Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwurf eines Systems zur Identifikation von Einstiegspatterns für die Umsetzung von Quantenalgorithmen

Timm Pankratz

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Frank Leymann
Betreuer/in:	M. Sc. Daniel Vietz M. Sc. Benjamin Weder
Beginn am:	27. April 2023
Beendet am:	27. Oktober 2023

Kurzfassung

Durch den Fortschritt im Bereich der Quanteninformatik können eine steigende Anzahl an Quantenalgorithmen tatsächlich auf einem Quantencomputer realisiert werden. Um diese Algorithmen umsetzen zu können ist Fachwissen erforderlich. Die Funktionsweise ist allein mit dem Verständnis von klassischen Algorithmen nicht nachzuvollziehen. Es ist ein System notwendig, welches die Umsetzung und Implementierung von Quantenalgorithmen unterstützt, ohne hohe Anforderungen an Fachwissen zu stellen. Diese Arbeit stellt ein Konzept zur Realisierung eines solchen Systems dar. Durch Angabe einer textuellen Beschreibung des umzusetzenden Problems, werden Quantenalgorithmen vorgeschlagen, die zur Lösung des Problems verwendet werden können. Mithilfe von Natürlicher Sprachverarbeitung wird ein Textvergleich durchgeführt, der die Eingabe mit den Beschreibungen und Informationen verschiedener Quantenalgorithmen vergleicht und die zutreffenden Ergebnisse präsentiert. Zur Unterstützung der Umsetzung der Quantenalgorithmen werden Muster, auch Patterns genannt, verwendet. Als bewährtes Konzept der Informatik bieten diese Lösungen zu bekannten Problemen an. Die Muster werden den Quantenalgorithmen zugeordnet und liefern Ansätze, die zur Umsetzung verwendet werden können. Um den Einstieg bei der Umsetzung weiter zu erleichtern, werden sogenannte Einstiegsmuster beziehungsweise Einstiegspatterns identifiziert. Diese stellen eine Hilfestellung für den Startpunkt der Implementierung dar. Die Ergebnisse des Systems werden visuell mithilfe eines Graphen dargestellt. Außerdem enthalten die Ergebnisse Referenzen zu den vorgeschlagenen Quantenalgorithmen, sowie den zugehörigen Mustern, wodurch alle notwendigen Ressourcen zur Umsetzung der Quantenalgorithmen vorhanden sind. Eine prototypische Implementierung des Konzepts, welche auf dem Patternatlas basiert, wird ebenfalls beschrieben. Das Konzept wird mithilfe dieses Prototypen validiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

GUI Graphical User Interface / Graphische Benutzerschnittstelle. 40, 42

HTTP Hypertext Transfer Protocol. 49, 51

JPA Java Persistence API. 24, 51, 52

JSON JavaScript Objekt Notation. 52

NISQ Noisy-Intermediate-Scale-Quantum. 15, 20

NLP Natural Language Processing / Natürliche Sprachverarbeitung. 22, 60

PlanQK Plattform und Ökosystem für Quantenapplikationen. 38, 48, 49, 51, 52, 56, 58, 59, 60

QAOA Quantum Approximate Optimization Algorithmus. 7, 32, 39, 56, 57

REST Representational State Transfer. 23, 49

SQL Structured Query Language. 24

VQE Variational Quantum Eigensolver. 32

1 Einleitung

Quantencomputer sind in den Augen vieler Unternehmen im aktuellen Zustand noch nicht ausgereift und deshalb noch ein Zukunftsprojekt. Trotzdem investieren weltweit viele Unternehmen in Quantencomputing [SCCT+16], denn durch die rasante Entwicklung im Bereich des Quantencomputings stehen immer leistungsfähigere Quantencomputer zur Verfügung. Es wird prognostiziert, dass diese in absehbarer Zeit in verschiedenen Bereichen der Industrie verwendet werden können [LTKT19]. Die Anzahl an Qubits, die ein Quantencomputer verwenden kann, steigt von Jahr zu Jahr stark an. Viele praxisrelevante Probleme, die bisher nicht mit Quantencomputern lösbar waren, können mit der aktuellen Anzahl an Qubits bereits gelöst werden. Ein Beispiel dafür ist der *Variational Quantum Factoring Algorithmus* [AOAC19], der zum Faktorisieren verwendet wird. Dadurch werden Quantencomputer immer attraktiver für die Industrie. Um Quantenvorteile, wie eine schnellere Ausführungszeit im Vergleich zu herkömmlichen Algorithmen, auszunutzen, müssen sogenannte Quantenalgorithmen entwickelt und implementiert werden. Diese Art von Algorithmen sind auf die Funktionsweise von Quantencomputern angepasst und können mit Qubits umgehen. Sie versprechen exponentielle Beschleunigung im Vergleich zu normalen Algorithmen. Jedoch benötigen viele Quantenalgorithmen für sinnvolle Anwendungen eine größere Anzahl an Qubits. Die aktuell zur Verfügung stehenden Qubits sind nicht ausreichend. Die vielversprechendsten Algorithmen sind deshalb zu diesem Zeitpunkt nicht die puren Quantenalgorithmen, sondern hybride Algorithmen. Sie kombinieren klassische Berechnungen mit der Berechnung von Quantencomputern und ermöglichen dadurch den Einsatz von Quantencomputer mit limitierter Anzahl an Qubits zur Ausführung des Algorithmus. Solche Quantencomputern werden als *Noisy Intermediatre-Scale Quantum Computer* (NISQ) bezeichnet und gehören zur aktuellen NISQ Ära [LLSK22]. Durch die Kombination von klassischen Von-Neumann-Rechnern und NISQ Quantencomputern, können bereits viele Probleme durch Verwendung von hybriden Algorithmen mit Ausnutzen von Quantenvorteilen gelöst werden. Die in Gebrauch befindlichen NISQ Quantencomputer sind auf einige hundert Qubits limitiert, weshalb sie noch verrauscht sind. Diese Anzahl an Qubits ist nicht ausreichend, um eine Rauschunterdrückung durchzuführen. Eine Rauschunterdrückung benötigt zusätzliche Qubits, um einen Schaltkreis zur Fehlererkennung und Fehlerkorrektur zu realisieren [Rof19]. Deshalb sind die aktuellen Quantencomputer noch fehleranfällig. Um Zugriff auf einen Quantencomputer zu bekommen, kann einer von vielen Anbietern wie beispielsweise IBM [IBM23] ausgewählt werden. Die Anbieter stellen eine wachsende Anzahl an Cloud-basierten Quantencomputern zur Verfügung. Somit wird die Nutzung auch für Privatpersonen erleichtert.

Obwohl eine exponentielle Beschleunigung durch Quantencomputing erreicht werden kann, ist es nicht immer vorteilhaft, ein gegebenes Problem mithilfe eines Quantencomputers zu lösen. Quantenalgorithmen haben nicht in jedem Fall eine bessere Laufzeit als normale Algorithmen. In manchen Fällen ist noch kein passender Quantenalgorithmus für ein Problem bekannt. Dadurch ergibt sich ein komplett neuer Problembereich: In welchem Fall ist es vorteilhaft, einen Quantencomputer zu nutzen? Diese Frage muss möglichst früh in der Entwicklung beantwortet werden, da sich Quantenalgorithmen deutlich von klassischen Algorithmen unterscheiden und dies

bei der Implementierung berücksichtigt werden muss. Eine Antwort kann nur von qualifizierten Fachkräften geliefert werden, die Kenntnisse im Bereich der Quanteninformatik haben. Da dieser Bereich noch zu einer relativ neuen Thematik gehört, ist die Anzahl an Fachkräfte nicht ausreichend. Um dieses Problem zu umgehen, muss ein Weg gefunden werden, Antworten auf diese Frage zu liefern, ohne auf Fachkräfte angewiesen zu sein. Erste Arbeiten in diesem Themenbereich führen eine textuelle Analyse einer angegebenen Problembeschreibung durch [MAl22]. Durch Natürlicher Sprachverarbeitung werden Vergleiche mit Informationen aus einer Datenbank durchgeführt, um das Entscheidungsproblem zu lösen. Das Ergebnis beschränkt sich hierbei auf die Wahl einer Quanten- beziehungsweise klassischen Lösung. Ein konkreter Algorithmus wird nicht vorgeschlagen. Falls das Ergebnis die Nutzung von Quantenalgorithmen vorschlägt, ergeben sich weitere Probleme. In vielen Fällen existieren mehrere Quantenalgorithmen, die dasselbe Problem lösen. Um die richtige Wahl zu treffen, ist wieder Fachwissen erforderlich. Weiterhin muss der Algorithmus implementiert werden. Trotz vorhandener Bibliotheken, Entwicklungskits wie Qiskit [WVN19] und vorhandenen Implementierungen sind oft Kenntnisse in Quantencomputing notwendig, um den zu implementierenden Quantenalgorithmus an die gegebenen Bedingungen anzupassen. Daraus ergibt sich die generelle Forschungsfrage der Arbeit:

Wie kann die Umsetzung eines Quantenalgorithmus zur Lösung eines Problems unterstützt werden, auch wenn nur wenig Fachwissen vorhanden ist?

Um diese Forschungsfrage zu lösen, sind mehrere Ansätze notwendig, welche die Umsetzung eines Quantenalgorithmus erleichtern sollen. Der erste Ansatz beschäftigt sich mit der Angabe des Problems. Es muss zuerst ein passender Algorithmus ausgewählt werden, der zur Lösung des Problems geeignet ist. Dieser soll sich aus der Problembeschreibung ergeben. Um die Problembeschreibung auszuwerten, sind Methoden aus der Natürlichen Sprachverarbeitung notwendig. Durch Vergleich der Angabe mit Beschreibungen verschiedener Algorithmen, kann ein geeigneter Algorithmus gefunden werden. Dies erfordert eine Sammlung von Informationen über verschiedene Algorithmen. Nach der Wahl des Algorithmus, soll der zweite Ansatz die Umsetzung erleichtern. Dazu muss eine Möglichkeit gefunden werden, das Wissen zu diesem Algorithmus kompakt darzustellen. Eine gute Möglichkeit hierfür sind Muster, auch Patterns genannt. Als gängiges Konzept im Bereich der Informatik werden Muster dazu benutzt, Lösungen zu wiederkehrenden Problemen zu dokumentieren. Muster wurden ursprünglich im Bereich der Architektur eingeführt und breiten sich inzwischen auf zahlreiche Domänen aus. Auch für Quantencomputing existieren bereits Muster-Sprachen. Leymann et al. [Ley19] beschreiben beispielsweise die Kernelemente von Quantenalgorithmen. Weitere Muster beschreiben hybride Algorithmen [WBLV21] und Fehlerbehandlung auf Quantencomputern [BBL+22]. Um einen Quantenalgorithmus zu implementieren, reicht ein einzelnes Muster nicht aus. Zur Lösung von komplexen Problemstellungen müssen verschiedene Muster kombiniert werden. Die passende Menge an Mustern, sowie die richtigen Muster auszuwählen, erfordert wieder Fachwissen. Deshalb ist ein weiterer Ansatz notwendig, der verschiedenen Algorithmen entsprechende Muster zuordnet. Dazu muss die Funktionsweise der Algorithmen und Muster analysiert werden, um eine entsprechende Zuordnung durchzuführen. Nach der Zuordnung sind alle Muster eines Algorithmus bekannt, jedoch kann die Anzahl von Mustern bei komplexen Algorithmen groß sein. Ein Einstiegspunkt zu Beginn der Umsetzung ist erforderlich, was einen weiteren Ansatz darstellt. Der Zusammenhang der Muster muss untersucht werden, um einen solchen Einstiegspunkt zu finden. Die Untersuchung führt zu Wahl von Einstiegsmuster, beziehungsweise Einstiegspatterns, welche als Startpunkt zur Implementierung verwendet werden können. Außerdem können anhand der Relationen zwischen verschiedenen Mustern, die nächsten

geeigneten Muster nach den Einstiegsmustern für Implementierung ausgewählt werden. Zuletzt ist ein Ansatz notwendig, welcher es ermöglicht, die Ergebnisse zusammenzufassen und darzustellen. Hierfür ist eine visuelle Darstellung geeignet, um eine klare Übersicht zu schaffen.

Um diese Probleme zu adressieren, wird in dieser Arbeit ein Konzept vorgestellt, welches die zuvor genannten Ansätze umsetzt. Das Konzept verwendet die Textvergleichstechniken *Keyword Extraction* und *Kosinus Ähnlichkeit* aus dem Bereich der Natürlichen Sprachverarbeitung, um einen passenden Algorithmus zur Umsetzung eines angegebenen Problems auszuwählen. Dabei wird das angegebene Problem mit verschiedenen Algorithmenbeschreibungen verglichen. Die Quantencomputing Muster werden zur Darstellung der Informationen des Algorithmus verwendet. Dem ausgewählten Algorithmus werden die zugehörigen Muster zugeordnet. Dazu wird eine Untersuchung von Zusammenhängen zwischen verschiedenen Mustern, sowie Muster und Algorithmen durchgeführt. Das Ergebnis wird mithilfe eines Wissensspeichers gesichert, welcher die Ergebnisse der Zuordnungen von Mustern zu verschiedenen Algorithmen enthält. Um einen Startpunkt zur Umsetzung zu finden, werden die zum Algorithmus gehörigen Muster untersucht und Einstiegsmuster identifiziert. Anhand der Einstiegsmuster kann eine Reihenfolge festgelegt werden, mithilfe derer die Muster zur Umsetzung des Algorithmus verwendet werden können. Für die Ergebnisdarstellung wird ein Graph verwendet, welcher die Muster des Algorithmus beinhaltet und die Einstiegsmuster kennzeichnet. Außerdem sind Referenzen vorhanden, welche weitere Informationen zu Mustern und zum Algorithmus enthalten. Es wird eine prototypische Umsetzung des Konzepts auf Basis des Patternatlas [LB21] implementiert, welche die im Konzept beschriebenen Ansätze validiert.

Aufbau der Arbeit

Diese Arbeit ist wie folgt strukturiert:

Kapitel 2 - Hintergründe
: Dieses Kapitel beschreibt Themen wie Muster und Quantenalgorithmen, die zum Verständnis der Arbeit wichtig sind, sowie einige der zur Implementierung verwendeten Techniken.

Kapitel 3 - Verwandte Arbeiten
: Kapitel drei zeigt relevante Arbeiten, die aufgrund ihres Inhaltes einen Einfluss auf diese Arbeit haben. Die Arbeiten werden in drei Themengebiete eingeteilt, welche unterschiedliche Bereiche der Arbeit beeinflussen.

Kapitel 4 - Motivierendes Szenario
: In Kapitel vier wird ein Szenario beschrieben, was die Notwendigkeit dieser Arbeit zeigt. Dieses Szenario enthält die Vorgehensweise zum Finden eines passenden Algorithmus und zugehörigen Mustern, ohne die Verwendung des Konzepts.

Kapitel 5 - Konzept
: Kapitel fünf enthält das Konzept der Arbeit. Hier wird eine Methode vorgestellt, die das Finden von passenden Algorithmen, sowie zugehörigen Mustern beschleunigt. Außerdem werden die Systembestandteile und umsetzungsbedingte Anpassungen besprochen, die sich aus dem Konzept ergeben.

Kapitel 6 - Implementierung

Eine Implementierung des in Kapitel fünf vorgestellten Konzepts wird in Kapitel sechs gezeigt. Auf die Bestandteile und Besonderheiten der Implementierung wird genauer eingegangen.

Kapitel 7 - Evaluation

Dieses Kapitel untersucht die Qualität des Ergebnisses. Dabei wird ermittelt, ob die Verwendung des Konzepts zur Implementierung hilfreich für das Szenario in Kapitel 4 ist.

Kapitel 8 - Zusammenfassung und Ausblick

Das letzte Kapitel enthält eine Zusammenfassung der Arbeit. Außerdem werden noch weitere Problembereiche genannt, die in zukünftigen Arbeiten gelöst werden müssen.

2 Hintergründe und Technologien

In diesem Kapitel wird das Hintergrundwissen geliefert, welches zum Verständnis dieser Arbeit notwendig ist. Abschnitt 2.1 befasst sich mit den Besonderheiten von Quantenalgorithmen. In Abschnitt 2.2 wird genauer auf Muster und Mustersprachen eingegangen. Die Verwendung von Natürlicher Sprachverarbeitung zum Textvergleich, sowie verschiedene Techniken hierfür, werden in Abschnitt 2.3 beschrieben. Anschließend wird in Abschnitt 2.4 die Verwendung von verschiedenen Technologien, die zur Implementierung des Konzepts verwendet werden, beschrieben.

2.1 Quantenalgorithmen

Gewöhnliche Algorithmen, die auf klassischen Computern ausgeführt werden, können nicht direkt für Quantencomputer verwendet werden. Das liegt daran, dass die Funktionsweise von Quantencomputern anders ist. Sie verwenden keine Bits, weshalb reguläre Algorithmen nicht funktionieren. Stattdessen arbeiten sie mit sogenannten Qubits, welche das Äquivalent zu den Bits darstellen. Qubits haben einen bedeutenden Unterschied zu Bits. Sie haben keinen festen Zustand, sondern bilden ein Zustandssystem. Dieses Zustandssystem stellt ein Qubit dar und entspricht genau einem Punkt auf der Bloch-Sphäre. Diese Sphäre ist eine Kugel, bei der jeder Punkt genau einen Einheitsvektor entfernt vom Mittelpunkt ist. Dadurch existieren überabzählbar viele Qubits auf der Bloch-Sphäre, während die normalen Bits nur einen festen Wert von null oder eins annehmen können. Dieser Unterschied wird in Abbildung 2.1 gezeigt. Ein fester Zustand kann nur durch Messen erzeugt werden. Bei der Messung wird jedoch das Zustandssystem zerstört, weshalb dieses nicht mehr weiterverwendet werden kann. Für den Umgang mit Qubits benötigt man spezielle Algorithmen. Diese Algorithmen werden Quantenalgorithmen genannt und werden explizit für die Verwendung auf Quantencomputern entworfen. Es existieren viele Arbeiten zu Quantenalgorithmen, wie beispielsweise von Shor [Sho02] oder Strubell [Str11], die genauer auf die Thematik eingehen. Zur Ausführung von Quantenalgorithmen auf Quantencomputern werden Quantenschaltkreise genutzt. Quantenschaltkreise verwenden spezielle Operatoren, die zur Manipulation von Qubits genutzt werden. Diese Operatoren stellen eine unitäre Transformation auf den Zustand der Qubits dar. Gängige Beispiele sind der CNOT-Operator oder der Rotations-Operator. Einer der wichtigsten Transformationen wird mithilfe des Hadamard-Operators durchgeführt. Durch Verwenden dieses Operators kann ein Basiszustand in Superposition gebracht werden. Dadurch entsteht eine Überlagerung des Zustandes. Eine weitere wichtige Eigenschaft von Qubits ist die Verschränkung. Verschränkte Zustände sind voneinander abhängig. So kann bei der Messung eines Qubits in einem maximal verschränkten Zustand der Wert der restlichen Qubits zur selben Zeit bestimmt werden. Durch diese Eigenschaft wird zum Beispiel die Quantenteleportation [Zei00] möglich. Quantenalgorithmen nutzen Quantenphänomene wie die Superposition und Verschränkung aus, um Probleme in kürzerer Zeit als gewöhnliche Algorithmen zu lösen. Es wird teilweise eine exponentielle Beschleunigung erreicht.

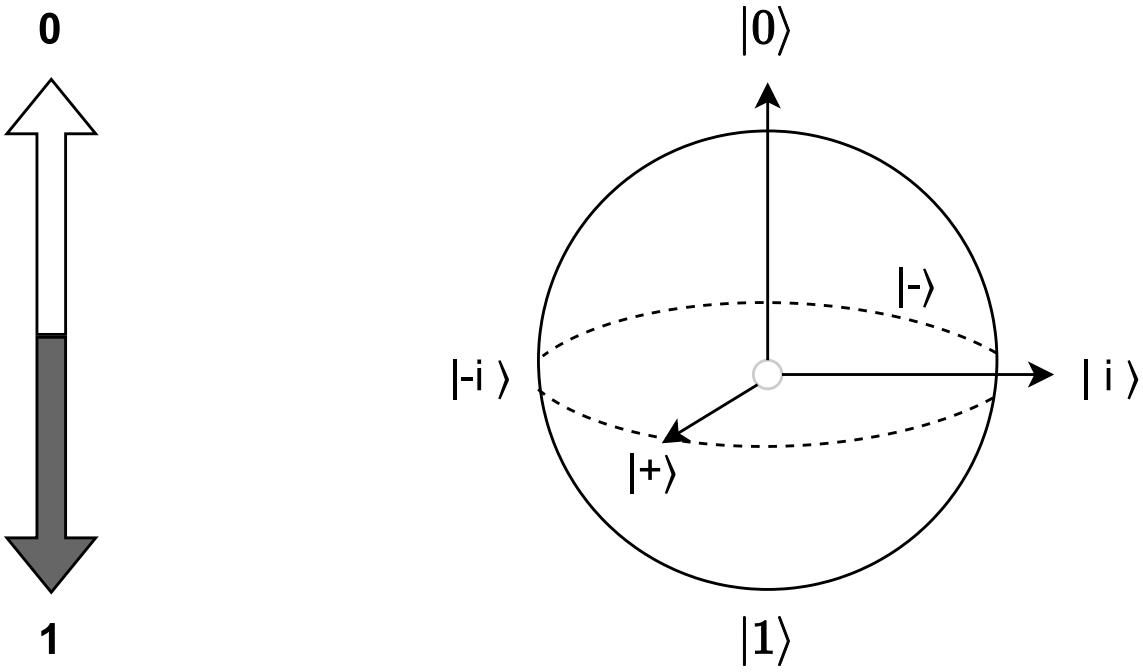

Abbildung 2.1: Unterschiede von Bits und Qubits

Quantenalgorithmen sind in fünf Teile aufgebaut. Sie bestehen aus der Vorverarbeitung, der Zustandsvorbereitung, der Unitären Transformation, gefolgt vom Messen und der Nachbearbeitung. Die Vorverarbeitung passt die Eingabe an die Gegebenheiten des zu verwendenden Quantencomputers an. Nach dieser Verarbeitung kann die Eingabe in die Quantenregister eingefügt werden. Dies passiert bei der Zustandsvorbereitung. Die Eingabe liegt bei diesem in Form von Qubits in den Quantenregistern vor und wird gegebenenfalls noch mittels Operatoren in den benötigten Startzustand des Quantenalgorithmus gebracht. Der eigentliche Algorithmus wird durch die Anwendung unitäre Transformationen auf den Startzustand durchgeführt. Das Ergebnis wird durch Messen des Endzustands erzielt. Das Messergebnis wird anschließend, falls notwendig, noch nachbearbeitet. Je nach Algorithmus wird der gesamte Vorgang mehrmals wiederholt.

Eine Teilgruppe der Quantenalgorithmen sind die sogenannten hybriden Algorithmen. Diese kombinieren die Nutzung von Quantenalgorithmen auf Quantencomputern mit der Verwendung von klassischen Computern. Hybride Algorithmen versuchen, Teile des Algorithmus, die keine Quantenvorteile ausnutzen, so weit wie möglich auf klassische Computer auszulagern. Dadurch wird nur für einen Teil des Algorithmus tatsächlich ein Quantencomputer benötigt. Die Zustandsvorbereitung, unitäre Transformation und das Messen finden im Quantencomputer statt, während die Vor- und Nachbearbeitung einen klassischen Computer verwendet. Die Anpassung wird vorgenommen, da aktuelle Quantencomputer in ihrer Anzahl an Qubits begrenzt sind und deshalb viele Algorithmen noch nicht komplett ausführen können. Außerdem kann aufgrund der Anzahl von Qubits noch keine Fehlerkorrektur durchgeführt werden [Rof19]. Deshalb sind die aktuellen NISQ Quantencomputer noch verrauscht. Eine Auslagerung von Berechnungen auf klassische Rechner reduziert die Anzahl an Fehlern. Aufgrund all diesen Eigenarten sind Quantenalgorithmen äußerst komplex und schwer zu entwickeln, da sehr viel Wissen über Quantenmechaniken erforderlich ist. Deshalb ist es hilfreich, Muster zu verwenden, um die Komplexität bei der Entwicklung zu verringern.

2.2 Patterns und Patternsprachen

Patterns sind Entwurfsmuster, die in vielen verschiedenen Gebieten zum Einsatz kommen. Sie werden als Ansatz verwendet, um bekannte Probleme effizient zu lösen. Ihren Ursprung haben Muster im Bereich der Architektur, wo sie zum ersten Mal von Christopher Alexander in seinem Buch, *A Pattern Language* [Ale77], verwendet werden. Alexander beschreibt Muster als Objekte, die wiederkehrende Probleme beschreiben. Der Aufbau eines Musters ist genau definiert, um Klarheit zu schaffen. Jedes Muster erhält ein Bild oder ein Symbol, welches ein Beispiel für das Problem darstellt. Die Bilder sind so gewählt, dass das Problem möglichst direkt erkannt werden kann. Nach dem Bild folgt eine Einführung, die den Kontext des Musters beschreibt. Der Einführung folgt ein kurzer Paragraph oder sogar nur ein Satz, welcher den Kern des Problems beschreibt. Anschließend wird das Problem beschrieben. Diese Beschreibung ist meist ausführlich und enthält den Hintergrund des Problems, wie es auftauchen kann und auf welche Art und Weise es weitere Probleme verursachen kann. Nach der Beschreibung, welche oft der längste Teil des Musters ist, folgt die Lösung zum zuvor beschriebenen Problem. Die Lösung darf nicht nur Lösungsansätze enthalten, sondern muss konkrete Anweisungen besitzen, die zur Lösung des Problems führen. Dadurch ist garantiert, dass bei der Verwendung von Mustern eine Lösung vorhanden ist. Nach Alexander muss auch ein Diagramm vorhanden sein, welches die Lösung des Problems zeigt. Dies ist je nach Anwendungsbereich nicht immer möglich und deshalb oft nicht vorhanden. Bei modernen Musterdarstellungen werden meistens auch Beispiele für die Nutzung des Musters angegeben. Zum Schluss werden noch verwandte Muster genannt.

Die Muster in diesem Buch beschreiben verschiedene Bereiche, wie Ansätze für die Platzierung von Fenstern in einem Raum. Muster, die ein bestimmtes Themengebiet beschreiben, werden unter einer Muster-Sprache zusammengefasst. Im Fall von Christopher Alexander werden die Muster in seinem Buch zur Sprache der Architekturmuster zugeordnet [Ale77]. Eine Muster-Sprache wird als Netzwerk von Mustern beschrieben, in welchem Gruppierungen von Mustern enthalten sind, die ähnliche Probleme beschreiben. Verbindungen zwischen diesen Gruppen können betrachtet werden, um eine Übersicht über die gesamte Muster-Sprache zu erhalten. Mit dem Ansatz von Muster-Sprachen können viele weitere Themengebiete zusammengefasst und beschrieben werden. Im Bereich der Informatik existieren bereits mehrere Muster-Sprachen, unter anderem die Cloud-Computing Muster sowie die für diese Arbeit relevanten Quantencomputing Muster. Diese Muster beschreiben, ähnlich wie die Architekturmuster, Ansätze für bestimmte Probleme in den entsprechenden Gebieten ihrer Sprache. Die Ansätze liefern dabei Vorschläge für die Implementierung des Problems. In dieser Arbeit werden Quantencomputing Muster verwendet, um die Bestandteile verschiedener Quantenalgorithmen darzustellen. Dadurch sollen diese besser verständlich sein, sowie die Implementierung durch Kombinationen von Mustern erleichtert werden. Gruppierungen von Mustern sind auch hier zu finden. Muster zur Fehlerbehandlung oder Muster zur Vorbereitung von Daten für einen Quantenalgorithmus gehören zu dieser Art von Gruppen. Die Quantencomputing Muster werden in Abschnitt 3.2 genauer beschrieben. Für komplexere Probleme können Muster aus verschiedenen Muster-Sprachen kombiniert werden. Dadurch können auch Ansätze für Probleme, die mehrere Themengebiete betreffen, gefunden werden.

2.3 Natürliche Sprachverarbeitung und Textvergleich

Natürliche Sprachverarbeitung ist eine Machine Learning Technologie, die es erlaubt, Texte oder Sprache zu analysieren. Diese Technologie hat verschiedene Anwendungsbereiche wie die Archivierung von Dokumenten, die Automatisierung des Kundenservice mithilfe von Chatbots oder dem Klassifizieren von Texten. Der Textvergleich ist ebenfalls Teil der Natürlichen Sprachverarbeitung und wird verwendet, um Texteingaben oder Anfragen zu verarbeiten und Vergleiche durchzuführen. In dieser Arbeit wird der Textvergleich verwendet, um Eingaben mit vorhandenen Daten zu vergleichen. Alle möglichen Ergebnisse müssen mit der Eingabe auf Ähnlichkeiten überprüft werden. Da die Anzahl an vorhandenen Daten oft sehr groß ist und gefiltert werden muss, ist es notwendig, diese auf effiziente Weise darzustellen. Der direkte Textvergleich ist zu aufwändig und braucht zu viele Ressourcen, da jedes einzelne Wort aus jedem Text verarbeitet werden muss. Um dieses Problem zu lösen, werden verschiedene Techniken zur Einschränkung und verbesserter Darstellung der Datenmenge verwendet. Eine häufig verwendete Technik hierfür ist *Keyword Extraction*. Dafür gibt es verschiedene Ansätze [FNAD20]. Ein trivialer Ansatz ist das Verwenden einer Liste, die zu filternde Wörter enthält. Hier werden häufig verwendete Stoppwörter entfernt. Dies sind Wörter wie *und* oder *zu*, welche keine wichtige Bedeutung haben. Methoden wie *TextRank* [MT04] teilen Texte in lexikalische Einheiten ein, die aus einem, oder in seltenen Fällen, mehreren Wörtern bestehen. Durch Nutzen von *Co-Occurence* [WBS10] Relationen werden die Distanzen zwischen den lexikalischen Einheiten untersucht. Einheiten, die unter einem bestimmten Wert liegen, werden zu einem Textgraphen hinzugefügt. Alle lexikalischen Einheiten im Graphen werden anschließend bewertet. Die Einheiten mit den besten Ergebnissen stellen die Schlüsselwörter dar. Diese Methode fällt in den NLP Bereich des nicht überwachten Lernens.

Alternativ können genetische Algorithmen wie *GenEx* [Tur00] verwendet werden, welche durch Trainieren mit großen Datenmengen Schlüsselwörter finden. Weitere Methoden wie Rake [RECC10] verwenden die Häufigkeit eines Wortes im Text in Kombination mit anderen Faktoren wie der schon erwähnten *Co-Occurence*, um Schlüsselwörter zu finden. Anhand der Schlüsselwörter wird dann mithilfe eines Ähnlichkeitsmaßes der Vergleich mit der Eingabe gemacht. Für die Ähnlichlichkeitsmaße gibt es ebenfalls verschiedene Möglichkeiten. Zu den bekanntesten Ähnlichkeitsmaßen gehört zum Beispiel das *Tf-Idf Maß* [SW10], welches den Schlüsselwörtern einen Wichtigkeitsgrad zuweist. Dadurch erhält der relevanteste Text beim Vergleich mit der Eingabe den höchsten Wichtigkeitsgrad. Das Ähnlichkeitsmaß, welches in dieser Arbeit verwendet wird, ist die *Kosinus Ähnlichkeit*. Dabei werden die sowohl die herausgefilterten Schlüsselwörter als auch die Eingabe als Vektoren betrachtet. Anschließend wird der Kosinus des Winkels bestimmt, der zwischen den beiden Vektoren vorliegt. Je ähnlicher die Vektoren sind, desto relevanter ist das Ergebnis. Ein Beispiel für die Verwendung von *Kosinus Ähnlichkeit* kann in der Arbeit von Gunawan et al. [GSB18] betrachtet werden. Das Verfahren des Textvergleichs wird in dieser Arbeit verwendet, um passende Algorithmen zur Umsetzung eines angegebenen Problems zu finden. In Kombination mit der Zuordnung von passenden Mustern zu Algorithmen stellt dies eines der wichtigsten Bestandteile des in dieser Arbeit vorgestellten Konzepts dar und der prototypischen Implementierung dar.

2.4 Technologien

Für die Implementierung des Konzepts werden mehrere unterschiedliche Technologien verwendet. Diese bieten eine Hilfestellung in verschiedenen Bereichen der Implementierung. Angular wird zur Implementierung des Frontends verwendet und wird vor allem bei der Ergebnisdarstellung genutzt. Docker erlaubt einen flexiblen Austausch von Bestandteilen. Das Java Persistence API erleichtert die Speicherung von Daten, was bei der Erstellung des Wissensspeichers zum Einsatz kommt.

2.4.1 Angular

Die Plattform Angular [Goo23] basiert auf der Programmiersprache TypeScript[1], die Aspekte von Javascript verwendet und mit Typisierung kombiniert. Angular erleichtert das Designen und die Implementierung von Front-End Architekturen. Es werden verschiedene Möglichkeiten geboten, Interaktionen mit Nutzerinnen/Nutzern zu gestalten und sich mit anderen Bestandteilen der Anwendung zu verständigen. Angular Anwendungen werden in Modulen aufgeteilt, die miteinander interagieren und kommunizieren können. Dies erleichtert vor allem die Implementierung von Dialogen zur Interaktion von Nutzerinnen/Nutzern. Außerdem unterstützt Angular die Verwendung von Services, welche typischerweise Funktionalitäten und kleine Erweiterungen für die Anwendung enthalten. Services werden oft zur Kommunikation mit anderen Bestandteilen des Systems verwendet. Die Implementierung des in dieser Arbeit beschriebenen Konzepts kommuniziert durch Nutzung eines Services mit den restlichen Bestandteilen des Systems, um Informationen zu erhalten. Mithilfe von Dialogen werden außerdem Schnittstellen zur Kommunikation mit dem Backend erstellt, welche ebenfalls Services verwenden. Durch den modularen Aufbau von Angular, können diese Dialoge direkt in das Gesamtsystem eingebunden werden, ohne andere Bestandteile zu beeinflussen. Eine Erweiterung der Implementierung kann dadurch ebenfalls einfach durchgeführt werden.

2.4.2 Docker

Docker[2] ist eine Software, welche es erlaubt, Teile einer Anwendung oder sogar eine komplette Anwendung in virtuelle Container zu verpacken. Diese Container sind isoliert voneinander und ermöglichen eine separate Bereitstellung. Dadurch wird die Flexibilität der Bereitstellung erhöht, da ein Docker-Container zu jedem Zeitpunkt kontrolliert gestartet oder gestoppt werden kann. Außerdem sind die Container leicht tragbar, da sie als sogenannte Docker-Images vorliegen, was erlaubt, sie an verschiedenen Orten zu unterschiedlichen Zwecken zu verwenden. Solche Images enthalten die notwendigen Informationen über die verpackte Anwendung, sowie zum Starten des Docker-Containers. In der Implementierung des Patternatlas [LB21] können mehrere Teile des Systems in Form von Docker-Containern verwendet werden. Die Datenbank wird beispielsweise als Docker-Image von PostgreSQL [The23] verwendet. Außerdem können die REST-Services zur Kommunikation zwischen den verschiedenen Teilen des Systems ebenfalls mithilfe eines Docker-Containers ausgeführt werden. Durch ihre Flexibilität können Docker-Container leicht ausgetauscht werden. Es kann eine Erweiterung des Inhalts eines bestimmten Docker-Containers

[1] https://www.typescriptlang.org/
[2] https://www.docker.com/

durchgeführt werden, indem die Erweiterung das Original ersetzt. Durch die Isolationseigenschaft der Docker-Container kann die Erweiterung problemlos mit den restlichen Containern beziehungsweise Bestandteilen des Systems verwendet werden. Diese Eigenschaften wurden bei der Entwicklung der prototypischen Implementierung des Konzepts ausgenutzt.

2.4.3 Java Persistence API

Das Java Persistence API[3] ist eine Spezifikation, welche das persistente Speichern von Daten in einer Datenbank programmatisch vereinfacht. Dadurch wird erhöhte Komplexität bei der Implementierung vermieden, da beispielsweise keine expliziten SQL Aufrufe zur Anpassung der Daten notwendig sind. Für die JPA Spezifikation gibt es verschiedene Implementierungen. In dieser Arbeit wird das Hibernate[4] Framework verwendet. Hibernate führt eine Abbildung von Java-Objekten zu relationalen Daten durch, was ermöglicht, Objekte direkt in einer relationalen Datenbank abzuspeichern. Die Attribute der Objekte werden zu Einträgen in der Datenbank. Zusätzliche Informationen wie das Festlegen eines primären Schlüssels können ebenfalls innerhalb der Objekte angegeben werden. Dies geschieht durch Annotationen der Attribute und falls notwendig dem Objekt selbst. Mithilfe von Annotationen können die Relationen zwischen verschiedenen Tabellen, sowie ihre gewünschten Beziehungen festgelegt werden. Ein Beispiel hierfür sind die *OnetoOne* oder *OneToMany* Beziehungen. Sie legen fest, dass beispielsweise für eine Tabelle bei *OnetoOne* nur exakt eine weitere zugehörige Tabelle existieren darf. Im Fall von *OneToMany* dürfen viele Tabellen für genau eine ursprüngliche Tabelle existieren. Der Zugriff auf die Daten ist ebenfalls einfach. Durch die Verwendung von regulären *getter* und *setter* Methoden, können die Attribute des Objektes abgerufen werden. Da diese auf die Einträge in den Tabellen der Datenbank abgebildet werden, erfolgt dadurch ein Zugriff auf die Datenbank.

[3] https://www.oracle.com/java/technologies/persistence-jsp.html
[4] https://hibernate.org/

3 Verwandte Arbeiten

In dieser Arbeit werden Arbeiten aus verschiedenen Themenbereichen betrachtet, um ein Konzept aus der Kombination von Kerninhalten dieser Arbeiten zu erstellen. Dieses Kapitel fasst grundlegende Arbeiten der Themenbereiche zusammen und gibt einen Einblick in verwandte Arbeiten, dessen Kerninhalte relevant für das Konzept sind. Abschnitt 3.1 beschreibt Arbeiten, die sich mit der Eignung von Quantenalgorithmen zur Problemlösung, sowie ihrer Umsetzung in verschiedenen Anwendungsfällen widmen. In Abschnitt 3.2 werden Muster für Quantencomputing, sowie Systeme, die Datenbanken zum Speichern dieser Muster enthalten, behandelt. Zuletzt wird in Abschnitt 3.3 auf Arbeiten eingegangen, die sich mit der Darstellung von Text als Graphen und dem damit zusammenhängenden Textvergleich beschäftigen.

3.1 Eignung von Quantenalgorithmen zur Lösung eines spezifischen Problems

Eine Problemstellung dieser Arbeit ist die Identifikation von geeigneten Quantenalgorithmen anhand einer Problemangabe. Dazu existieren bereits einige Arbeiten, die verschiedene Problembereiche untersuchen. Arbeiten wie die von Tarrataca und Wichert [TW11], untersuchen die Möglichkeit für die Verwendung von Quantencomputern zur Lösung von allgemeinen Problemen wie Tiefensuche. Dazu wird spezifisch nach einer Lösung für das Schiebeblock-Puzzle beliebiger Größe gesucht. Das bekannteste Puzzle dieser Art ist das 15-Puzzle, welches fünfzehn Puzzleteile enthält, die in einem vier mal vier großen Quadrat angebracht sind. Jedes der Puzzleteile ist nummeriert und enthält eine Zahl zwischen eins und fünfzehn. Anstatt eines sechzehnten Puzzleteils ist ein Loch vorhanden, welches das Schieben der Puzzleteile ermöglicht. Das Schiebeblock-Puzzle gilt als gelöst, wenn die Puzzleteile in aufsteigender Reihenfolge angeordnet sind. Tarrataca und Wichert verwenden eine hybride Implementierung zur Lösung des Problems. Für dieses Problem wird eine Beschleunigung im Vergleich zur puren klassischen Lösung erreicht. Allerdings ist damit nicht sichergestellt, ob jedes Problem der Problemklasse immer eine schnellere Lösung bei der Nutzung von Quantenalgorithmen liefert. Die Schwierigkeit, mit Quantenalgorithmen zu arbeiten, wird dadurch deutlich. Dies kann auch für andere Problemklassen gelten. Außerdem ist nicht klar, ob optimalere Algorithmen für das gleiche Problem existieren. Das ist ein Ansatz, auf den im Konzept dieser Arbeit eingegangen wird. Kataloge wie PlanQK [Pla23] fassen Quantenalgorithmen für unterschiedliche Probleme zusammen, um einen Überblick über ihre Einsatzbereiche und Anwendungszwecke zu schaffen. Dies kann genutzt werden, falls bekannt ist, in welchem Bereich der Quantenalgorithmus verwendet werden soll. In Kombination dazu können Musterdatenbanken, wie in Abschnitt 3.2 beschrieben, verwendet werden, um weitere Informationen zu erhalten.

Um zu entscheiden, ob ein Quantenalgorithmus zur Lösung eines Problems geeignet ist, muss zuvor festgestellt werden, ob es überhaupt sinnvoll ist, das Problem mit Quantencomputing zu lösen. Arbeiten in diesem Themenbereich vergleichen deshalb klassische Algorithmen mit den

Quantenalgorithmen für das selbe Problem. Loke et al. [LTR+16] führen einen solchen Vergleich anhand von *PageRanking* durch. Ein weiteres Beispiel ist der Vergleich von Quanten- und klassischen genetischen Algorithmen [LC12]. Arbeiten wie QCSplit [MAl22], welche von Mohamad Altaweel vorgestellt wird, gehen einen Schritt weiter und versuchen eine Hilfestellung bei der Lösung des Entscheidungsproblems zu bieten. Das Ziel von QCSplit ist es, ein System zur Unterstützung der Entscheidung, ob ein klassischer Algorithmus oder Quantenalgorithmus zur Lösung eines Problems verwendet werden soll, zu erstellen. Altaweel stellt hierfür einen iterativen Ansatz vor. Es wird mithilfe von Referenzen begonnen, aus welchen die Informationen extrahiert und als Fragen umgeschrieben werden. Die Fragen und dazu passende Antworten werden in einer Datenbank abgespeichert. Dieser Vorgang wird mit Referenzen für weitere Probleme durchgeführt, was die Datenbank ständig erweitert. Dadurch wird die Datenbank immer weiter verfeinert und kann Entscheidungsprobleme mit größerer Wahrscheinlichkeit lösen. Zur Speicherung des Entscheidungsgraphen wird eine Graphdatenbank verwendet. Diese erlaubt die direkte Speicherung von Graphen, was die Struktur auf Kosten von Leistung beibehält. Die Nutzung von Graphen bringt allerdings auch Probleme mit sich. Nicht in allen Fällen kann ein deterministisches Ergebnis erzielt werden. Zusätzlich zu den Fragen, die der Nutzerin/dem Nutzer gestellt werden, wird deshalb noch eine Möglichkeit geboten, eine Problembeschreibung als Text zu formulieren und dazu Referenzen zu finden. Dadurch können anhand von vorhandener Forschung Probleme gruppiert, und Lösungsansätze in den Referenzen in Form von Text gespeichert werden. Mithilfe von Natürlicher Sprachverarbeitung der Referenzen kann ein besseres Ergebnis erzielt werden. Altaweel wählt hierfür die Methode *Keyword Extraction*. Bei dieser Methode werden die wichtigsten Informationen aus den Referenzen in Form von Schlüsselwörtern entnommen. Den Schlüsselwörtern werden anhand einer Skala Wichtigkeiten zugeordnet, die bei der Auswertung berücksichtigt werden müssen. Wie wichtig ein Wort ist, wird durch die Bedeutung des Wortes in Hinsicht auf die Beschreibung von Algorithmen festgelegt. Zum Schluss wird die Ähnlichkeit von Eingabe und Referenzen geprüft, indem diese mittels *Kosinus Ähnlichkeit* von Eingabe und Referenzen berechnet wird. Je größer die Ähnlichkeit, desto höher ist die Übereinstimmung. Der Algorithmus mit der höchsten Übereinstimmung wird als Ergebnis ausgewählt.

Das QCsplit System wird in drei Schichten eingeteilt, welche jeweils das Frontend, Backend und die Datenbank enthalten. QuCFace stellt das Frontend dar und kümmert sich um alle Interaktionen mit der Nutzerin/dem Nutzer, wie der Fragestellung und Auswertung der Texteingabe. Die Datenbank enthält den Entscheidungsgraphen, der zur Problemlösung aufgerufen wird. Um die Interaktionen zu verarbeiten und passende Informationen aus der Datenbank abzurufen, wird QuCMixx verwendet. Dieser Teil vom Backend ist zuständig für die Auswertung des Ergebnisses. Außerdem hat QuCMixx Zugriff auf Referenzen, welche im Falle der Texteingabe zur Natürlichen Sprachverarbeitung verwendet werden. Um Einträge in der Datenbank zu verändern, wird der Service QuPie verwendet, welcher speziell dafür ausgelegt ist. Die Endpunkte dieses Services können von verschiedenen Anwendungen genutzt werden, um die Datenbank iterativ weiter zu verfeinern.

QCSplit enthält viele Parallelen zu dieser Arbeit, da in dieser ebenfalls ein Entscheidungssystem gebaut werden soll. Dieses Entscheidungssystem knüpft thematisch direkt an das Ergebnis von QCSplit an, sollte es sich um die Quantenlösung handeln. Methoden zu *Keyword Extraction* und *Kosinus Ähnlichkeit* für das Ähnlichkeitsmaß zur Lösung des Entscheidungsproblems werden ebenfalls in beiden Arbeiten verwendet, um ein Ergebnis zu erhalten. Die Grundlagen und Funktionsweisen dieser Methoden werden in Abschnitt 2.3 behandelt.

3.2 Muster für Quantencomputing

Die Verwendung von Mustern zur Umsetzung von Quantenalgorithmen ist ein wichtiger Bestandteil dieser Arbeit. Abschnitt 2.2 beschreibt den Aufbau, sowie die Vorteile bei der Verwendung von Mustern für die Darstellung von Informationen. Es existieren viele verschiedene Muster-Sprachen, die die Muster eines Themengebietes zusammenfassen. Die Muster, die für das Quantencomputing verwendet werden, sind unter der Muster-Sprache der Quantencomputing Muster zusammengefasst. Leymann et al. [Ley19] stellen dazu die Kernelemente der Quantencomputing Muster dar. Zu diesen gehören unter anderem das *Initialization* Muster, welches die Initialisierung eines Algorithmus beschreibt, sowie das *Creating Entanglement* Muster, das beschreibt, wie eine Verschränkung erzeugt werden kann. Muster, die oft verwendete Konzepte wie das Orakel, welches wie eine Black-Box funktioniert, oder der *Amplitude Amplification*, die bei wiederholter Ausführung eines Algorithmus basierend auf Annäherung der Lösung bessere Ergebnisse verspricht, sind ebenfalls enthalten. Weitere Muster sind die *Encoding* Muster [WBLS21], wie das *Amplitude-Encoding* Muster oder das *Basis-Encoding* Muster, welche Methoden beschreiben, um den initialen Zustand der Qubits basierend auf den Daten der Eingabe festzulegen. Die Quantencomputing Muster sind vergleichsweise relativ neu, weshalb diese oft mit neuen Gruppen von Mustern erweitert werden. Das *Quantum-Classical Split* Muster ist zum Beispiel der Startpunkt einer solchen Erweiterung. Von diesem Muster ausgehend werden Muster für Hybride Algorithmen [WBLV21], wie dem *Variational Quantum Algorithm* Muster, das einen Ansatz für die Ausführung von hybriden Algorithmen liefert, hinzugefügt. Eine weitere Erweiterung enthält Muster, die die Behandlung von Fehlern beschreiben [BBL+22]. Die neusten Gruppen von Quantencomputing Mustern, sind einerseits die *Execution* Muster [GBB+23], die verschiedene Möglichkeiten zur Ausführung von auf Quantencomputing basierenden Anwendungen beschreiben. Ein Beispiel für ein solches Muster ist das *Standalone Circuit Execution* Muster, welches die Ausführung ohne Integrations- oder Bereitstellungsanforderungen beschreibt. Andererseits wurden die *Quantum Software Development* Muster [BBB+23] hinzugefügt, die auf spezifische Probleme bei der Softwareentwicklung mit Quantenalgorithmen eingehen. Solche Probleme beinhalten das Erstellen von Modulen, die Berechnungen sowohl auf dem Quantencomputer, als auch auf klassischen Computern ausführen müssen. Wie ein solches Modul erstellt werden kann, wird mithilfe des *Hybrid Module* Musters beschrieben. Diese Gruppen von Mustern sind wahrscheinlich nicht die letzten, denn die Forschung zur Erweiterung der Quantencomputing Muster wird fortgeführt.

Eine weiterer Forschungsbereich ist das Speichern von Mustern. Um Informationen von Mustern zu speichern, ist wie für alle anderen Arten von Daten eine Datenbank notwendig. Wegen des Informationsgehalts, sowie dem Zusammenspiel und den Relationen von Mustern, ist ein auf diese angepasstes System notwendig. Der Pattern Atlas ist ein solches System, welches von Leymann et al. [LB21] entwickelt wird. Dieses System umfasst mehrere Muster-Sprachen, sowie die Zusammenhänge zwischen den Mustern und ihre Eigenschaften. Zu diesem Zweck enthält es eine Datenbank von Mustern, die auf dem PatternPedia [FBFL15] beruht. Diese Datenbank beinhaltet die Muster-Sprachen und entsprechende Muster, die vom Patternatlas verwendet werden. Außerdem besitzt die Implementierung des Patternatlas eine Ansicht mit Informationen über die verschiedenen Muster, ihrem Nutzen und Referenzen auf andere Muster. Das ermöglicht eine einfache Übersicht über die Muster und erleichtert dadurch die Nutzung von Mustern zur Unterstützung der Implementierung. Der Patternatlas besitzt eine weitere Ansicht, die einen Graphen enthält. Dieser zeigt die Verbindungen der Muster der ausgewählten Muster-Sprache zueinander.

Die Relationen, welche durch die Kanten im Graphen dargestellt werden, können leicht abgerufen werden. Diese Visualisierung erlaubt es schnell zu erfassen, welche Muster zu einer Gruppe gehören, Ähnlichkeiten haben und welche aufeinander aufbauen.

Verbindungen zwischen verschiedenen Muster-Sprachen sind in dieser Ansicht nicht zu sehen. Das Ziel des Pattern Atlas System ist es deshalb, die Verbindungen zwischen verschiedenen Muster-Sprachen (besser) darzustellen. Die Idee dahinter ist eine sogenannte *Pattern-View*, welche sich aus mehreren Mustern aus verschiedenen Muster-Sprachen zusammensetzt. Dabei verwendete Muster sind nur Teile der gesamten Muster-Sprache und bilden somit einen Ausschnitt dieser. Das Ergebnis ist eine Ansicht, die über die Grenzen der Muster-Sprachen hinausgeht. Leymann et al. definieren diese grenzüberschreitenden Verbindungen als Links, welche Transitionsfunktionen beschreiben. Diese Definition ist von der Kartographie inspiriert, da bei diesem Themengebiet die kurvigen Karten auf dem Globus in ihre zwei-dimensionale Repräsentation transformiert werden.

Die bereits vorhandenen Funktionen des Patternatlas bieten eine passende Grundlage für die Implementierung dieser Arbeit. Die Graphansicht der Muster, spezifisch der Quantencomputing Muster, welche im Pattern Pedia und im Patternatlas enthalten sind, bietet einen ausdrucksvollen Ansatz für die Darstellung der Ergebnisse. Das Prinzip von *Pattern-Views* zur Darstellung eines Ausschnitts der Muster-Sprache ist hilfreich und wird ebenfalls bei der Ergebnisdarstellung verwendet. Die Ansichten werden auf die gegebenen Anforderungen angepasst. Die wichtigen Muster innerhalb der Ansicht können markiert werden und Informationen zu den entsprechenden Mustern stehen ebenfalls zur Verfügung. Deshalb werden Teile des Patternatlas bei der prototypischen Implementierung des Konzepts in dieser Arbeit verwendet und mit weiteren Funktionen erweitert.

3.3 Graph basierte Textdarstellung und Textvergleich

Die Darstellung von Text als Graphen, sowie die Analyse von Graphen und Texten durch Vergleichen sind wichtige Bestandteile der modernen Textverarbeitung. Graphen werden oft zur Darstellung verwendet, weil diese eine gute Übersicht über Daten schaffen. Da viele verschiedene Typen und Vorgehensweisen beim Erstellen von textbasierten Graphen existieren, ist die Auswahl des passenden Graphen umso wichtiger. Osman und Barakub [OB20] untersuchen diese Thematik und fassen die Ergebnisse zu verschiedenen Graph-Repräsentationen und Techniken zusammen. Der erste Punkt, der analysiert wird, ist das Textrepräsentationsschema von Graphen. Dieses wird in vier Typen eingeteilt. Der erste Typ ist das Konzeptuelle Graph Schema [VV04], welches sich durch Flexibilität und Textverständnis auszeichnet, jedoch anhand des Aufbaus auf Daten beschränkt ist. Dieses Schema ist in der Lage, die Semantik von Texten darzustellen. Dazu werden zusätzlich zu den normalen Knoten im Graph spezielle Knoten hinzugefügt, die nur für die Semantik zuständig sind. Das Resultat ist deshalb oft sehr komplex. Der zweite Typ ist der Abhängigkeitsgraph [WNS+11], welcher die Abhängigkeiten von Wörtern zueinander darstellt. Dieses Schema ist zur Verbesserung der Visualisierung von Texten geeignet und ist ebenfalls sehr flexibel, da der Graph eine unabhängige Sprache darstellt. Dadurch kann dieses Schema für verschiedene Sprachen verwendet werden. Ein weiterer Typ ist der *Conzept Frame Graph* [RT02], welcher Beschreibungen für die verschiedenen Abschnitte und Wörter eines Textes liefert. Hierfür sind Vorverarbeitungsschritte, wie die Bestimmung des Wortstamms, notwendig. Obwohl dieses Schema nicht viele Informationen zur Semantik liefert, werden andere Informationen wie die Platzierung eines Wortes innerhalb des Textes relativ zu anderen Wörtern sehr genau dargestellt.

Der letzte Typ ist die Formale Konzeptanalyse [WL08], eine der ältesten Techniken. Dieser Typ erlaubt die Darstellung von Verbindungen zwischen verschiedenen Wörtern. Die Verbindungen werden mithilfe von Ähnlichkeitsmaßen festgelegt. Im Vergleich zu den anderen Schemen ist die Berechnungszeit hier ziemlich hoch.

Unabhängig von den Darstellungsschemen lassen sich Graph Repräsentationen auch in Typen einteilen. Osman und Barakub [OB20] analysieren verschiedene Graph Repräsentationen spezifisch für Dokumente im Internet, wodurch sich mehrere Typen ergeben. Die Standardrepräsentation verwendet Wortstamm-Algorithmen, um jedes Wort als einzigartigen Knoten darzustellen. Die Position im Text relativ zu anderen Wörtern wird durch die Kanten im Graphen angezeigt. Ähnlich zur Standardrepräsentation gibt es eine simplere Version, welche anders als bei der Standardrepräsentation Titel und Metadaten ignoriert und die Kanten nicht beschriftet. Die Anwendungsbereiche dieser beiden Darstellungen sind unter anderem *Zusammenfassen*, *Klassifikation* und *Kategorisierung* von Text. Für Bereiche wie Muster-Erkennung werden Methoden wie die N-Distanz verwendet [SBLK05]. Diese beachtet nur jedes N-te Wort und verbindet sie mit Kanten, welche die Distanz zwischen den Wörtern enthalten. Weitere Methoden, die auf der simplen Standardrepräsentation aufbauen, sind die Absoluten und Relativen Frequenz Repräsentationen. Diese enthalten für jedes Wort zusätzlich noch die Anzahl an Vorkommnissen im Dokument. Ein Anwendungsbereich hierfür ist zum Beispiel das Feststellen der Lesbarkeit eines Textes. Die Knoten eines Graphen müssen aber nicht zwingend nur aus einzelnen Wörtern bestehen, oder gesamte Texte oder Dokumente repräsentieren. Anwendungsbereiche wie die maschinelle Übersetzung verwenden Graphen, die semantische und syntaktische Informationen zu Sätzen enthalten. Auch Sätze als direkter Teil des Graphen können verwendet werden. Ein Anwendungsbereich dafür ist zum Beispiel das Erkennen von Plagiaten. Für Bereiche wie *Keyword Extraction* ist die Koexistenz von Wörtern hilfreich.

Bei der Textanalyse werden einzelne Dokumente nicht nur verarbeitet, sondern müssen auch verglichen werden. Hier kommen Vergleichstechniken ins Spiel. Osman und Barakub [OB20] teilen die Techniken wieder in verschiedene Bereiche auf. Das Vergleichen von Graphen kann strukturell, semantisch oder auf Ähnlichkeit basierend durchgeführt werden. Das strukturelle Vergleichen basiert auf Tiefensuche, welche auch Ähnlichkeiten wie Nachbarknoten im Graph überprüft. Diese Art von Vergleich hat im schlechtesten Fall exponentielle Laufzeit. Um Komplexität zu verringern, wird versucht, unnötige Teilgraphen zu entfernen. Weitere Möglichkeiten, diese Komplexität zu verringern, werden untersucht. Diese verlieren jedoch oft zu viel Genauigkeit und sind deshalb schwer zu verwenden. Das semantische Vergleichen verwendet Graphen mit semantischer Repräsentation von Text. Diese Graphen enthalten sowohl semantische als auch syntaktische Knoten, weshalb diese Technik auf strukturellem Vergleichen aufbaut. Die Beschreibung, beziehungsweise Bedeutung von Knoten und Kanten in einem solchen Graphen wird zum Vergleichen verwendet. Weitere semantische Ansätze verwenden Heuristiken, um unähnliche Knoten und Kanten direkt zu entfernen und davon ausgehend ähnlichere Knoten zu identifizieren. Auf Ähnlichkeit basierendes Vergleichen verwendet Subgraphen mit maximaler Ähnlichkeit, um Ergebnisse zu erhalten. Die Graphen müssen dadurch nicht mehr komplett untersucht werden, da sie sich auf die Teile des Graphen beschränken, die die größte Ähnlichkeit haben. Dabei werden zum Beispiel Distanzmaße verwendet, um die strukturellen Unterschiede von Graphen festzulegen.

Die verschiedenen Vergleichstechniken können je nach Ansatz auf optimale oder approximierte Ergebnisse zielen. Wenn genaue Übereinstimmungen erforderlich sind, können Ansätze verwendet werden, die einen exakten Vergleich durchführen. Dies wird beispielsweise bei Algorithmen verwendet, die zur Suche nach bestimmten Wörtern verwendet werden und gehört zum Bereich

des strukturellen Vergleichs. Alternativ wird grobes, ungenaues Vergleichen verwendet, um beispielsweise unterschiedlich aufgebaute Graphen zu vergleichen. Approximierte Ergebnisse werden beim semantischen Vergleich verwendet, da Graphen mit semantischer Textrepräsentation komplex sind und ein optimales Ergebnis schwer zu finden ist. Ein Anwendungsbereich ist die Bestimmung der semantischen Ähnlichkeit von Texten. Bei auf Ähnlichkeit basierten Vergleichen können beispielsweise die aus dem Vergleich erhaltenen Distanzmaße zur Fehlerkorrektur verwendet werden, was je nach Ansatz mit optimalen oder approximierten Ergebnissen durchgeführt wird. Osman und Barakub [OB20] beschreiben noch offene Probleme der Graphdarstellung von Texten. Zu diesen Problemen gehören unter anderem eine fehlende Analyse von Textdokumenten mit riesigen Ausmaßen, sowie die Forschung nach optimaleren Formen der Textrepräsentation.

Die verschiedenen Darstellungsschemen und Vergleichstechniken bieten eine Hilfestellung für die Umsetzung des Textvergleichs dieser Arbeit, sowie den Umgang mit Graphen zur Darstellung von Informationen. Mithilfe dieser Einblicke werden für diese Arbeit passende Methoden ausgewählt.

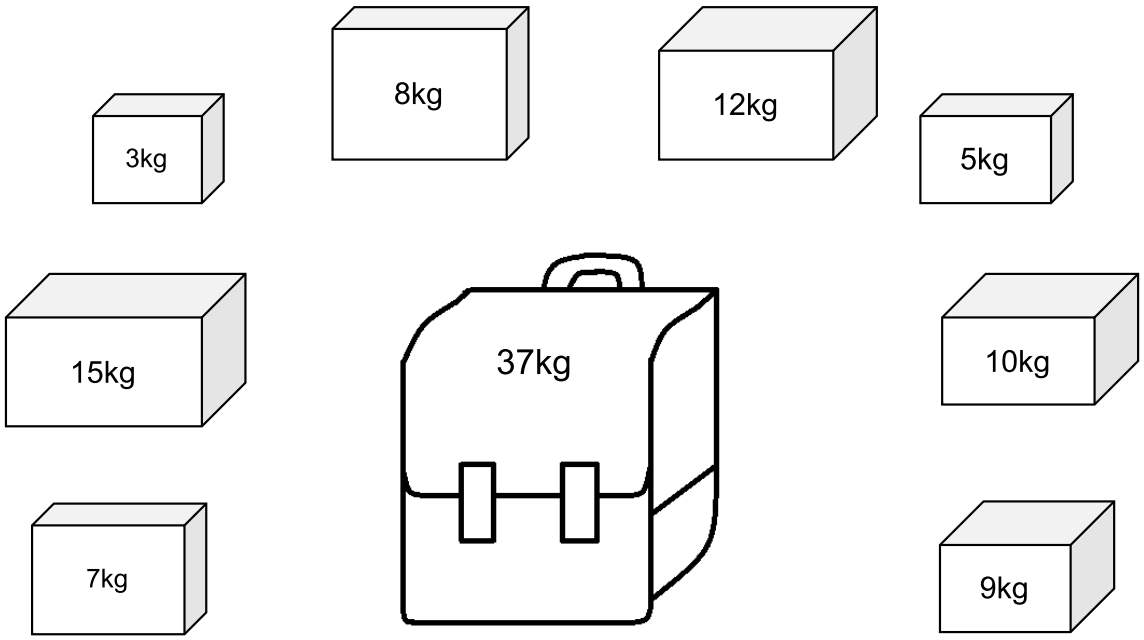

Abbildung 4.1: Rucksackproblem mit Schranke von 37 kg

4 Motivierendes Szenario

In diesem Szenario soll ein Optimierungsproblem, spezifisch das Rucksackproblem [MT90], mithilfe eines Quantenalgorithmus gelöst werden. Bei diesem Problem soll das Gewicht der Objekte, die zum Füllen des Rucksacks verwendet werden, maximiert werden. Die Objekte sind dabei unterschiedlich schwer. Der Rucksack besitzt eine Gewichtsschranke, die nicht überschritten werden darf. Abbildung 4.1 zeigt ein Beispiel für ein solches Problem mit einer Schranke von 37 kg. Es ist zu entscheiden, mit welchen Objekten der Rucksack gefüllt werden soll, um sich so nah wie möglich der Schranke anzunähern. In diesem Beispiel kann durch Ausprobieren verschiedener Kombinationen die Schranke erreicht werden. Eine Möglichkeit ist es, zwei Objekte mit einem Gewicht von 15 kg und ein Objekt mit dem Gewicht von 7 kg zu wählen. Damit wird zusätzlich die Anzahl an Objekten minimiert. Wird eine weitere Einschränkung ergänzt, die nur die einmalige Verwendung eines Objekts erlaubt, wird das Problem komplexer. In diesem Fall gibt es verschiedene Kombinationen, die die Schranke erreichen. Ein Beispiel dafür sind die Objekte mit 15 kg, 10 kg, 9 kg und 3 kg. Alternativ können auch die Kombination von Objekten mit 15 kg, 12 kg, 7 kg und 3 kg verwendet werden. Da das Problem aus Abbildung 4.1 nur eine geringe Schranke und nur wenige verfügbare Objekte hat, kann relativ schnell eine Lösung gefunden werden. Wenn die Schranke viel größer ist und Tausende von Objekten existieren, kann das Problem nicht mehr so einfach gelöst werden. Es werden Algorithmen und ausreichend Rechenleistung benötigt, um eine Lösung zu erhalten. Bei der Verwendung von klassischen Computern kann ein solches Problem zum Beispiel

mithilfe dynamischer Programmierung gelöst werden. Durch Verwendung von Quantencomputern kann das Problem effizienter gelöst werden. Es wird jedoch angenommen, dass kein Vorwissen zu Quantenalgorithmen vorhanden ist.

Um die Implementierung des Rucksackproblems mithilfe von Quantenalgorithmen durchzuführen, muss zuerst herausgefunden werden, welcher Algorithmus dafür geeignet ist. Dies kann zum Beispiel durch eine Suche im Internet herausgefunden werden. Da oft mehrere Algorithmen existieren, die das gleiche Problem lösen können, ist die Suche nicht trivial. Sie führt zu verschiedenen Quantenalgorithmen, wie beispielsweise dem Shor Algorithmus [Sho97], dem VQE [PMS+14] Algorithmus, dem QAOA-Algorithmus [FGG14] und einigen weiteren. Von diesen Algorithmen den geeignetsten zu finden, ist ohne Fachwissen schwierig und erfordert eine ausführliche Recherche zu jedem Algorithmus, da die meisten zur Lösung des Problems verwendet werden können. Deshalb müssen die Algorithmen miteinander verglichen und der passendste ausgewählt werden. Selbst wenn Quantencomputing-Kenntnisse vorhanden sind, kostet dieser Schritt bei mehr als zwei geeigneten Algorithmen viel Zeit. Ist das nicht der Fall, wird noch mehr Zeit für das Recherchieren verwendet. Deshalb lohnt sich allein schon die Automatisierung dieses Prozesses.

Nach der Auswahl des Algorithmus muss dieser implementiert werden. Auch dieser Schritt ist ohne Fachwissen schwierig, da Quantenalgorithmen durch Verwendung von Qubits eine andere Funktionsweise als klassische Algorithmen haben. Zu Beginn der Implementierung sind Ansätze hilfreich, um mit dieser beginnen zu können. Vor allem, wenn der Algorithmus Teil eines größeren Projektes ist. Dazu wird eine Methode verwendet, die auf Mustern basiert. Wie in Abschnitt 2.2 beschreiben, sind diese als Lösungsansatz gut geeignet. Für Implementierungen von Quantenalgorithmen existieren bereits die Muster der Quantencomputing Muster-Sprache. Die Quantencomputing Muster sind gut dokumentiert und leicht zu finden. Die Muster beschreiben jedoch auch komplexere Probleme, die keinen Ansatz für den Beginn der Implementierung liefern. Es ist schwierig, aus der Menge der Muster passende Einstiegspunkte zu finden. Um die sogenannten Einstiegsmuster, die Einstiegspuntke liefern, zu finden, müssen diese zuerst auf den zu implementierenden Quantenalgorithmus eingeschränkt werden. Dazu ist eine Zuordnung von Mustern zum Algorithmus notwendig. Eine solche Zuordnung ist jedoch selten vorzufinden, denn für die meisten Algorithmen werden nur ein oder zwei Muster genannt, die relevant sind. Weitere Muster können nur gefunden werden, indem der Artikel, welcher den Algorithmus vorstellt, genau untersucht wird und mit den Lösungsansätzen der restlichen Muster verglichen wird. Sind die zugehörigen Muster identifiziert, können aus diesen die Einstiegsmuster ausgewählt werden. Diese Vorgehensweise, die eine intensive Untersuchung von Artikeln benötigt, ist ein großer Zeitaufwand. Informationen über Algorithmen und Mustern sind zwar separat vorhanden, können aber ohne die Durchführung einer Zuordnung nicht optimal genutzt werden. Deshalb ist ein System notwendig, welches diese Zuordnung automatisch durchführt und die Informationen über Algorithmen, sowie zugehörige Muster zusammen darstellt. Wird dies mit der Automatisierung des Auswahlprozesses für Algorithmen kombiniert, kann ein Großteil der Zeit für die Implementierung, die für die Recherche verwendet wird, gespart werden. Außerdem macht dieses System das Thema Quantenalgorithmen für Personen zugänglich, die nur wenige Kenntnisse im Bereich der Quanteninformatik haben.

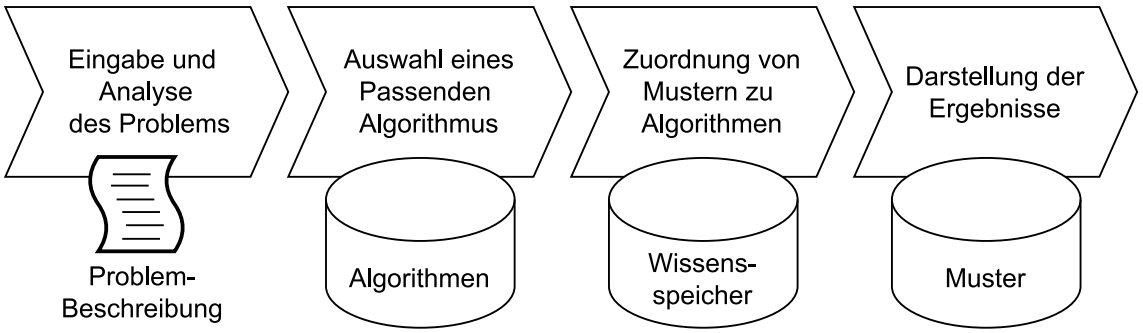

Abbildung 5.1: Ablauf des Konzepts

5 Konzept

In diesem Kapitel wird das Konzept der Arbeit vorgestellt, welches bei der Eingabe eines Problems mithilfe vom Textvergleich passende Quantenalgorithmen sowie zugehörige Muster vorschlägt. Das Kapitel ist in zwei Teile eingeteilt. Abschnitt 5.1 beschreibt die vier Schritte im Ablauf des Konzepts. Die Vorgehensweise innerhalb der jeweiligen Schritte werden dabei erläutert. In Abschnitt 5.2 wird auf die konzeptionelle Umsetzung mithilfe eines Systems eingegangen. Die notwendigen Systembestandteile werden hier beschrieben.

5.1 Ablauf des Konzepts

Der Ablauf des Konzepts ist in vier Schritte eingeteilt, welche in Abbildung 5.1 zu sehen sind. Zuerst wird das Problem beschrieben. Da die Verarbeitung den Textvergleich verwendet, muss die Angabe der Beschreibung des Problems in Textform vorliegen. Die Beschreibung wird ausgewertet und ein passender Algorithmus wird ausgewählt. Die Auswertung erfolgt dabei im nächsten Schritt durch die Durchführung eines Textvergleichs der Eingabe mit den Algorithmenbeschreibungen. Um die Auswertung durchzuführen, werden Informationen zu allen Algorithmen benötigt. Diese müssen in einer Datenbank vorliegen. Der Algorithmus mit dem besten Ergebnis wird ausgewählt. Anschließend werden dem ausgewählten Algorithmus im dritten Schritt passende Muster zugeordnet. Dazu wird ein Wissensspeicher verwendet, welcher eine Zuordnung von zugehörigen Mustern enthält. Ist dieser noch nicht vorhanden, muss er zuerst erstellt werden. Nach der Zuordnung wird der Algorithmus mit den entsprechenden Mustern zur Ergebnisdarstellung weitergeleitet. Im letzten Schritt wird das Ergebnis visuell dargestellt. Informationen zum Algorithmus, sowie zugehöriger Muster, können abgerufen werden. Dazu wird ebenfalls ein Speicher benötigt, der Informationen zu den Mustern enthält, um auf diese zugreifen zu können. Abschnitt 5.1.1 beschreibt genaueres zur Problemangabe. In Abschnitt 5.1.2 wird darauf eingegangen, wie ein passender Algorithmus zur Problemlösung gefunden wird und welche Methoden dafür notwendig sind. Anschließend wird

in Abschnitt 5.1.3 erklärt, wie den ausgewählten Algorithmen passende Muster zugeordnet werden und welche Vorarbeit dafür vorgenommen werden muss. Abschnitt 5.1.4 beschreibt die Darstellung der Ergebnisse, sowie die zur Verfügung stehenden Informationen.

5.1.1 Eingabe und Analyse des Problems

Um mit der Analyse eines Problems zu beginnen, muss zuerst eine Möglichkeit geboten werden, das Problem anzugeben. Deshalb muss zu Beginn eine Benutzerschnittstelle (User Interface) vorhanden sein, um mit der Anwendung kommunizieren zu können. Die Benutzerschnittstelle enthält ein Textfeld, welches zur Eingabe der Problembeschreibung verwendet wird. Damit kann die textuelle Beschreibung des Problems, die mithilfe eines Quantenalgorithmus gelöst werden soll, zur Auswertung an die Anwendung weitergegeben werden. Diese Übertragung wird je nach Zusammensetzung der Anwendung durch Nutzung eines Services durchgeführt. Aufgrund dieser Eigenschaften gibt es keine großen Beschränkungen für die Implementierung. Außerdem ist bei der Nutzung sofort erkennbar, was zu tun ist, da eine geringe Komplexität vorliegt.

Nachdem die textuelle Problembeschreibung angegeben und an das Backend gesendet wird, muss ein Algorithmus anhand der Beschreibung ausgewählt werden. Die Problembeschreibung wird deshalb zuerst analysiert. Hierzu wird in diesem Konzept eine Technik aus dem Bereich der Natürlichen Sprachverarbeitung verwendet. Die gewählte Technik wird *Keyword Extraction* genannt und ist dafür zuständig, die Problembeschreibung zu untersuchen und in Schlüsselwörter aufzuteilen. Dieser Vorgang ist je nach verwendeter *Keyword Extraction* Methode unterschiedlich und beruht in den meisten Fällen auf dem Entfernen von Stoppwörtern und der Auswertung der übrigen Wörter. Anschließend werden die Schlüsselwörter im nächsten Schritt weiterverwendet, um die Auswahl des Algorithmus zu treffen.

5.1.2 Auswahl eines passenden Algorithmus

Nach der Analyse der Problembeschreibung sind die zugehörigen Schlüsselwörter bekannt und können zur Auswahl eines Algorithmus verwendet werden. Für die Auswahl muss die Problembeschreibung mit den vorhandenen Informationen aller Algorithmen verglichen werden. Abbildung 5.2 zeigt den Ablauf dieses Vorgehens. Für die Informationen der Algorithmen muss entsprechend ein Speicher beziehungsweise eine Datenbank vorhanden sein, um diese abrufen zu können. Da verschiedene Aspekte wie Problembeschreibung, Lösungsbeschreibung und Schlüsselwörter der Algorithmen in Betracht gezogen werden, um bessere Ergebnisse bei der Auswertung zu erhalten, muss einfacher Zugriff gewährleistet sein. Dazu wird eine Datenbank verwendet, die alle Informationen über die Algorithmen in Form von Beschreibungen enthält. Um nicht alles Wort für Wort vergleichen zu müssen, werden Techniken aus dem Bereich der Natürlichen Sprachverarbeitung verwendet. Eine geeignete Technik, die zum Vergleichen verwendet werden kann, ist der Textvergleich (Textmatching). Wie schon in Abschnitt 2.3 vorgestellt, existieren verschiedene Möglichkeiten zum Vergleichen von Texten. Die für dieses Konzept gewählte Technik verwendet das Ähnlichkeitsmaß der *Kosinus Ähnlichkeit* für den Vergleich. Den aus der Datenbank entnommenen Algorithmenbeschreibungen werden, wie zuvor der Problembeschreibung in Abschnitt 5.1.1, mithilfe von *Keyword Extraction* die wichtigsten Schlüsselwörter entnommen. Dieser Anwendungsschritt kann im Voraus ausgeführt

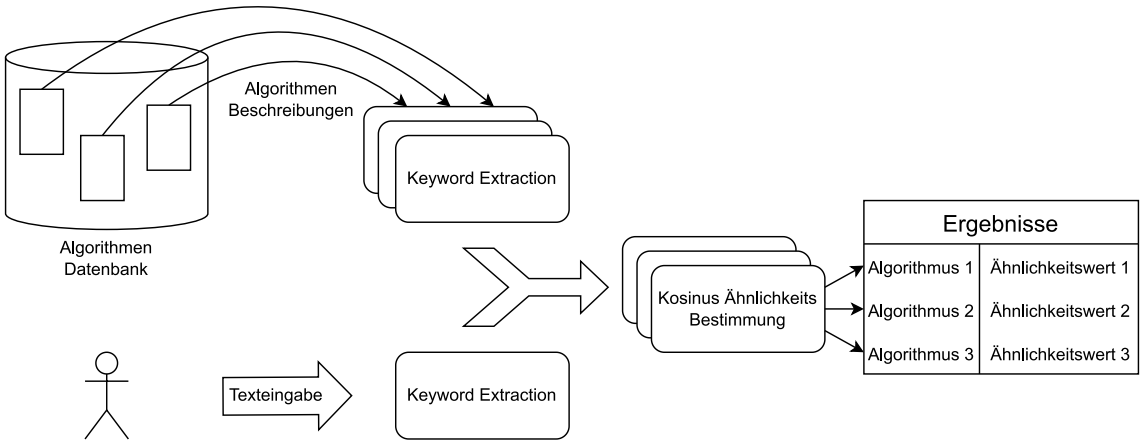

Abbildung 5.2: Ablauf der Algorithmenauswahl

werden, sofern keine Änderungen in der Algorithmendatenbank geschehen. Nach einer Änderung müssen die Schlüsselwörter erneut bestimmt werden. Die gefundenen Schlüsselwörter werden zum Vergleich weitergeleitet. Nach dem Vergleichen werden die Ergebnisse zusammengefasst.

Die Vergleichsmethode, welche die *Kosinus Ähnlichkeit* verwendet, benötigt als Eingabe zwei Vektoren. Diese Methode wird gewählt, da die Ergebnisse beim Anwenden von *Keyword Extraction* auf einfache Weise mit einem Vektor dargestellt werden können. Mithilfe der *Kosinus Ähnlichkeitsformel* wird die Ähnlichkeit zwischen den beiden Vektoren bestimmt. Das Ergebnis liefert dabei eine Zahl zwischen null und eins, wobei die Ähnlichkeit im Fall von eins sehr groß ist, beziehungsweise die Vektoren identisch sind. Im Fall von null haben die Vektoren keine Gemeinsamkeiten. Sie sind dementsprechend unabhängig voneinander. Als Eingabevektoren werden zum einen die Schlüsselwörter verwendet, welche aus der Problemstellung extrahiert werden. Diese Schlüsselwörter werden durch die *Keyword Extraction* Methode als Vektor gespeichert. Zum anderen werden die Beschreibungen der Algorithmen verwendet, welche aus der Datenbank geholt werden. Die Schlüsselwörter dieser Beschreibungen werden ebenfalls extrahiert und als Vektor dargestellt. Dies geschieht separat für jeden in der Datenbank vorhandenen Algorithmus. Zum Bestimmen der *Kosinus Ähnlichkeiten* werden alle Vektoren, die aus den Algorithmen erstellt werden, mit dem Vektor der Eingabe verglichen. Der Algorithmus mit der größten Ähnlichkeit erzeugt den größten Wert. Für das Ergebnis können mehrere Fälle auftreten: (i) Keine Übereinstimmungen, (ii) Ähnliche beziehungsweise Identische Werte, (iii) Ein eindeutiges bestes Ergebnis. Der erste Fall tritt höchstwahrscheinlich auf, wenn die Problembeschreibung zu kurz ist oder die Formulierung nicht ausdrucksvoll genug ist. Falls sie zu kurz ist, können nicht genügend, beziehungsweise keine Schlüsselwörter gefunden werden, was automatisch zu einer Ähnlichkeit von null führt. Wenn die Formulierung nicht ausdrucksvoll genug ist, werden ebenfalls kaum Schlüsselwörter gefunden, was den Vergleich nicht sinnvoll macht, da es an Ausdruckskraft fehlt. Im zweiten Fall gibt es ebenfalls mehrere Möglichkeiten. Es kann das gleiche Problem wie im ersten Fall auftreten, mit dem Unterschied, dass trotzdem einige Schlüsselwörter gefunden werden. Sind diese Schlüsselwörter universal verwendete Wörter, treffen diese auf die meisten Algorithmenbeschreibungen zu. Dadurch haben die meisten Algorithmen ähnliche, beziehungsweise identische Werte für die *Kosinus Ähnlichkeit*. Die andere Möglichkeit ist, dass sich zwei Algorithmen sehr ähnlich sind oder zum Lösen ähnlicher oder sogar derselben Problemen verwendet werden. Dementsprechend ist die *Kosinus Ähnlichkeit* vergleichbar oder sogar identisch. Deshalb sollte bei der Darstellung

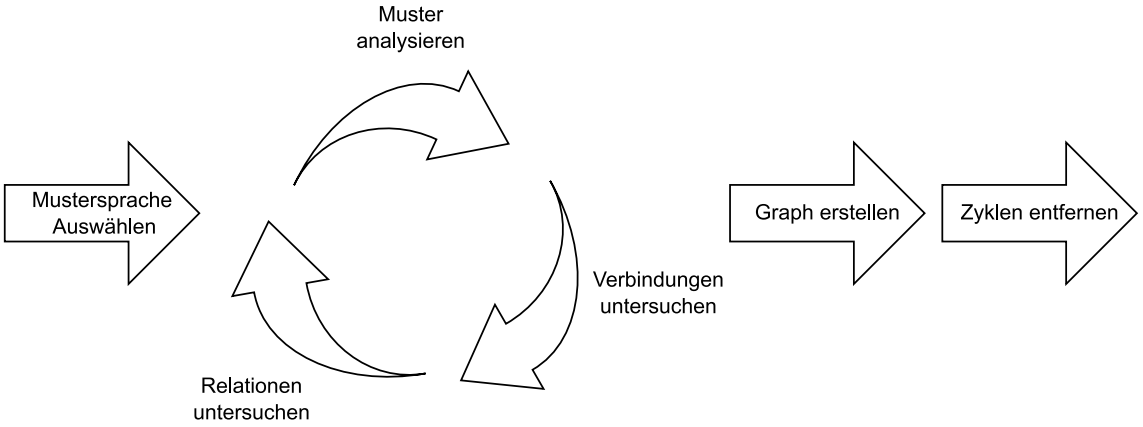

Abbildung 5.3: Notwendige Schritte vor dem Aufbau des Wissensspeichers. Das Ziel dabei ist es, einen azyklischen Graphen zu Erstellen.

der Ergebnisse nicht nur das Ergebnis mit dem größten Ähnlichkeitswert, sondern eine Auflistung mehrerer Ergebnisse mit absteigenden Ähnlichkeitswerten gezeigt werden. In diesem Fall können die relevanten Algorithmen direkt verglichen werden und bieten somit eine größere Anzahl an Möglichkeiten an. Der letzte Fall ist bestenfalls der Normalfall. Hier existiert ein eindeutiges bestes Ergebnis. Da jedoch aufgrund von Fall zwei mehrere Ergebnisse angezeigt werden, ist es notwendig, das beste Ergebnis hervorzuheben. Der Wert der *Kosinus Ähnlichkeit* wird deshalb ebenfalls angezeigt, um die verschiedenen Algorithmen zu differenzieren.

5.1.3 Zuordnung von Mustern zu Algorithmen

Die vom Textvergleich erzielten Ergebnisse erlauben die Auswahl des am besten geeignetsten Algorithmus. Dies ist bereits ein großer Vorteil. Um eine Implementierung dieses Algorithmus zu erleichtern, ist das jedoch nicht ausreichend. Deshalb werden zur Unterstützung noch zugehörige Muster zum ausgewählten Algorithmus angegeben. Als bewährte Methode zur Lösung von Problemen bieten Muster Lösungsansätze für die Implementierung des Algorithmus. Um die Vorteile von Mustern ausnutzen zu können, muss zuerst eine Zuordnung von Mustern zu Algorithmen durchgeführt werden. Dazu wird als Vorarbeit ein Wissensspeicher erstellt, der für jeden Algorithmus alle zugehörigen Muster enthält. Dieser Wissensspeicher muss bestenfalls nur einmal erstellt werden. Alle zukünftigen Algorithmen werden nach dem Zuordnen der Muster zum bestehenden Speicher hinzugefügt, um diesen zu erweitern.

Bevor mit dem Erstellen eines Wissensspeichers begonnen wird, muss zuerst eine ausführliche Analyse von Mustern, ihren Verbindungen untereinander und ihren Relationen durchgeführt werden. Der Ablauf für eine solche Analyse wird in Abbildung 5.3 dargestellt. Im ersten Schritt wird der Rahmen der Analyse festgelegt. Für dieses Konzept ist dieser schon vorgegeben und beinhaltet die Quantencomputing Muster. Der Ansatz kann auch generell verwendet werden, um mehrere Muster-Sprachen zu vergleichen und übergreifend Verbindungen zwischen den Mustern zu finden. Dies überschreitet jedoch den Rahmen des Konzepts und wird deshalb nicht weiter erwähnt. Nach der Auswahl der Muster-Sprache werden die Muster und vor allem die Verbindungen und Relationen zwischen den Mustern betrachtet. Arbeiten wie der Patternatlas [LB21], welcher in Abschnitt 3.2 beschrieben wird, liefern bereits die meisten Verbindungen, die zwischen den Mustern vorhanden

sind, sowie die Muster selbst. Dadurch kann dieser als Datenbank für Muster und Musterrelationen verwendet werden. Nicht alle Relationen aus dem Patternatlas sind jedoch relevant. Die Relationen zwischen den Mustern können zum Beispiel nur die Zugehörigkeit zum gleichen Anwendungsschritt eines Quantenalgorithmus ausdrücken, ohne in Verbindung zu stehen. Deshalb ist es notwendig, die verschiedenen Relationen genauer zu betrachten. Eine Auflistung dieser Relationen ist in Tabelle 5.1 zu sehen. Die Tabelle enthält die Relationen, ihre umgedrehte Version, sowie eine Beschreibung der jeweiligen Relation. Relationen wie *isRelatedTo* beschreiben nur die Verwandtschaft zweier Muster, die sich zum Beispiel durch Beschreibung alternativer Methoden im selben Anwendungsschritt eines Quantenalgorithmus ergibt. Diese Art von Relation erhält deshalb bei der Analyse eine niedrige Priorität und kann, falls notwendig, verändert oder entfernt werden. Außerdem kann sie ohne Nebeneffekte umgedreht werden. Die Relation *uses* verbindet Muster, die eine direkte Abhängigkeit haben. Sie sind meist nicht direkt verwandt, haben aber Eigenschaften, die für das verbundene Muster unerlässlich sind. Das dominante Muster ist hierbei dasjenige, welches das Andere verwendet. Ähnlich dazu gibt es die *canBeUsedWith* Relation. Hier gilt das Gleiche wie bei *uses*, mit dem Unterschied, dass die Verbindung optional ist und kein dominantes Muster existiert. Bei der Zuordnung zu entsprechenden Algorithmen bedeutet das, dass das andere Muster nicht automatisch zum Algorithmus gehört. Im Gegensatz dazu impliziert bei der Relation *uses* die Präsenz des dominanten Musters im Algorithmus, dass das nicht dominante Muster ebenfalls zum Algorithmus gehören muss. Für die Relation *dependsOn* gilt das Gleiche. Sie unterscheidet sich jedoch anhand der Beschreibung, da nur eine Abhängigkeit besteht und die Muster sich nicht direkt gegenseitig verwenden können. Diese beiden Relationen sind semantisch unterschiedlich, erzielen aber bei der Analyse wegen der Abhängigkeit das gleiche Ergebnis. Die letzte wichtige Relation ist *refines*. Sie beschreibt ebenfalls eine direkte Abhängigkeit. Das besondere dabei ist, dass das dominante Muster eine direkte Erweiterung des anderen Musters ist. Bei der Zuordnung zu den Algorithmen besteht die Möglichkeit, anhand solcher Relationen, auch eine Verbindung zwischen Algorithmen festzustellen, die einerseits die erweiterte Form des Musters benutzen und andererseits nicht. Nach der Analyse der Muster und Relationen müssen die Ergebnisse geordnet werden.

Um die beste Übersicht zu erhalten und die Zuordnung zu Algorithmen im nächsten Schritt zu erleichtern, ist es deshalb vorteilhaft, die Muster in einem gerichteten azyklischen Graphen darzustellen. Der Darstellungsgraph des Patternatlas ist beispielsweise nicht inhärent azyklisch. Einige Relationen des Patternatlas müssen deshalb umgedreht werden, um Zyklen zu vermeiden. Dies ist ein weiterer Grund für die ausführliche Untersuchung der Relationen. Relationen wie *isRelatedTo* können ohne Probleme umgedreht werden, da in dieser Relation kein dominantes Muster existiert. Aufgrund der niedrigen Priorität der Relation kann diese auch bei Bedarf entfernt werden, wenn eine stärkere Relation für dieselbe Verbindung existiert, oder wenn eine Verbindung in beide Richtungen gleichzeitig existiert. Beim Umdrehen anderer Relationen muss auf die Namensgebung geachtet werden. Die Relation *uses* muss zum Beispiel zu *usedBy* geändert werden, um die Semantik beizubehalten und das dominante Muster zu bestimmen. Dies gilt auch für die restlichen Relationen. *refines* wird zu *refinedBy* und *dependsOn* wird zu *isDependantOf*. Das Ziel ist es eine Darstellungsform zu erreichen, welche von Mustern ausgeht, die nicht von anderen Mustern abhängig sind. Dazu wird eine Kopie des originalen Graphen erzeugt, welche alle notwendigen Anpassungen enthält. Dieser Graph wird in den Wissensspeicher mit aufgenommen, um die spätere Zuordnung zu erleichtern. Durch die Betrachtung eines Musters können alle weiteren Muster, von denen das Ausgewählte abhängig ist, durch Verfolgung der Pfade gefunden werden.

Relation	Umgedrehte Relation	Beschreibung der Relation
isRelatedTo	isRelatedTo	Muster sind verwandt, aber nicht voneinander abhängig
uses	isUsedBy	Muster benutzt anderes Muster, Abhängigkeit besteht
canBeUsedWith	canBeUsedWith	Muster kann zusammen mit dem anderem Muster verwendet werden, keine Abhängigkeit
dependsOn	isDependantOf	Muster ist abhängig von vorherigem Muster, aber nicht direkt verwandt
refines	is RefinedBy	Muster ist eine Erweiterung des vorherigen Musters
consistsOf	isPartOf	Muster verwendet vorheriges Muster direkt

Tabelle 5.1: Relationstypen von Mustern

Nach dem Entfernen der Zyklen ist erkennbar, dass bestimmte Muster existieren, die Startpunkte des Graphen darstellen. Somit kann zum Beispiel das *Initialization* Muster als erstes generelles Einstiegsmuster der Quantumcomputingpatterns festgelegt werden. Die Zuordnung der Muster zu den Algorithmen erfordert die Analyse des Aufbaus der Algorithmen. Anhand ihrer Beschreibung und Funktionsweise wird entschieden, welche Muster ihnen zugeordnet werden. Ein Beispiel dafür ist die Art der Codierung. Durch Untersuchen eines Algorithmus kann ihm ein *Encodingmuster* anhand der zu verwendenden Art der Codierung zugeordnet werden. Je nach Algorithmus wird zum Beispiel das *Basis-Encoding* Muster oder das *Amplitude-Encoding* Muster zugeteilt. Mithilfe von Arbeiten wie der PlanQK Plattform [Pla23] für Quantenalgorithmen können Informationen für diese einfach entnommen werden. Sind die Informationen nicht ausreichend, weisen Referenzen zu wissenschaftlichen Arbeiten für den entsprechenden Algorithmus hin. Anhand der Untersuchung solcher Arbeiten können die restlichen relevanten Muster zugeordnet werden. Durch die Verwendung der PlanQK Plattform ergeben sich weitere Vorteile, wie die Zusammenhänge zwischen verschiedener Algorithmen. Ist ein Algorithmus etwa eine Erweiterung eines anderen Algorithmus, können die meisten Muster des ersten Algorithmus für den zweiten übernommen werden. Die Änderungen geben ebenfalls Hinweise über weitere Muster, die hinzukommen können. Dadurch kann, ähnlich wie bei den Mustern, eine Hierarchie von Abhängigkeiten zwischen verschiedenen Algorithmen aufgestellt werden. Ist die Zuordnung für einen Algorithmus abgeschlossen, wird diese in den Wissensspeicher aufgenommen. Nach der Aufnahme sind die vorherigen Schritte nicht mehr notwendig. Die Zuordnung steht fest und kann beliebig oft wiederverwendet werden. Wird später ein neuer Algorithmus hinzugefügt, muss nur der Zuordnungsschritt durchgeführt werden. Danach kann dieser einfach in den Wissensspeicher mit aufgenommen werden. Die einzige Ausnahme ist das Hinzufügen neuer Muster zur Muster-Sprache. Das kann durchaus vorkommen, da Muster-Sprachen ständig weiterentwickelt werden. Eine Untersuchung des zuvor erstellten azyklischen Graphen wird dadurch notwendig, denn neue Muster bringen neue Verbindungen zu bestehenden Mustern mit sich, welche wiederum zu Zyklen führen können. Deshalb müssen in diesem Fall einige der zuvor beschriebenen Schritte erneut durchgeführt werden, um den Graphen aktuell zu halten.

Um die Implementierung des Algorithmus noch weiter zu erleichtern, können ein oder mehrere Muster als Einstiegsmuster festgelegt werden. Einstiegsmuster erlauben es, einen Startpunkt festzulegen, an dem die Implementierung beginnen kann. Bei einem Graphen, der nur einen Startknoten hat, kann das Einstiegsmuster direkt gefunden werden. Dies ist jedoch nicht immer der Fall, da nicht jeder Graph von Mustern einer Muster-Sprache diese Form hat. Wie zuvor

beschrieben hat der Graph der Quantencomputing Muster mindestens ein Startpunkt. Dieser ist das *Initialization* Muster. Ob bei der Zuordnung noch weitere Einstiegsmuster hinzukommen, ist allerdings noch offen. Einstiegsmuster sind ein vages Konzept und sind auch nicht immer vorhanden. Um weitere Einstiegsmuster zu finden, wird die von Reinfurt et al. [RFL20] vorgestellte Methode verwendet. Diese Methode schlägt vor, in drei Schritten vorzugehen um Einstiegsmuster ausfindig zu machen. Im ersten Schritt wird das Problem und der Kontext beschrieben und Anforderungen gestellt. Dieser Schritt wird *Situation Assesment* genannt. Im nächsten Schritt, der *Treatment Selection*, wird eine Muster-Sprache ausgewählt und mithilfe der Informationen aus dem vorherigen Schritt Einstiegsmuster ausgewählt. Reinfurt et al. definieren einen Algorithmus für die Wahl eines Einstiegsmusters für den Fall, dass sich bei der Analyse der gesammelten Informationen keines ergibt. Dieser Algorithmus benötigt alle verfügbaren Muster der Muster-Sprache, sowie die aktuelle Situation als Eingabe. Die aktuelle Situation ist ein Set von Mustern, welche eventuell zur Lösung beitragen können. Das Set enthält sowohl Muster, die zur Lösung gehören, als auch Muster, die nicht zur Lösung gehören. Als erstes erstellt der Algorithmus ein weiteres Set, welches alle möglichen Pfade zwischen den Mustern enthält. Jeder dieser Pfade wird auf die aktuelle Situation angepasst. Diese wird so modifiziert, dass alle Muster des Pfades zur Lösung gehören. Im nächsten Schritt wird für jede auf diese Art erzeugt Situation berechnet, wie viele Probleme nicht Teil der Lösung sind. Anschließend werden alle Pfade ausgewählt, welche die größte Übereinstimmung mit der gewünschten Lösung haben. Dies sind die Pfade, die die geringste Menge an Problemen haben, die nicht zur Lösung gehören. Zum Schluss wählt der Algorithmus aus den übrig gebliebenen Pfaden die kürzesten aus. Nach dem Algorithmus sind die Einstiegsmuster genau die ersten Muster der kürzesten Pfade. Mithilfe der Einstiegsmuster kann nun der letzte Schritt, *Treatment Application* genannt, durchgeführt werden. Hier werden alle Pfade, die beim Berechnen der Einstiegsmuster gefunden werden, verfolgt. Dabei werden alle Muster zum Ergebnis hinzugefügt. Liegt eine Situation vor, die mehrere Muster-Sprachen beinhaltet, können alle Muster beider Sprachen als Eingabe verwendet werden. Dies erlaubt die Lösung noch komplexerer Probleme mithilfe des Algorithmus. Die beschriebene Methode kann auch für die Zuordnung von Mustern zu Algorithmen nützlich sein. Für den ersten Schritt der Methode von Reinfurt et al. kann die Algorithmusbeschreibung verwendet werden, da diese bekannt ist. Die Muster-Sprache ist ebenfalls schon festgelegt. Durch Nutzen der Algorithmusbeschreibung zur Auswahl der Muster können diese zur Ausführung des Algorithmus zum Finden von Einstiegsmustern verwendet werden. Das erlaubt einen schnelleren Vorgang bei der Zuordnung von Mustern, da der erste zeitaufwändigste Schritt weitgehend übersprungen wird.

Ein weiterer Zusatz, der sich bei der Zuordnung von Mustern zu Algorithmen ergibt, sind optionale Muster. Optionale Muster sind Muster, die bei der Anwendung eines Algorithmus verwendet werden können, aber nicht zwingend Teil des Algorithmus sind. Sie bieten Erweiterungen für den Algorithmus, die diesen etwas verändern. Diese Veränderungen sind meist nicht groß genug, um den entstandenen Algorithmus als neuen Algorithmus festzulegen. Stattdessen wird der ursprüngliche Algorithmus um das entsprechende Muster erweitert. Ein Beispiel hierfür ist die Verwendung des *Warm Start* Musters beim QAOA-Algorithmus [FGG14]. Die Funktionsweise des Algorithmus bleibt erhalten, aber der Startpunkt beziehungsweise die Art des Starts wird angepasst. Deshalb müssen die optionalen Muster von den restlichen Mustern unterscheidbar sein. Eine Möglichkeit dafür ist es, optionale Muster graphisch anders darzustellen. Dabei muss vorsichtig vorgegangen werden, da ein Muster zwar für einen bestimmten Algorithmus optional sein kann, aber bei anderen Algorithmen nicht optional ist. Die Zuordnung von optionalen Muster ist abhängig vom Algorithmus. Bei der Darstellung müssen sie also dynamisch zugeordnet werden.

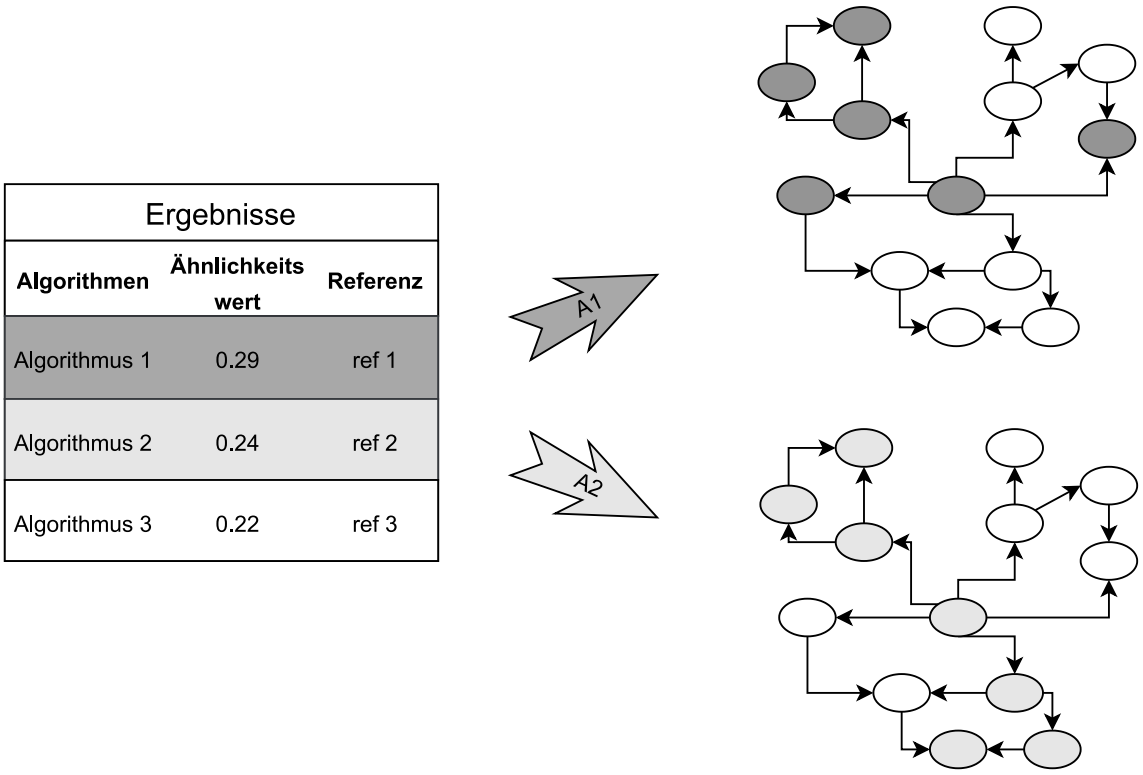

Abbildung 5.4: Illustration der Ergebnisdarstellung. Die Referenzen führen zu Graphen, die den Algorithmen zugeordnete Muster markieren.

5.1.4 Darstellung der Ergebnisse

Nach der Verarbeitung der Eingabe, Auswahl des Algorithmus und Zuordnung der Muster, wird das Ergebnis zuletzt präsentiert. Dazu kann die zur Eingabe verwendete Benutzerschnittstelle wiederverwendet werden, oder eine neue Schnittstelle zur Ergebnisdarstellung wird erzeugt. Da die Ergebnisse wie in Abschnitt 5.1.2 beschrieben aus mehreren Algorithmen bestehen, ist es sinnvoll, sie in Form einer Liste oder Tabelle darzustellen. Das beste Ergebnis wird dabei hervorgehoben. Dadurch sind alle Ergebnisse leicht überschaubar. Auf diese Darstellungsart können jedoch nicht alle Informationen der Algorithmen widergespiegelt werden. Ihre zugehörigen Muster, sowie Einstiegs- und optionale Muster können nicht in einer Tabelle dargestellt werden, da aufgrund der vielen Informationen die Übersicht verloren werden kann. Muster können graphisch besser und übersichtlicher dargestellt werden. Deshalb wird zusätzlich zur Tabellendarstellung ein Graphische Benutzerschnittstelle, kurz GUI, verwendet, um die Muster der Algorithmen zu visualisieren. Um eine hohe Benutzerfreundlichkeit zu erreichen, ist in der Tabelle der Ergebnisdarstellung eine Referenz zum GUI für jeden aufgelisteten Algorithmus vorhanden. Dadurch können beide Schnittstellen miteinander verknüpft werden, was die Navigation möglich macht.

Sind weitere Informationen zu einem Quantenalgorithmus notwendig, muss die entsprechende Referenz ausgewählt werden. Das GUI wird daraufhin geöffnet, welches eine Visualisierung der Muster des Algorithmus zeigt. Außerdem werden die Relationen zwischen den Mustern, sowie Referenzen zu weiteren Informationen über die individuellen Muster dargestellt. Eine passende Visualisierungsform ist dafür notwendig. Die Idee ist deshalb, einen Graphen zur Darstellung von

Algorithmus 5.1 Muster des Teilgraphen festlegen

Require: Set aller Muster $S(P_{all})$, Set aller Muster des Algorithmus $S(P_A)$

```
1: result ← []
2: for all P in S(P_all) do
3:     if P ∈ S(P_A) then
4:         if P is optional then
5:             markiere P als optionales Muster
6:         end if
7:         result ← result ∪ P
8:     end if
9: end for
```

Mustern und ihren Relationen zu verwenden. Mit den Mustern als Knoten und den Relationen als Kanten können die Relationen übersichtlich dargestellt werden. Hier kommt der Graph im Wissensspeicher ins Spiel. Dieser enthält bereits alle existierenden Muster und Relationen. Außerdem ist er schon durch Umformen azyklisch und gerichtet. Das kann zur Darstellung für die Muster der Algorithmen ausgenutzt werden. Abbildung 5.4 zeigt beispielsweise, wie Referenzen von verschiedenen Algorithmen zu unterschiedlichen Graphdarstellungen führen. Wird Algorithmus eins ausgewählt, werden entsprechend zugehörige Muster markiert. Wird Algorithmus zwei ausgewählt, ist eine andere Markierung zu sehen. Die zugehörigen Muster bilden eine Teilmenge des Graphen im Wissensspeicher. Indem diese Muster mit den Mustern des kompletten Graphen geschnitten werden, wird ein Teilgraph erzeugt, welcher die exakt zum Algorithmus zugehörigen Muster enthält. Dadurch kann mit wenig Aufwand für jeden im Wissensspeicher enthaltenen Algorithmus eine Graphdarstellung erzeugt werden. Wie in Algorithmus 5.1 zu sehen, werden dabei die optionalen Muster in Zeile fünf markiert, damit sie später anders dargestellt werden als die restlichen Muster. Die restlichen zugehörigen Patterns werden ebenfalls durch Iteration über alle vorhandenen Muster in Zeile 7 zum Teilgraphen zugeordnet. Die irrelevanten Muster werden nicht angezeigt. Weitere Informationen in Bezug auf die Muster werden ebenfalls im Teilgraphen angezeigt. Wichtige Muster wie die Einstiegsmuster werden besonders behandelt. Durch eine topologische Sortierung des Teilgraphen, können die Einstiegsmuster immer an erster Stelle des Graphen dargestellt werden. Dies erleichtert die Identifizierung.

Algorithmus 5.2 zeigt, wie eine solche Sortierung ablaufen kann. In Zeile zwei wird über die Länge aller zugehörigen Muster des Algorithmus iteriert, um in jedem Fall die Zuordnung aller Muster abzudecken. In jedem Schritt werden alle Muster ohne eingehende Kanten entfernt und dem Teilgraphen zugeordnet. Dazu wird in Zeile sechs und sieben untersucht, ob das Muster eingehende Kanten hat, indem jede vorhandene Kante aller Muster überprüft wird. Ist das Ziel einer Kante identisch mit dem aktuellen Muster, wird eine entsprechende Flagge in Zeile acht gesetzt. Die Zuordnung zum Teilgraphen in Zeile 12 des Algorithmus erfolgt nur, wenn die Flagge nicht gesetzt wurde. Ist dies der Fall, wird das Muster, sowie die aktuelle Iteration in der das Muster als Ergebnis festgelegt wird, zu den Ergebnissen hinzugefügt. Für alle Muster, die in einer Iteration hinzugefügt werden, wird in Zeile 13 eine Markierung gesetzt. Dadurch werden die Muster zum Entfernen festlegt und anschließend aus der Menge der restlichen Muster in Zeile 17 entfernt. Die ausgehenden Kanten dieser Muster werden in Zeile 18 ebenfalls entfernt, wodurch im nächsten Schritt neue Muster ohne eingehende Kanten vorhanden sind. Dies wird so lange wiederholt, bis alle Muster im Teilgraphen angeordnet sind. Zusätzlich können die Einstiegsmuster markiert, oder anders visualisiert werden,

Algorithmus 5.2 Topologische Sortierung zur Erstellung des Teilgraphen

Require: Muster des Algorithmus P_{alg}, Kanten zwischen den Mustern E_{alg}

```
1:  result ← []
2:  for i in range length(P_alg) do
3:      nodesToRemove ← []
4:      for all P in S(P_alg) do
5:          hasNoIncomingEdge = true
6:          for all E in E_alg do
7:              if E.target == P then
8:                  hasNoIncomingEdge = false
9:              end if
10:         end for
11:         if hasNoIncomingEdge then
12:             result.push(node : P, level : i)
13:             nodesToRemove.push(P)
14:         end if
15:     end for
16:     for all node in nodesToRemove do
17:         E_alg ← E_alg.filter(edge => edge.source! = node)
18:         P_alg.remove(node)
19:     end for
20: end for
```

um sie auffälliger zu machen. Außerdem müssen die optionalen Muster eines Algorithmus anders gestaltet werden. Dies muss auf eine Arte und Weise passieren, die den Unterschied zwischen optional und notwendig klarmacht, ohne die Ansicht stark zu beeinflussen. Eine Veränderung der Form des entsprechenden Knotens im Graphen, oder eine leichte Färbung ist deshalb ausreichend. Falls kein Bedarf an optionalen Mustern besteht, kann die Option bereitgestellt werden, diese auch auszublenden. Als Endergebnis wird ein Teilgraph erhalten, der alle Muster, Einstiegsmuster, optionale Muster und Relationen darstellt, die für das Ergebnis, also den ausgewählten Algorithmus relevant sind. Um die zur Verfügung gestellten Informationen zu erweitern, wird für jedes Muster im Graphen noch eine Referenz hinzugefügt, die zur Beschreibung des Musters führt. Zusätzlich zur GUI Referenz wird in der Tabelle noch eine Referenz zur Beschreibung des Algorithmus ergänzt. Dadurch werden alle Informationen zur Verfügung gestellt, die über Algorithmen, Muster, deren Zusammengehörigkeit und Relationen vorhanden sind.

5.2 Umsetzung des Systems

Das Konzept beschreibt bisher nur die Vorgänge der Anwendung. Diese Vorgänge müssen entsprechenden Systembestandteilen zugeordnet werden, um eine funktionale Anwendung zu erhalten. Dazu wird das System in drei Hauptbestandteile eingeteilt. Eine Illustration des Systems ist in Abbildung 5.5 zu sehen. Das Frontend verwendet eine, beziehungsweise mehrere Benutzerschnittstellen für die Eingabe des Problems und die Ergebnisdarstellung. Ein GUI zur visuellen Darstellung der Muster des Ergebnisses ist ebenfalls bei der Ergebnisdarstellung enthalten. Auf Referenzen für das

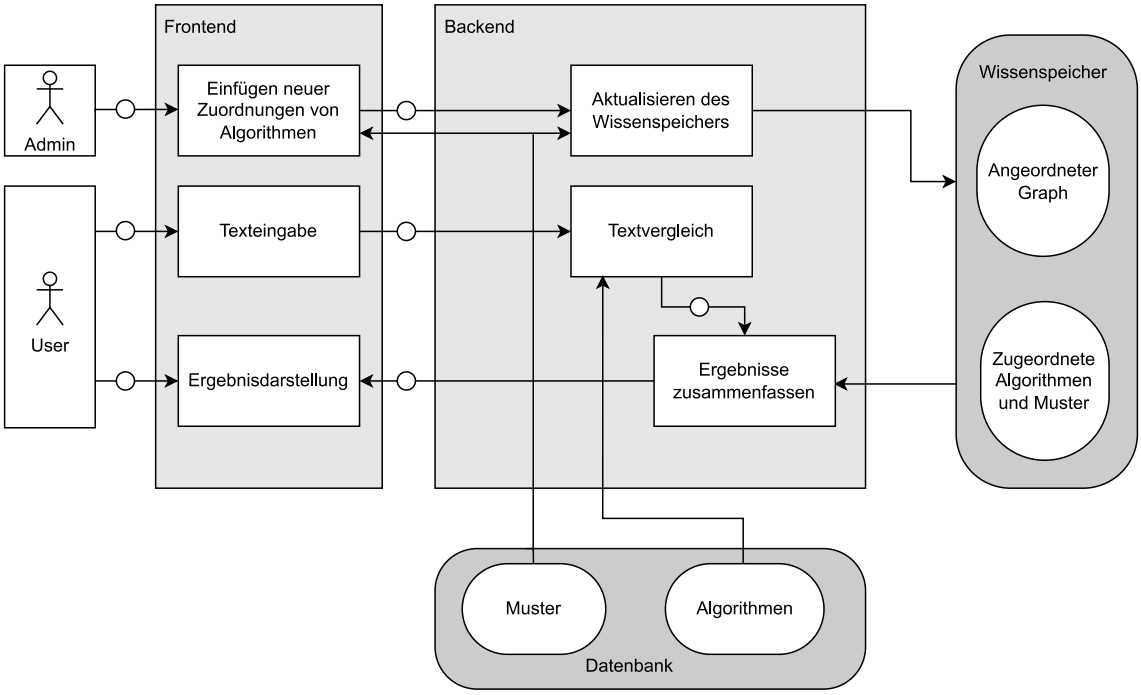

Abbildung 5.5: Systemaufbau

Abrufen von zusätzlichen Informationen kann dort jederzeit zugegriffen werden. Im Backend wird die Eingabe verarbeitet und der Textvergleich durchgeführt. Die Ergebnisse werden zusammengefasst, um sie für das Frontend nutzbar zu machen. Außerdem wird das Hinzufügen neuer Algorithmen und Muster abgehandelt. Ein Wissensspeicher in Form einer Datenbank enthält die zugehörigen Muster zu den Algorithmen, sowie den azyklischen Graphen, der bei der Ergebnisdarstellung genutzt wird. Damit alles einwandfrei verwendet werden kann, sind noch weitere Bestandteile notwendig. Die Kommunikation zwischen den Bestandteilen des Systems muss noch geklärt werden. Außerdem sind noch weitere Daten wie die Algorithmeninformationen für den Textvergleich notwendig. Diese Daten müssen ebenfalls gespeichert werden.

5.2.1 Kommunikation

Ein wichtiger, bisher nicht erwähnter, Bestandteil ist die Kommunikation zwischen den verschiedenen Bereichen der Anwendung. Diese ist je nach Implementierung anders und sorgt für das Zusammenspiel der einzelnen Bestandteile. Die Kommunikation kann zum Beispiel durch Nutzen von Services durchgeführt werden. Vor allem zur Abfrage von Daten aus einer Datenbank ist ein solcher Service notwendig. Je nach Aufbau der Anwendung müssen auch Front- und Backend mithilfe von Services kommunizieren. Die genaue Struktur eines solchen Services ist nicht vorgegeben, weshalb eine beliebige Implementierung dafür verwendet werden kann.

5.2.2 Speicherung von Daten

Der Wissensspeicher, welcher in Abschnitt 5.1.3 genannt wird, enthält nach Erstellung die Informationen der Zugehörigkeit von Mustern zu jedem Algorithmus. Muster werden bei der Zuordnung als Einstiegsmuster oder optionale Muster gekennzeichnet, falls sie eine besondere Eigenschaft haben. Das heißt, dass beim Zugriff auf die Daten eines Algorithmus, nur alle zugehörigen Muster ohne ihre Relationen geliefert werden. Dies ist kein Problem, da im Speicher ebenfalls der Graph aller existierenden Muster existiert. Die Relationen sind dort enthalten. Bei der visuellen Darstellung der Ergebnisse wird auf diesen Graphen zugegriffen, um die Relationen beim Schnitt zu erhalten, wie in Abschnitt 5.1.4 beschrieben. Informationen, wie die Beschreibung von jedem Muster, sind jedoch nicht in diesem Speicher vorhanden. Die Beschreibung von Algorithmen, welche für den Textvergleich benötigt werden, sind ebenfalls nicht in diesem Speicher enthalten. Deshalb sind zwei weitere Datenbanken notwendig, um die weiteren Informationen zu speichern. Alternativ können die Muster- und Algorithmenbeschreibungen in einer Datenbank zusammengeführt, oder ebenfalls in den Wissensspeicher hinzugefügt werden. Dazu liegen keine Einschränkungen vor. Da jedoch für sowohl Musterbeschreibungen als auch Algorithmenbeschreibungen bereits externe Datenbanken existieren, ist es nicht notwendig, diese Informationen extra zu kopieren und abzuspeichern.

5.2.3 Backend

Das Backend ist für die Datenverarbeitung und Zusammenfassen der Ergebnisse zuständig. Informationen über ein Problem werden vom Frontend in Textform erhalten. Mit den Informationen wird der Textvergleich durchgeführt, indem diese mit Algorithmenbeschreibungen verglichen werden. Hierbei werden Vergleichstechniken wie die *Kosinus Ähnlichkeit* verwendet, um die Algorithmen mit dem größten Ähnlichkeitswert zu bestimmen. Für den Vergleich sind die Algorithmeninformation aus der Datenbank erforderlich. Nach der Durchführung des Vergleichs werden die Ergebnisse anschließend intern zusammengefasst. Es werden weitere Informationen, die zur darauf folgenden Darstellung im Frontend notwendig sind, hinzugefügt. Zu diesen Informationen gehören alle zu den Algorithmen gehörige Muster. Nach dem Zusammenfassen werden alle Informationen zur Ergebnisdarstellung an das Frontend geschickt. Dort wird der im Wissensspeicher vorliegende azyklische Graph verwendet, um die visuelle Darstellung im Frontend zu erleichtern.

5.2.4 Frontend

Das Frontend ist für die Eingabe des Problems in Textform, sowie die Ergebnisdarstellung zuständig. Durch eine Benutzerschnittstelle kann das zu lösende Problem angegeben werden. Die Schnittstelle sendet die Problembeschreibung zur Verarbeitung an das Backend. Sind die Ergebnisse vorhanden, werden diese vom Backend an das Frontend geschickt. Im Frontend werden die Ergebnisse visuell Dargestellt. Diese Darstellung enthält Graphen, welche alle relevanten Muster für einen Algorithmus enthalten, sowie optionale Muster und Einstiegsmuster kennzeichnen.

5.2.5 Erweitern des Wissensspeichers

In einem sich ständig weiterentwickelndem Feld der Forschung wie der Quanteninformatik ist es unabdingbar, dass häufig neue Algorithmen oder sogar neue Muster entstehen. Deshalb ist eine Funktion zum Aktualisieren der Datenbankeinträge notwendig. Je nach Aufbau der Anwendung ist eine solche Funktion für die Datenbanken, die Algorithmeninformationen und Musterinformationen enthalten, notwendig. Dies ist der Fall, wenn die Datenbanken nicht extern vorliegen. Im anderen Fall fällt das nicht in den Aufgabenbereich der Anwendung. Die Datenbank, welche den Wissensspeicher mit der Zuordnung von Mustern zu Algorithmen enthält, benötigt in jedem Fall diese Funktion. Damit die Funktionalität der Anwendung erhalten bleibt, müssen so viele Informationen wie möglich über die neuen Algorithmen beziehungsweise Muster bekannt sein. Der Textvergleich liefert genauere Ergebnisse, wenn mehr Informationen als nur die Beschreibung zur Verfügung stehen. Die Verbindungen der Muster untereinander und die Zugehörigkeit von Mustern zum neu hinzugefügten Algorithmus müssen bekannt sein, um diese zum Wissensspeicher hinzuzufügen. Das Hinzufügen dieser Informationen erfordert eine Interaktion mit der Nutzerin/dem Nutzer der Anwendung. Deshalb wird eine weitere Benutzerschnittstelle benötigt, welche die Möglichkeit bietet, diese Angaben zu machen. Da dadurch eine Änderung am Wissensspeicher ermöglicht wird, ist es sinnvoll, den Zugriff auf diese Benutzerschnittstelle einzuschränken. Um diese Änderung durchzuführen und neue Einträge in die Datenbank hinzuzufügen, ist zusätzlich ein Systembestandteil im Backend notwendig. Die Bestandteile benötigten beide Zugriff auf die Datenbank, welche die Muster enthält. Um die Zugehörigkeit zu einem neuen Algorithmus festzulegen, wird eine Liste der Muster angegeben, die anhand der Datenbank erstellt wird. Durch Auswählen der entsprechenden Muster ordnet die Anwendung diese dem neuen Algorithmus zu, bevor er im Wissensspeicher abgespeichert wird. Zusätzlich kann ausgewählt werden, welche Muster optional sind. In Abschnitt 5.1.3 wird bereits erwähnt, dass beim Erweitern der Muster-Sprache mit neuen Mustern Probleme im Graphen des Wissensspeichers auftreten können. Deshalb muss der Graph im Wissensspeicher durch das Backend aktualisierbar sein, um Komplikationen zu vermeiden.

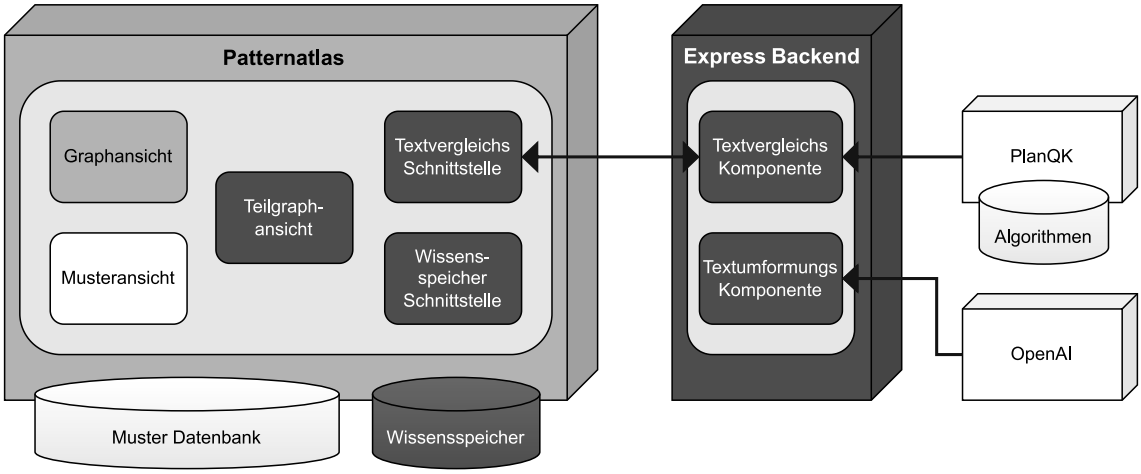

Abbildung 6.1: Systemarchitektur der Implementierung (Weiß: bereits vorhandene Bestandteile, Grau: erweiterte Bestandteile, Schwarz: neue Bestandteile)

6 Implementierung

Dieses Kapitel beschreibt eine prototypische Implementierung des in Kapitel 5 vorgestellten Konzepts. Der Prototyp ist auf GitHub[1] zu finden. Im Folgenden werden die konkreten Bestandteile, die für die Implementierung verwendet werden, beschrieben. Die Implementierung beruht auf dem in Abschnitt 3.2 erwähnten Patternatlas [LB21]. Dadurch können die Vorteile, durch die Nutzung der vorhandenen Patternatlas Implementierung auf Basis von Angular [Goo23], ausgenutzt werden. Abbildung 6.1 zeigt eine Übersicht der implementierten Architektur. Die Implementierung besteht hauptsächlich aus der Erweiterung des Patternatlas, die in Abschnitt 6.1 beschrieben wird, sowie der Express Implementierung des Backends, auf die in Abschnitt 6.2 eingegangen wird. Abschnitt 6.3 beschreibt die Speicherung von Daten. Der Patternatlas wird um eine Schnittstelle zur Kommunikation mit dem Backend, einer Schnittstelle für den Zugriff auf den Wissensspeicher, sowie einer Ansicht zur visuellen Darstellung der Ergebnisse erweitert. Die Graphansicht des Patternatlas wird ebenfalls angepasst, um mit den neuen Bestandteilen zu interagieren. Das Express Backend enthält die Funktionalität des Textvergleichs und kann über die Schnittstelle mit dem Patternatlas kommunizieren, um Ergebnisse zu übermitteln. Zusätzlich enthält das Backend eine Textumformungskomponente zur Anpassung der Eingabe, falls eine Negation enthalten ist.

[1] https://github.com/Tpankv4/QAIsup

6.1 Erweiterung des Patternatlas

Wie bereits im vorherigen Abschnitt erwähnt, verwendet die Implementierung des Konzepts den Patternatlas [LB21] als Basis. Genauer werden nur die Bereiche verwendet, welche die Quantencomputing Muster enthalten. Der Grund für die Wahl des Patternatlas ist zum einen die Graphansicht, welche die Verbindungen und Relationen zwischen den verschiedenen Mustern zeigt. Diese enthält bereits einen kompletten Graphen für die gesamte Muster-Sprache, weshalb dieser nicht zusätzlich erstellt werden muss. Zum anderen enthält der Patternatlas eine Informationsansicht für jedes einzelne Muster. Diese Musteransicht ist bereits vollständig und muss nicht erweitert werden. Deshalb ist der Patternatlas für die Darstellungen der Ergebnisse hervorragend geeignet, da Informationen über Muster, die benötigt werden, direkt abrufbar sind. Weitere Muster können mit nur einem Mausklick auf die entsprechenden Referenzen aufgerufen werden.

Zur Ergebnisdarstellung bei der Auswahl eines Algorithmus wird die Graphansicht des Patternatlas verwendet. Diese wird angepasst, um mit den neuen Bestandteilen der Implementierung interagieren zu können. Die Graphansicht besitzt die Fähigkeit, beim Auswählen eines Musters, dieses hervorzuheben. Das wird ausgenutzt, um bei der Ergebnisdarstellung nach Bedarf mehrere Muster in den Vordergrund zu bringen. Um die Ergebnisdarstellung umzusetzen, wird eine Combobox hinzugefügt, welche alle im Wissensspeicher vorhandenen Algorithmen enthält. Bei der Auswahl eines Algorithmus werden die zum Algorithmus gehörigen Muster, sowie ihre Verbindungen zueinander im Graphen hervorgehoben. Die Opazität der nicht relevanten Muster wird dabei heruntergesetzt. Somit besteht eine klare Übersicht über die zum Algorithmus gehörigen Muster. Eine weitere Erweiterung ist das Hinzufügen einer Ansicht, welche die Reihenfolge der Muster für die Implementierung des Algorithmus und ihre Abhängigkeiten zueinander zeigt. Diese Teilgraphansicht wird mithilfe eines Angular [Goo23] Dialogs implementiert, der direkt mit dem Rest der Anwendung kommunizieren kann. Der Dialog kann über die entsprechende Schaltfläche geöffnet werden. Im Dialog ist ein Teilgraph der gesamten Quantencomputing Muster zu sehen, welcher nur die zum ausgewählten Algorithmus zugehörigen Muster enthält. Bei der Erstellung des Teilgraphen wird wie im Konzept vorgegangen. Eine Besonderheit ist hier, dass zuerst die Richtung aller Relationen umgedreht wird, da die komplexeren Muster standardmäßig fast nur ausgehenden Kanten im Graphen haben, was später bei der Anordnung Probleme macht.

Durch topologische Sortierung wird der Teilgraph geordnet. Diese Sortierung wird mit einem Algorithmus ähnlich dem aus Algorithmus 5.2 durchgeführt. Für den Algorithmus werden als Eingabe alle Muster für die Erstellung des Teilgraphen, sowie alle zwischen ihnen vorhandenen Kanten benötigt. Zuerst wird überprüft, ob ein Muster eingehende Kanten hat. Falls nicht, wird es zum ersten Level des Teilgraphen hinzugefügt. In diesem Fall wird es aus der Menge der noch vorhandenen Muster entfernt. Anschließend werden alle ausgehende Kanten der im vorherigen Schritt gefundenen Muster entfernt. Dies geschieht so lange, bis alle Muster angeordnet sind. Wie im Konzept beschrieben, werden die Einführungsmuster und optionale Muster anders gekennzeichnet. Die Einführungsmuster sind dabei als Erstes im Graphen abgebildet und werden im ersten Durchlauf des Algorithmus gefunden. Die optionalen Muster können bei Bedarf versteckt werden, um die Unterschiede bei der Verwendung dieser Muster besser darzustellen. Das Verstecken erfolgt über eine Schaltfläche in der Teilgraphansicht. Referenzen zur Musteransicht eines jeweiligen Musters im Patternatlas sind ebenfalls vorhanden. Das Gleiche gilt für den Algorithmus selbst, der eine Referenz zur Quelle der Algorithmusbeschreibung enthält. Diese weist auf die PlanQK [Pla23] Plattform hin, welche Informationen zu vielen verschiedenen Quntenalgorithmen besitzt.

Zur Kommunikation mit dem Backend Server wird eine Schnittstelle hinzugefügt. Diese ist für das Senden der Eingabe der Problembeschreibung zuständig. Diese Schnittstelle wird in Form eines Angular Dialogs implementiert, welcher ein Textfeld für die Eingabe enthält. Nach der Eingabe wird diese an das Backend geschickt. Eine weitere Schnittstelle ist für die Erweiterung des Wissensspeichers zuständig. Diese wird ebenfalls durch einen Angular Dialog implementiert und ermöglicht das Hinzufügen sowie das Entfernen von Algorithmen aus dem Wissensspeicher. Beim Hinzufügen wird ausgewählt, welche Muster zum neu hinzuzufügenden Algorithmus gehören und welche dieser Muster optional sind. Beim Entfernen steht eine Liste der im Wissensspeicher enthaltenen Algorithmen zur Verfügung, aus der einer oder mehrere zu löschende Algorithmen ausgewählt werden können, falls notwendig. Beide Schnittstellen verwenden den in Angular integrierten *Angular HttpClient*, welcher die benötigten REST Funktionen implementiert.

6.2 Express Backend

Die Implementierung des Backends erfolgt mithilfe von Express[2]. Express bietet ein Gerüst zur Implementierung von kleinen Anwendungen und Services auf Basis von *Node.js*[3]. Node.js ist eine Laufzeitumgebung, welche die Programmiersprache Javascript verwendet. Mithilfe von Express und *Node.js* wird das Backend als Server implementiert. Dieser Server kommuniziert durch eine kleine Anwendungsschnittstelle mit der entsprechenden Patternatlas Schnittstelle. Auf diese Art werden die Informationen für den Textvergleich weitergegeben. Das Backend beinhaltet eine Textvergleichskomponente, welche den Textvergleich durchführt, sowie eine Textumformungskomponente, die gegebenenfalls die Texteingabe anpasst oder umformuliert.

Um mit dem Textvergleich zu beginnen, muss der Textinhalt zuerst angepasst werden. Dazu werden mithilfe eines *keyword extractors* die Schlüsselwörter der Eingabe bestimmt, ebenso wie die Schlüsselwörter aller Algorithmen, beziehungsweise ihrer Beschreibungen. Diese Beschreibungen werden mittels HTTP-Aufrufen aus der Algorithmen Datenbank von PlanQK [Pla23] erhalten. Es stehen zwei *keyword extractor* Methoden zur Verfügung. Eine davon ist eine generische[4], welche eine Liste der gängigsten Stoppwörter verwendet, um zwischen wichtigen und unwichtigen Wörtern eines Textes zu unterscheiden. Als Ergebnis wird ein Array erzeugt, welches die Schlüsselwörter enthält. Die andere Methode verwendet Rake [RECC10], um Schlüsselwörter zu finden. Rake verwendet ebenfalls eine Liste von Stoppwörtern, um die unwichtigen Wörter zu entfernen. Zusätzlich dazu werden die restlichen Wörter in sogenannte *Candidate keywords* eingeteilt. Diese bestehen aus einzelnen oder mehreren Wörtern. Für diese Schlüsselwörter wird eine *Co-Occurence* Matrix berechnet, welches das gleichzeitige Vorkommen der Wörter enthält. Anhand dieser Matrix werden den Schlüsselwörtern eine Punktzahl zugeordnet, welche anhand von verschiedenen Metriken, wie der Häufigkeit des Vorkommens im Text, sowie dem Vorkommen in längeren, aus mehreren Wörtern bestehenden Schlüsselwörtern, berechnet wird. Die N Schlüsselwörter mit den höchsten Punktzahlen werden als Ergebnis gewählt, wobei N meist ein Drittel aller gefunden Schlüsselwörter entspricht. Rake *Keyword Extraction* liefert qualitativ hochwertigere Schlüsselwörter als die generische Methode, generiert vergleichsweise jedoch weniger Schlüsselwörter. Die generische Methode hat dagegen eine größere Anzahl an Schlüsselwörtern.

[2] https://expressjs.com/

[3] https://nodejs.org/de

[4] https://www.npmjs.com/package/keyword-extractor

Methoden	Endpunkte
einfacher keywordextractor	/api/matcher
einfacher keywordextractor + OpenAI	/api/matcher/openai
Rake	/api/matcher/rake
Rake + OpenAI	/api/matcher/rake/openai

Tabelle 6.1: Endpunkte des Backend Servers

Die Schlüsselwörter werden anschließend verwendet, um die *Kosinus Ähnlichkeit* zu berechnen. Da die Schlüsselwörter in einem Array vorliegen, muss zuerst die Anzahl von identischen Wörtern festgestellt werden. Dazu wird eine Map erstellt, die für jedes Wort im Array die Anzahl des Auftretens des Wortes zuordnet. Das Vorkommen der Wörter für jeden Algorithmus wird dadurch zusammengefasst. Um im nächsten Schritt die *Kosinus Ähnlichkeit* zu bestimmen, muss noch eine Umformung durchgeführt werden. Zur Berechnung dieses Ähnlichkeitswertes sind zwei Vektoren notwendig. Deshalb müssen die Einträge in der Map in einem Vektor abgespeichert werden. Dabei ist es wichtig, dass jeder Index in jedem Vektor immer für dasselbe Wort steht, da der Vergleich sonst nicht durchgeführt werden kann. Sind der Algorithmus und die Eingabe als Vektoren vorhanden, wird die *Kosinus Ähnlichkeit* nach der folgenden Formel berechnet.

$$cos(\alpha) = \frac{vecA * vecB}{\|vecA\| * \|vecB\|}$$

Der Winkel α stellt dabei den Winkel zwischen den beiden Vektoren dar. Je näher das Ergebnis der Berechnung an eins liegt, desto ähnlicher sind die Vektoren. Bei einem Ergebnis von exakt eins sind die Vektoren identisch, während sie bei einem Ergebnis von null orthogonal zueinander sind. Das beste Ergebnis hat dementsprechend den größten Wert.

Durch die Verwendung von Textvergleichs Methoden zur Bestimmung der Ergebnisse, sind diese stark von der Formulierung der Texteingabe, die für den Vergleich verwendet wird, abhängig. Enthält die Eingabe eine Negation, wird diese nicht erkannt, da Wörter, wie zum Beispiel *nicht*, bei der Bestimmung der Schlüsselwörter entfernt werden. Wird beispielsweise die Eingabe *Ich benötige einen Algorithmus, der nicht das Minimum bestimmt* getätigt, werden als Schlüsselwörter mit hoher Wahrscheinlichkeit nur *Algorithmus* und *Minimum* gefunden. Dementsprechend fällt die Negation weg und das Ergebnis enthält die falschen Informationen. Um dieses Problem zu umgehen, enthält das Backend die Textumformungskomponente. Diese verwendet das Sprachmodell *GPT-3.5 Turbo* von OpenAI[5], um die Eingabe umzuformulieren, damit keine Negationen enthalten sind. Ist die Optionen, OpenAI zu verwenden ausgewählt, wird die Eingabe an OpenAI gesendet, um die Umformulierung durchzuführen. Um diese Funktionalität nutzen zu können, ist ein persönlicher API-Schlüssel von OpenAI erforderlich. Das Ergebnis wird anschließend an die Textvergleichskomponente weitergegeben und wird wie zuvor beschrieben weiterverarbeitet.

In der Textvergleichsschnittstelle im Patternatlas kann ausgewählt werden, welche Methode zur Bestimmung der Schlüsselwörter verwendet werden soll. Zusätzlich kann bestimmt werden, ob die OpenAI Funktionalität zum Umformen der Eingabe verwendet werden soll. Nach der Auswahl wird der entsprechende Endpunkt des Backend Servers angesprochen. Eine Liste der Endpunkte ist in Tabelle 6.1 zu sehen. Alle Endpunkte benötigen als Eingabe einen Text, für den der Vergleich

[5] https://openai.com/

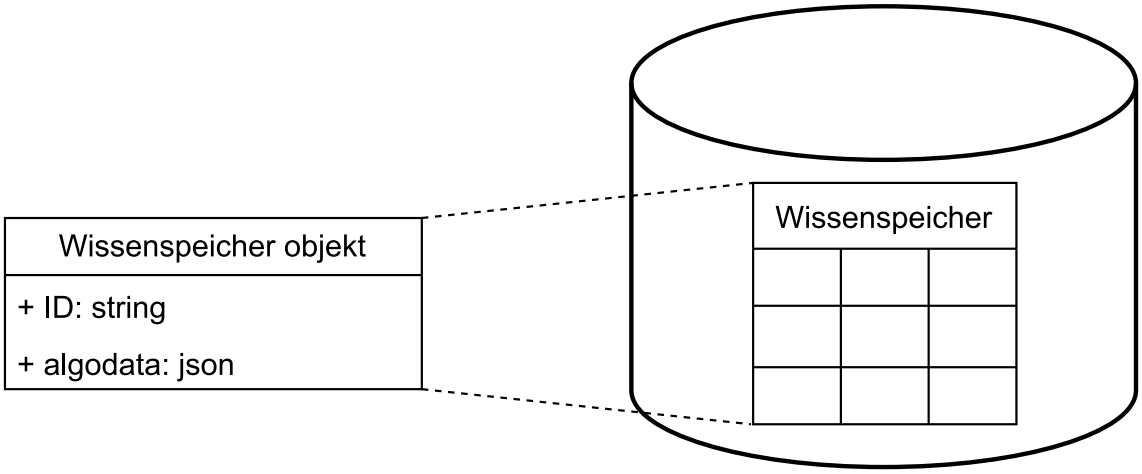

Abbildung 6.2: Abbildung des Wissensspeichers in die Datenbank

durchgeführt werden soll, sowie die Algorithmeninformationen von PlanQK. Als Ergebnis wird ein Array von Objekten gesendet, welches die Namen der Algorithmen, sowie ihre Ähnlichkeitswerte enthalten. Die Objekte im Array sind bereits absteigend nach ihren Ähnlichkeitswerten sortiert.

6.3 Datenbank

Algorithmen, Muster und Relationen zwischen den Mustern müssen immer verfügbar sein, damit sie für den Textvergleich und die Ergebnisdarstellung verwendet werden können. Dazu müssen diese Informationen in einer Datenbank vorliegen. Die Datenbank, welche die Musterinformationen enthält, ist eine Implementierung von PostgreSQL [The23] und gehört zum Patternatlas. PostgreSQL ist eine der größten relationalen open-source Datenbanken, welche bereits seit über 30 Jahren verfügbar ist und immer noch weiterentwickelt wird. Diese Bewährtheit macht sie zu einem guten Kandidaten zum Speichern und Verwalten von Daten. Als relationale Datenbank liegen die Musterinformationen in Tabellen vor, welche diese zum einen zu einer Muster-Sprache zuordnen und zum anderen ihre Relationen zueinander beschreiben. Die Patternatlas Implementierung enthält die Daten zu verschiedenen Muster-Sprachen. Für diese Implementierung sind aber nur die Muster der Quantencomputing Muster-Sprache relevant. Diese werden mithilfe eines HTTP-Services in Angular [Goo23] aus der Datenbank geholt. Der Zugriff auf die Datenbank erfolgt dabei mithilfe eines Java Persistence APIs[6]. Diese API verwendet das Hibernate[7] Framework, was eine Abbildung von Java Objekten auf Tabellen in der Datenbank erlaubt. Die Informationen der Muster können durch Zugriff auf das JPA aus der Datenbank erhalten werden. Der Wissensspeicher, welcher die Zuordnung von Algorithmen und Mustern enthält, wird als Tabelle in der PostgreSQL Datenbank des Patternatlas umgesetzt, wie in Abbildung 6.2 zu sehen. Dies erlaubt eine Interaktion mit den anderen Informationen der Muster, falls notwendig. Um das möglich zu machen, muss das JPA um ein Objekt erweitert werden, welches den Wissensspeicher repräsentiert. Dieser muss, genauso wie die Muster, durch passende Annotationen auf eine Tabelle in der Datenbank abgebildet

[6] https://www.oracle.com/java/technologies/persistence-jsp.html
[7] https://hibernate.org/

werden. Da die Informationen im Wissensspeicher aus Zuordnungen von Algorithmen und Mustern bestehen, sind diese entsprechend komplex. Deshalb werden die Informationen als JSON Daten gespeichert, was durch Typdefinitionen von Hibernate möglich gemacht wird. Das Objekt, welches den Wissensspeicher repräsentiert, muss also einen entsprechenden Datentypen besitzen, welcher JSON Daten zugeordnet werden kann. Der gewählte Datentyp ist eine Liste, welche Maps enthält. Jeder Eintrag in der Liste entspricht der Information zu einem Algorithmus. Die Maps besitzen einen String als Schlüssel und ein Objekt als Wert. Mithilfe dieses Datentyps lassen sich JSON Daten speichern. Zusätzlich dazu besitzt das Objekt eine ID, welche den primären Schlüssel in der Tabelle darstellt. Dieser primäre Schlüssel der Wissensspeicher Tabelle entspricht dabei der ID der Quantencomputing Muster-Sprache, was eine doppelte Erstellung der Tabelle verhindert. Dadurch existiert jederzeit nur ein Wissensspeicher. Muss dieser durch Hinzufügen eines neuen Algorithmus und entsprechenden Mustern erweitert werden, kann die Tabelle einfach aktualisiert werden, indem mithilfe des JPA auf das zur Tabelle gehörige Objekt zugegriffen wird.

Der Wissensspeicher besitzt nach der Beschreibung im Konzept eine Zuordnung von Mustern zu Algorithmen. Um die Implementierung verwenden zu können, muss dieser Speicher zuerst gefüllt werden. Dazu wurden verschiedene Algorithmen aus der Datenbank von PlanQK [Pla23], sowie Artikel zu den entsprechenden Algorithmen auf Merkmale untersucht, die auf Quantencomputing Muster hindeuten. Das Ergebnis der Untersuchung für mehrere Algorithmen ist in Tabelle 6.2 zu sehen. Auf diese Art alle relevanten Quantencomputing Muster ausfindig zu machen ist schwierig, da oft nicht alle Aspekte, die auf bestimmte Muster oder Mustergruppen andeuten, auf PlanQK oder in den Artikeln der Algorithmen angesprochen werden. Deshalb ist keine Garantie vorhanden, dass für jeden untersuchten Algorithmus alle relevanten Muster gefunden wurden. Einige Quantenalgorithmen bauen auf anderen auf, was bei der Musterzuweisung berücksichtigt werden kann. Ein Beispiel hierfür ist der HHL-Algorithmus [HHL09], der die Quantum Fourier Transformation [NC10] verwendet. Weitere Algorithmen sind Varianten der Basisversion. Dazu gehören verschiedene Anwendungen des *Quantum Annealing* Algorithmus [Cha17], wie das Lösen des *Quadratic Assignment Problems* [CDA22]. Diese sind größtenteils nicht in Tabelle 6.2 aufgelistet. Außerdem ist anzumerken, dass das *Initialization* Muster aus Platzgründen in der Tabelle weggelassen wurde, da dieses Muster bei den meisten Quantenalgorithmen vorkommt.

Quantenalgorithmen	Quantencomputing Muster
Anomaly Detection with QGANs [HOR21]	Variational Quantum Algorithm, Quantum Classical Split, Alternating Operator Ansatz
Deutsch Algorithm [DR92]	Uncompute, Oracle, Uniform Superposition, Creating Entanglement, Basis Encoding, Function Table
Dürr-Høyer Quantum Minimization Algorithm [DH99]	Uncompute, Oracle, Amplitude Amplification, Uniform Superposition, Creating Entanglement, Function Table, Speedup by Verification, Quantum Classical Split
Grover's Algorithm [Gro96]	Uncompute, Oracle, Amplitude Amplification, Uniform Superposition, Creating Entanglement, Function Table, Speedup by Verification
HHL Algorithm [HHL09]	Uncompute, Matrix Encoding, QRAM Encoding, Post Selective Measurement
Hybrid Transfer Learning [MBI+20]	Angle Encoding, Variational Quantum Algorithm, Quantum Classical Split, Post Selective Measurement
Inverse Quantum Fourier Transform [NC10]	Amplitude Encoding, Uniform Superposition, Creating Entanglement
Quantum Annealing [Cha17]	Basis Encoding, Creating Entanglement, Uniform Superposition
Quantum Approximate Optimization Algorithm [FGG14]	Basis Encoding, Variational Quantum Algorithm, Alternating Operator Ansatz, Warm Start
Quantum-Assisted Genetic Algorithm [KMB+19]	Basis Encoding, Creating Entanglement, Uniform Superposition, Warm Start, Quantum Classical Split
Quantum Boltzmann Machine [AAR+18]	Basis Encoding, Creating Entanglement, Uniform Superposition, Quantum Classical Split, Variational Quantum Algorithm, Post-Selective Measurement
Quantum Fourier Transform [NC10]	Amplitude Encoding, Uniform Superposition, Creating Entanglement
Quantum Generative Adversarial Network [RA19]	Variational Quantum Algorithm, Quantum Classical Split, Alternating Operator Ansatz
Quantum K-Means Clustering [LMR13]	Basis Encoding, Variational Quantum Algorithm, Alternating Operator Ansatz, Warm Start, Uncompute, Oracle, Amplitude Amplification, Uniform Superposition, Creating Entanglement, Function Table, Speedup by Verification, Quantum Classical Split
Quanvolutional Neural Network [HSPC19]	Angle Encoding, Variational Quantum Algorithm, Quantum Classical Split, Post Selective Measurement
Reverse Annealing [D-W17]	Basis Encoding, Creating Entanglement, Uniform Superposition, Warm Start, Quantum Classical Split
Variational Quantum Eigensolver [PMS+14]	Quantum Classical Split, Variational Quantum Algorithm, Warm Start, Alternating Operator Ansatz

Tabelle 6.2: Zuordnung von Mustern zu Algorithmen im Wissensspeicher der Implementierung basierend auf den Informationen von PlanQK [Pla23] und weiteren Arbeiten.

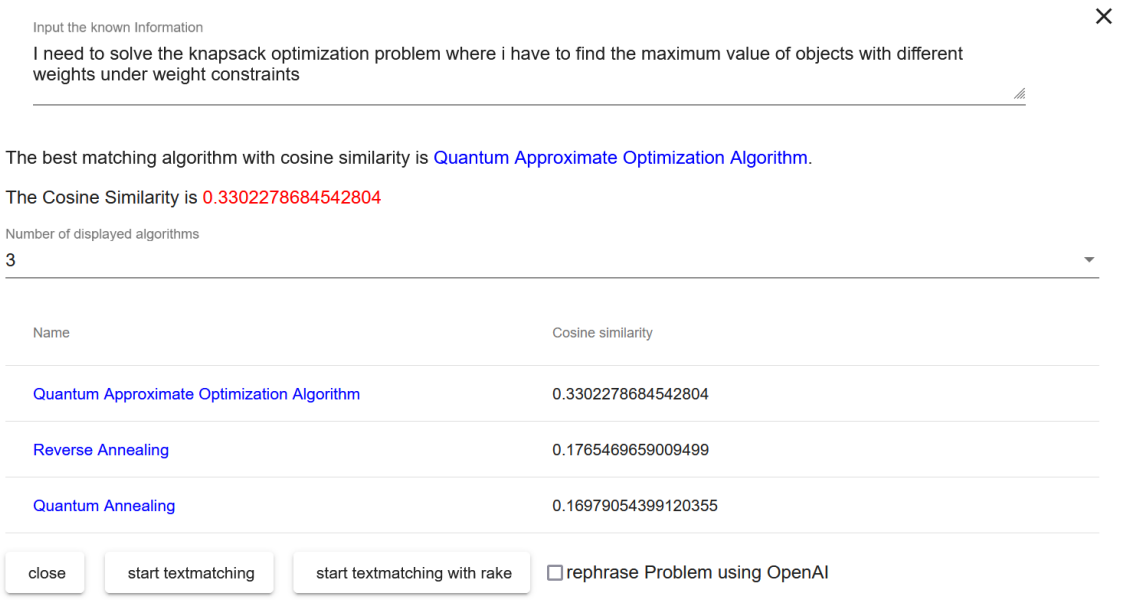

Abbildung 7.1: Darstellung der Ergebnisse des Textvergleichs

7 Evaluation

In diesem Kapitel wird eine Überprüfung des Konzepts durchgeführt. Es wird evaluiert, ob die beschriebenen Probleme gelöst werden können und das Konzept, sowie die Implementierung des Konzepts eine zufriedenstellende Lösung liefern. Anschließend werden noch Erweiterungsmöglichkeiten vorgestellt. Zuerst wird nochmal auf das Szenario aus Kapitel 4 eingegangen. In diesem Szenario soll das Rucksackproblem mithilfe von Quantenalgorithmen gelöst werden. Dabei ist kein Vorwissen zu Quantenalgorithmen vorhanden. Das Szenario beschreibt, wie bei der Implementierung ohne das Konzept vorgegangen wird. Ein großes Problem ist der Zeitverlust, der bei der Wahl des passenden Algorithmus sowie vor allem bei der Identifizierung von passenden Mustern zur Erleichterung der Implementierung, existiert. Wird eine Implementierung des Konzepts verwendet, muss nicht mehr selbst nach einem passenden Algorithmus gesucht werden. Selbst wenn mehrere passende Algorithmen zur Problemlösung existieren, wird der passendste mithilfe vom Textvergleich ausgewählt. Es ist nichts weiter zu tun, als die Problembeschreibung anzugeben und den zutreffenden Algorithmus auszuwählen. Nach der Auswahl stehen sowohl alle Informationen zum Algorithmus, sowie ein Überblick über die zu verwendenden Muster in Form von Referenzen zur Verfügung. Die Muster sind übersichtlich in einem Graphen dargestellt und ihre Informationen können jederzeit direkt abgerufen werden. Ein solcher Graph existiert für jeden einzelnen Algorithmus, der bei den Ergebnissen aufgelistet ist. Der Graph enthält Einstiegsmuster, welche einen Ansatz für den Start

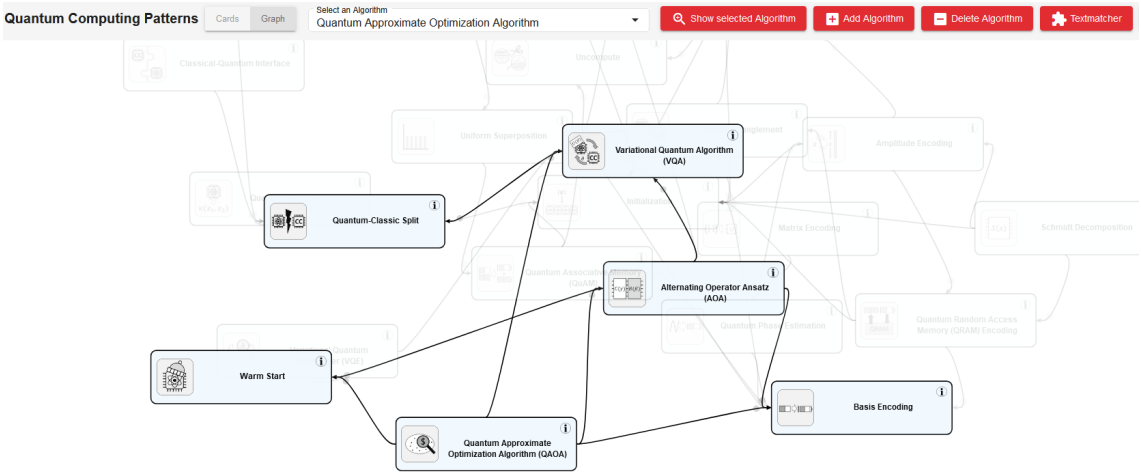

Abbildung 7.2: Muster Graphansicht des QAOA Algorithmus

der Implementierung liefern. Sind weitere Optionen nötig, können die optionalen Muster überprüft werden, welche ebenfalls im Graph enthalten sind. Es werden alle notwendigen Hilfsmittel geliefert, um die Recherche zu überspringen und direkt mit der Implementierung zu beginnen.

Durch die Verwendung der in Kapitel 6 beschriebenen Implementierung, sollen Lösungen zu den im Szenario genannten Problemen geliefert werden. Die Eingabe erfolgt mithilfe der Textvergleichsschnittstelle. Es wird eine Beschreibung des Rucksackproblems in Textformat angegeben. Die Problembeschreibung wird an das Backend gesendet, wo sie ausgewertet wird. Zur Darstellung der Ergebnisse werden diese nach dem Textvergleich zurück zur Erweiterung des Patternatlases geschickt. Es wird beispielsweise die Beschreibung: *I need to solve the knapsack optimization problem where i have to find the maximum value of objects with different weights under weight constraints* verwendet. Diese Beschreibung führt bei der Durchführung des Textvergleichs bei Verwendung von Rake [RECC10] zum Ergebnis in Abbildung 7.1. Die Ergebnisdarstellung erfolgt zunächst in Tabellenform, in welcher die Ergebnisse in absteigender Relevanz zu sehen sind. Relevantere Ergebnisse haben dabei höhere Ähnlichkeitswerte. Das Ergebnis mit dem höchsten Ähnlichkeitswert ist hier der QAOA-Algorithmus [FGG14]. Es sind aktuell drei Ergebnisse sichtbar, die jeweils die besten Ähnlichkeitswerte beim Textvergleich erzielen. Es kann ausgewählt werden, wie viele Ergebnisse auf einmal angezeigt werden sollen. Nach Bedarf kann diese Anzahl jederzeit erhöht oder gesenkt werden. Für jeden Algorithmus ist eine Referenz zur PlanQK [Pla23] Webseite des jeweiligen Algorithmus verfügbar, welche alle Informationen zu diesem enthält. Ganz oben in der Ansicht in Abbildung 7.1 ist ebenfalls die Eingabe zu sehen, die zur Auswertung verwendet wird. Sind Änderungen an der Eingabe erforderlich, können diese direkt in diesem Fenster vorgenommen werden. Dadurch kann die Problembeschreibung ohne viel Aufwand angepasst werden.

Wird ein Algorithmus aus der Ergebnisdarstellung ausgewählt, indem auf ihn geklickt wird, öffnet sich die Graphansicht des Patternatlas [LB21]. Für den QAOA-Algorithmus ist dies in Abbildung 7.2 sichtbar. Auf dieser Ansicht sind die zugehörigen Muster zu sehen. Die restlichen Muster haben eine geringe Opazität, um diese auszublenden. Alle Muster können durch Anklicken ausgewählt werden, um Informationen zu ihren Verbindungen und Relationen zu erhalten. Außerdem enthalten sie eine Referenz zur eingebauten Musterübersicht des Patternatlas, welche genaue Informationen zu jedem

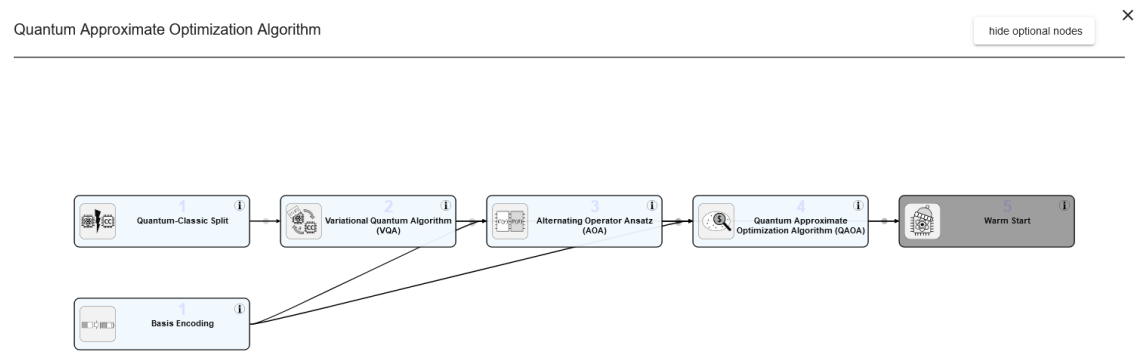

Abbildung 7.3: Teilgraph des QAOA Algorithmus

Muster enthält. Besteht Interesse zu einem anderen Algorithmus, kann dieser aus der Liste der verfügbaren Algorithmen ausgewählt werden. Die zugehörigen Pattern werden nach der Auswahl markiert. Für eine kompaktere Übersicht mit weiteren Informationen kann eine Algorithmen spezifische Ansicht geöffnet werden, indem die Schaltfläche neben der Algorithmenwahl betätigt wird. Dadurch wird ein neues Fenster, mit einer geordneten Übersicht des zum Algorithmus gehörigen Teilgraphen, angezeigt. In diesem Graphen sind die Muster, wie in Abbildung 7.3 zu sehen, nach ihren Verbindungen geordnet. Diese Ordnung bietet eine Hilfestellung zur Implementierung des Algorithmus. Alle Muster, die hier sichtbar sind, entsprechen den hervorgehobenen Mustern in der vorherigen Ansicht. Die Muster ganz links, welche nur ausgehende Verbindungen haben, stellen dabei die Einstiegsmuster dar. Die dunkelgrau markierten Muster sind optional und können durch betätigen einer Schaltfläche ausgeblendet werden. Für den QAOA-Algorithmus sind die Einstigsmuster das *Basis-Encoding* Muster sowie das *Quantum-Classical Split* Muster. Diese führen zum *Variational Quantum Algorithm* Muster, *Alternating Operator Ansatz* Muster und schlussendlich zum QAOA Muster. Das *Warm Start* Muster ist hier optional, da dieses bei Bedarf zusammen mit dem QAOA Muster verwendet werden kann.

Durch das Zusammenfassen aller Informationen zu Mustern und Algorithmen sparen potenzielle Nutzerinnen/Nutzer sich sehr viel Zeit, die normalerweise für die Recherche verwendet werden muss. Dies löst das Problem aus Kapitel 4. Da mehrere Lösungsvorschläge für Algorithmen gemacht werden, kann auch im Fall von ähnlichen Ähnlichkeitswerten schnell eine Alternative gefunden werden. Baut ein Algorithmus auf einem anderen auf, können die Unterschiede durch Auswählen der Algorithmen anhand der Unterschiede im Mustergraphen untersucht werden. Werden neue Algorithmen entworfen, die noch nicht in der Datenbank enthalten sind, können diese auf einfache Art und Weise hinzugefügt werden. Die Benutzerschnittstelle enthält eine Schaltfläche, welche einen Dialog öffnet, der zum Hinzufügen verwendet wird. Hier werden entsprechende Muster sowie optionale Muster angegeben, um die Datenbank zu erweitern. Diese Funktion stellt sicher, dass das System erweiterbar und dadurch auch zukunftssicher ist. Für das Löschen eines Algorithmus ist ebenfalls eine Schaltfläche vorhanden, falls notwendig.

Das Konzept ist jedoch nicht ohne Nachteile. Da die Auswahl der Algorithmen auf dem Textvergleich basiert, ist die Qualität der Ergebnisse nicht nur von der Eingabe abhängig, sondern auch von den Informationen zu den Algorithmen, mit denen der Vergleich durchgeführt wird. Trotz Verwendung

verschiedener Methoden zum Finden von Schlüsselwörtern, besteht keine Garantie, dass alle wichtigen Begriffe gefunden werden. Dementsprechend kann die Genauigkeit nicht gewährleistet werden. Gerade bei kurzen Eingaben, die nicht viele Informationen enthalten, sind die Ergebnisse nicht sehr genau. Eine gute Formulierung des Eingabeproblems liefert bessere Ergebnisse, vor allem wenn Fachbegriffe verwendet werden, die auch in den Algorithmenbeschreibungen vorhanden sind. Außerdem werden alle Informationen zu den Algorithmen aktuell nur von einer Quelle, der PlanQK Quantenalgorithmen Datenbank, geholt und sind von dieser abhängig. Das Verwenden von weiteren Quellen könnte genauere Ergebnisse erzeugen, setzt aber voraus, dass die anderen Quellen die gleiche Qualität von Informationen über Algorithmen wie PlanQK besitzen.

Erweiterungsmöglichkeiten

Die im Konzept dieser Arbeit vorgestellte Vorgehensweise ermöglicht viele Ansätze zur Erweiterung des Ablaufs. Es kann beispielsweise die Eingabe einer Problembeschreibung analysiert und bearbeitet werden, um beim Textvergleich präzisere Ergebnisse zu erhalten. Auch beim Textvergleich können andere Vergleichsmethoden verwendet werden. Eine Untersuchung von verschiedenen Vergleichsmethoden wird in dieser Arbeit nicht durchgeführt. Im Hinblick auf die Genauigkeit der gewählten Ergebnisse wäre solch eine Untersuchung hilfreich, um optimale Ergebnisse zu gewährleisten. In Abschnitt 5.2 wird ein Beispiel für den Systemaufbau bei der Verwendung des Konzepts gezeigt. Die einzelnen Bestandteile liegen separat vor, was das Erweitern sowie den Austausch erleichtert. So können andere Vergleichsmethoden eingebaut und getestet werden, ohne das restliche System zu beeinflussen. Auch andere Bestandteile können ausgetauscht werden, falls sie anders implementiert werden sollen.

Der Patternatlas, welcher zur Implementierung des Konzepts erweitert wird, besitzt die Fähigkeit, sogenannte *Views* zu erstellen. Diese sind Muster-Sprachen übergreifend. Dadurch können Ansichten erstellt werden, die Muster aus verschiedenen Muster-Sprachen enthalten. Diese Fähigkeit kann ausgenutzt werden, um nach der Bestimmung von passenden Algorithmen auch Muster andere Muster-Sprachen miteinzubeziehen. Das ermöglicht die Darstellung von immer vielseitigeren und komplexeren Quantenalgorithmen. Da vor allem hybride Algorithmen immer relevanter werden und die Entwicklung dieser sich über eine wachsende Anzahl an Anwendungsbereichen ausdehnt, ist eine Muster-Sprachen übergreifende Darstellung der Ergebnisse ein guter Ansatz für die Erweiterung des Konzepts. Zum Speichern der Muster wird die zum Patternatlas gehörende Datenbank verwendet. Diese dient zusätzlich als Wissensspeicher, welche die Zuordnung von Mustern und Algorithmen enthält. Die zu den Mustern im Wissensspeicher gehörigen Kanten werden aktuell nicht im Speicher abgelegt, sondern erst beim Aufrufen der Daten berechnet. Um einen besseren Überblick über die Daten zu erhalten, wäre das automatische Speichern der Kanten im Wissensspeicher im Hinblick auf zukünftige Interaktionen mit anderen Teilen der Datenbank eine hilfreiche Erweiterung.

8 Zusammenfassung und Ausblick

Durch die den rasanten Fortschritt im Bereich der Quanteninformatik können immer mehr Algorithmen, die bisher nur theoretisiert wurden, auch auf richtigen Quantencomputern getestet werden. Da diese Algorithmen jedoch auf Quantenmechaniken basieren, muss für den Entwurf und die Implementierung entsprechendes Wissen vorhanden sein. Dies stellt ein Problem dar, denn die Anzahl an Entwicklern beziehungsweise Programmierern mit ausführlichen Kenntnissen im Bereich der Quanteninformatik ist eher gering. Deshalb ist ein System notwendig, das auch ohne viel Wissen zu Quanteninformatik erlaubt, Quantenalgorithmen umzusetzen.

Um dieses Vorhaben zu unterstützen, stellt diese Arbeit ein Konzept vor, welches Hilfsmittel in Form von Informationen liefert. Es wird Natürliche Sprachverarbeitung in Form des Textvergleichs verwendet, um eine angegebene Problembeschreibung zu analysieren und passende Algorithmen zur Lösung des Problems anzubieten. Mehr als fünfzehn Quantenalgorithmen und ihre Anwendungsmöglichkeiten, sowie die 38 zu diesem Zeitpunkt existierenden Quantencomputing Muster wurden untersucht, um eine passende Zuordnung zu erstellen. Dadurch wird es möglich, zu vielen Algorithmen jeweils zugehörige Muster anzugeben, welche als Ansatz für die Umsetzung der verschiedenen Aspekte der Algorithmen verwendet werden können. Um den Einstieg noch weiter zu erleichtern, werden Einstiegsmuster ermittelt, mit denen die Umsetzung begonnen werden kann. Die Ergebnisse werden graphisch dargestellt, um eine gute Übersichtlichkeit zu erhalten. Für jeden Algorithmus, der als Lösung zur Verfügung steht, wird ein Graph erzeugt, der die zugehörigen Muster enthält und ihre Verbindungen zueinander darstellt. Für genauere Informationen zu Muster und Algorithmen werden Referenzen zu ihren Beschreibungen zur Verfügung gestellt. Dies erlaubt potenziellen Nutzerinnen und Nutzern, die ein Problem mithilfe eines Quantenalgorithmus umsetzen wollen, durch textueller Angabe der Problembeschreibung eine Empfehlung für einen geeigneten Algorithmus zu erhalten. Zusätzlich dazu werden Hilfsmittel in Form von Mustern und Visualisierungen geliefert, die relevante Informationen für die Umsetzung enthalten.

Zum Validieren des Konzepts wurde eine Implementierung realisiert, welche die im Konzept verwendete Architektur verwendet. Die Funktionsweise wurde anhand eines konkreten Szenarios evaluiert. Die Implementierung verwendet den Patternatlas [LB21], da dieser schon Informationen zu verschiedenen Mustern enthält und bereits einen Mustergraphen zur Verfügung stellt. Informationen zu den Algorithmen werden extern von der PlanQK [Pla23] Plattform erhalten. Durch Verwenden der Benutzerschnittstelle kann eine textuelle Problembeschreibung übergeben werden. Diese wird mithilfe von Textvergleichs Techniken aus dem Bereich der Natürlichen Sprachverarbeitung ausgewertet. Dabei können unterschiedliche Methoden zum Finden von Schlüsselwörtern ausgewählt werden. Außerdem ist es möglich, durch Verwendung der OpenAI Funktionalität die Eingabe umformulieren zu lassen, um Probleme durch Negationen in der Eingabe zu umgehen.

Ausblick

Eine Ausdehnung der Quellen, die für den Textvergleich verwendet werden, bietet einen guten Ansatz für zukünftige Forschung. Die Implementierung, die in Kapitel 6 beschrieben wird, nutzt für den Vergleich bisher nur die Algorithmenbeschreibungen von PlanQK. Die Auswirkung der Nutzung weiterer Quellen auf die Qualität und Genauigkeit der Ergebnisse könnte untersucht werden. Ein weitere Forschungsansatz ist die Verwendung von Künstlicher Intelligenz, um weitere Algorithmenbeschreibungen zu erhalten. Die Problembeschreibung könnte zum Beispiel an NLP Programme wie ChatGPT [DL23] weitergegeben werden. Durch Textvergleich der dadurch erhaltenen Informationen kann untersucht werden, ob es lohnenswert ist, Künstliche Intelligenz für einen solchen Anwendungsfall einzusetzen. Die Qualität der Ergebnisse in diesem Fall sind ebenfalls interessant. Weiterhin könnte nach einer Möglichkeit geforscht werden, Muster automatisch zu Algorithmen zuzuordnen. Dieser Vorgang erfolgt zurzeit noch manuell und könnte zum Beispiel durch das Trainieren einer Künstlichen Intelligenz für diesen Zweck automatisiert werden.

Literaturverzeichnis

[AAR+18] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, R. Melko. „Quantum Boltzmann Machine“. In: *Phys. Rev. X* 8 (2 Mai 2018), S. 021050. DOI: 10.1103/PhysRevX.8.021050 (zitiert auf S. 53).

[Ale77] C. Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford university press, 1977 (zitiert auf S. 21).

[AOAC19] E. Anschuetz, J. Olson, A. Aspuru-Guzik, Y. Cao. „Variational Quantum Factoring“. In: *Quantum Technology and Optimization Problems*. Hrsg. von S. Feld, C. Linnhoff-Popien. Cham: Springer International Publishing, 2019, S. 74–85. URL: https://link.springer.com/chapter/10.1007/978-3-030-14082-3_7 (zitiert auf S. 15).

[BBB+23] F. Bühler, J. Barzen, M. Beisel, D. Georg, F. Leymann, K. Wild. „Patterns for Quantum Software Development“. In: *Proceedings of the 15th International Conference on Pervasive Patterns and Applications (PATTERNS 2023)*. Xpert Publishing Services (XPS), Juni 2023, S. 30–39. ISBN: 978-1-68558-049-0 (zitiert auf S. 27).

[BBL+22] M. Beisel, J. Barzen, F. Leymann, F. Truger, B. Weder, V. Yussupov. „Patterns for Quantum Error Handling“. In: *Proceedings of the 14th International Conference on Pervasive Patterns and Applications (PATTERNS 2022)*. Xpert Publishing Services (XPS), Apr. 2022, S. 22–30. ISBN: 978-1-61208-953-9 (zitiert auf S. 16, 27).

[CDA22] P. Codognet, D. Diaz, S. Abreu. „Quantum and Digital Annealing for the Quadratic Assignment Problem“. In: *2022 IEEE International Conference on Quantum Software (QSW)*. 2022, S. 1–8. DOI: 10.1109/QSW55613.2022.00016 (zitiert auf S. 52).

[Cha17] N. Chancellor. „Modernizing quantum annealing using local searches“. In: *New Journal of Physics* 19.2 (Feb. 2017), S. 023024. DOI: 10.1088/1367-2630/aa59c4 (zitiert auf S. 52, 53).

[D-W17] D-Wave. *Reverse Quantum Annealing for Local Refinement of Solutions*. D-Wave Systems Inc. Nov. 2017. URL: https://www.dwavesys.com/media/5hsjmvom/14-1018a-a_reverse_quantum_annealing_for_local_refinement_of_solutions.pdf (zitiert auf S. 53).

[DH99] C. Durr, P. Hoyer. *A Quantum Algorithm for Finding the Minimum*. 1999. arXiv: quant-ph/9607014 [quant-ph] (zitiert auf S. 53).

[DL23] J. Deng, Y. Lin. „The Benefits and Challenges of ChatGPT: An Overview“. In: *Frontiers in Computing and Intelligent Systems* 2.2 (Jan. 2023), S. 81–83. DOI: 10.54097/fcis.v2i2.4465 (zitiert auf S. 60).

[DR92] D. David, J. Richard. „Rapid solution of problems by quantum computation“. In: *Proc. R. Soc.* London, UK: The Royal Society Publishing, 1992, S. 553–558. DOI: 10.1098/rspa.1992.0167 (zitiert auf S. 53).

[FBFL15] C. Fehling, J. Barzen, M. Falkenthal, F. Leymann. „PatternPedia – Collaborative Pattern Identification and Authoring“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. epubli GmbH, Juni 2015 (zitiert auf S. 27).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph] (zitiert auf S. 32, 39, 53, 56).

[FNAD20] N. Firoozeh, A. Nazarenko, F. Alizon, B. Daille. „Keyword extraction: Issues and methods“. In: *Natural Language Engineering* 26.3 (2020), S. 259–291. DOI: 10.1017/S1351324919000457 (zitiert auf S. 22).

[GBB+23] D. Georg, J. Barzen, M. Beisel, F. Leymann, J. Obst, D. Vietz, B. Weder, V. Yussupov. „Execution Patterns for Quantum Applications“. In: *Proceedings of the 18th International Conference on Software Technologies - ICSOFT*. SciTePress, Juli 2023, S. 258–268. ISBN: 978-989-758-665-1. DOI: 10.5220/0012057700003538 (zitiert auf S. 27).

[Goo23] Google. 2023. URL: https://angular.io (zitiert auf S. 23, 47, 48, 51).

[Gro96] L. K. Grover. „A Fast Quantum Mechanical Algorithm for Database Search“. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, S. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866 (zitiert auf S. 53).

[GSB18] D. Gunawan, C. A. Sembiring, M. A. Budiman. „The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents“. In: *Journal of Physics: Conference Series* 978.1 (März 2018), S. 012120. DOI: 10.1088/1742-6596/978/1/012120 (zitiert auf S. 22).

[HHL09] A. W. Harrow, A. Hassidim, S. Lloyd. „Quantum Algorithm for Linear Systems of Equations“. In: *Phys. Rev. Lett.* 103 (15 Okt. 2009), S. 150502. DOI: 10.1103/PhysRevLett.103.150502 (zitiert auf S. 52, 53).

[HOR21] D. Herr, B. Obert, M. Rosenkranz. „Anomaly detection with variational quantum generative adversarial networks“. In: *Quantum Science and Technology* 6.4 (Juli 2021), S. 045004. DOI: 10.1088/2058-9565/ac0d4d (zitiert auf S. 53).

[HSPC19] M. Henderson, S. Shakya, S. Pradhan, T. Cook. *Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits*. 2019. arXiv: 1904.04767 [quant-ph] (zitiert auf S. 53).

[IBM23] IBM. *IBM Quantum*. 2023. URL: https://www.ibm.com/quantum (zitiert auf S. 15).

[KMB+19] J. King, M. Mohseni, W. Bernoudy, A. Fréchette, H. Sadeghi, S. V. Isakov, H. Neven, M. H. Amin. *Quantum-Assisted Genetic Algorithm*. 2019. arXiv: 1907.00707 [quant-ph] (zitiert auf S. 53).

[LB21] F. Leymann, J. Barzen. *Pattern Atlas*. Hrsg. von M. Aiello, A. Bouguettaya, D. A. Tamburri, W.-J. van den Heuvel. Cham: Springer International Publishing, 2021, S. 67–76. ISBN: 978-3-030-73203-5. DOI: 10.1007/978-3-030-73203-5_5 (zitiert auf S. 17, 23, 27, 36, 47, 48, 56, 59).

[LC12] Z. Laboudi, S. Chikhi. „Comparison of Genetic Algorithm and Quantum Genetic Algorithm“. In: *International Arab Journal of Information Technology* 9.3 (Mai 2012), S. 243–249. URL: https://www.researchgate.net/publication/268381602_Comparison_of_Genetic_Algorithm_and_Quantum_Genetic_Algorithm (zitiert auf S. 26).

[Ley19] F. Leymann. „Towards a Pattern Language for Quantum Algorithms“. In: *Quantum Technology and Optimization Problems*. Bd. 11413. Lecture Notes in Computer Science (LNCS). Cham: Springer International Publishing, 2019, S. 218–230. DOI: 10.1007/978-3-030-14082-3_19 (zitiert auf S. 16, 27).

[LLSK22] J. W. Z. Lau, K. H. Lim, H. Shrotriya, L. C. Kwek. „NISQ computing: where are we and where do we go?“ In: *AAPPS Bulletin* 32.1 (Sep. 2022), S. 27. ISSN: 2309-4710. DOI: 10.1007/s43673-022-00058-z (zitiert auf S. 15).

[LMR13] S. Lloyd, M. Mohseni, P. Rebentrost. *Quantum algorithms for supervised and unsupervised machine learning*. 2013. arXiv: 1307.0411 [quant-ph] (zitiert auf S. 53).

[LTKT19] M. Langione, C. Tillemann-Dick, A. Kumar, V. Taneja. „Where Will Quantum Computers Create Value—and When?“ In: *Boston Consulting Group* (2019), S. 19. URL: https://www.bcg.com/publications/2019/quantum-computers-create-value-when (zitiert auf S. 15).

[LTR+16] T. Loke, J. W. Tang, J. Rodriguez, M. Small, J. B. Wang. „Comparing classical and quantum PageRanks“. In: *Quantum Information Processing* 16.1 (Dez. 2016), S. 25. ISSN: 1573-1332. DOI: 10.1007/s11128-016-1456-z (zitiert auf S. 26).

[MAl22] M.Altaweel. „QuCSplit: A Decision Support System for Quantum-Classical Splitting“. In: (2022). DOI: http://dx.doi.org/10.18419/opus-12630 (zitiert auf S. 16, 26).

[MBI+20] A. Mari, T. R. Bromley, J. Izaac, M. Schuld, N. Killoran. „Transfer learning in hybrid classical-quantum neural networks“. In: *Quantum* 4 (Okt. 2020), S. 340. ISSN: 2521-327X. DOI: 10.22331/q-2020-10-09-340 (zitiert auf S. 53).

[MT04] R. Mihalcea, P. Tarau. „TextRank: Bringing Order into Text“. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, Juli 2004, S. 404–411. URL: https://aclanthology.org/W04-3252 (zitiert auf S. 22).

[MT90] S. Martello, P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990. DOI: dl.acm.org/doi/abs/10.5555/98124 (zitiert auf S. 31).

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667 (zitiert auf S. 52, 53).

[OB20] A. H. Osman, O. M. Barukub. „Graph-Based Text Representation and Matching: A Review of the State of the Art and Future Challenges“. In: *IEEE Access* 8 (2020), S. 87562–87583. DOI: 10.1109/ACCESS.2020.2993191 (zitiert auf S. 28–30).

[Pla23] PlanQK. *Plattform und Ökosystem für Quantenapplikationen*. 2023. URL: https://planqk.de (zitiert auf S. 25, 38, 48, 49, 52, 53, 56, 59).

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien. „A variational eigenvalue solver on a photonic quantum processor". In: *Nature Communications* 5.1 (Juli 2014). DOI: 10.1038/ncomms5213 (zitiert auf S. 32, 53).

[RA19] J. Romero, A. Aspuru-Guzik. *Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions*. 2019. arXiv: 1901.00848 [quant-ph] (zitiert auf S. 53).

[RECC10] S. Rose, D. Engel, N. Cramer, W. Cowley. *Automatic Keyword Extraction from Individual Documents*. John Wiley & Sons, Ltd, 2010. Kap. 1, S. 1–20. ISBN: 9780470689646. DOI: https://doi.org/10.1002/9780470689646.ch1 (zitiert auf S. 22, 49, 56).

[RFL20] L. Reinfurt, M. Falkenthal, F. Leymann. „Where to begin: on pattern language entry points". In: *SICS Software-Intensive Cyber-Physical Systems* 35.1 (Aug. 2020), S. 127–139. ISSN: 2524-8529. DOI: 10.1007/s00450-019-00417-6 (zitiert auf S. 39).

[Rof19] J. Roffe. „Quantum error correction: an introductory guide". In: *Contemporary Physics* 60.3 (2019), S. 226–245. DOI: 10.1080/00107514.2019.1667078 (zitiert auf S. 15, 20).

[RT02] K. Rajaraman, A.-H. Tan. „Knowledge Discovery from Texts: A Concept Frame Graph Approach". In: *Proceedings of the Eleventh International Conference on Information and Knowledge Management*. CIKM '02. McLean, Virginia, USA: Association for Computing Machinery, 2002, S. 669–671. ISBN: 1581134924. DOI: 10.1145/584792.584914 (zitiert auf S. 28).

[SBLK05] A. Schenker, H. Bunke, M. Last, A. Kandel. *Graph-Theoretic Techniques for Web Content Mining*. Bd. 62. World Scientific, Mai 2005. ISBN: 981-256-339-3 (zitiert auf S. 29).

[SCCT+16] R. Srivastava, I. Choi, T. Cook, N. U. E. Team et al. „The Commercial Prospects for Quantum Computing". In: *Networked Quantum Information Technologies* (2016). URL: https://nqit.ox.ac.uk/sites/www.nqit.ox.ac.uk/files/2018-10/Commercial%20Prospects%20for%20Quantum%20Computing%20Dec%202016.pdf (zitiert auf S. 15).

[Sho02] P. W. Shor. „Introduction to Quantum Algorithms". In: *Proceedings of Symposia in Applied Mathematics*. Bd. 58. 2002, S. 143–160. DOI: https://doi.org/10.1090/psapm/058 (zitiert auf S. 19).

[Sho97] P. W. Shor. „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), S. 1484–1509. DOI: https://doi.org/10.1137/S0097539795293172 (zitiert auf S. 32).

[Str11] E. Strubell. „An Introduction to Quantum Algorithms". In: *COS498 Chawathe Spring* 13 (2011), S. 19. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=cad3e3f789c2e3015f1d70e18c418d11fbd4fc13 (zitiert auf S. 19).

[SW10] „TF–IDF". In: *Encyclopedia of Machine Learning*. Hrsg. von C. Sammut, G. I. Webb. Boston, MA: Springer US, 2010, S. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832 (zitiert auf S. 22).

[The23] The PostgreSQL Global Development Group. *PostgreSQL*. 2023. URL: https://www.postgresql.org (zitiert auf S. 23, 51).

[Tur00] P. D. Turney. „Learning Algorithms for Keyphrase Extraction“. In: *Information retrieval* 2 (2000), S. 303–336. DOI: https://doi.org/10.1023/A:1009976227802 (zitiert auf S. 22).

[TW11] L. Tarrataca, A. Wichert. „Problem-solving and Quantum Computation“. In: *Cognitive Computation* 3.4 (Dez. 2011), S. 510–524. ISSN: 1866-9964. DOI: 10.1007/s12559-011-9103-6 (zitiert auf S. 25).

[VV04] I. Valatkaite, O. Vasilecas. „Automatic Enforcement Of Business Rules As ADBMS Triggers From Conceptual Graphs Model“. In: *Information Technology and Control* 31.2 (2004), S. 26–42. URL: https://itc.ktu.lt/index.php/ITC/article/view/11852 (zitiert auf S. 28).

[WBLS21] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Encoding patterns for quantum algorithms“. In: *IET Quantum Communication* 2.4 (2021), S. 141–152. DOI: https://doi.org/10.1049/qtc2.12032 (zitiert auf S. 27).

[WBLV21] M. Weigold, J. Barzen, F. Leymann, D. Vietz. „Patterns for Hybrid Quantum Algorithms“. In: *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer International Publishing, Sep. 2021, S. 34–51. DOI: 10.1007/978-3-030-87568-8_2 (zitiert auf S. 16, 27).

[WBS10] C. Wartena, R. Brussee, W. Slakhorst. „Keyword Extraction Using Word Co-occurrence“. In: *2010 Workshops on Database and Expert Systems Applications*. 2010, S. 54–58. DOI: 10.1109/DEXA.2010.32 (zitiert auf S. 22).

[WL08] L. Wang, X. Liu. „A new model of evaluating concept similarity“. In: *Knowledge-Based Systems* 21.8 (2008), S. 842–846. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2008.03.042 (zitiert auf S. 29).

[WNS+11] Y. Wang, X. Ni, J.-T. Sun, Y. Tong, Z. Chen. „Representing Document as Dependency Graph for Document Clustering“. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM '11. Glasgow, Scotland, UK: Association for Computing Machinery, 2011, S. 2177–2180. ISBN: 9781450307178. DOI: 10.1145/2063576.2063920 (zitiert auf S. 28).

[WVN19] R. Wille, R. Van Meter, Y. Naveh. „IBM's Qiskit Tool Chain: Working with and Developing for Real Quantum Computers“. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, S. 1234–1240. DOI: 10.23919/DATE.2019.8715261 (zitiert auf S. 16).

[Zei00] A. Zeilinger. „QUANTUM TELEPORTATION“. In: *Scientific American* 282.4 (2000), S. 50–59. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/26058672 (besucht am 12. 09. 2023) (zitiert auf S. 19).

Alle URLs wurden zuletzt am 20. 10. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Waiblingen, den 25.10.2023 T. Pankratz

Ort, Datum, Unterschrift