietet die nötige Flexibilität und Einfachheit, um komplexe Datenstrukturen effizient darzustellen. Diese Struktur ermöglicht eine reibungslose Integration der Daten in die interaktiven Visualisierungen des Graph Visualizers. Durch diese Architektur wird eine klare Trennung zwischen den Datenformaten im Backend und der Datenpräsentation im Frontend gewährleistet, was die Wartbarkeit und Erweiterbarkeit des Systems verbessert.

3.1.4 Zusammenfassung der Systemarchitektur

Die Systemarchitektur des Graph Visualizers ist in die Client-Seite und die Server-Seite unterteilt. Auf der Client-Seite läuft das Frontend als Single Page Application im Browser des Users. Als JavaScript-Framework wird Svelte eingesetzt. Es ist für die Benutzeroberfläche und die Präsentation der Visualisierungen verantwortlich, wobei D3.js zur Darstellung der Knowledge Graphen sowie TailwindCSS und DaisyUI für das Styling eingesetzt werden. Das Frontend kommuniziert über API-Anfragen mit dem Backend.

Das Backend, entwickelt mit ASP.NET Core, übernimmt die Geschäftslogik und den Datenzugriff. Es empfängt API-Anfragen vom Frontend, verarbeitet die Daten von DBpedia, die über einen SPARQL-Endpunkt im RDF-Format abgerufen werden, und wandelt sie in JSON um, das für das Frontend leicht verarbeitbar ist. Das Backend fungiert somit als zentrale Schnittstelle zwischen der Datenquelle und der Präsentationsebene.

Diese Struktur ist zur Verdeutlichung in der Abbildung 3.1 dargestellt.

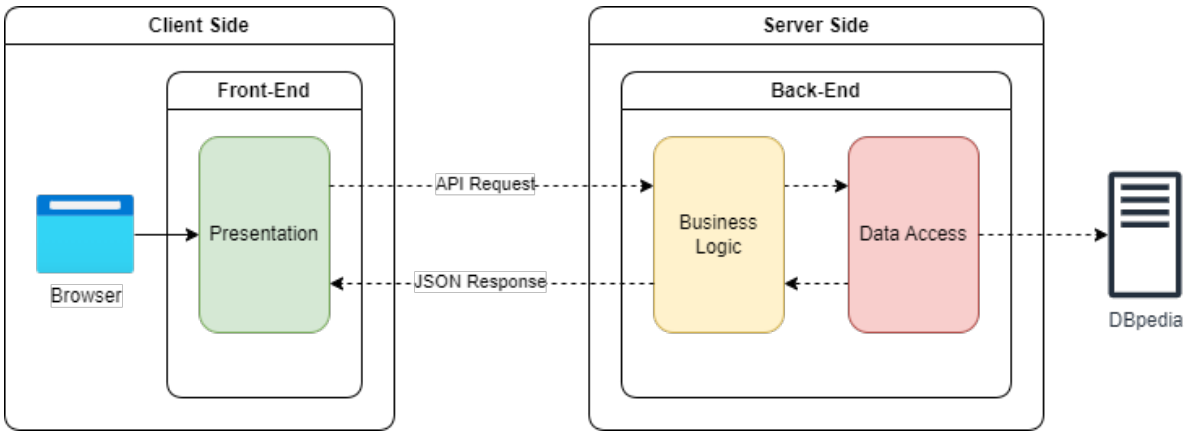

Abbildung 3.1: Geplante Architektur des Graph Visualizers.

3.2 Implementierung

In diesem Kapitel wird die praktische Umsetzung des Graph Visualizers beschrieben. Aufbauend auf den in den vorherigen Kapiteln erarbeiteten Anforderungen, liegt der Schwerpunkt dieses Kapitels auf der konkreten Implementierung der Features. Das Kapitel ist nach den zentralen Funktionalitäten der Anwendung strukturiert. Innerhalb dieser Kapitel wird dann jeweils auf die Implementierung im Front- und Backend eingegangen, je nach Relevanz. Ziel ist es, die Techniken und Ansätze zu erläutern, die verwendet wurden, um die Anwendung mit all ihren Anforderungen zu realisieren.

3.2.1 Initialisierung der Anwendung

Die Initialisierung der Anwendung begann mit der Einrichtung der beiden zentralen Komponenten. Der Svelte-Anwendung für das Frontend und der ASP.NET Core-Anwendung für das Backend.

Zunächst wurde die Svelte-Anwendung erstellt, die als Frontend des Graph Visualizers dient. Hierfür wurde ein neues Svelte-Projekt mit dem Svelte-CLI-Tool initialisiert. Diese Initialisierung legte die grundlegende Verzeichnisstruktur und die erforderlichen Konfigurationsdateien fest, die für die Entwicklung einer SPA notwendig sind. Im Anschluss daran wurden die benötigten Abhängigkeiten installiert. Dazu gehörten TailwindCSS für das Styling und DaisyUI als ergänzende UI-Komponentenbibliothek. Zudem wurde D3.js integriert, um die Visualisierung der Knowledge Graphen zu ermöglichen. Diese Bibliotheken wurden hinzugefügt, um die Entwicklung einer interaktiven und benutzerfreundlichen Oberfläche zu unterstützen.

Parallel dazu wurde eine ASP.NET Core-Anwendung initialisiert, die das Backend des Systems bildet. Diese Anwendung wurde mit dem .NET CLI-Tool erstellt, das eine Standard-Verzeichnisstruktur und grundlegende Konfigurationsdateien bereitstellt. Das Backend ist dafür verantwortlich, die Svelte-Anwendung auszuliefern und API-Endpunkte für die Datenverarbeitung bereitzustellen.

Um die Integration zwischen Frontend und Backend zu realisieren, wurde Vite so konfiguriert, dass es die Svelte-Anwendung kompiliert und die generierten Dateien, in das wwwroot-Verzeichnis der ASP.NET Core-Anwendung platziert. Diese Dateien werden dann von der ASP.NET Core-Anwendung als statische Dateien ausgeliefert.

Um die Auslieferung der statischen Dateien zu ermöglichen, wurden in der Startup.cs-Datei der ASP.NET Core-Anwendung die Middleware-Komponenten *app.UseDefaultFiles()* und *app.UseStaticFiles()* konfiguriert. Diese Middleware sorgt dafür, dass Anfragen an den

Server automatisch die entsprechenden Dateien aus dem wwwroot-Verzeichnis laden und an den Client ausliefern.

Nachdem die Konfiguration abgeschlossen war, wurde die ASP.NET Core-Anwendung gestartet, um sicherzustellen, dass die Svelte-Anwendung korrekt ausgeliefert wird. Bei einem Zugriff auf den Server durch den Browser wurde die Anwendung wie erwartet geladen, was den erfolgreichen Abschluss der Initialisierung bestätigte. Diese Schritte legten die Grundlage für die weitere Implementierung der spezifischen Features des Graph Visualizers.

3.2.2 Implementierung der Startseite

Die Implementierung der Startseite des Graph Visualizers legt besonderen Wert auf eine minimalistische und benutzerfreundliche Gestaltung, um den Nutzer sofort auf den Hauptzweck der Anwendung zu fokussieren. Dies entspricht der in den nicht-funktionalen Anforderungen (NFR05) festgelegten Priorität auf Benutzerfreundlichkeit.

Als erstes wurde das Layout der Startseite so entworfen, dass die Benutzer nur mit den notwendigsten Elementen interagieren müssen. Im Zentrum der Seite befindet sich ein auffälliges Suchfeld, das zur Eingabe von Suchbegriffen dient. Diese Reduktion auf das Wesentliche soll sicherstellen, dass die Benutzer direkt zur Hauptfunktion der Anwendung geführt werden, nämlich der Visualisierung von Knowledge Graphen. Die Startseite ist in Abbildung 3.2 dargestellt.

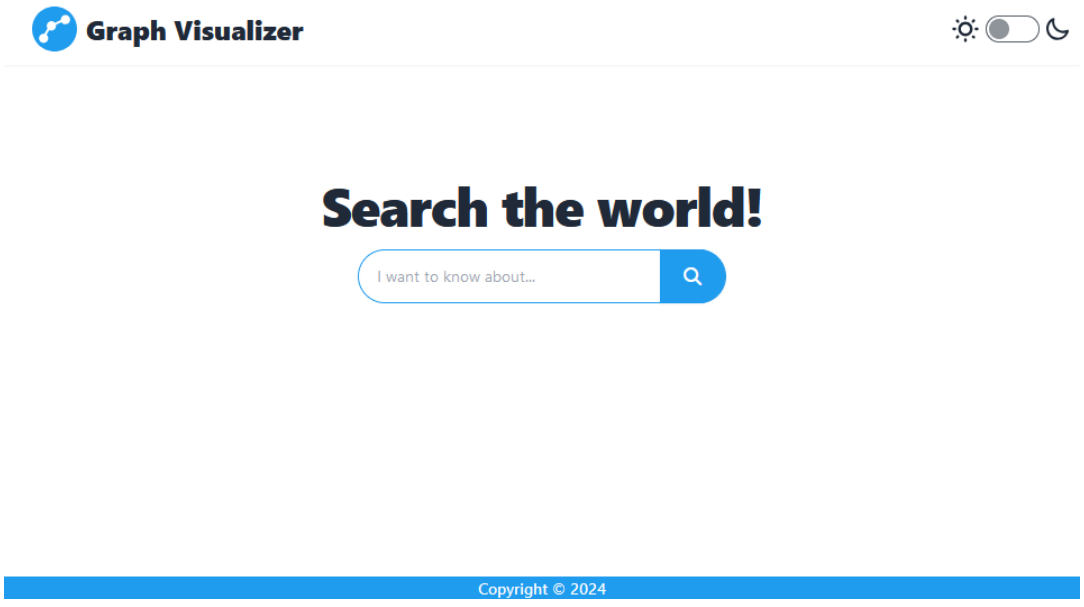

Abbildung 3.2: Startseite des Graph Visualizers.

Ein Feature der Startseite und der gesamten Anwendung ist die Unterstützung eines Dark Modes. Diese Funktion wurde implementiert, um den Komfort für die Benutzer zu erhöhen, insbesondere bei längeren Nutzungszeiten oder in Umgebungen mit schwacher Beleuchtung. Durch einen Schalter in der oberen rechten Ecke der Seite kann der Benutzer zwischen dem hellen und dunklen Modus wechseln. Dies sorgt für eine angenehmere visuelle Erfahrung und unterstützt das Ziel einer anpassbaren und benutzerfreundlichen Oberfläche. Das Feature ist in Abbildung 3.3 zu sehen.

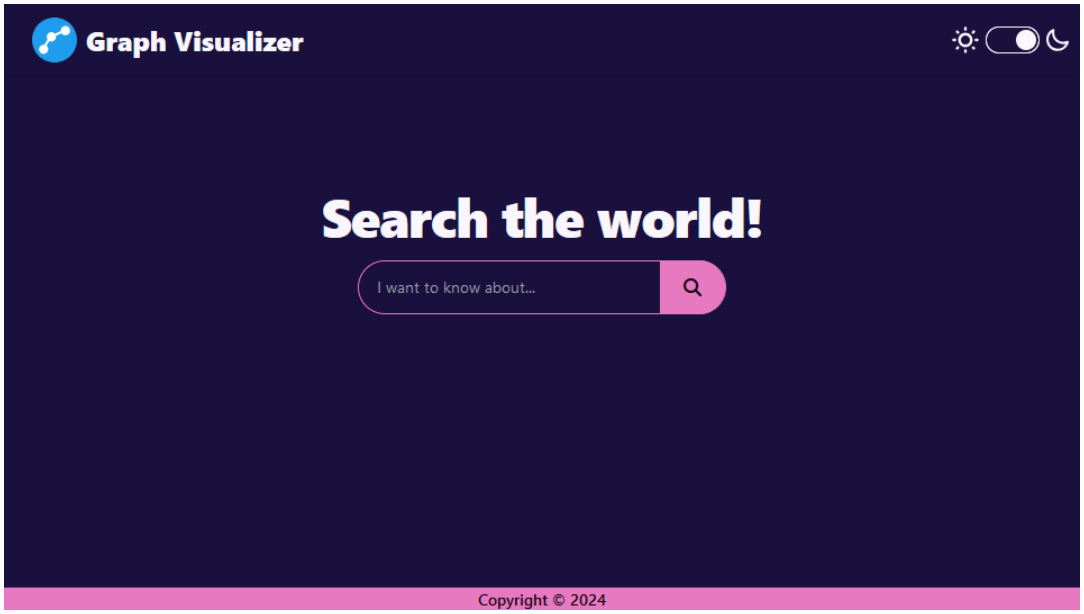

Abbildung 3.3: Startseite des Graph Visualizers im Dark Mode.

Wenn der Benutzer einen Suchbegriff in das Eingabefeld eingibt und die Suche ausführt, wird er automatisch auf die Suchseite weitergeleitet. Die URL der Suchseite enthält den Suchbegriff als Parameter (z.B. */#/search?query=Space%20Center*). Bereits beim Aufbau der Suchseite wird im Hintergrund eine Anfrage an den Search Endpunkt des SparqlControllers im ASP.NET Core Backend gestellt. Dieser Controller verwendet das SparqlRepository, das mithilfe des SparqlQueryClient aus der Bibliothek dotNetRdf.Core eine SPARQL-Abfrage an den DBpedia-Endpunkt sendet.

```
1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2
3 SELECT DISTINCT ?resource ?label WHERE {
4   ?resource rdfs:label ?label .
5   FILTER(regex(?label, "{searchTerm}", "i") && langMatches(lang(?
        label),"EN"))
6 }
7 LIMIT 100
```

Listing 3.1: SPARQL-Abfrage zur Filterung von Ressourcen anhand des Labels

In der Abfrage aus Listing 3.1 wird zunächst jede Ressource identifiziert, die ein Label besitzt. Der FILTER-Ausdruck sorgt dafür, dass nur solche Labels berücksichtigt werden, die dem eingegebenen Suchbegriff entsprechen. Zusätzlich werden die Ergebnisse auf Labels in englischer Sprache eingeschränkt, um eine Kontinuität für dieses Projekt zu erreichen. Schließlich wird die Anzahl der zurückgegebenen Ergebnisse auf 100 beschränkt, um die Performance der Abfrage zu verbessern.

Die Ergebnisse dieser Abfrage bestehen aus einer Liste von Ressourcen (URI) und ihren entsprechenden Labels, die dann in eine JSON-Struktur umgewandelt und an das Frontend übermittelt werden. Auf der Suchseite werden diese Ergebnisse als klickbare Links angezeigt, die den Benutzer weiter in die Exploration des Knowledge Graphen führen können. Diese Darstellung ist in der Abbildung 3.4 zu sehen.

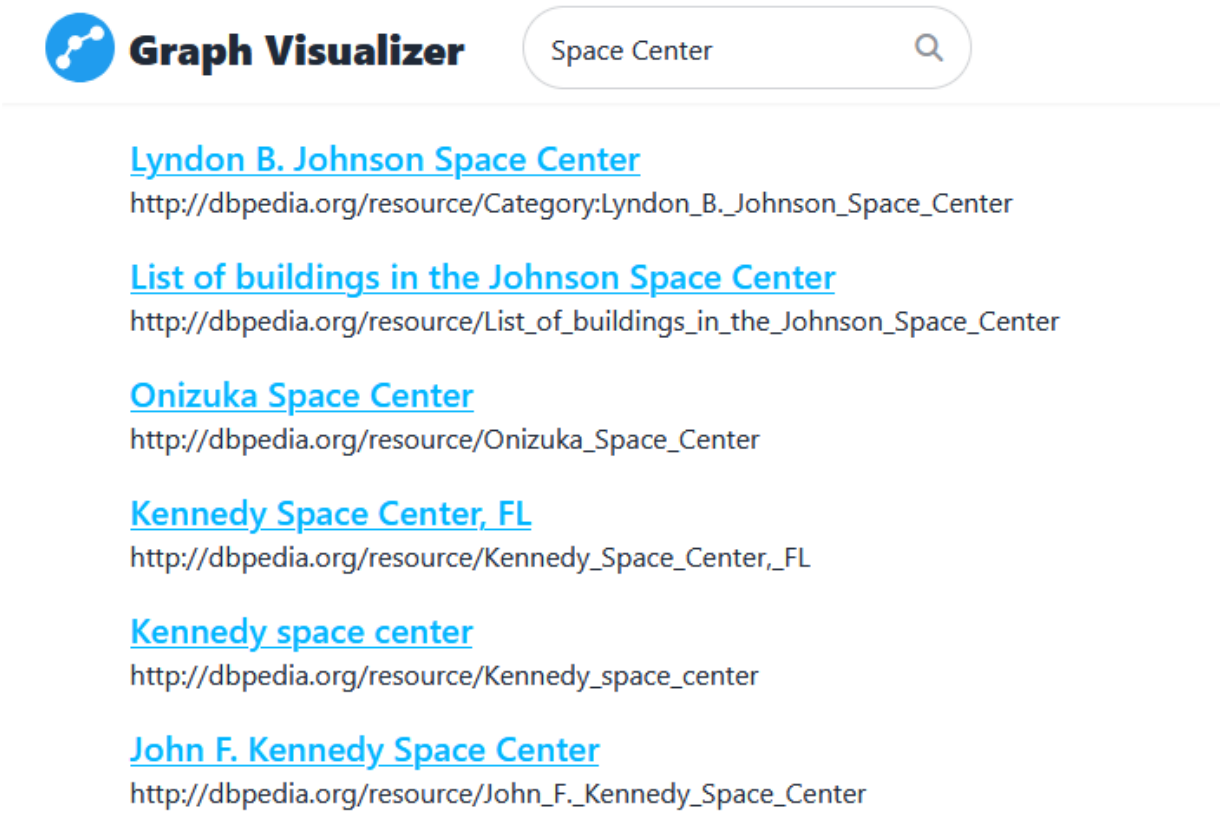

Abbildung 3.4: Suchseite des Graph Visualizers.

3.2.3 Implementierung der Graph-View

Die Graph-View ist das zentrale Element des Graph Visualizers und ermöglicht dem Benutzer die interaktive Erkundung von Knowledge Graphen. Die Seite bietet dem Benutzer verschiedene Optionen, um die Darstellung des Graphen anzupassen und die Daten zu filtern, um eine möglichst intuitive und effiziente Visualisierung zu gewährleisten.

Aufbau der Graph-View

Die Graph-View stellt den zentralen Bereich der Anwendung dar, in dem die Visualisierung des Knowledge Graphen erfolgt. Diese Ansicht ermöglicht es dem Benutzer, den Graphen interaktiv zu erkunden und anzupassen. Auf der linken Seite befindet sich ein Bedienfeld, das verschiedene Einstellungsmöglichkeiten bietet, um die Darstellung und das Verhalten des Graphen zu beeinflussen. Die Graph-View ist in Abbildung 3.5 dargestellt.

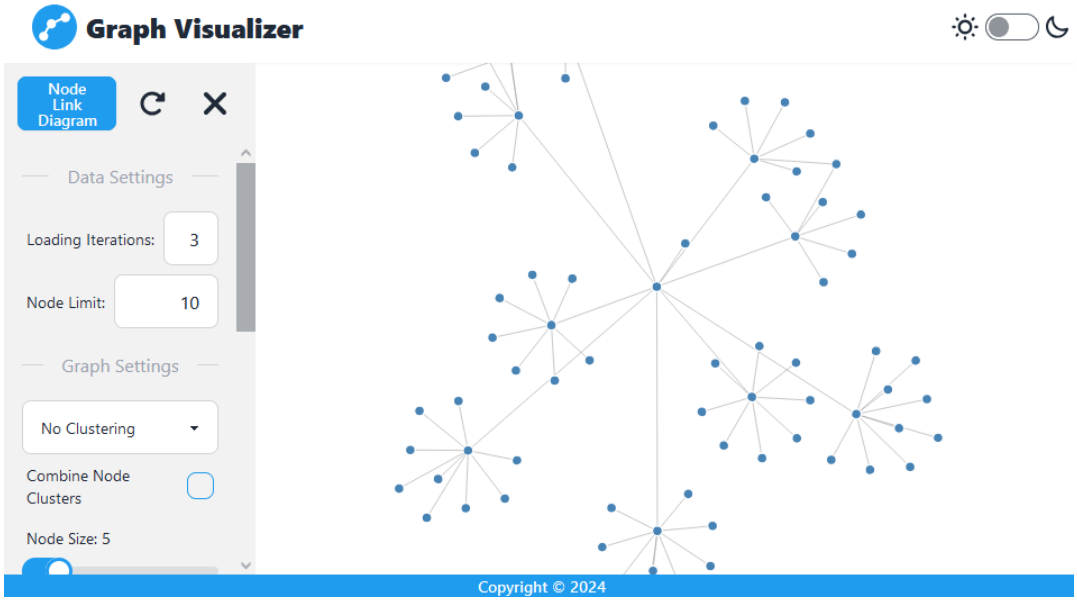

Abbildung 3.5: Graph-View des Graph Visualizers.

In dem Menu auf der linken Seite gibt es mehrere Panels mit unterschiedlichen Einstellungen. Zu den Data Settings gehört die Möglichkeit, die Anzahl der Ladeiterationen sowie das Knotenlimit festzulegen. Diese Einstellungen bestimmen, wie tief der Graph geladen wird und wie viele Knoten gleichzeitig dargestellt werden. Diese Parameter erlauben es, die Performance der Anwendung zu kontrollieren und zu testen, welche Techniken zu einer Leistungssteigerung der Anwendung führen.

Der Graph selbst wird standardmäßig als Node-Link Diagramm dargestellt, da es eine intuitive Visualierung der Beziehungen und der Knoten bietet. Auf die Erstellung des Graphen wird in Kapitel 3.2.4 noch genauer eingegangen.

Laden des Graphen

Beim Laden der Graph-View wird die URI der Ressource aus der URL extrahiert (z.B. */#/graph/?uri=http://dbpedia.org/resource/Kennedy_Space_Center,_FL*). Diese URI wird an das Backend übermittelt, das den entsprechenden Graphen von DBpedia abruft und als JSON an das Frontend zurückgibt. Das Abrufen des Graphen geschieht im Backend, da es eine bessere Performance bietet. Das parallele Abrufen von Ressourcen beschleunigt das Laden großer Graphen deutlich. Ein zentraler Bestandteil dieses Prozesses ist die *Get*-Methode des SparqlRepository. Diese Methode übernimmt die Aufgabe, den Graphen zu laden und ihn für die Anzeige vorzubereiten.

Ein wichtiger Bestandteil der Implementierung ist der Cache, der zur Verbesserung der Performance dient. Der Cache speichert zuvor geladene Graphen, um unnötige Anfragen an die externe Datenquelle zu vermeiden. Wenn eine Anfrage an das Backend gestellt wird, prüft die *Get*-Methode zunächst, ob der angeforderte Graph bereits im Cache vorhanden ist. Falls der Graph im Cache vorhanden ist, wird er direkt aus dem Cache zurückgegeben, was die Ladezeit erheblich verkürzt. Ist der Graph jedoch nicht im Cache vorhanden, wird er mittels der *LoadGraphAsync* Methode von DBpedia abgerufen und anschließend in den Cache aufgenommen.

Der Einsatz eines Caches kann die Performance der Anwendung signifikant verbessern, insbesondere wenn häufig auf dieselben Graphen zugegriffen wird. Allerdings ist der Speicherplatz des Caches begrenzt, und bei sehr großen Datenmengen könnte der Cache an seine Grenzen stoßen. In einer produktiven Umgebung könnte es daher sinnvoll sein, eine dedizierte Datenbank zur Zwischenspeicherung der Graphen zu verwenden, um eine noch robustere Lösung zu bieten.

In Listing 3.2 ist die *Get*-Methode des SparqlRepository abgebildet. Der Code enthält die Logik zum Abrufen des Graphen, entweder aus dem Cache oder von DBpedia.

```
1     public async Task<KnowledgeGraph> Get(string uri, int
          loadingDepth, int limit)
2     {
3         if (!_cache.TryGetValue(GraphCacheKey, out Dictionary<string,
              KnowledgeGraph>? knowledgeGraphDictionary))
4         {
5             knowledgeGraphDictionary = [];
6             _cache.Set(GraphCacheKey, knowledgeGraphDictionary);
7         }
8
9         if (!knowledgeGraphDictionary!.TryGetValue(uri, out
              KnowledgeGraph? knowledgeGraph))
10        {
```

```
11              Graph? graph = null;
12              try
13              {
14                  graph = (Graph?)await LoadGraphAsync(uri);
15              }
16              catch (Exception ex)
17              {
18                  Console.WriteLine($"Failed to load graph for URI: {
                        uri}. Exception: {ex.Message}");
19              }
20
21              if (graph == null)
22              {
23                  knowledgeGraph = new KnowledgeGraph
24                  {
25                      Nodes = new Dictionary<string, Node> { { uri, new
                            Node { Uri = uri } } }
26                  };
27              }
28              else
29              {
30                  knowledgeGraph = GraphHelper.
                        ConvertGraphToKnowledgeGraph(graph);
31              }
32
33              knowledgeGraphDictionary[uri] = knowledgeGraph;
34          }
35
36          // Limit the number of nodes returned
37          KnowledgeGraph limitedKnowledgeGraph = ApplyNodeLimit(
                knowledgeGraph, limit);
38
39          if (loadingDepth > 1)
40          {
41              await LoadSubGraphsAsync(limitedKnowledgeGraph,
                    loadingDepth - 1, limit, knowledgeGraphDictionary);
42          }
43
44          return limitedKnowledgeGraph;
45      }
```

Listing 3.2: Get-Methode des SparqlRepository zum Abrufen des Graphen mit Cache.

Erweiterung des Graphen

Zusätzlich zum primären Graphen kann die Anwendung auch Subgraphen laden. Dies ist sogar notwendig, um einen „echten“ Knowledge Graphen zu erhalten. Wenn ein Graph aus der DBpedia-Datenbank geladen wird, wird die angeforderte Ressource inklusive all ihrer direkten Links und Informationen abgerufen. Das entspricht einem zentralen Knoten und seinen Verbindungen zu anderen Ressourcen. Ein solcher Graph ist in Abbildung 3.6 dargestellt.

Abbildung 3.6: Visualierung einer Ressource von DBpedia.

Diese Darstellung ist sicherlich nicht das, was man sich unter einem großen Knowledge Graphen vorgestellt. Um diesen zu erhalten, muss man die Ressourcen, mit welchen der Graph verbunden ist, abrufen und den originalen Graphen um diese Informationen erweitern.

Die Methode *LoadSubGraphsAsync* wird verwendet, um diese weiterführenden Ressourcen zu laden und in den bestehenden Graphen zu integrieren. Diese Methode lädt die verknüpften Ressourcen, indem sie für jede verlinkte Ressource erneut die Get-Methode aufruft. Die geladenen Subgraphen werden dann in den ursprünglichen Graphen integriert. Dieser rekursive Prozess ermöglicht es dem Benutzer, die Exploration des Knowledge Graphen zu vertiefen und weitere Verbindungen und Zusammenhänge zu entdecken, ohne dabei die Übersichtlichkeit zu verlieren. Der Code der Methode ist in Listing 3.3 abgebildet.

```
1     private async Task LoadSubGraphsAsync(KnowledgeGraph
          knowledgeGraph, int remainingDepth, int limit, Dictionary<
          string, KnowledgeGraph> knowledgeGraphDictionary)
2     {
3         var nodeUris = knowledgeGraph.Nodes.Keys.ToList();
4
5         var tasks = nodeUris.Select(async uri =>
6         {
7             try
8             {
```

```
9                   var subGraph = await Get(uri, remainingDepth, limit);
10
11                  lock (knowledgeGraph.Nodes)
12                  {
13                      foreach (var subNode in subGraph.Nodes.Values)
14                      {
15                          if (knowledgeGraph.Nodes.TryGetValue(subNode.
                                Uri, out var existingNode))
16                          {
17                              GraphHelper.MergeNodes(existingNode,
                                    subNode);
18                          }
19                          else
20                          {
21                              knowledgeGraph.Nodes[subNode.Uri] =
                                    subNode;
22                          }
23                      }
24                  }
25              }
26              catch (Exception ex)
27              {
28                  Console.WriteLine($"Failed to load subgraph for URI:
                        {uri}. Exception: {ex.Message}");
29              }
30          });
31
32          await Task.WhenAll(tasks);
33      }
```

Listing 3.3: *LoadSubGraphsAsync*-Methode zum rekursiven Laden der Graph-Ressourcen

Durch diese Herangehensweise beim Abrufen des Graphen stellt die Implementierung sicher, dass komplexe Netzwerke effizient geladen werden können.

3.2.4 Implementierung der Graph-Visualisierungen

Die Implementierung der Graph-Visualisierungen im Graph Visualizer umfasst zwei wesentliche Diagrammarten. Das Node-Link Diagramm und die Adjazenzmatrix. Diese beiden Visualisierungstechniken bieten unterschiedliche Perspektiven auf die zugrunde liegenden Daten und ermöglichen eine flexible Analyse von Knowledge Graphen. Bevor jedoch die Implementierung dieser Diagramme beschrieben wird, ist es wichtig, die Struktur der vom Backend gelieferten Daten zu verstehen und wie diese Daten für die Visualisierung aufbereitet werden müssen.

Datenstruktur und Aufbereitung

Die Daten, die vom Backend an das Frontend übermittelt werden, bestehen aus einer Liste von Knoten, die als JSON formatiert sind. Jeder Knoten enthält die folgenden Attribute:

- Uri: Die eindeutige Ressource-Identifier-URI des Knotens.
- Label: Ein menschenlesbarer Name oder Titel, der den Knoten beschreibt.
- Properties: Ein Dictionary von Eigenschaften, das zusätzliche Informationen über den Knoten enthält. Die Schlüssel im Wörterbuch sind die Namen der Eigenschaften, und die Werte sind die entsprechenden Inhalte.
- Links: Ein Dictionary, das Verbindungen zu anderen Knoten beschreibt. Die Schlüssel sind die Typen der Verbindungen, und die Werte sind Listen von URIs, die auf die verlinkten Knoten verweisen.

Ein solcher Knoten im JSON Format ist in Listing 3.4 dargestellt.

```
{
    "Uri": "http://dbpedia.org/resource/Kennedy_Space_Center,_FL",
    "Label": "Kennedy␣Space␣Center",
    "Properties": {
        "Location": "Florida",
        "Operator": "NASA"
    },
    "Links": {
        "located_in": ["http://dbpedia.org/resource/Florida"],
        "operated_by": ["http://dbpedia.org/resource/NASA"]
    }
}
```

Listing 3.4: Informationen einer Node im JSON Format.

Damit diese Daten von D3.js für die Visualisierung genutzt werden können, müssen sie zunächst in ein Format umgewandelt werden, das die Bibliothek versteht. Für das Node-Link Diagramm bedeutet dies, dass die Knoten und die Kanten extrahiert und entsprechend strukturiert werden müssen. Ein Beispiel für die transformierten Daten ist in Listing 3.5 dargestellt.

```
    const nodes = [
        { id: "http://dbpedia.org/resource/Kennedy_Space_Center,_FL",
            label: "Kennedy␣Space␣Center" },
        { id: "http://dbpedia.org/resource/Florida", label: "Florida"
            },
```

```
4         { id: "http://dbpedia.org/resource/NASA", label: "NASA" }
5     ];
6
7     const links = [
8         { source: "http://dbpedia.org/resource/Kennedy_Space_Center,
              _FL", target: "http://dbpedia.org/resource/Florida", type
              : "located_in" },
9         { source: "http://dbpedia.org/resource/Kennedy_Space_Center,
              _FL", target: "http://dbpedia.org/resource/NASA", type: "
              operated_by" }
10    ];
```

Listing 3.5: Informationen einer Node im JSON Format.

Diese Datenstruktur ermöglicht es D3.js, Knoten und Verbindungen darzustellen, wobei jeder Knoten durch eine eindeutige id identifiziert wird und die Kanten durch Quell- und Ziel-Eigenschaften spezifiziert sind.

Node-Link Diagramm

Das Node-Link Diagramm ist eine der zentralen Visualisierungsformen des Graph Visualizers und stellt die Beziehungen zwischen den Entitäten eines Knowledge Graphen intuitiv und visuell ansprechend dar. Die Implementierung des Node-Link Diagramms erfolgt mittels D3.js. Die Knoten werden als Kreise dargestellt, deren Positionen durch ein Force-Directed Layout bestimmt werden. Dieses Layout modelliert die Knoten als physikalische Objekte, die durch Kräfte zueinander in Beziehung stehen, ähnlich wie Federn und Ladungen in einem physikalischen Modell. Knoten, die durch Kanten verbunden sind, werden durch diese Kräfte näher zusammengeführt, während unverbundene Knoten weiter auseinanderliegen. Die Oberfläche ist in Abbildung 3.7 dargestellt.

Die Darstellung der Knoten und Kanten im Node-Link Diagramm kann durch verschiedene Einstellungen im „Graph Settings“-Panel angepasst werden. Hier können die Benutzer die Größe der Knoten, den Kollisionsradius, die Ladungsstärke und die Link-Distanz konfigurieren. Die Größe der Knoten bestimmt, wie prominent die einzelnen Entitäten im Diagramm erscheinen, während der Kollisionsradius beeinflusst, wie nah die Knoten zueinander liegen dürfen, bevor sie sich abstoßen. Die Ladungsstärke reguliert die Abstoßungskraft zwischen den Knoten, was bei der Entzerrung dicht verbundener Bereiche des Graphen hilfreich ist. Die Link-Distanz steuert die Länge der Verbindungen zwischen den Knoten, was die Gesamtstruktur des Graphen entweder dichter oder weiter gefasst erscheinen lässt. Das Ergebnis einer Änderung der Einstellungen ist in Abbildung 3.8 zu sehen.

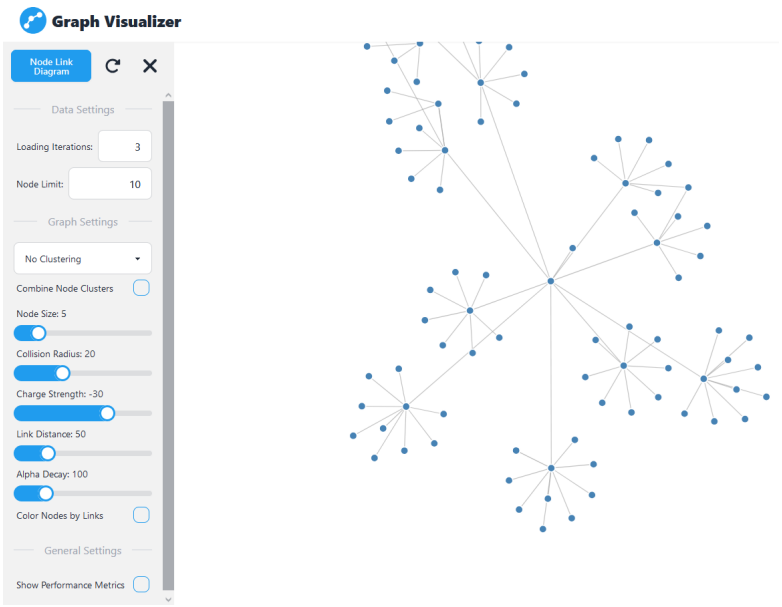

Abbildung 3.7: Oberfläche des Graph Visualizers mit Node-Link Diagramm.

Eine weitere Einstellung ist die Farbgebung der Knoten. Um die Visualisierung zugänglicher zu machen, wurde eine Option eingebaut, die es ermöglicht, die Farbe der Knoten basierend auf der Anzahl Verbindungen berechnen zu lassen. Durch die Farbgebung können Benutzer auf einen Blick erkennen, welche Knoten besonders stark vernetzt sind oder spezifische Verbindungen aufweisen. Ein Beispiel hierfür ist in Abbildung 3.9 dargestellt.

Zusätzlich ist die Einstellung Alpha Decay ein wesentlicher Parameter, der die Stabilisierung des Force-Directed Layouts beeinflusst. Dieser Wert bestimmt, wie schnell das Layout konvergiert und zur Ruhe kommt. Ein hoher Alpha Decay-Wert sorgt dafür, dass sich das Layout schneller stabilisiert, was bei der Visualisierung großer Graphen von Vorteil sein kann, da die Knotenpositionen rascher endgültig festgelegt werden. Andererseits kann die Darstellung dann unübersichtlicher sein als bei einer Simulation, die länger läuft. Diese Diskrepanz ist in den Abbildungen 3.10 und 3.11 dargestellt. In der linken Abbildung wurde der Simulation viel Zeit gegeben, während die rechte Darstellung einen hohen Alpha Decay hatte. Hier zeigt sich deutlich, dass das Zentrum der rechten Darstellung deutlich unübersichtlicher ist.

Die umfangreichen Anpassungsmöglichkeiten des Node-Link Diagramms sind darauf ausgelegt, die in den nicht-funktionalen Anforderungen definierten Ziele zu erreichen. Dazu gehört unter anderem die Forderung nach einer hohen Benutzerfreundlichkeit und Flexibilität bei der Darstellung von komplexen Knowledge Graphen. Die interaktiven Anpassungsoptionen ermöglichen es den Benutzern, das Diagramm an ihre spezifischen Bedürfnisse anzupassen, was insbesondere bei der Analyse großer und komplexer Graphen von entscheidender Bedeutung ist.

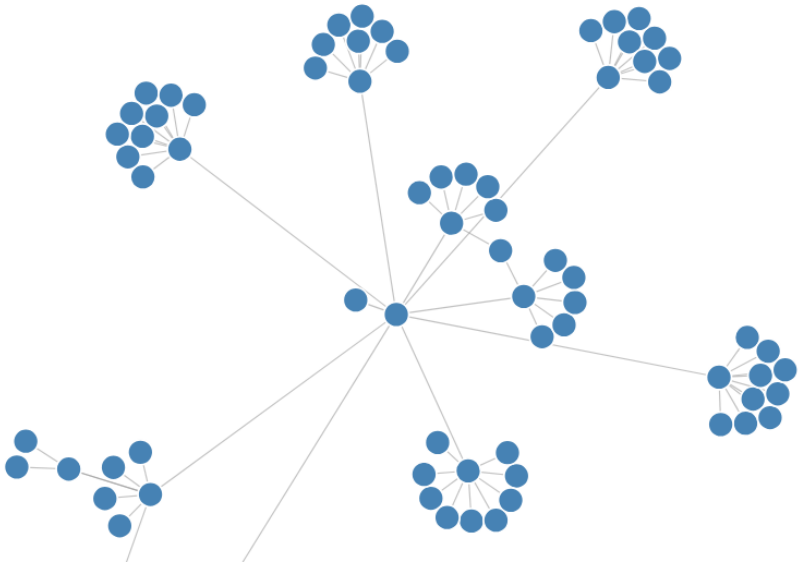

Abbildung 3.8: Node-Link Diagramm mit veränderten Einstellungen.

Adjazenzmatrix

Die Adjazenzmatrix ist eine alternative Visualisierungsmethode, die sich besonders für die Analyse von großen und komplexen Graphen eignet, bei denen die Verbindungen zwischen den Knoten im Vordergrund stehen. In einer Adjazenzmatrix werden die Knoten sowohl in den Zeilen als auch in den Spalten einer Matrix angeordnet. Jede Zelle der Matrix repräsentiert eine mögliche Verbindung zwischen den Knoten, die durch die entsprechende Zeile und Spalte definiert sind. Ist eine Verbindung vorhanden, wird die Zelle entsprechend markiert. Diese Visualisierungsmethode eignet sich besonders gut, um Cluster und dichte Netzwerke innerhalb des Graphen schnell zu identifizieren. Durch die Anordnung der Knoten in einer Matrix können Verbindungen direkt und ohne Überlappungen dargestellt werden, was bei komplexen Graphen mit vielen Knoten und Kanten eine deutlich verbesserte Lesbarkeit ermöglicht. Der Nutzer kann im Graph Visualizer über das Settings-Panel auf die Adjazenzmatrixdarstellung umschalten. Diese Darstellung ist in Abbildung 3.12 zu sehen.

Die Adjazenzmatrix im Graph Visualizer bietet nur eine Anpassungsmöglichkeit. Sie ermöglicht es, die Umrandung der Zellen ein- oder auszuschalten. Die Anzeige der Umrandung hilft dabei, die einzelnen Zellen klar voneinander zu trennen, was besonders bei dichten Graphen nützlich ist. Die Darstellung der Trennlinien kostet aber auch Performance, weshalb dem Nutzer die Möglichkeit gegeben wird, diese auszuschalten.

Die Adjazenzmatrix erfüllt mehrere der im Projekt definierten Anforderungen, insbesondere in Bezug auf die Darstellung komplexer Netzwerkstrukturen und die Benutzerfreundlichkeit. Durch die klare und strukturierte Darstellung von Verbindungen ermöglicht die

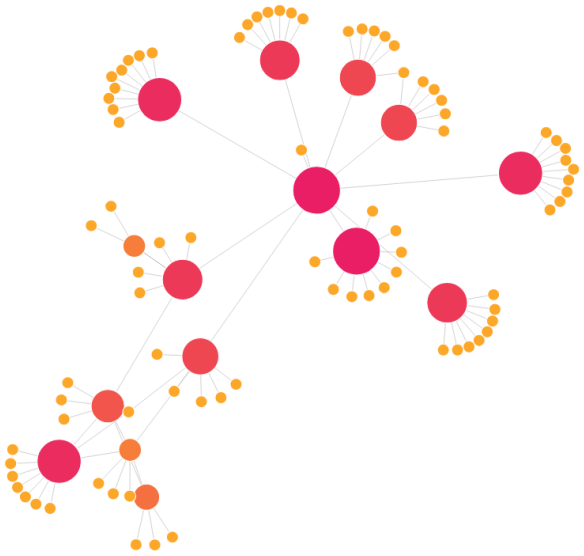

Abbildung 3.9: Node-Link Diagramm mit der Einstellung „Color Nodes by Links“.

Adjazenzmatrix eine einfache Identifikation von Clustern und Mustern im Graphen, was die Analyse und das Verständnis der zugrunde liegenden Daten erheblich erleichtert.

3.2.5 Implementierung einer Clustering-Lösung

Die Implementierung einer Clustering-Lösung innerhalb des Graph Visualizers stellt einen wichtigen Schritt dar. Sie ermöglicht es, große und komplexe Netzwerke effizienter darzustellen. Cluster werden als Gruppen von Knoten verstanden, die innerhalb eines Netzwerks eng miteinander verbunden sind. Durch die visuelle Gruppierung solcher Knoten wird die Komplexität des Graphen reduziert und dem Benutzer ermöglicht, Muster und Strukturen innerhalb des Netzwerks leichter zu erkennen.

Visualierung der Cluster im Node-Link Diagramm

In der aktuellen Version des Graph Visualizers wurde der Louvain-Algorithmus zur Clustererkennung verwendet. Dieser Algorithmus zeichnet sich dadurch aus, dass er keine vorherige Festlegung der Anzahl der zu erkennenden Cluster erfordert und speziell für die Analyse großer Netzwerke entwickelt wurde. Der Louvain-Algorithmus optimiert die sogenannte Modularität, ein Maß für die Dichte der Verbindungen innerhalb von Clustern im Verhältnis zu den Verbindungen zwischen verschiedenen Clustern.

Nach der Berechnung der Cluster werden die Knoten in der Visualisierung entsprechend eingefärbt, um die Zugehörigkeit zu einem bestimmten Cluster deutlich zu machen. Diese

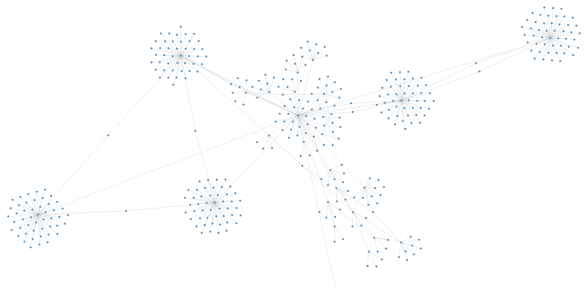

Abbildung 3.10: Node Link Diagramm niedrigem Aplha Decay.

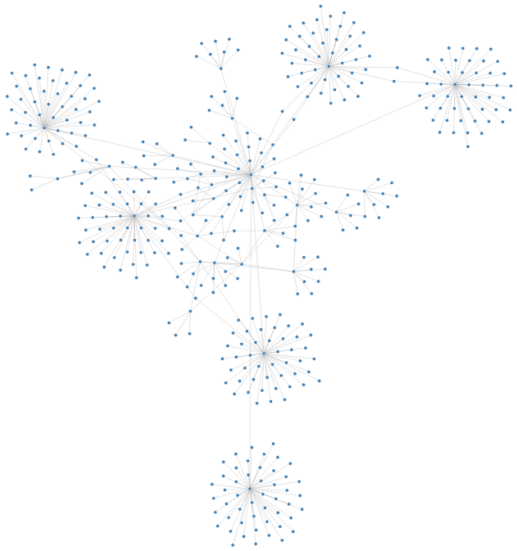

Abbildung 3.11: Node Link Diagramm hohem Aplha Decay.

farbliche Differenzierung erleichtert dem Benutzer die Identifikation der verschiedenen Gruppen innerhalb des Netzwerks. Diese Darstellung ist in Abbildung 3.13 zu sehen.

Zusammenfassung der Cluster-Nodes

Ein zusätzliches Feature ist die Möglichkeit, Cluster aus einzelnen Knoten zu großen Nodes zusammen zu fassen. Dieses Feature verbessert die Übersichtlichkeit der Visualisierung, insbesondere bei der Analyse sehr großer und komplexer Netzwerke.

Die Zusammenfassung von Cluster-Nodes erfolgt, indem alle Knoten innerhalb eines Clusters zu einem einzigen großen Knoten zusammengefasst werden. Dieser neue, aggregierte Knoten repräsentiert alle ursprünglichen Knoten des Clusters und fasst ihre Verbindungen entsprechend zusammen. In der Benutzeroberfläche kann der Benutzer diese Funktion durch Aktivierung der Option „Combine Node Clusters“ im „Graph Settings“-Panel nutzen.

Die Umsetzung dieser Funktion basiert auf der Berechnung der Cluster durch den Louvain-Algorithmus. Sobald die Cluster berechnet sind, wird für jedes Cluster ein neuer virtueller Knoten erstellt, der die Position und die Verbindungen der Knoten im Cluster übernimmt. Die Größe des neuen Knotens ist proportional zur Anzahl der Knoten im Cluster, was dem Benutzer eine schnelle visuelle Einschätzung der Clustergröße ermöglicht. Darüber hinaus werden die Verbindungen, die aus einem Knoten innerhalb des Clusters zu einem Knoten außerhalb des Clusters führen, auf den zusammengefassten Knoten übertragen. In Abbildung 3.14 ist das Feature zu sehen.

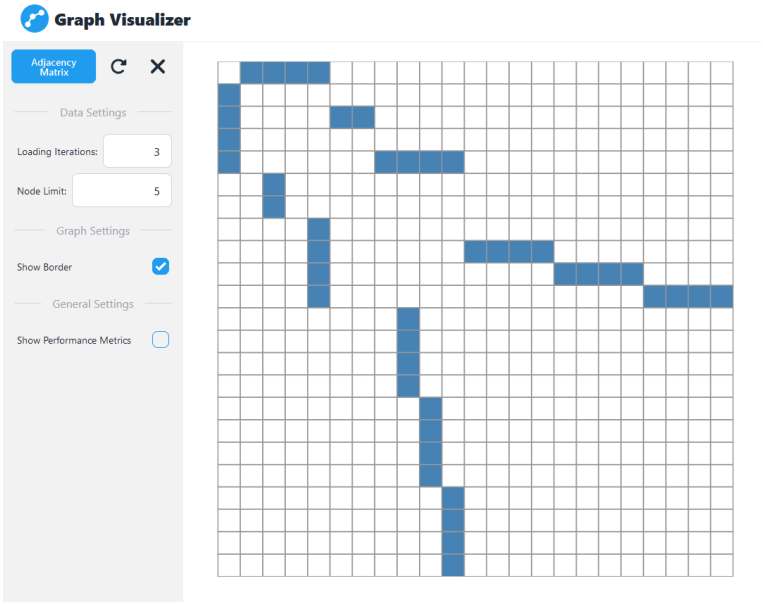

Abbildung 3.12: Adjazenzmatrixdarstellung im Graph Visualizer.

Dieses Feature wurde entwickelt, um den Anforderungen nach besserer Performance (NFR01) und Benutzerfreundlichkeit (NFR05) gerecht zu werden. Insbesondere wenn Benutzer mit sehr großen Graphen arbeiten, wird die Performance reduziert und die Übersichtlichkeit sinkt aufgrund der vielen Knoten und Verbindungen. Die Zusammenfassung reduziert die visuelle Komplexität und die Anzahl der Knoten, die dargestellt werden müssen. Dieses Feature stellt somit eine wertvolle Ergänzung zur bereits implementierten Clustererkennung dar und erweitert die Analysemöglichkeiten im Graph Visualizer.

3.2.6 Implementierung einer Detailansicht für Nodes

Die Implementierung der Detailansicht für Nodes basiert auf der Verarbeitung und Aufbereitung von RDF-Triples. Die Triples stammen von DBpedia und enthalten grundlegende Informationen in Form von Subjekt, Prädikat und Objekt. Sie bilden die Grundlage für die semantische Darstellung des Knowledge Graphen. Das Backend unterteilt die Triples in zwei Hauptkategorien, in Eigenschaften und Links zu anderen Knoten. Diese Informationen werden dann in strukturierte Daten umgewandelt, die vom Frontend visualisiert werden können.

Die Detailansicht für einzelne Nodes wird aufgerufen, wenn der Benutzer auf einen bestimmten Knoten im Node-Link Diagramm klickt. Diese Ansicht zeigt die spezifischen Informationen der ausgewählten Ressource. Dazu gehören die URI, die als eindeutige Adresse der Ressource dient, das Label, das den Namen des Knotens beschreibt, und eine Liste von Eigenschaften, die dem Knoten zugeordnet sind. Diese Detailansicht wurde ent-

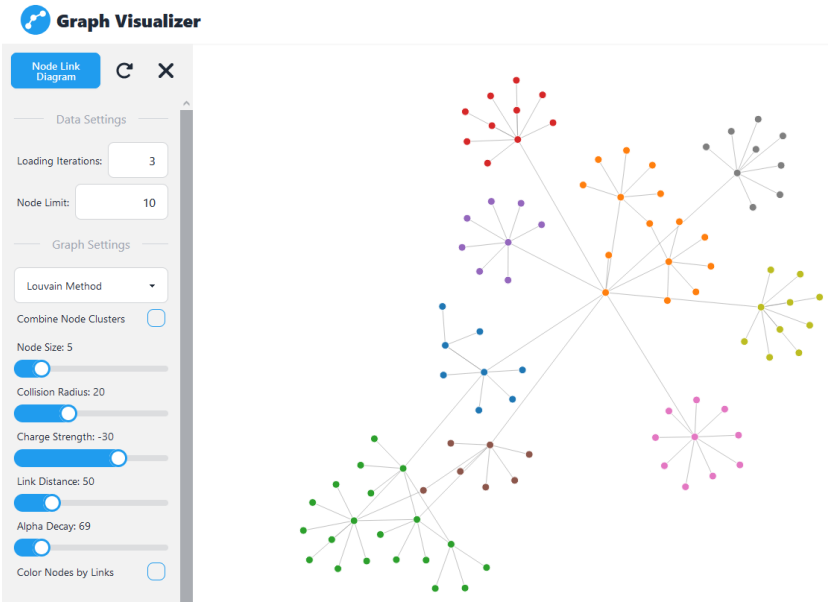

Abbildung 3.13: Node-Link Diagramm mit Clustering.

wickelt, um dem Benutzer einen schnellen und tiefgehenden Überblick über die Metadaten eines Knotens zu ermöglichen und ist in Abbildung 3.15 dargestellt.

Bei der Auswahl eines Clusters von Knoten wird die Detailansicht für Node-Cluster aktiviert. Diese Ansicht zeigt eine Übersicht über die Knoten, die innerhalb des Clusters zusammengefasst sind. Das Cluster-Label gibt an, wie das Cluster vom Louvain-Algorithmus identifiziert wurde, und die Anzahl der Knoten im Cluster wird ebenfalls angezeigt. Die Knoten innerhalb des Clusters werden mit ihrer URI und ihrem Label aufgelistet, und der Benutzer kann auf einzelne Knoten klicken, um deren spezifische Detailansicht zu öffnen. Die Detailansicht für Cluster ist in Abbildung 3.16 dargestellt.

Durch die Implementierung der Detailansicht für Nodes und Cluster wird eine der Kernanforderungen des Projekts erfüllt: die Möglichkeit, große und komplexe Graphen effizient zu analysieren. Der Benutzer kann sich durch die intuitive und interaktive Detailansicht sowohl auf einzelner Knotenebene als auch auf Clusterebene durch die Daten navigieren. Dies trägt erheblich zur Verständlichkeit und Übersichtlichkeit der Visualisierung bei. Die klare und strukturierte Darstellung von Informationen unterstützt den Benutzer bei der Analyse und ermöglicht es ihm, tiefere Einblicke in die zugrunde liegenden Daten des Knowledge Graphs zu gewinnen.

Zusammenfassend erfüllen die Detailansichten die funktionalen Anforderungen an die Bereitstellung von spezifischen und strukturierten Informationen über die Ressourcen im Knowledge Graph (FR02) sowie die nicht-funktionalen Anforderungen an Benutzerfreundlichkeit (NFR05), indem sie die Daten übersichtlich und interaktiv darstellen. Gleichzeitig werden die Anforderungen an Skalierbarkeit (NFR03) und Performance (NFR01) durch

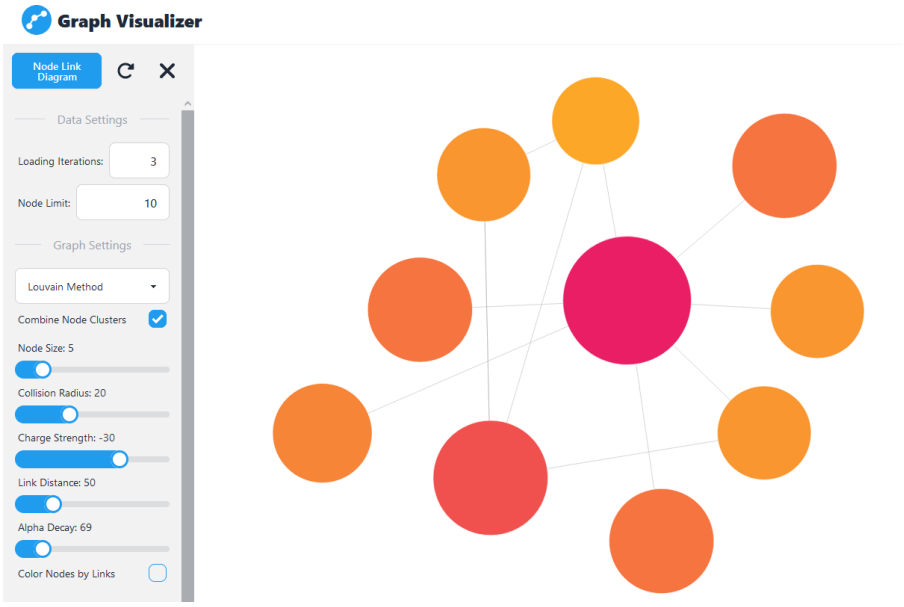

Abbildung 3.14: Zusammengefasste Cluster im Node-Link Diagramm.

die optimierte Verarbeitung und Darstellung der Knoten und Cluster im Backend gewährleistet.

3.3 Vergleich und Auswertung

In diesem Kapitel werden die im Graph Visualizer implementierten Diagrammarten und Visualisierungstechniken miteinander verglichen und ausgewertet. Ziel ist es, die Erfüllung der im Projekt definierten funktionalen und nicht-funktionalen Anforderungen zu überprüfen.

Der Vergleich der Diagrammarten Node-Link Diagramm und Adjazenzmatrix soll zeigen, welche dieser Techniken sich besser für die Darstellung großer und komplexer Knowledge Graphen eignet. Dabei wird bewertet, wie gut die Diagramme die Übersichtlichkeit bewahren und wie effizient sie mit zunehmender Größe der Netzwerke umgehen.

Die Auswertung der Visualisierungstechniken wie das Force-Directed Layout und die Clustererkennung fokussiert sich darauf, inwieweit diese Techniken den Nutzern ermöglichen, die Netzwerkstrukturen klar zu erkennen und zu analysieren. Auch hier stehen die Anforderungen an Anpassbarkeit und Performance im Vordergrund.

Insgesamt zielt dieses Kapitel darauf ab, auf Basis der praktischen Implementierung eine fundierte Bewertung der verwendeten Methoden vorzunehmen und daraus Schlüsse für künftige Entwicklungen zu ziehen.

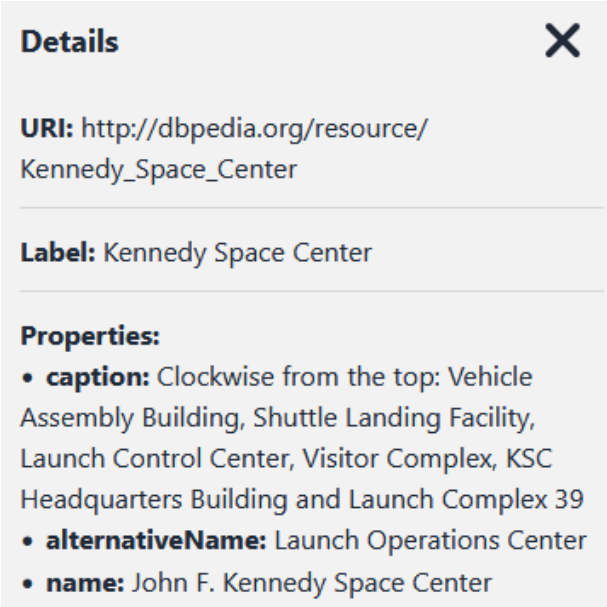

Abbildung 3.15: Detailansicht für eine Node.

3.3.1 Vergleich der Diagrammarten

Im Rahmen der Implementierung des Graph Visualizers wurden zwei verschiedene Diagrammarten zur Visualisierung von Knowledge Graphen verwendet: das Node-Link Diagramm und die Adjazenzmatrix. Beide Diagrammarten bieten unterschiedliche Ansätze zur Darstellung von Knoten und ihren Verbindungen und bringen jeweils spezifische Stärken und Schwächen mit sich. In diesem Kapitel werden die beiden Diagrammarten anhand ihrer Eignung für die jeweiligen Anforderungen verglichen und ihre Einsatzmöglichkeiten ausgewertet. Dabei werden die Erkenntnisse aus der Implementierung und den durchgeführten Tests einbezogen, um eine fundierte Bewertung vorzunehmen.

Node-Link Diagramm

Das Node-Link Diagramm bietet eine intuitive und visuell ansprechende Darstellung von Netzwerken, bei der Knoten durch Linien verbunden werden. Diese Darstellung eignet sich besonders gut, um die Beziehungen zwischen Entitäten im Graphen auf einen Blick sichtbar zu machen. Zu den großen Stärken des Node-Link Diagramms zählt seine Benutzerfreundlichkeit, da die Verbindungen zwischen den Knoten visuell leicht nachvollziehbar sind. Diese Eigenschaft trägt wesentlich zur Erfüllung der Anforderung nach einer klaren und verständlichen Darstellung (NFR05) bei.

Allerdings zeigte sich, dass das Node-Link Diagramm bei Netzwerken mit mehreren Tausend Knoten an seine Grenzen stößt. Ab einer bestimmten Komplexität kann es zu Überlappungen und Unübersichtlichkeit kommen, was die Skalierbarkeit beeinträchtigt.

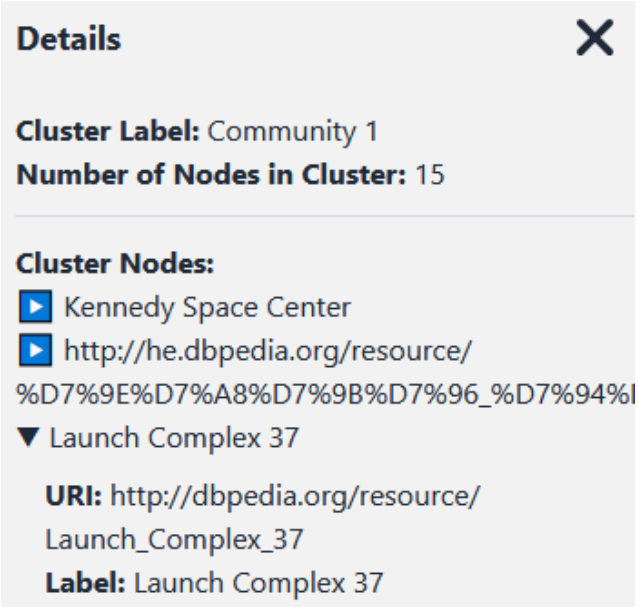

Abbildung 3.16: Detailansicht für ein Cluster.

Dies stellt ein Problem in Bezug auf die Performance und die Fähigkeit, auch große Graphen übersichtlich darzustellen, dar. Trotz dieser Einschränkungen bleibt das Node-Link Diagramm für Netzwerke mit bis zu einigen Tausend Knoten ein wertvolles Werkzeug, insbesondere aufgrund seiner visuellen Anschaulichkeit und interaktiven Natur. Viele der Nachteile können auch durch verschiedene Visualisierungstechniken ausgeglichen werden. Diese werden in dem folgenden Kapitel näher betrachtet.

Adjazenzmatrix

Die Adjazenzmatrix bietet eine strukturierte und effiziente Möglichkeit, Netzwerke darzustellen, indem Knoten sowohl in den Zeilen als auch in den Spalten einer Matrix angeordnet und Verbindungen durch markierte Zellen visualisiert werden. Diese Methode ist besonders nützlich, um Netzwerke mit vielen Verbindungen übersichtlich darzustellen, da Überlappungen vermieden werden. Dies spricht vor allem die Anforderungen an die Skalierbarkeit (NFR03) und Performance (NFR01) an, da auch bei einer großen Anzahl von Verbindungen die Struktur des Graphen klar dargestellt wird.

Eine überraschende Erkenntnis war jedoch, dass die Adjazenzmatrix schon bei Netzwerken mit einer geringeren Anzahl von Knoten als das Node-Link Diagramm an ihre Grenzen stößt. Während das Node-Link Diagramm mehrere Tausend Knoten handhaben kann, zeigt die Adjazenzmatrix bei größeren Netzwerken ab etwa 200 Knoten eine abnehmende Übersichtlichkeit. Dies könnte auf die Implementierung zurückzuführen sein und müsste mit anderen Technologien weiter untersucht werden. Dennoch zeigt die Adjazenzmatrix ihre Stärken insbesondere bei Netzwerken mit vielen Verbindungen, aber einer vergleichsweise

geringen Anzahl an Knoten (bis zu etwa 200). In einem Test, bei dem statische Daten mit vielen Verbindungen im Adjazenzmatrix Diagramm dargestellt wurden, konnte die Effizienz der Matrix bei stark verbundenen Netzwerken demonstriert werden. Eine Abbildung der Darstellung eines stark verbundenen Graphen ist in Abbildung 3.17 zu sehen.

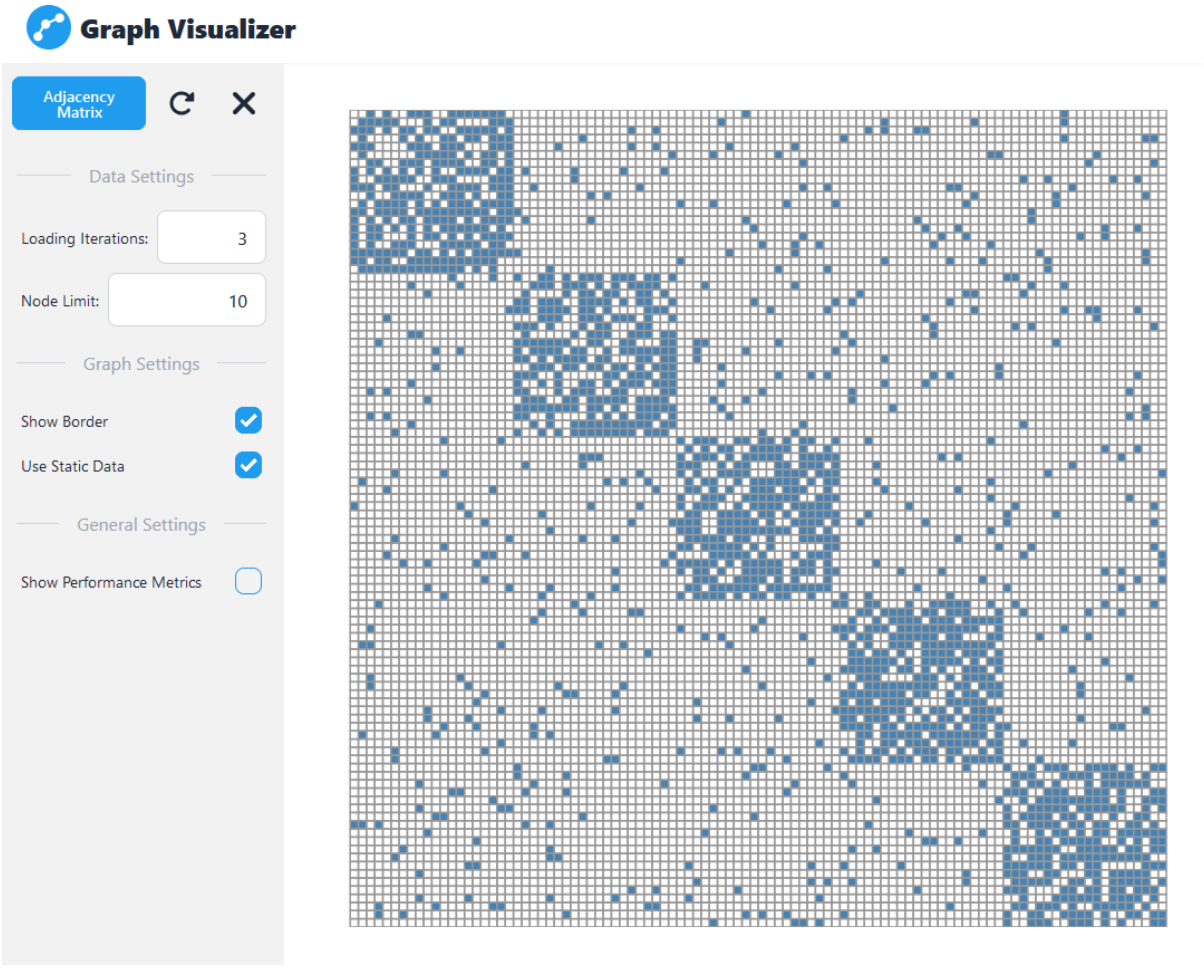

Abbildung 3.17: Beispiel für die Darstellung stark Verbundener Graphen in einem Adjazenzmatrix Diagramm.

Fazit des Vergleichs

Zusammenfassend lässt sich sagen, dass Node-Link Diagramme eine ausgezeichnete Wahl für kleinere bis mittelgroße Netzwerke mit bis zu mehreren Tausend Knoten sind. Seine Stärke liegt in der visuellen Klarheit und der einfachen Benutzbarkeit. Allerdings stößt es bei sehr großen Netzwerken mit Tausenden von Knoten an seine Grenzen. Die Adjazenzmatrix hingegen zeigt ihre Stärken bei Netzwerken mit vielen Verbindungen, aber einer relativ geringen Knotenanzahl. Sie ist weniger intuitiv, bietet jedoch bei dichten Netzwerken mit weniger Knoten eine klarere Darstellung. Ab einer bestimmten Anzahl von Knoten nimmt jedoch auch hier die Übersichtlichkeit ab, was auf die aktuelle Implementierung

zurückzuführen sein könnte und eine weitere Untersuchung mit alternativen Technologien erfordert.

3.3.2 Auswertung der Visualisierungstechniken

Die im Graph Visualizer eingesetzten Visualisierungstechniken spielen eine entscheidende Rolle, um die Benutzerfreundlichkeit und Effektivität der Anwendung zu gewährleisten. Zwei wesentliche Techniken, die im Rahmen der Implementierung verwendet wurden, sind das Force-Directed Layout sowie das Clustering. In diesem Kapitel werden beide Techniken hinsichtlich ihrer Effektivität, Benutzerfreundlichkeit und Performance bewertet.

Force-Directed Layout

Das Force-Directed Layout ist eine zentrale Technik im Node-Link Diagramm und sorgt dafür, dass Knoten durch physikalische Kräfte angeordnet werden. Diese Technik simuliert Knoten als Objekte, die durch anziehende und abstoßende Kräfte interagieren, was zu einer visuellen Darstellung führt, die verwandte Knoten näher zueinander positioniert. Dies erleichtert es dem Benutzer, die Beziehungen im Netzwerk zu erkennen, da Cluster und zentrale Knoten schnell erkennbar werden.

In der Implementierung des Graph Visualizers wurden zahlreiche Anpassungsmöglichkeiten eingebaut, die es dem Benutzer ermöglichen, das Verhalten des Force-Directed Layouts individuell zu steuern. Dazu gehören Einstellungen wie Node-Größe, Kollisionsradius, Ladungsstärke und Link-Distanz. Diese Anpassungsmöglichkeiten ermöglichen eine flexible Darstellung und tragen zur Benutzerfreundlichkeit bei, da der Benutzer die Visualisierung an die jeweiligen Anforderungen und die Größe des Netzwerks anpassen kann.

In Tests zeigte sich, dass das Force-Directed Layout für Netzwerke mit moderater Knotenanzahl sehr effektiv ist. Bei großen Netzwerken nimmt jedoch die Performance ab, was sich in längeren Ladezeiten und einer langsameren Reaktion auf Benutzerinteraktionen äußert. Dennoch bleibt das Force-Directed Layout aufgrund seiner Anpassungsfähigkeit und intuitiven Darstellung eine der geeignetsten Techniken zur Visualisierung mittelgroßer Netzwerke.

Clustererkennung und -darstellung

Die Clustererkennung ermöglicht die Darstellung von Netzwerken mit mehreren zehntausend Knoten, indem Knoten, die eng miteinander verbunden sind, zu Clustern zusammengefasst werden. Diese Cluster werden dann im Graphen als größere Knoten dargestellt,

was die Komplexität des Netzwerks erheblich verringert. Die Knoten im Cluster bleiben vorerst verborgen, und der Benutzer sieht nur die Cluster, wodurch das Netzwerk deutlich übersichtlicher wird. Die Darstellung der Cluster bietet dem Benutzer eine vereinfachte Sicht auf das Netzwerk und ermöglicht es, schnell die Verbindungen zwischen den Clustern zu erkennen. Dies erleichtert die Analyse großer Netzwerke und hilft dabei, Muster und Strukturen besser zu identifizieren. Ein großer Vorteil der Clusterdarstellung ist, dass die Benutzer je nach Bedarf die Details einzelner Cluster betrachten können, um die darin enthaltenen Knoten und Verbindungen zu erkunden.

Diese Technik erfüllt die Anforderungen an die Skalierbarkeit (NFR03), da sie die Komplexität des Netzwerks auf ein für den Benutzer handhabbares Maß reduziert, ohne wichtige Informationen zu verlieren. Netzwerke mit mehreren zehntausend Knoten lassen sich so auch bei großer Dichte der Verbindungen noch effektiv visualisieren und analysieren.

Ein wesentliches Problem tritt jedoch bei der Darstellung von Verbindungen zwischen Clustern auf. In sehr großen Netzwerken entstehen oft zahlreiche Verbindungen zwischen verschiedenen Clustern. Wenn diese Verbindungen im Force-Directed Layout dargestellt werden, kann dies zu Instabilitäten führen. Die physikalische Simulation des Force-Directed Layouts versucht, die Knoten auf Basis ihrer Verbindungen auszurichten, doch bei sehr vielen Verbindungen beginnt das Diagramm zu „zittern“. Dieses Zittern entsteht, weil die Kraftsimulation Schwierigkeiten hat, die Knoten in einer stabilen Position zu halten, was zu langen Stabilisierungszeiten führt.

Diese Instabilität beeinträchtigt die Benutzererfahrung und wirkt sich negativ auf die Performance aus. Je mehr Verbindungen zwischen den Clustern bestehen, desto stärker wird die Simulation beansprucht, was dazu führt, dass die Darstellung langsamer reagiert und die Benutzerinteraktion weniger flüssig wird. Dies ist besonders bei Netzwerken mit vielen Verbindungen zwischen den Clustern ein Problem, da die große Anzahl an Kanten die Stabilität des Layouts belastet und die Performance mindert.

Fazit

Die Clustererkennung und -darstellung im Graph Visualizer ermöglicht eine effiziente Visualisierung großer Netzwerke, indem sie die Komplexität durch die Aggregation von Knoten in Clustern reduziert. Dies verbessert die Übersichtlichkeit und Skalierbarkeit erheblich und erfüllt die entsprechenden Anforderungen des Projekts. Die Kombination dieser Technik mit dem Force-Directed Layout bietet eine visuell ansprechende und interaktive Darstellung von Netzwerken.

Allerdings stößt das Force-Directed Layout bei Netzwerken mit vielen Verbindungen zwischen Clustern an seine Grenzen. Das Zittern und die langen Stabilisierungszeiten

beeinträchtigen die Performance und die Benutzererfahrung. Eine Reduktion der Anzahl angezeigter Verbindungen oder die Anwendung einer alternativen Visualisierungstechnik könnte dieses Problem mildern und die Stabilität des Graphen bei sehr großen Netzwerken verbessern.

3.3.3 Auswertung der Interaktivität und Benutzerfreundlichkeit

Die Interaktivität und Benutzerfreundlichkeit des Graph Visualizers sind entscheidend für die effektive Analyse von Netzwerken. In diesem Abschnitt werden die wichtigsten interaktiven Features und ihre Auswirkung auf die Benutzerfreundlichkeit sowie die Erfüllung der Anforderungen bewertet.

Filterung und Detailansicht

Die Graph-Suche reduziert die Komplexität großer Netzwerke, indem irrelevante Daten ausgeblendet werden. Dies verbessert die Übersichtlichkeit und erfüllt die Anforderung an Benutzerfreundlichkeit (NFR05). Die Detailansicht zeigt bei einem Klick auf einen Knoten die zugehörigen Informationen und ermöglicht eine gezielte Analyse. Diese Funktion ist besonders nützlich im Node-Link Diagramm, fehlt jedoch noch in der Adjazenzmatrix.

Anpassungsmöglichkeiten der Diagramme

Benutzer können verschiedene Parameter wie Node-Größe, Kollisionsradius und Link-Distanz im Force-Directed Layout anpassen. Diese Flexibilität erhöht die Interaktivität und ermöglicht eine individuelle Anpassung der Visualisierung an die Netzwerkstruktur. Diese Funktion trägt erheblich zur Benutzerfreundlichkeit (NFR05) bei und erleichtert die Analyse komplexer Netzwerke.

Clusterdarstellung und Navigation

Die Clusterdarstellung ermöglicht es, große Netzwerke effizienter zu navigieren, indem stark verknüpfte Knoten zusammengefasst werden. Dies verbessert die Übersichtlichkeit und erfüllt die Anforderungen an Skalierbarkeit (NFR03). Das Ein- und Ausklappen von Clustern bietet eine flexible Möglichkeit, zwischen globaler und detaillierter Ansicht zu wechseln.

Performance und Stabilität der Interaktivität

Bei sehr großen Netzwerken zeigt das Force-Directed Layout Schwächen in der Performance, insbesondere durch das Zittern und lange Stabilisierungszeiten. Dies beeinträchtigt die Benutzererfahrung, da die Reaktionsfähigkeit der Anwendung abnimmt. Trotz dieser Einschränkungen funktioniert die Interaktivität bei mittelgroßen Netzwerken gut.

Fazit

Die Interaktivität und Benutzerfreundlichkeit des Graph Visualizers sind insgesamt gut umgesetzt. Funktionen wie die Filterung, Detailansicht und Anpassungsmöglichkeiten bieten eine flexible und intuitive Nutzung, erfüllen die Anforderungen an Skalierbarkeit (NFR03) und Benutzerfreundlichkeit (NFR05) weitgehend. Schwächen bei der Performance großer Netzwerke sollten jedoch weiter optimiert werden.

4 Zusammenfassung und Ausblick

Die vorliegende Arbeit hatte das Ziel, die geeignetste Visualisierungstechnik für Knowledge Graphen zu ermitteln und eine entsprechende Implementierung zu entwickeln. Die Visualisierung von Knowledge Graphen stellt eine besondere Herausforderung dar, da sowohl die Struktur der Daten als auch die Darstellung großer Netzwerke effizient und benutzerfreundlich sein müssen. In den vorangegangenen Kapiteln wurden verschiedene Visualisierungstechniken wie das Node-Link Diagramm und die Adjazenzmatrix analysiert und implementiert, um ihre Stärken und Schwächen im praktischen Einsatz zu bewerten. In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und wichtige Erkenntnisse reflektiert, bevor ein Ausblick auf mögliche Weiterentwicklungen und Optimierungen gegeben wird.

4.1 Zusammenfassung der Ergebnisse

In dieser Arbeit wurde das Ziel verfolgt, die beste Visualisierungstechnik für Knowledge Graphen zu ermitteln und eine darauf basierende Visualisierungslösung zu implementieren. Im Laufe der Untersuchung und Implementierung wurden zwei zentrale Visualisierungstechniken analysiert und in den Graph Visualizer integriert. Diese Techniken wurden hinsichtlich ihrer Stärken und Schwächen evaluiert, insbesondere im Kontext der Darstellung großer Netzwerke.

4.1.1 Analyse der Visualisierungstechniken

Das Node-Link Diagramm erwies sich als eine intuitive und visuell ansprechende Methode, um kleinere bis mittelgroße Netzwerke zu visualisieren. Es ermöglicht eine klare Darstellung der Beziehungen zwischen Knoten, stößt jedoch bei sehr großen Netzwerken mit mehreren Tausend Knoten an seine Grenzen. Die Performance nimmt mit der Anzahl der Verbindungen stark ab, was zu einer beeinträchtigten Stabilität und Benutzererfahrung führt.

Die Adjazenzmatrix zeigte ihre Stärken bei Netzwerken mit vielen Verbindungen und einer vergleichsweise geringen Knotenanzahl. Bei Netzwerken mit mehr als 200 Knoten wurden jedoch Schwächen in der Übersichtlichkeit erkennbar. Trotz dieser Einschränkungen konnte

die Adjazenzmatrix in stark verbundenen Netzwerken ihre Vorteile zeigen, indem sie eine klare und strukturierte Darstellung von Verbindungen ohne Überlappungen ermöglichte.

4.1.2 Erfüllung der Anforderungen

Die in Kapitel 3.1.1 festgelegten funktionalen Anforderungen wurden vollständig erfüllt. Die nicht-funktionalen Anforderungen wurden größtenteils erfüllt:

- Performance (NFR01): Bei mittelgroßen Netzwerken war die Performance zufriedenstellend. Bei sehr großen Netzwerken mit vielen Verbindungen zwischen Clustern traten jedoch Instabilitäten im Force-Directed Layout auf, die zu längeren Ladezeiten und Zittern führten.
- Zuverlässigkeit (NFR02): Die Anwendung erwies sich als stabil, abgesehen von den Performance-Einschränkungen bei großen Netzwerken. Kleinere bis mittelgroße Netzwerke wurden zuverlässig und ohne signifikante Probleme dargestellt.
- Kompatibilität (NFR03): Die Verwendung von Svelte im Frontend und ASP.NET Core im Backend ermöglichte eine problemlose Ausführung der Anwendung in verschiedenen Umgebungen und stellte eine gute Grundlage für Erweiterbarkeit und zukünftige Anpassungen sicher.
- Wartbarkeit (NFR04): Die modulare Architektur des Graph Visualizers erleichtert zukünftige Wartung und Erweiterungen. Durch klare Trennung von Backend und Frontend sowie die Verwendung etablierter Frameworks wird eine hohe Wartbarkeit gewährleistet.
- Benutzerfreundlichkeit (NFR05): Die intuitive Benutzeroberfläche mit interaktiven Features wie Filterung, Detailansicht und flexiblen Anpassungsmöglichkeiten der Visualisierung erfüllte die Anforderungen an die Benutzerfreundlichkeit. Besonders die Möglichkeit, Cluster zu erkennen und die Visualisierung dynamisch zu modifizieren, trug zur positiven Benutzererfahrung bei.

Insgesamt konnte die Implementierung des Graph Visualizers die funktionalen und nicht-funktionalen Anforderungen weitgehend erfüllen. Besonders die Skalierbarkeit und Benutzerfreundlichkeit wurden durch die Clustererkennung und die flexiblen Visualisierungstechniken erfolgreich umgesetzt. Die Performance bei sehr großen Netzwerken mit vielen Verbindungen bleibt jedoch ein Bereich, der weiter optimiert werden muss.

4.2 Erkenntnisse aus der Arbeit

Die Arbeit hat wertvolle Einsichten zur Visualisierung von Knowledge Graphen geliefert und zeigte, welche Techniken besonders gut geeignet sind. Eine überraschende Erkenntnis war, wie viele Knoten das Node-Link Diagramm trotz der anfänglichen Erwartung, bei größeren Netzwerken unperfomant zu werden, darstellen konnte. Selbst bei Netzwerken mit mehreren Tausend Knoten blieb die Performance akzeptabel. Dies macht das Node-Link Diagramm zu einer starken Option für mittelgroße Netzwerke mit moderaten Verbindungen.

Die Adjazenzmatrix hingegen zeigte ihre Stärken bei dichten Netzwerken mit vielen Verbindungen, jedoch nur bis zu einer bestimmten Knotenanzahl. Bei Netzwerken mit mehr als 200 Knoten verlor die Adjazenzmatrix an Übersichtlichkeit und litt unter Performanceproblemen, was ihre Anwendung auf große Netzwerke einschränkt. Diese Technik erwies sich dennoch als nützlich für kleinere, stark verbundene Netzwerke, in denen die Verbindungen klar dargestellt werden können, ohne dass Überlappungen auftreten.

Eine der wichtigsten Erkenntnisse war die zentrale Rolle der Clustererkennung und Clusterdarstellung. Durch das Zusammenfassen stark verbundener Knoten in Cluster wurde die Komplexität großer Netzwerke erheblich reduziert, was die Visualisierung von Netzwerken mit zehntausenden Knoten ermöglichte. Die Clusterdarstellung verbesserte die Übersichtlichkeit und Skalierbarkeit der Visualisierung signifikant, indem sie Netzwerke in überschaubare Einheiten aufteilte. Zudem bot sie den Nutzern die Möglichkeit, bei Bedarf tiefer in die Cluster einzutauchen, um die enthaltenen Verbindungen zu untersuchen.

Jedoch zeigte sich, dass Verbindungen zwischen Clustern bei sehr großen Netzwerken problematisch sind. Trotz der Reduktion sichtbarer Knoten bleibt die Vielzahl an Verbindungen bestehen, was zu Instabilitäten im Force-Directed Layout und Performance-Problemen führte. Diese Erkenntnis zeigt, dass alternative Ansätze zur Darstellung von Verbindungen in großen Netzwerken notwendig sind, um Stabilität und Performance zu verbessern.

Zusammengefasst zeigte die Arbeit, dass sowohl das Node-Link Diagramm als auch die Adjazenzmatrix jeweils spezifische Stärken für bestimmte Netzwerkstrukturen bieten. Die Clustererkennung ist dabei ein Schlüssel, um sehr große Netzwerke handhabbar zu machen. Während das Node-Link Diagramm in Kombination mit der Clusterdarstellung die beste Balance aus Übersichtlichkeit und Skalierbarkeit bietet, bleibt die Optimierung der Darstellung von Verbindungen zwischen Clustern eine Herausforderung für zukünftige Arbeiten.

4.3 Ausblick

Die Ergebnisse dieser Arbeit haben gezeigt, dass die Visualisierung von Knowledge Graphen durch Techniken wie das Node-Link Diagramm, die Adjazenzmatrix und die Clustererkennung erheblich verbessert werden kann. Gleichzeitig sind während der Auswertung der Implementierung und der Tests neue Ideen und Optimierungsmöglichkeiten aufgekommen, die in zukünftigen Arbeiten berücksichtigt werden sollten.

4.3.1 Kombination von Node-Link und Adjazenzmatrix

Eine interessante Idee, die sich aus den Stärken und Schwächen der beiden Diagrammarten entwickelt hat, ist die Möglichkeit, Node-Link und Adjazenzmatrix auf eine innovative Weise zu kombinieren. Da die hohe Konnektivität zwischen Communities im Node-Link Diagramm zu Problemen führt, aber gleichzeitig ein Vorteil der Adjazenzmatrix ist, könnte man diese beiden Ansätze verbinden: Die Communities werden zunächst als Adjazenzmatrix dargestellt, wobei die Verbindungen zwischen den stark verknüpften Gruppen übersichtlich und ohne Überlappungen dargestellt werden. Durch einen Klick auf eine Community innerhalb der Matrix könnte dann eine detaillierte Darstellung der Knoten innerhalb dieser Gruppe im Node-Link Diagramm erfolgen. Dies würde die Stärken beider Techniken vereinen und könnte die Visualisierung sehr großer Netzwerke erheblich verbessern.

4.3.2 Weitere Techniken zur Verbesserung der Visualisierung

Es gibt mehrere Visualisierungstechniken, die in zukünftigen Implementierungen getestet werden sollten, um die Benutzerfreundlichkeit und Übersichtlichkeit weiter zu optimieren. Diese Techniken wurden in dieser Arbeit nicht getestet, da es der zeitliche Rahmen nicht erlaubt hat.

- Edge Bundling: Diese Technik könnte dabei helfen, die visuelle Komplexität von Netzwerken zu reduzieren, indem ähnliche Kanten zu Bündeln zusammengefasst werden. Besonders in stark vernetzten Netzwerken könnte dies die Übersichtlichkeit verbessern und die Darstellung entlasten.
- Highlighting von Kanten bei Hover über Knoten: Diese Funktion würde es den Benutzern ermöglichen, die Verbindungen eines spezifischen Knotens schnell und übersichtlich zu sehen, was die Interaktivität und Analyse des Netzwerks erleichtert.

Neben den Visualisierungstechniken ist auch die Performance ein zentrales Thema für Optimierungen. Vor allem bei großen Netzwerken und vielen Verbindungen zwischen Clustern treten Performance-Probleme auf. Alternativen wie die Bibliothek sigma.js könnten getestet werden, um zu prüfen, ob sie eine bessere Performance und Stabilität bieten.

Auch die Überarbeitung der Clustering-Ansicht ist notwendig. Derzeit führt die Hover-Detection dazu, dass kleine Knoten verschwinden und wieder auftauchen, wenn der Benutzer über Community-Nodes fährt. Eine Verbesserung der Erkennung würde die Benutzererfahrung deutlich konsistenter und reibungsloser gestalten.

4.3.3 Fazit

Insgesamt bietet die Arbeit eine solide Grundlage für die Visualisierung von Knowledge Graphen, hat aber auch Bereiche aufgezeigt, in denen zukünftige Entwicklungen und Tests notwendig sind. Durch die Kombination von Node-Link und Adjazenzmatrix, den Einsatz neuer Visualisierungstechniken und Performance-Optimierungen können zukünftige Implementierungen robuster und effizienter gestaltet werden, um den Anforderungen großer und komplexer Netzwerke gerecht zu werden.

Literatur

[BBC+21] M. Burch, K.B. ten Brinke, A. Castella u. a. *Dynamic graph exploration by interactively linked node-link diagrams and matrix visualizations*. Techn. Ber. Visual Computing for Industry, Biomedicine, und Art, 2021.

[BHJ09] Mathieu Bastian, Sebastien Heymann und Mathieu Jacomy. „Gephi: An Open Source Software for Exploring and Manipulating Networks“. In: *Proceedings of the International AAAI Conference on Web and Social Media* 3.1 (März 2009), S. 361–362. DOI: 10.1609/icwsm.v3i1.13937. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/13937.

[Bos] Mike Bostock. *GitHub - d3*. URL: https://github.com/d3/d3 (besucht am 26. 08. 2024).

[Cha+22] Vinay Chaudhri u. a. „Knowledge Graphs: Introduction, History and, Perspectives“. In: *AI Magazine* 43.1 (2022), S. 17–29. DOI: 10.1002/aaai.12033. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/19119.

[Che+19] Yi Chen u. a. *A survey on visualization approaches for exploring association relationships in graph data*. Techn. Ber. The Visualization Society of Japan, 2019.

[HW09] D.H.R. Holten und J.J. Wijk, van. „Force-directed edge bundling for graph visualization“. In: *Computer Graphics Forum* 28.3 (2009). ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2009.01450.x.

[JD88] Anil K. Jain und Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[JPS] Alexis Jacomy, Guillaume Plique und Benoît Simard. *GitHub - sigma.js*. URL: https://github.com/jacomyal/sigma.js (besucht am 26. 08. 2024).

[NKI20] Rungsiman Nararatwong, Natthawut Kertkeidkachorn und Ryutaro Ichise. *Knowledge Graph Visualization: Challenges, Framework, and Implementation*. Techn. Ber. 2020, S. 174–178. DOI: 10.1109/AIKE48582.2020.00034.

[OJK19] Mershack Okoe, Radu Jianu und Stephen Kobourov. „Node-Link or Adjacency Matrices: Old Question, New Insights“. In: *IEEE Transactions on Visualization and Computer Graphics* 25.10 (2019), S. 2940–2952. DOI: 10.1109/TVCG.2018.2865940.

[Que+15] Xinyu Que u. a. „Scalable Community Detection with the Louvain Algorithm“. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. 2015, S. 28–37. DOI: 10.1109/IPDPS.2015.59.

[SB92] Manojit Sarkar und Marc H. Brown. „Graphical fisheye views of graphs“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 1992, S. 83–91. ISBN: 0897915135. DOI: 10.1145/142750.142763. URL: https://doi.org/10.1145/142750.142763.

[Sha+03] Paul Shannon u. a. *Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks*. Techn. Ber. Cold Spring Harbor Laboratory, 2003. URL: http://www.genome.org/cgi/doi/10.1101/gr.1239303.

[SW] Lorenzo Sauras-Altuzarra und Eric W. Weisstein. *Adjacency Matrix*. URL: https://mathworld.wolfram.com/AdjacencyMatrix.html (besucht am 10. 07. 2024).

[Thea] The Cytoscape Consortium. *Navigation and Layout*. URL: https://manual.cytoscape.org/en/stable/Navigation_and_Layout.html (besucht am 19. 08. 2024).

[Theb] The Data Visualisation Catalogue. *Network Diagram*. URL: https://datavizcatalogue.com/methods/network_diagram.html (besucht am 10. 02. 2024).

[Unw20] Antony Unwin. *Why is Data Visualization Important? What is Important in Data Visualization?* Techn. Ber. Harvard Data Science Review, 2020.

[w3o] w3.org. *RDF 1.1 Concepts and Abstract Syntax*. URL: https://www.w3.org/TR/rdf11-concepts/ (besucht am 06. 09. 2024).

[Wil96] Robin J. Wilson. *Introduction to Graph Theory*. 4. Aufl. Longman, 1996. ISBN: 0-582-24993-7.

[XW05] Rui Xu und Donald Wunsch. „Survey of Clustering Algorithms“. In: *Neural Networks, IEEE Transactions on* 16 (Juni 2005), S. 645–678. DOI: 10.1109/TNN.2005.845141.

[yWoa] yWorks. *Clustering Graphs and NetworksClustering Graphs and Networks*. URL: https://www.yworks.com/pages/clustering-graphs-and-networks (besucht am 19. 08. 2024).

[yWob] yWorks. *Force-Directed Graph Layout*. URL: https://www.yworks.com/pages/force-directed-graph-layout (besucht am 19. 08. 2024).