Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

KI-gestützte Clusterung von studentischen Programmierlösungen zur Verbesserung automatisierter Feedbackprozesse

AI-supported clustering of student programming solutions to improve automated feedback processes

Gregor Germerodt

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Michael Striewe

Trier, 07.07.2025

Vorwort

Ein Vorwort ist nicht unbedingt nötig. Falls Sie ein Vorwort schreiben, so ist dies der Platz, um z.B. das Unternehmen vorzustellen, in der diese Arbeit entstanden ist, oder um den Personen zu danken, die in irgendeiner Form positiv zur Entstehung dieser Arbeit beigetragen haben.

Auf keinen Fall sollten Sie im Vorwort die Aufgabenstellung näher erläutern oder vertieft auf technische Sachverhalte eingehen.

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

Abstract

The same in English.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1

Einleitung und Problemstellung

Für Lehrkräfte an Hochschulen oder Universitäten kann das Kontrollieren und Bewerten von studentischen Einreichungen je nach Menge zu einer großen Herausforderung werden. Gerade bei einer hohen Anzahl an Abgaben steigt der Korrekturaufwand erheblich, was den zeitlichen Rahmen für individuelles Feedback einschränken kann. Eine Untersuchung zeigte, dass das Kontrollieren und Bewerten von Arbeiten der Hauptfaktor für Arbeitsbelastung und Beeinträchtigung des Wohlbefindens ist (vgl. [JS21]). In der Informatik könnte es sich hier auf Programmieraufgaben beziehen. Dabei müssen Lehrkräfte konsistente Bewertungen abliefern, während jede Abgabe unterschiedliche Syntax und Semantik beinhalten kann.

Eine Lösung dieses Problems bieten etablierte Systeme zur automatischen Auswertung von Programmieraufgaben. Systematische Übersichtsarbeiten zeigen, dass viele Werkzeuge vorwiegend auf Unit-Tests oder statische Analysen setzen, was meist zu eher generischem Feedback führt (vgl. [MBKS]). Die Hoschule Trier benutzt beispielsweise ASB - Automatische Software-Bewertung[1]. Nachdem hierbei Studierende die von der Lehrkraft gestellte Aufgabe bearbeitet haben, können sie online ihre Lösungen hochladen. Das Programm prüft danach nach statischen Kriterien, ob z. B. alle benötigen Dateien hochgeladen wurden, ob sie der Namenskonvention entsprechen, etc. Daraufhin wird das hochgeladene Programm mit Testdaten ausgeführt und geprüft, ob die zu erwarteten Ergebnisse ausgegeben werden. Sollte das nicht der Fall sein, wird eine Fehlermeldung ausgegeben, dass das Programm oder bestimmte Module nicht erwartungsgemäß funktionieren.

Das erzeugte Feedback solcher statischen Systeme dient zur Orierentierung, jedoch weniger zur Fehlersuche, da es recht allgemein gehalten ist. Um die Feedbackgenerierung zu verbessern könnten KI-gestütze Verfahren eingesetzt werden. Damit befassten sich beispielsweise die Autoren der wissenschaftlichen Arbeiten ... (Hier Quellen einfügen und erläutern).

Dazu wurde in dieser Arbeit versucht, einen Schritt vor der Feedbackgenerierung zu entwickeln. Er befasst sich mit der KI-gestützten Clusterung studentischer Programmierlösungen. Dieser Ansatz ermöglicht es für mehrere Einreichungen ein gemeinsames Feedback zu generieren, indem sie nach Ähnlichkeit in Bezug auf Syn-

[1] https://www.hochschule-trier.de/informatik/forschung/projekte/asb

tax und Semantik geclustert bzw. gruppiert werden. Weiterführende Prorgamme oder auch Lehrkräfte könnten dann einen Kandidat pro Cluster wählen, Feedback erzeugen und dieses an alle anderen Teilnehmer des Clusters weiterleiten. Dies könnte eine erhebliche Zeitersparnis zur Folge haben und weiterhin mehr Spielraum für präziseres individuelles Feedback ermöglichen.

2

Theoretische Grundlagen

2.1 Entwicklungswerkzeuge

Das Projekt wurde über die frei verfügbare Entwicklungsumbegung Visual Studio Code (VSC) implementiert und über Git verwaltet. Aufgrund der weltweiten Etablierung und die dadurch gegebene vielfältige Auswahl an Bibilotheken und online verfügbare Hilfestellungen, fiel die Wahl der Prorgammiersprache auf Python. Weiterhin wurde das Textsatzsystem LaTeX unter einer von der Hochschule Trier bereitgestellten Vorlagen zum Verfassen wissenschaftlicher Arbeiten[1] genutzt. Folgend eine Auflistung von Erweiterungen die VSC notwendigerweise oder unterstützend hinzugefügt wurden.

- Notwendig:
  - LaTeX Workshop (Kompilierung, Vorschau, Autovervollständigung)
  - Python
  - Pylance (effizienter language Server für Python)
- Unterstützend:
  - LaTeX language support (Syntax-Highlighting und Sprache für .tex-Dateien)
  - Python Debugger
  - isort (automatische Sortierung von Python-Imports)
  - Git Graph (Visualisierung von Arbeitsverlauf)

2.2 Künstliche Intelligenz

Künstliche Intelligenz (KI) bezeichnet die Wissenschaft und Technik der Entwicklung von Maschinen, die Aufgaben ausführen können, für die normalerweise menschliche Intelligenz erforderlich ist (Russell & Norvig, 2021, S. 1). Der Mensch hat sich damit Systeme geschaffen, um die Kapazitäten menschlicher Intelligenz für bestimmte Aufgaben zu schonen oder zu erweitern. Dabei übernimmt die KI den Teil der Automatisierung monotoner Arbeit, also jenen Part, der durch menschliches Handeln erwiesenermaßen fehleranfällig ist, um konsistente Ergebnisse bereitzustellen. Automatisierung bedeutet in diesem Zusammenhang, dass Maschinen

[1] `https://www.hochschule-trier.de/informatik`

bestimmte Entscheidungs- und Handlungsprozesse eigenständig durchführen, ohne dass eine Person direkt in jeden einzelnen Schritt eingreifen muss. Während sich solche Systeme ursprünglich vor allem in der industriellen Fertigung etablierten, stellt sich zunehmend die Frage, wie sich vergleichbare Konzepte auf andere Bereiche übertragen lassen, wie etwa auf das Bildungswesen und hier speziell auf die Bewertung und Rückmeldung (Feedback) zu studentischen Programmierlösungen. Wie kann eine intelligente Automatisierung Lehrkräfte dabei unterstützen, qualitativ hochwertiges und individualisiertes Feedback zu generieren, ohne jede Lösung einzeln manuell prüfen zu müssen?

In dieser Arbeit wird KI anhand verschiedener Open-Source-Bibliotheken genutzt. Das System kombiniert mehrere KI-Techniken wie

- Deep Learning - ein Teilbereich des Machine Learnings (ML), der künstliche neuronale Netze mit vielen Schichten verwendet, um komplexe Muster in Daten zu erkennen,
- Unsupervised Machine Learning - ein Verfahren, bei denen Modelle ohne beschriftete Trainingsdaten Muster oder Strukturen in den Daten erkennen, z.B. durch Clustering, und
- andere verschiedene Machine Learning Methoden, bei denen Computer aus Beispieldaten eigenständig Muster und Zusammenhänge erkennen, um daraus Vorhersagen oder Entscheidungen abzuleiten.

Das Zusammenspiel dieser Methoden führt letztlich zur Unterstützung der Lehrkräfte zur Feedbackgenerierung, was in den Ergebnissen zu sehen sein wird.

2.3 Verwendete Algorithmen

2.3.1 Einbettung (Embedding)

Einer der ersten Schritte des Programms ist das Einbetten von Quellcode-Text der Programmierlösungen, dessen Erstellung im späterem Verlauf der Arbeit erklärt wird. Der Quellcode-Text wird in numerische hochdimensionale Vektor-Repräsentationen (Embeddings) umgewandelt. Diese numerischen Vektoren fassen semantische und strukturelle Merkmale des Codes zusammen und machen die Daten für anschließende Verfahren wie Dimensionsreduktion, Clustering und Visualisierung nutzbar. Hier wurde eine erste Erwähnung solch eines Verfahrens in der Arbeit von Orvalho et al. (2022, [OJM]) erfasst. Das dort beschriebene Verfahren CodeBERT stellte sich durch weitere Recherche für diese Arbeit als geeignet heraus. CodeBERT ist ein auf die Transformer-Architektur[2] basiertes Modell, das gleichzeitig mit natürlicher Sprache und Programmiercode (u.a. Java und Python) trainiert wurde. Es verwendet dabei machine learning (ML) wie Masked Language

[2] Neuronales Netzwerkmodell, das mithilfe von Self-Attention-Mechanismen die Beziehungen zwischen Elementen in einer Sequenz erfasst und dadurch besonders leistungsfähig für Aufgaben mit Text- oder Code-Daten ist

Modeling[3] und Replaced Token Detection[4], um inhaltlich sinnvolle und strukturierte Vektorrepräsentationen (Embeddings) für Code zu erzeugen (vgl. [FGT+]).

2.3.2 Dimensionsreduktion

Weiterhin wurden Verfahren eingebunden die die durch das Embedding erstellten hochdimensionale Vektoren in ihren Dimensionen reduzieren, um sie ebenso für weiterführende Prozesse wie z. B. zur Clusterung und besonders zur Visualisierung im zwei- oder dreidimensionalem Raum nutzbar zu machen. In dieser Arbeit wurden folende Verfahren benutzt:

- Principal component analysis (PCA) - projiziert hochdimensionale Daten in einen lineareren Unterraum mit geringerer Dimension, indem neue Achsen, entlang derer die Daten am stärksten streuen, berechnet werden und stellt die Daten entlang dieser Achsen dar (vgl. [Kar01]),
- t-distributed stochastic neighbor embedding (t-SNE) - visualisiert hochdimensionaler Daten, indem es berechnet wie ähnlich Punkte zu ihren Originaldaten sind, um sie entsprechend weit auseinander oder nahe zusammen zu platzieren (vgl. [Lau08]), und
- Uniform Manifold Approximation and Projection (UMAP) - nutzt Topologie und Geometrie, um skalierbare, strukturtreue EIinbettungen in niedrigere Dimensionen zu erreichen (vgl. [MHM]).

Diese Algorithmen wurden bevorzugt, da sie aufgrund ihrer Verfügbarkeit in öffentlichen Bibliotheken in das vorgestellte Python Projekt einfach eingebunden werden konnten.

2.3.3 Gruppierung (Clustering)

Das zentral angesprochene Verfahren ist das Clustering. Inspiration für diese Arbeit wurde aus aktuellen Werken entnommen, wie Orvalho et al. (2022), die mit InvAASTCluster ein Verfahren zur Clusterung von Programmierlösungen mittels dynamischer Invarianten-Analyse vorstellen (vgl. [OJM]); aus Paiva et al. (2024) die AsanasCluster, ein inkrementelles k-means-basiertes Verfahren, zur Clusterung von Programmierlösungen für automatisiertes Feedback entwickelt haben (vgl. [PLF24]); und Tang et al. (2024) die Large Language Models[5] (LLMs) und Clustering kombinieren, um personalisiertes Feedback in Programmierkursen zu skalieren (vgl. [TWH+]).

Die Algorithmen dieser Quellen und weitere Recherche dienten zum Kennenlernen und zum späteren Einbinden von Algorithmen, die aufgrund ihrer besonderen Eignung zur Clusterung von Programmieraufgaben herausstachen wie

[3] Trainingsmethode, bei der zufällig Wörter im Text verdeckt werden und das Modell lernen soll, diese fehlenden Wörter richtig vorherzusagen

[4] Trainingsmethode, bei der das Modell lernt zu erkennen, welche Wörter im Text durch andere ersetzt wurden, um so bessere Sprachrepräsentationen zu entwickeln.

[5] auf Textdaten trainierte KI-Modelle, die natürliche Sprache verarbeiten und generieren

- k-means - teilt N Beobachtungen in k Cluster auf, wobei jede Beobachtung zu dem Cluster mit dem nächstgelegenen Mittelwert gehört, der als Prototyp des Clusters dient (vgl. [Mac67]), und
- hdbscan - erweitert des Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithmus, indem es eine Hierarchie von Clustern aufbaut und die stabilen Cluster über unterschiedliche Dichteebenen hinweg extrahiert (vgl. [CMS]),

2.3.4 Visualisierung

Zur Visualisierung der zuvor geclusterten studentischen Programmierlösungen wurden die Python-Bibliotheken pandas und plotly.express verwendet:

- pandas: Dient zur effizienten Verarbeitung und Analyse von tabellarischen Daten. In diesem Fall wurden damit die Cluster-Zuordnungen und die zugehörigen Punktkoordinaten in einer DataFrame-Struktur verwaltet.
- plotly.express: Eine High-Level-Bibliothek für interaktive Diagramme. Sie wurde genutzt, um die studentischen Lösungen als farbige Punkte in einem Streudiagramm darzustellen, wobei die Farbe jeweils das zugehörige Cluster repräsentiert.

So konnte die Qualität und Trennschärfe der Clusterung visuell überprüft werden.

2.3.5 Evaluierung

Das Projekt wurde so erstellt, dass für ein beliebigen Clustering-Algorithmus ein Diagramm erstellt wird, in denen farbige Punkte die entsprechenden studentischen Lösungen repäsentieren. Um diesen Prozess zu bewerten wurden Evaluierungsverfahren etabliert. Nach Halkidi et al. 2001 bewerten interne Clustering-Evaluierungsverfahren die Qualität einer Clusterlösung anhand der Dichte innerhalb der Cluster und der Trennung zwischen den Clustern, ohne dabei externe Referenzdaten heranzuziehen (vgl. [HBV01]). In dieser Arbeit wurden erstmalig Erwähnungen solcher Verfahren in [You24] entdeckt, wobei daraus nur zwei der benutzen Verfahren und erst durch weitere Recherche ein drittes hier eingebunden wurde. Folgende Auflistung beschreibt die benutzten Verfahren:

- Silhouette Score - berechnet für jeden Datenpunkt einen Silhouette-Wert, der die Qualität der Clusterzuordnung anhand der Abstände innerhalb und zwischen Clustern bewertet (vgl. [Rou87])
- Caliński-Harabasz Index - bewertet die Clusterqualität anhand des Verhältnisses von Streuung zwischen und innerhalb der Cluster. Das Verfahren wurde ursprünglich von Calinski und Harabasz (1974, [Cal74]) eingefügt, eine Beschreibung findet sich in [HBV01].
- Davies-Bouldin Index - bewertet die Clusterqualität anhand des Verhältnisses von Intra-Cluster-Distanzen zu den Distanzen zwischen den Clustermittelpunkten. Das Verfahren wurde ursprünglich von Davies und Bouldin (1979,

[DB79]) eingeführt, eine Beschreibung findet sich beispielsweise in der scikit-learn-Dokumentation[6].

[6] https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

3

Vorgehensweise und Methodik

3.1 Themenfindung

Die zunehmende Digitalisierung und der technologische Fortschritt eröffnen vielfältige Einsatzmöglichkeiten für Methoden der künstlichen Intelligenz (KI). Besonders im Bildungsbereich entsteht dadurch die Chance, Prozesse zu optimieren und Lehrkräfte bei Routineaufgaben zu entlasten. Vor diesem Hintergrund wurde das Thema dieser Arbeit gewählt. Ziel ist es, das Potenzial von KI-gestützten Verfahren im Kontext der automatisierten Auswertung studentischer Programmierlösungen zu untersuchen. Durch die Clusterung ähnlicher Lösungen können neue Ansätze für Feedbackprozesse entwickelt werden, die eine effizientere und gezieltere Betreuung von Studierenden ermöglichen.

3.2 Recherche und Vorbereitung

Zu Beginn wurden diverse Quellen herausgesucht, die sich im Rahmen dieses Themas bewegen. Besonders oft stach dabei der k-Means Algorithmus heraus oder wie dieser als Grundlage für erweiternde Algorithmen wie den InvAASTCluster (vgl. [OJM]) oder AsanasCluster (vgl. [PLF24]) benutzt wurde um bessere Ergebnisse zu liefern. Anhand eines Rankings wurde die Relevanz der Quellen festgelegt, um das weitere Vorgehen einzugrenzen. Andere Methoden wie der Caliński-Harabasz Index oder der Silhouette Score wurden erwähnt die zur Evaluation des Clusterings dienten. Es wurde klar, dass der Verlauf von studentischen Programmierlösungen bis zu nutzbaren Daten zur Feedbackgenerierung ein mehrschrittiger Prozess sein wird. Erst mit Hilfe des KI-basierten Sprachmodell-Chatbots von OpenAI (ChatGPT) wurde der Prozess bzw. die Pipeline für das Programm grob definiert.

3.3 Implementierungsverlauf

3.3.1 Erster Implementierungsabschnitt

Durch den ersten Prototyp der Pipeline ergaben sich folgende grobe Anforderungen an das Programm:

- Das Programm muss Java-Dateien einlesen können.
- Das Programm muss eine YAML-Konfigurationsdatei laden können.
- Das Programm muss für eingelesene Java-Dateien ein Embedding erstellen können.
- Das Programm muss Embeddings in ihren Dimensionen reduzieren können.
- Das Programm muss Embeddings clustern können.
- Das Programm muss Cluster evaluieren können.
- Das Programm muss Cluster visuell darstellen können.

Erst nachdem die einzelnen Prozessschritte in der Pipeline platz gefunden hatten, wurde klar welche Module dafür implementiert werden mussten. Als erstes mussten die Java-Dateien der studentischen Programmierlösugen eingelesen werden können. Diese wurden als Datensätze durch den betreuenden Prof. Dr. Striewe bereitgestellt. Der Datensatz an denen das Programm fortlaufend getestet wurde, bestand aus mehreren Überordner und final aus drei Java-Dateien und einer Text-Datei, welche den Studenten vorgegeben waren und vervollständigt werden mussten, jedoch soll die Menge der Java-Dateien keine überwiegende Rolle spielen. Das Programm musste also in der Lage sein Ordner zu durchsuchen und Java-Dateien zu erkennen. Dies ermöglichte eine Methode des ersten implementierten Moduls `data_loader.py`. Es extrahierte Quellcode-Text unspeicherte pro Datei `code_snippets` in eine Liste und gab diese an die Pipeline zurück. Anfgans entstanden durch problematische Zeichen innerhald der Java-Dateien Fehlermeldungen und zudem war das Modul nur daraus ausgelegt einen Ordner zu durchsuchen und nicht auch deren Unterordner. Der Code wurde auf Hinsicht der Flexibilität und Vertragbarkeit angepasst, sodass die Anzahl der Unterordner keine Rolle mehr spielte und problematische Zeichen ignoriert werden.

Als nächstes folgte das Modul zum einbetten (Embedding) dieser `code_snippets`, um sie für das Clustering vorzubereiten. Dafür wurde das Modul `embedding_model.py` erstellt, welches anhand einer Methode, importierte vortrainierter Sprachmodelle aus der Transformers-Bibliothek von Python (hier CodeBERT) und passende Tokenizer, die den Code vorbereitend für den Transformer in Tokens zerlegt. Zusammen werden dadurch die `code_snippets` in numerische Vektoren umwandelt[1]. Das entstandene Embedding wwurde anschließend in an die Pipeline zurückgegeben und in eine Liste abgespeichert.

Nun mussten die Embeddings geclustert werden. Das entsprechend erstellte Modul `clustering_engine.py` importierte dafür die beiden Cluster Algorithmen k-Means aus der scikit-learn Bibliothek[2] für machine learning und der separaten hdbscan Python-Bibliothek[3]. Das benutzte Modell wird in der Config-Datei festgelegt. An die Pipeline werden schließlich mit Markierungen (labels) versehene Cluster zurückgegeben.

Anschließend wurde ein Evaluierungsverfahren eingebaut, um die Qualität der Cluster zu bewerten. Dafür wurde das Modul `evaluation_metrics.py` implemen-

[1] `https://huggingface.co/docs/transformers`
[2] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`
[3] `https://hdbscan.readthedocs.io/en/latest/api.html#hdbscan.hdbscan.HDBSCAN`

tiert. Darin wird jeder Cluster nach den Evaluierungsverfahren Silhouette Score[4], Caliński-Harabasz Index[5] und den Davies-Bouldin Index[6] getesten, welche ebenfals aus der scikit-learn Bibliothek importiert wurden. Zurückgegeben wird ein Dictionary mit den drei Zahlenwerten der Bewertungsmetriken.

Jedes implementierte Modul wurde einzeln getestet. Dazu wurden die Ergebnisse und die Zeit, die für den jeweiligen Prozessschritt notwendig war durch print-Anweisungen ausgegeben. Dabei stellte sich heraus, dass Embedding und besonders Imports relativ viel Zeit in Anspruch nahmen. Da die zu testenden Datensätze teilweise aus mehreren hunderten Dateien bestanden, wurde dazu caching eingeführt, um das Testen des Zusammenspiels der einzelnen Module zu beschleunigen, was jedoch später wieder entfernt wurde. Versuche die Imports durch lazy loading zu beschleunigen waren in diesem Fall nur geringfügig in dem nächsten beschriebenen Schritt verwendbar. Des Weiteren wurde eine Requirements.txt-Datei erstellt. Diese beeinhaltet sämltiche Information über die aktuell im Projekt benutzen Versionen der importierten Bibliotheken. Sie sorgt, dass für andere Nutzende gleiche Bedingungen wie auch in der Entwicklung herrschen um ein lauffähiges Programm zu gewährleisten. Die noch später hinzugefügte Datei HowToInstall.txt dient dabei als Schritt-für-Schritt-Anleitung. Eine config.yaml-Datei diente als Ansprechquelle der Pipeline für Parameter

Bevor das Visualisierungs-Modul sinnvoll eingesetzt werden konnte, wurde noch ein weiterer ProzessSchritt eingebunden, das Dimensionsreduktionsverfahren. Das entsprechende Modul `dimensionality_reducer.py` importiert auch hier Verfahren aus der scikit-learn Bibliothek, die Principal Component Analysis (PCA)[7] und t-Distributed Stochastic Neighbor Embedding (t-SNE)[8]. Das dritte Verfahren Uniform Manifold Approximation and Projection (UMAP)[9] stammt aus der eigenständigen umap-learn Bibliothek. Dieser Prozessschritt findet zwischen Embedding und Clustering statt. Er reduziert die Embeddings bzw. Vektoren in ihren Dimensionen (hier 2D oder 3D), welche danach zur Clusterung weitergereicht werden können.

Die Pipeline besteht zu diesem Zeitpunkt aus folgendem Ablauf: **Daten laden → einbetten → Dimensionen reduzieren → clustern → evaluieren → visualisieren**

Schließlich wurde noch das Modul `cluster_plotter.py` implementiert, welches anhand der dimensions-reduzierten Embeddings und der aus dem Clustering hervorgegangenen labels ein statisches Diagramm (engl. plot) in einem separatem Fenster erstellt. Hierfür wurden aus der Matplotlibs die Plotting-Module pyplot[10]

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[9] https://umap-learn.readthedocs.io/en/latest/
[10] https://matplotlib.org/stable/api/pyplot_summary.html

und `mpl_toolkits.mplot3d`[11] importiert, die für 2D- und 3D-Visualisierung (falls gewünscht) zuständig sind. Die zuvor erstellten Module gleichten sich durch ihre einheitlichen Klassenstruktur, da jedoch für dieses Modul keine Zwischenspeicherung von Zuständen anhand einer Instanz notwendig war, wurden dessen Methoden statisch definiert, um sie direkt aufrufen zu können.

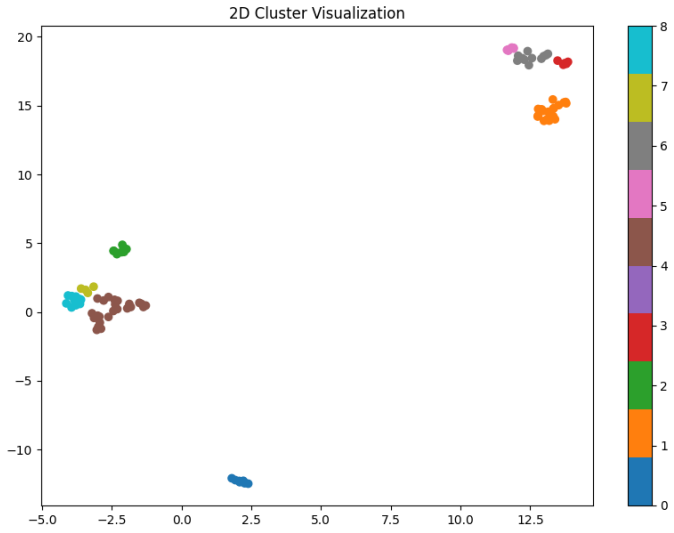

Abbildung 3.1: Clustering-Diagramm einer Clusterung von 147 Java-Dateien. Die unterschiedlichen Farben kennzeichnen die Zugehörigkeit der Punkte zu einem Cluster.

3.3.2 Zweiter Implementierungsabschnitt

In Abbildung 3.2 sind zwar farbige Punkte in unterschiedlicher Anzahl pro Cluster zu erkennen, jedoch wurden keine labels angezeigt, die Informationen über die Punkte anzeigen sollten. Als vorbereitender Schritt für weiterführende Prozesse zur automatischen Feedbackgenerierung, müssen sie zumindest fürs Erste visuell zuordenbar sein. Aus diesem Grund wurde ein weiteres Modul zur Visualisierung implementiert. Das entstandene Modul `interactive_plot.py` nahm dafür wieder Embeddings, labels und zustätzlich noch den Namen der aktuellen Java-Datei entgegen, die im `data_loader.py`-Modul gespeichert wurden. Dazu wurden aus der Python-Bibliothek die Module Pandas[12] und Plotly Express[13] importiert, die einerseits zur Erstellung von Tabellenstrukturen (DataFrames) und andererseits für einfache und interaktiave Plots zuständig sind. Ausgeführt, öffnete sich ein

[11] `https://matplotlib.org/stable/gallery/mplot3d/2dcollections3d.html#sphx-glr-gallery-mplot3d-2dcollections3d-py`
[12] `https://pandas.pydata.org/docs/`
[13] `https://plotly.com/python/plotly-express/`

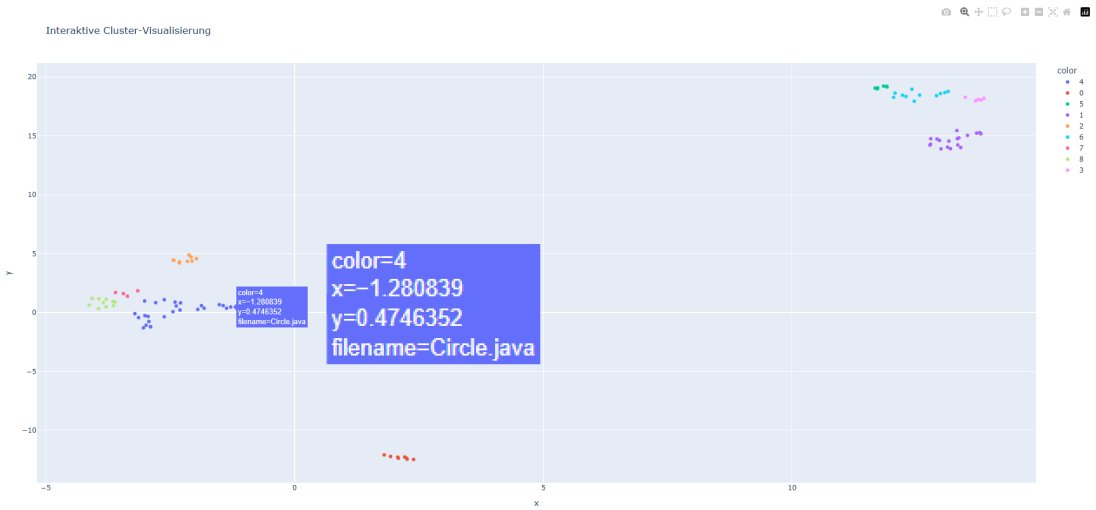

Abbildung 3.2: Interaktives Clustering-Diagramm einer Clusterung von 147 Java-Dateien. Durch das Halten der Maus über Punkte werden Informationen über sie angezeigt (im Bild wiederholt vergrößert dargestellt). Am rechten Rand sind die Mengen der Cluster angezeigt.

Fenster im Webbrowser mit von der Gestalung ähnlichem Plot wie in der statischen Variante (3.1).

Beim mehrmaligen Austesten verschiedener Dateimengen und Überprüfen der ausgegeben Punkte, fiel auf, dass Cluster bei größeren Datenmengen nicht mehr konsistent sind, obwohl akzeptable Ergebnisse im Evaluationsverfahren erreicht wurden. Genauer genommen wurden vereinzelt unterschiedliche Dateien in einem gemeinsamen Cluster platziert (gezeigt in 4.2 in Abschnitt Forschungsergebnisse). So kamen beispielsweise Point.java-Dateien in einem Cluster mit sonst nur Circle.java-Dateien vor. Da Clustering-Algorithmen nach ähnlicher Syntax und Semantik clustern, kann solch ein Verhalten durchaus vor kommen, ist hier jedoch nicht von praktischen Nutzen. Andererseits könnten ebenso Fehler in den Algorithmen oder Inkonsistenzen in den gegebenen Datensätzen die fehlerhafte Clusterung verursachen. Selbst Dateien die abhängig von der gestellten Aufgabe zu den Einreichungen bereits gegeben waren, nicht bearbeitet werden sollten und überall identisch waren, wurden in verschiedenen Clustern angeordnet. Als erste Maßnahme gegen diese Probleme, wurden die nicht zu bearbeitenden Dateien ignoriert, indem nicht mehr allgemein nach „.java“-Dateien gesucht wird, sondern nur noch nach bestimmten Namen. Im weiterem Verlauf wurde zudem in der pipeline eine Schleife ergänzt, die die Prozesse ab Embedding bis zur Visualisierung für die gewünschten, in der config-Datei festgelegten Namen bzw. Java-Dateien wiederholt. Dadurch werden gemischte Cluster verhindert und für jede unterschiedene Java-Datei ein Plot erstellt. Weiterhin eim Testen einer niedriger Anzahl von Dateien ($n \leq 4$) ist aufgefallen, dass der Embedding-Algorithmus nicht mehr funtkioniert, jedoch ist solche eine Clusterung ohnehin nicht von Nutzen.

Um die Evaluationsverfahren einfach testen zu können, wurde eine separate experimentierungs-Pipeline erstellt. Der Vorteil bestand hierbei, dass die verschiedenen Kombinationen der Dimensionsreduktionsverfahren und Clustering-Algorithmen mit Parametern anhand dictionaries innerhalb der Pipeline definiert wurden. In den Tabellen 4.1 und 4.2 im Abschnitt Forschungsergebnisse wurde gezeigt, welche Algorithmen-Kombination für verschiedene Dateimengen geeignet sind. Die Evaluationsergebnisse wurden anschließend in einer CSV-Datei im Projektverzeichnis festgehalten.

Um den Punkten im Diagramm mehr Informationen entnehmen zu können, wurde neben den Dateinamen nun noch der Name des Ordners hinzugefügt. Da die erreichten Punktzahlen der Einreichungen Teil des Ordnernames sind, konnte jetzt die Clusterung besser nachvollzogen und überprüft werden. Weiterhin wurde das `interactive_plot.py`-Modul um die Option das Diagramm als dreidimensionale Umgebung darzustellen erweitert. Sollten mehrere Cluster im 2D-Diagramm aufeinanderliegen, so kann durch die dritte Achse bessere Einsicht gewährleisten.

3.3.3 Dritter Implementierungsabschnitt

Auch wenn für jede durch Namen getrennte Art Datei ein separates Diagramm erstellt wird, besteht die Möglichkeit dass die vollständige Einreichung bzw. Lösung zu den gestellten Aufgaben gesamt betrachtet werden sollte, da es sonst zu verminderter Information führen könnte. Um die Dateien als ein Ganzes zu betrachten, wurde das `data_loader.py`-Modul um eine Konkatenations-Funktionalität ergänzt. Die entstandene Methode konkateniert alle gesuchten Java-Dateien eines Einreichungsordners. Der im Diagramm gezeigte Dateiname eines Punktes, setzt sich nun aus den konkatenierten Namen zusammen. Die Schleife in der Pipeline die die Prozessschritte für jede gesuchte Art Datei wiederholte, wurde wieder entfernt.

Auch wenn das Programm in der Lage ist die Embeddings ohne Dimensionsreduktionsverfahren zu behandeln um sie für weiterführende Projekte vorzubereiten bzw. zu clustern, ist es zum Testen immer noch mehr geeignet sie sinnvoll visuell anzeigen zu können. Daneben ist durch die Anzeige der Ordnernamen pro Punkt ein weiters Problem aufgefallen. So werden konkatenierte Dateien in einen Cluster gesteckt, dessen Ordnernamen sich durch stark variierende Punktzahlen unterscheiden, wie in Abbildung 5.1 gezeigt.

4

Forschungsergebnisse

4.1 Rangliste der Evaluationsverfahren

In den folgenden beiden Tabellen 4.1 und 4.2 sind die Ergebnisse der Evaluationsverfahren für den zweiten Implementierungsabschnitt enthalten, welche nach Rang bzw. aufsteigend nach bester Algorithmus-Kombination geordnet sind. Der Rang ergibt sich dabei aus dem Mittelwert der normalisierten Werte der Metriken (Davies-Bouldin Index invertiert normalisiert). Die optimalen Werte für die verschiedenen Verfahren sind wie folgt:

- Silhouette Score: 0,5 oder höher
- Caliński-Harabasz Index: höchster Wert aus allen Tests
- Davies-Bouldin Index: niedrigester Wert zwischen 0,3 und 0,7

Tabelle 4.1 legt nahe, dass UMAP das geeigneteste Dimensionsreduktionsverfahren ist, unabhängig vom Clustering-Algorithmus, wobei t-SNE und PCA mittelwertige Ergebnisse mit k-Means und schlechtere Ergebnisse mit HDBSCAN liefern. Tabelle 4.2 zeigt, dass sich die Eignung nicht großartig ändert. So sind sowohl für kleine, als auch große Dateimengen beide Clustering-Algorithmen zusammen mit UMAP geeignet.

Kombination	**Silhouette**	**Calinski-Harabasz**	**Davies-Bouldin**	**Rang**
umap_kmeans	0.763	3593.558	0.317	1
umap_hdbscan	0.728	3248.739	0.425	2
tsne_kmeans	0.640	173.360	0.273	3
pca_kmeans	0.609	168.268	0.444	4
pca_hdbscan	0.573	85.872	0.777	5
tsne_hdbscan	0.600	2.318	5.856	6

Tabelle 4.1: Evaluationsergebnisse mit 40 Dateien.

Kombination	Silhouette	Calinski-Harabasz	Davies-Bouldin	Rang
umap_kmeans	0.595	6554.361	0.390	1
umap_hdbscan	0.671	1398.606	0.608	2
tsne_kmeans	0.579	490.225	0.609	3
pca_kmeans	0.716	36.378	0.863	4
pca_hdbscan	0.408	201.754	0.780	5
tsne_hdbscan	0.421	267.712	0.814	6

Tabelle 4.2: Evaluationsergebnisse mit 320 Dateien.

4.2 Clustern unterschiedlicher Dateien in einem Diagramm

Folgende Abbildungen zeigen Clusterungen mit steigender Anzahl Dateien, die sich auf den zweiten Implementierungsabschnitt beziehen. Dabei trat das Problem auf, dass bei größeren Dateimengen die Wahrscheinlichkeit für gemischte Cluster, also mit unterschiedlichen Dateien in einem Cluster anstieg. In den Abbildungen 4.1 und 4.2 ist die Clusterung von 160 Point.java- (orangene Punkte) und 160 Circle.java-Dateien (blaue Punkte) und das beschriebene Problem zu sehen. Die genutzte Algorithmen-Kombination war UMAP mit HDBSCAN. Andere Kombinationen wie PCA mit k-Means ergaben deutlich andere Ergebnisse, jedoch stieg hier die Anzahl der gemischten Cluster ebenso an.

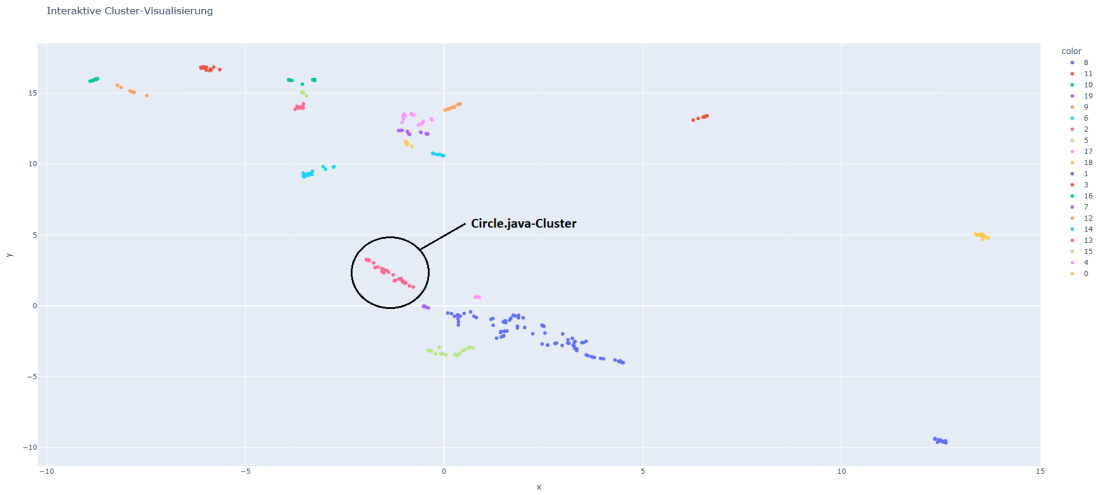

Abbildung 4.1: Clustering-Diagramm einer Clusterung von 320 Java-Dateien. Der eingekreiste Cluster ist ein Circle.java-Cluster.

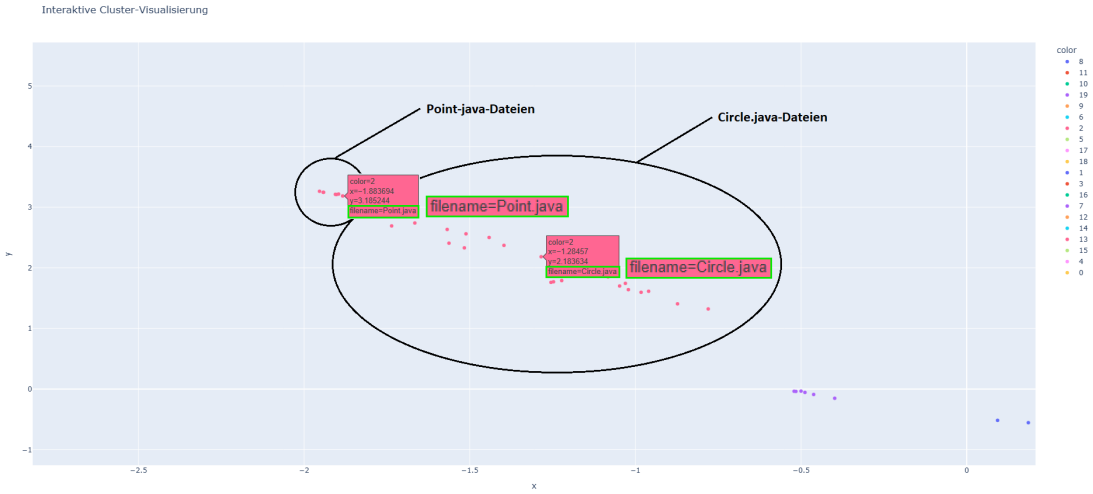

Abbildung 4.2: Vergrößerter Circle.java-Cluster aus Abbildung 4.1

5

Weitere Kapitel

Die Gliederung hängt natürlich vom Thema und von der Lösungsstrategie ab. Als nützliche Anhaltspunkte können die Entwicklungsstufen oder -schritte z.B. der Software-Entwicklung betrachtet werden. Nützliche Gesichtspunkte erhält und erkennt man, wenn man sich

- in die Rolle des Lesers oder
- in die Rolle des Entwicklers, der die Arbeit z.B. fortsetzen, ergänzen oder pflegen soll,

versetzt. In der Regel wird vorausgesetzt, dass die Leser einen fachlichen Hintergrund haben - z.B. Informatik studiert haben. Nur in besonderen Fällen schreibt man in populärer Sprache, so dass auch Nicht-Fachleute die Ausarbeitung prinzipiell lesen und verstehen können.

Die äußere Gestaltung der Ausarbeitung hinsichtlich Abschnittformate, Abbildungen, mathematische Formeln usw. wird im Folgenden kurz dargestellt.

5.1 Bausteine

Der Text wird in bis zu drei Ebenen gegliedert:

1. Kapitel (`\chapter{Kapitel}`)
2. Abschnitte (`\section{Abschnitt}`)
3. Unterabschnitte (`\subsection{Unterabschnitt}`)

5.2 Abschnitt

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua [**?**]. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.2.1 Unterabschnitt

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

5.3 Abbildungen und Tabellen

Abbildung und Tabellen werden zentriert eingefügt. Grundsätzlich sollen sie erst dann erscheinen, nachdem sie im Text angesprochen wurden (siehe Abbildung 5.1). Abbildungen und Tabellen (siehe Tabelle 5.1) können im Fließtext (`h=here`), am Seitenanfang (`t=top`), am Seitenende (`b=bottom`) oder auch gesammelt auf einer nachfolgenden Seite (`p=page`) oder auch ganz am Ende der Ausarbeitung erscheinen. Letzteres sollte man nur dann wählen, wenn die Bilder günstig zusammen zu betrachten sind und die Ausarbeitung nicht zu lang (< 20 Seiten) ist.

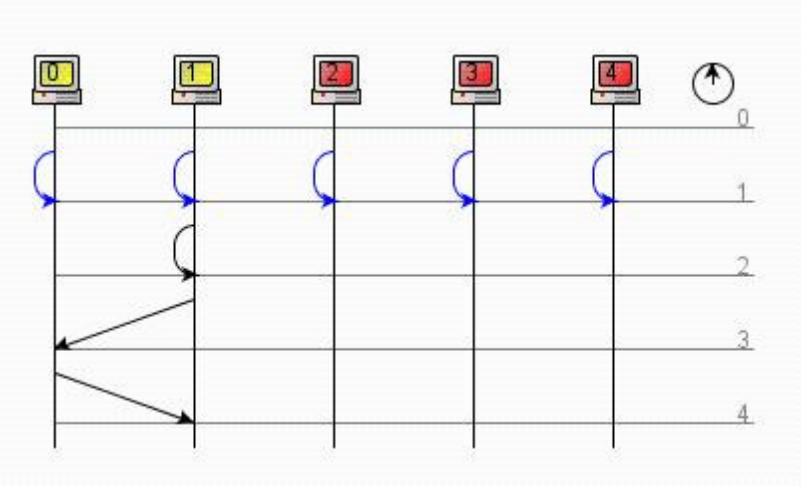

Abbildung 5.1: Bezeichnung der Abbildung

5.4 Listings

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua (siehe Listing 21, Zeile 8).

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$			
P_2		$W(x)2$		
P_3		$R(x)2$		$R(x)1$
P_4			$R(x)2$	$R(x)1$

Tabelle 5.1: Bezeichnung der Tabelle

```
1  def quicksort(arr):
2  less = []
3  pivotList = []
4  more = []
5  if len(arr) <= 1:
6      return arr
7  else:
8      pivot = arr[0]  # the pivot element
9      for i in arr:
10         if i < pivot:
11             less.append(i)
12         elif i > pivot:
13             more.append(i)
14         else:
15             pivotList.append(i)
16     less = quicksort(less)
17     more = quicksort(more)
18     return less + pivotList + more

20 print(quicksort[4, 65, 2, -31, 0, 99, 83, 782, 1]))
```

Listing 5.1: Quicksort-Implementierung in Python

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua (siehe Listing 10, Zeilen 3 und 5).

```
1  function quicksort([pivot, ...others]) {
2      return pivot === undefined ? [] : [
3          ...quicksort(others.filter(n => n < pivot)),
4          pivot,
5          ...quicksort(others.filter(n => n >= pivot))
6      ];
7  }

9  console.log(quicksort([11.8, 14.1, 21.3, 8.5, 16.7, 5.7]));
```

Listing 5.2: Quicksort-Implementierung in JavaScript

Größere Code-Fragmente sollten im Anhang eingefügt werden. [**?**]

5.5 Mathematische Formel

Mathematische Formeln bzw. Formulierungen können sowohl im Fließtext (z.B. $y = x^2$) oder abgesetzt und zentriert im Text erscheinen. Gleichungen sollten für Referenzierungen nummeriert werden (siehe Formel 5.1).

$$e_i = \sum_{i=1}^{n} w_i x_i \tag{5.1}$$

Entscheidungsformel:

$$\psi(t) = \begin{cases} 1 & 0 <= t < \frac{1}{2} \\ -1 & \frac{1}{2} <= t < 1 \\ 0 & sonst \end{cases} \tag{5.2}$$

Matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \tag{5.3}$$

Vektor:

$$\overline{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \tag{5.4}$$

5.6 Sätze, Lemmata, Definitionen, Beweise, Beispiele

Sätze, Lemmata, Definitionen, Beweise und Beispiele können in speziell dafür vorgesehenen Umgebungen erstellt werden.

Definition 5.1. *(Optimierungsproblem) Ein* Optimierungsproblem $\mathcal{P}$ *ist festgelegt durch ein Tupel* $(I_\mathcal{P}, sol_\mathcal{P}, m_\mathcal{P}, goal)$ *wobei gilt*

1. $I_\mathcal{P}$ *ist die Menge der Instanzen,*
2. $sol_\mathcal{P} : I_\mathcal{P} \longmapsto \mathbb{P}(S_\mathcal{P})$ *ist eine Funktion, die jeder Instanz* $x \in I_\mathcal{P}$ *eine Menge zulässiger Lösungen zuweist,*
3. $m_\mathcal{P} : I_\mathcal{P} \times S_\mathcal{P} \longmapsto \mathbb{N}$ *ist eine Funktion, die jedem Paar* $(x, y(x))$ *mit* $x \in I_\mathcal{P}$ *und* $y(x) \in sol_\mathcal{P}(x)$ *eine Zahl* $m_\mathcal{P}(x, y(x)) \in \mathbb{N}$ *zuordnet (= Maß für die Lösung* $y(x)$ *der Instanz* x*), und*
4. $goal \in \{min, max\}$.

Beispiel 5.2. MINIMUM TRAVELING SALESMAN (MIN-TSP)

- $I_{MIN-TSP} =_{def}$ s.o., ebenso $S_{MIN-TSP}$
- $sol_{MIN-TSP}(m, D) =_{def} S_{MIN-TSP} \cap \mathbb{N}^m$
- $m_{MIN-TSP}((m, D), (c_1, \dots, c_m)) =_{def} \sum_{i=1}^{m-1} D(c_i, c_{i+1}) + D(c_m, c_1)$
- $goal_{MIN-TSP} =_{def} min$

□

Satz 5.3. *Sei $\mathcal{P}$ ein **NP**-hartes Optimierungsproblem. Wenn $\mathcal{P} \in$ **PO**, dann ist **P** = **NP**.*

Beweis. Um zu zeigen, dass **P** = **NP** gilt, genügt es wegen Satz A.30 zu zeigen, dass ein einziges **NP**-vollständiges Problem in **P** liegt. Sei also $\mathcal{P}'$ ein beliebiges **NP**-vollständiges Problem.

Weil $\mathcal{P}$ nach Voraussetzung **NP**-hart ist, gilt insbesondere $\mathcal{P}' \leq_T \mathcal{P}_C$. Sei R der zugehörige Polynomialzeit-Algorithmus dieser Turing-Reduktion. Weiter ist $\mathcal{P} \in$ **PO** vorausgesetzt, etwa vermöge eines Polynomialzeit-Algorithmus A. Aus den beiden Polynomialzeit-Algorithmen R und A erhält man nun leicht einen effizienten Algorithmus für $\mathcal{P}'$: Ersetzt man in R das Orakel durch A, ergibt dies insgesamt eine polynomielle Laufzeit.

Lemma 5.4. *Aus **PO** = **NPO** folgt **P** = **NP**.*

Beweis. Es genügt zu zeigen, dass unter der angegeben Voraussetzung KNAPSACK $\in$ **P** ist.

Nach Voraussetung ist MAXIMUM KNAPSACK $\in$ **PO**, d.h. die Berechnung von $m^*(x)$ für jede Instanz x ist in Polynomialzeit möglich. Um KNAPSACK bei Eingabe (x, k) zu entscheiden, müssen wir nur noch $m^*(x) \geq k$ prüfen. Ist das der Fall, geben wir 1, sonst 0 aus. Dies bleibt insgesamt ein Polynomialzeit-Algorithmus.

□

5.7 Fußnoten

In einer Fußnote können ergänzende Informationen[1] angegeben werden. Außerdem kann eine Fußnote auch Links enthalten. Wird in der Arbeit eine Software (zum Beispiel Java[2]) eingesetzt, so kann die Quelle, die diese Software zur Verfügung stellt in der Fußnote angegeben werden.

5.8 Literaturverweise

Jede verwendete Literatur wird im Literaturverzeichnis angegeben[3]. Jeder im Verzeichnis vorkommende Eintrag muss mindestens einmal im Text referenziert werden [**?**].

[1] Informationen die für die Arbeit zweitrangig sind, jedoch für den Leser interessant sein könnten.
[2] `https://www.oracle.com/java/technologies/`
[3] Dazu wird eine sogenannte BibTeX-Datei (literatur.bib) verwendet.

6

Beispiel-Kapitel

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

6.1 Warum existieren unterschiedliche Konsistenzmodelle?

Laut [**?**] sind mit der Replikation von Daten immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz der Daten. Die Form der Konsistenzsicherung bestimmt dabei, inwiefern das eine Kriterium erfüllt und das andere dementsprechend nicht erfüllt ist (Trade-off zwischen Verfügbarkeit und der Konsistenz der Daten). Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien der Daten existieren, dürfen keine Abweichungen auftreten. Die Verfügbarkeit der Daten ist hier jedoch stark eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten, wobei die Konsistenz nur an bestimmten Synchronisationspunkten gewährleistet wird. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen.

Nach [**?**] kann die Performanzsteigerung der schwächeren Konsistenzmodelle wegen der Optimierung (Pufferung, Code-Scheduling, Pipelines) 10-40 Prozent betragen. Wenn man bedenkt, dass mit der Nutzung der vorhandenen Synchronisierungsmechanismen schwächere Konsistenzmodelle den Anforderungen der strengen Konsistenz genügen, stellt sich der höhere programmiertechnischer Aufwand bei der Implementierung der schwächeren Konsistenzmodelle als ihr einziges Manko dar.

In [**?**] ist beschrieben, wie man sich Formen von DSM vorstellen könnte, für die ein beachtliches Maß an Inkonsistenz akzeptabel wäre. Beispielsweise könnte DSM verwendet werden, um die Auslastung von Computern in einem Netzwerk zu speichern, so dass Clients für die Ausführung ihrer Applikationen die am wenigsten ausgelasteten Computer auswählen können. Weil die Informationen dieser Art innerhalb kürzester Zeit ungenau werden können (und durch die Verwendung

der veralteten Daten keine großen Nachteile entstehen können), wäre es vergebliche Mühe, sie ständig für alle Computer im System konsistent zu halten [**?**]. Die meisten Applikationen stellen jedoch strengere Konsistenzanforderungen.

6.2 Klassifizierung eines Konsistenzmodells

Die zentrale Frage, die für die Klassifizierung (streng oder schwach) eines Konsistenzmodells von Bedeutung ist [**?**]: wenn ein Lesezugriff auf eine Speicherposition erfolgt, welche Werte von Schreibzugriffen auf diese Position sollen dann dem Lesevorgang bereitgestellt werden? Die Antwort für das schwächste Konsistenzmodell lautet: von jedem Schreibvorgang, der vor dem Lesen erfolgt ist, oder in der „nahen“ Zukunft, innerhalb des definierten Betrachtungsraums, erfolgten wird. Also irgendein Wert, der vor oder nach dem Lesen geschrieben wurde.

Für das strengste Konsistenzmodell, Linearisierbarkeit (atomic consistency), stehen alle geschriebenen Werte allen Prozessoren sofort zur Verfügung: eine Lese-Operation gibt den aktuellsten Wert zurück, der geschrieben wurde, bevor das Lesen stattfand. Diese Definition ist aber in zweierlei Hinsicht problematisch. Erstens treten weder Schreib- noch Lese-Operationen zu genau einem Zeitpunkt auf, deshalb ist die Bedeutung von „aktuellsten“ nicht immer klar. Zweitens ist es nicht immer möglich, genau festzustellen, ob ein Ereignis vor einem anderen stattgefunden hat, da es Begrenzungen dafür gibt, wie genau Uhren in einem verteilten System synchronisiert werden können.

Nachfolgend werden einige Konsistenzmodelle absteigend nach ihrer Strenge vorgestellt. Zuvor müssen wir allerdings klären, wie die Lese- und Schreibe-Operationen in dieser Ausarbeitung dargestellt werden.

Sei x eine Speicherposition, dann können Instanzen dieser Operationen wie folgt ausgedrückt werden:

- $R(x)a$ - eine Lese-Operation, die den Wert a von der Position x liest.
- $W(x)b$ - eine Schreib-Operation, die den Wert b an der Position x speichert.

6.3 Linearisierbarkeit (atomic consistency)

Die Linearisierbarkeit im Zusammenhang mit DSM kann wie folgt definiert werden:

- Die verzahnte Operationsabfolge findet so statt: wenn $R(x)a$ in der Folge vorkommt, dann ist die letzte Schreib-Operation, die vor ihr in der verzahnten Abfolge auftritt, $W(x)a$, oder es tritt keine Schreib-Operation vor ihr auf und a ist der Anfangswert von x. Das bedeutet, dass eine Variable nur durch eine Schreib-Operation geändert werden kann.
- Die Reihenfolge der Operationen in der Verzahnung ist konsistent zu den Echtzeiten, zu denen die Operationen bei der tatsächlichen Ausführung aufgetreten sind.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$		$W(y)2$	
P_2		$R(x)1$		$R(y)2$

Tabelle 6.1: Linearisierbarkeit ist erfüllt

Die Bedeutung dieser Definition kann an folgendem Beispiel (Tabelle 6.1) nachvollzogen werden. Es sei angenommen, dass alle Werte mit 0 vorinitialisiert sind.

Hier sind beide Bedingungen erfüllt, da die Lese-Operationen den zuletzt geschriebenen Wert zurückliefern. Interessanter ist es, zu sehen, wann die Linearisierbarkeit verletzt ist.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)0$	$R(x)2$

Tabelle 6.2: Linearisierbarkeit ist verletzt, sequentielle Konsistenz ist erfüllt.

In diesem Beispiel (Tabelle 6.2) ist die Echtzeit-Anforderung verletzt, da der Prozess P_2 immer noch den alten Wert liest, obwohl er von Prozess P_1 bereits geändert wurde. Diese Ausführung wäre aber sequentiell konsistent (siehe kommender Abschnitt), da es eine Verzahnung der Operationen gibt, die diese Werte liefern könnte ($R(x)0, W(x)1, W(x)2, R(y)2$). Würde man beide Lese-Operationen des 2. Prozesses vertauschen, wie in der Tabelle 6.3 dargestellt, so wäre keine sinnvolle Verzahnung mehr möglich.

Prozesse	**Zeit** $\rightarrow$			
P_1	$W(x)1$	$W(x)2$		
P_2			$R(x)2$	$R(x)0$

Tabelle 6.3: Linearisierbarkeit und sequentielle Konsistenz sind verletzt.

In diesem Beispiel sind beide Bedingungen verletzt. Selbst wenn die Echtzeit, zu der die Operationen stattgefunden haben, ignoriert wird, gibt es keine Verzahnung einzelner Operationen, die der Definition entsprechen würde.

7

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

8

Anhang

Literaturverzeichnis

Cal74. Calinski, T. and Harabasz, J.: *A dendrite method for cluster analysis.* Communications in Statistics, 3(1):1–27, 1974.

CMS. Campello, Ricardo J. G. B., Davoud Moulavi und Joerg Sander: *Density-Based Clustering Based on Hierarchical Density Estimates.* Seiten 160–172.

DB79. Davies, D. L. und D. W. Bouldin: *A Cluster Separation Measure.* IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1(2):224–227, 1979.

FGT+. Feng, Zhangyin, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang und Ming Zhou: *CodeBERT: A Pre-Trained Model for Programming and Natural Languages.*

HBV01. Halkidi, Maria, Yannis Batistakis und Michalis Vazirgiannis: *On Clustering Validation Techniques.* Journal of Intelligent Information Systems, 17(2-3):107–145, 2001.

JS21. Jerrim, John und Sam Sims: *When is high workload bad for teacher wellbeing? Accounting for the non-linear contribution of specific teaching tasks.* Teaching and Teacher Education, 105:103395, 2021.

Kar01. Karl Pearson: *On lines and planes of closest fit to systems of points in space.* Philosophical Magazine, 2(11):559–572, 1901.

Lau08. Laurens van der Maaten und Geoffrey Hinton: *Visualizing Data using t-SNE.* Journal of Machine Learning Research, 9:2579–2605, 2008.

Mac67. MacQueen, J.: *Some methods for classification and analysis of multivariate observations.* In: Le Cam, Lucien M. und Jerzy Neyman (Herausgeber): *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Band 5.1, Seiten 281–298. University of California Press, 1967.

MBKS. Messer, Marcus, Neil C. C. Brown, Michael Kölling und Miaojing Shi: *Automated Grading and Feedback Tools for Programming Education: A Systematic Review.*

MHM. McInnes, Leland, John Healy und James Melville: *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.*

OJM. Orvalho, Pedro, Mikoláš Janota und Vasco Manquinho: *InvAASTCluster: On Applying Invariant-Based Program Clustering to Introductory Programming Assignments*.

PLF24. Paiva, José Carlos, José Paulo Leal und Álvaro Figueira: *Clustering source code from automated assessment of programming assignments*. International Journal of Data Science and Analytics, Seiten 1–12, 2024.

Rou87. Rousseeuw, Peter J.: *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*. Journal of Computational and Applied Mathematics, 20:53–65, 1987.

TWH+. Tang, Xiaohang, Sam Wong, Marcus Huynh, Zicheng He, Yalong Yang und Yan Chen: *SPHERE: Scaling Personalized Feedback in Programming Classrooms with Structured Review of LLM Outputs*.

You24. Youssef Lahmadi, Mohammed Zakariae El Khattabi, Mounia Rahhali, Lahcen Oughdir: *Optimizing Adaptive Learning: Insights from K-Means Clustering in Intelligent Tutoring Systems*. International Journal of Intelligent Systems and Applications in Engineering, 12(3):1842–1851, 2024.

Index

A

Glossar

DisASTer	Distributed Algorithms Simulation Terrain, eine Plattform zur Implementierung verteilter Algorithmen [**?**]
DSM	Distributed Shared Memory
AC	Atomic Consistency (dt.: Linearisierbarkeit)
RC	Release Consistency (dt.: Freigabekonsistenz)
SC	Sequential Consistency (dt.: Sequentielle Konsistenz)
WC	Weak Consistency (dt.: Schwache Konsistenz)

B

Eigenständigkeitserklärung

☐ Die vorliegende Arbeit wurde als Einzelarbeit angefertigt.

☐ Die vorliegende Arbeit wurde als Gruppenarbeit angefertigt. Mein Anteil an der Gruppenarbeit ist im untenstehenden Abschnitt *Verantwortliche* dokumentiert:

☐ Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche kenntlich gemacht. Darüber hinaus erkläre ich, dass ich die vorliegende Arbeit in dieser oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht habe.

☐ Es ist keine Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln erfolgt.

☐ Die Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln wurde von der/dem Prüfenden ausdrücklich gestattet. Die von der/dem Prüfenden mit Ausgabe der Arbeit vorgegebenen Anforderungen zur Dokumentation und Kennzeichnung habe ich erhalten und eingehalten. Sofern gefordert, habe ich in der untenstehenden Tabelle *Nutzung von KI-Tools* die verwendeten KI-basierten text- oder inhaltgenerierenden Hilfsmittel aufgeführt und die Stellen in der Arbeit genannt. Die Richtigkeit übernommener KI-Aussagen und Inhalte habe ich nach bestem Wissen und Gewissen überprüft.

Datum

Unterschrift der Kandidatin/des Kandidaten

Verantwortliche

Die Tabellen unten führen auf, wer als Autor für die einzelnen Kapitel der vorliegenden Dokumentation beziehungsweise für einzelne Teile des Quellcodes hauptverantwortlich ist.

Insgesamt beteiligt sind die folgenden Personen:

- Autor 1
- Autor 2
- Autor 3

Dokumentation

Kapitel	**Überschrift**	**Autor**
1	Einleitung	Autor 1, Autor 2, Autor 3
2	Problemstellung	Autor 1, Autor 2, Autor 3
3	Aufgabenstellung und Zielsetzung	Autor 1, Autor 2, Autor 3
4	Übrige Abschnitte (Kapitel und Absätze)	Autor 1
4.1	Abschnitt	Autor 3
4.1.1	Unterabschnitt	Autor 2, Autor 3
4.2	Abbildungen und Tabellen	usw.
4.3	Mathematische Formel	
4.4	Sätze, Lemmas und Definitionen	
4.5	Fußnoten	
4.6	Literaturverweise	
5	Beispiel-Kapitel	
5.1	Warum existieren unterschiedliche Konsistenzmodelle?	
5.2	Klassifizierung eines Konsistenzmodells	
5.3	Linearisierbarkeit (atomic consistency)	

Quellcode

Paket	**Autor**
algorithms.search	Autor 1
algorithms.sort	Autor 3

Nutzung von KI-Tools

KI-Tool	**Genutzt für**	**Warum?**	**Wann?**	**Mit welcher Eingabefrage bzw. -aufforderung?**	**An welcher Stelle der Arbeit übernommen?**
ChatGPT	Konzept XY erklären lassen	Erklärung von Verständnisfragen zu...	Bei der Bearbeitung des Theorieteils der Arbeit	Welches sind die zentralen Merkmale des Konzepts XY?	S. 25, 30 ff.
DeepL Write	Neuformulierung meiner Textentwürfe	Bessere Lesbarkeit	Über die gesamte Arbeit hinweg	Formuliere die Kapitel 2 und 3 neu in einfachen und leicht verständlichen Sätzen!	S. 45 ff. , S. 67