Optimierte Clusteranalyse basierend auf minimalen Spannbäumen am Beispiel intensivmedizinischer Daten

Bachelorarbeit

Carla Victoria Matthews
469150

28. April 2025

Gutachter: Prof. Dr. Benjamin Blankertz
Prof. Dr. Marc Alexa

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Neurotechnologie

Kurzfassung

In dieser Arbeit werden zwei stark miteinander verflochtene Bereiche der Datenanalyse betrachtet: Clustering und Outlierdetection. Ausgangspunkt dieser Arbeit ist der Artikel von Li et al. [1], in dem ein Algorithmus zur Outlierdetection basierend auf minimalen Spannbäumen vorgestellt wird. Ziel dieser Arbeit ist es, die Laufzeit dieses Algorithmus zu optimieren und ihn für die Clusteranalyse weiterzuentwickeln.
Zur Optimierung der Laufzeit wurde der ursprünglich eingesetzte Prim-Algorithmus durch den Borůvka-Algorithmus sowie den Karger-Klein-Tarjan-Algorithmus ersetzt. Der Vergleich der Algorithmen zeigte, dass sich auf kleinen Datensätzen die Laufzeit dadurch nicht verbessert konnte, jedoch war die Verwendung von Borůvka für große Datensätze ein erheblicher Vorteil.
Die im Artikel von Li et al. präsentierten Kategorisierungsergebnisse der Inlier und Outlier konnten unter den angegebenen Parametern nicht repliziert werden. Die Einführung der Dopplung der MST-Kanten als Input für den Outlierdetectionalgorithmus führte in vielen Fällen zu einer erheblichen Verbesserung der Ergebnisse.
Statt der Nutzung der euklidischen Distanz als Kantengewichte bei der Bildung des Graphen aus dem Datensatz wurde ein anderes Ähnlichkeitsmaß getestet: eine Variante der Gaußschen radialen Basisfunktion. Dies sollte der Optimierung der Clustergenauigkeit dienen. Die Anpassung führte auf den genutzten Daten zu keiner Veränderung der Clustergenauigkeit. Es konnten jedoch Einsatzmöglichkeiten identifiziert werden, um beispielsweise Daten auf einen Ausschnitt der Inlier zu reduzieren.
Um den Outlierdetectionalgorithmus für die Clusteranalyse abzuwandeln, wurden die bei der Outlierberechnung entstehenden mini-MSTs als Minicluster betrachtet und nach zur Analyse nach ihrem meist enthaltenen Ergebnis gelabelt. Diese Minicluster wurden genutzt um Cluster aus einem Vorclustering zu strukturieren. Die Methode wurde erfolgreich auf einem Datensatz der MIMIC-III-Datenbank [2, 3, 4] getestet. Die Datenbank enthält Daten einer Intensivstation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Jahr 2023 wurden 132,4 Zettabyte an Daten generiert und repliziert. Für 2028 werden 383,9 Zettabyte an Daten prognostiziert [5].
Um diese Datenmengen nutzbar zu machen, gewinnt die Datenanalyse zunehmend an Bedeutung in der Informatik. In dieser Arbeit werden zwei stark miteinander verflochtene Bereiche der Datenanalyse betrachtet: Clustering und Outlierdetection.
Clustering dient dazu, ähnliche Datenpunkte zu finden und zu gruppieren. Dies hat vielfältige Anwendungen wie die Erkennung von Objekten und Strukturen in Bildern, die Gruppierung von Nutzern mit ähnlichen Interessen oder auch die Klassifizierung von Leukämietypen auf Basis ihrer Genexpressionmuster [6].
Outlierdetection findet Ausreißerwerte (Outlier) in den Daten. Diese Outlier können beispielsweise Produktionsfehler, ungewöhnliche Gesundheitswerte oder auch auf Messfehler hinweisen. Outlierdetection und Clustering hängen in der Praxis oft eng zusammen. Zum einen kann Clustering dazu genutzt werden, Outlier zu identifizieren, indem alle nicht klassifizierbaren Daten als Outlier gelten. Zum anderen können zunächst mittels Outlierdetection die Ausreißerwerte identifiziert und vor dem Clustering eliminiert werden, um bessere Ergebnisse zu erzielen. Damit bildet die Outlierdetection einen guten Ansatzpunkt für Clusteroptimierung.
In dem Artikel von Li et al. [7] wird ein Algorithmus zur Outlierdetection basierend auf minimalen Spannbäumen (Def. s. Kapitel 2) vorgestellt. Dieser berechnet einen minimalen Spannbaum (MST) über alle Datenpunkte und bestimmt basierend auf diesem kleinere Spannbäume, deren Datenpunkte als nicht-Ausreißerwerte (Inlier) gelten. Nach Ablauf des Algorithmus sind alle restlichen Datenpunkte Outlier.
Dies soll beispielhaft auf Datensätzen, die aus der MIMIC-III-Datenbank [2, 3, 4] extrahiert wurden, angewendet werden. Die MIMIC-III-Datenbank enthält gesammelte Daten einer Intensivstation. Daraus ergibt sich die folgende Forschungsfrage: **Wie kann der von Li et al. vorgestellte Algorithmus optimiert und für eine verbesserte Clusteranalyse werden?**
Da Optimierung ein weites Feld ist, beschränkt sich diese Arbeit auf die Überprüfung der nachfolgenden Hypothesen:

1.1 Hypothese 1

Die Outlierdetection lässt sich durch Parameteranpassungen verbessern.
Im Appendix [8] zum Artikel von Li et al. wurden Parameteranpassungen vorgestellt. Anhand dieser kann das Ergebnis des Algorithmus für verschiedene Datensätze optimiert werden.

1.2 Hypothese 2

Die bei der Outlierdetection entstehenden mini-MSTs können zur Clusteroptimierung und Strukturierung genutzt werden.
Wie bereits erläutert, sind Outlierdetection und Clusteranalyse oft eng miteinander verknüpft. Die mini-MSTs, die während der Outlierdetection des Algorithmus von Li et al. [7] entstehen, nehmen immer möglichst kurze Kanten hinzu. Daher ist es denkbar, dass diese besonders ähnliche Datenpunkte enthalten und somit sinnvolle Minicluster bilden, die besonders bei groben Clustern feinere Struktur innerhalb des Clusters bieten.
Die Minicluster können mit Hilfe eines groben Vorclusterings automatisch gelabelt und zusammengefasst werden. Da die Minicluster insbesondere nur Inlier enthalten, wird erwartet, dass mit dieser Methode eventuelles Datenrauschen der Vorcluster erheblich reduziert wird.

1.2.1 Hypothese 2.1

Das Clustering kann durch Nutzung eines anderen Ähnlichkeitsmaßes verbessert werden.
Die Genauigkeit der Cluster kann möglicherweise gesteigert werden, wenn statt der euklidischen Distanz ein alternatives Ähnlichkeitsmaß genutzt wird.

1.3 Hypothese 3

Die Laufzeit des Gesamtalgorithmus lässt sich durch die Ersetzung des MST-Algorithmus verbessern.
Um diese Hypothese zu überprüfen, wird der Prim-Algorithmus [9], der im Artikel von Li et al. genutzt wurde, mit dem Borůvka-Algorithmus [10] sowie dem Karger-Klein-Tarjan-Algorithmus [11], der auf dem Borůvka-Algorithmus basiert, ersetzt. Besonders der Karger-Klein-Tarjan-Algorithmus hat das Potential, schneller als Prim zu sein, da er nur eine erwartete lineare Laufzeit bezüglich der Kantenanzahl hat.

Im Verlauf der Arbeit wird zunächst die Erstellung der Datensätze und der daraus resultierenden Graphen erläutert. Anschließend werden die für die Überprüfung der Hypothesen relevanten Algorithmen beschrieben. Zudem werden die Unterschiede zwischen der Algorithmusbeschreibung im Artikel von Li et al. und der tatsächlichen Umsetzung in deren Code hervorgehoben, sowie die Unterschiede zum Code dieser Arbeit erläutert. Auch werden neue Anpassungen eingeführt.
Im Ergebnisteil sind die Unterschiede zwischen den im Artikel vorgestellten Ergebnissen und den Ergebnissen der Codereplikation in dieser Arbeit dargestellt. Zudem sind die Auswirkungen vielversprechender Anpassungen auf die Analyse der erstellten medizinischen Datensätze zu sehen.
Anschließend werden die Anpassungsergebnisse in Hinblick auf die drei Hypothesen diskutiert.

2 Definitionen

- **Baum:** ein Baum ist ein Graph, der aus einer Menge an Knoten und verindenden Kanten besteht. Ein Baum ist zyklenfrei, ungerichtet und zusammenhängend. Knoten mit nur einer Kante werden als Blattknoten bezeichnet. Besteht ein Graph aus mehreren Bäumen, so wird er als **Wald** bezeichnet.
  Ist eine Wurzel definiert und ein Knoten w mit einem betrachteten Knoten v über eine Kante verbunden, kann die Beziehung zwischen diesen Knoten genauer definiert werden. Ist w weiter von der Wurzel entfernt als v, d.h. der Pfad von w zur Wurzel ist länger als der von v zur Wurzel, ist w das Kind (engl. child) von v. Ist w näher an der Wurzel als v, wird w als Elter (engl. parent) von v bezeichnet.

- **Minimaler Spannbaum (MST):**
  Sei $G = (V, E)$ ein ungerichteter zusammenhängender Graph mit zugewiesenen Kantengewichten. Ein *Spannbaum* ist ein ungerichteter Baum, der alle Knoten V von G verbindet. Die *Kosten* eines Spannbaums sind dabei die Summe aller Kantengewichte des Baumes.
  Ein *minimaler Spannbaum* (engl. **m**inimum **s**panning **t**ree) ist ein Spannbaum, dessen Kosten minimal zu allen möglichen Spannbäumen von G ist. Ein Graph kann mehrere minimale Spannbäume haben.

- **Eulertour:** ein zyklischer Pfad, der jede Kante des Graphen genau einmal besucht

- **f-heavy Kante:** Sei G ein Graph mit gewichteten Kanten und $w(x, y)$ das Gewicht der Kante $\{x, y\}$. Sei ferner F ein Wald in G, dann ist $F(x, y)$ der Pfad der x und y in F verbindet. $w_F(x, y)$ bezeichnet das *maximale Gewicht* einer Kante auf dem Pfad $F(x, y)$. Existiert kein Pfad zwischen x und y in F, dann gilt $w_F(x, y) = \infty$.
  Eine Kante ist **f-heavy**, wenn $w(x, y) > w_F(x, y)$. D.h. die Kante aus G zwischen x und y ist schwerer als das maximale Kantengewicht im Pfad zwischen den Punkten in F. Ist eine Kante nicht f-heavy, wird sie als F-light bezeichnet. [11]

- **full branching tree:** ein Baum, in dem jedes Blatt auf demselben Level ist und jeder interne Knoten mindestens zwei Kinder hat [12]

- **Adjazenzmatrix:** Eine Adjazenzmatrix ist eine Matrix an der am Index i, j der Abstand zwischen den Datenpunkten i und j gespeichert ist. Folglich ist eine Adjazenzmatrix symmetrisch, da der Eintrag an Index i, j gleich dem Eintrag an der Stelle j, i ist. Zudem ist die Diagonale 0, da an den Einträgen i, i der Abstand eines Punktes zu sich selbst liegt.

- **Zusammenhangskomponente:** Ein *zusammenhängender* Graph ist ein Graph, bei dem von jedem Knoten aus ein Pfad zu jedem anderen Knoten existiert.
  Eine *Zusammenhangskomponente* (ZHK) ist der maximale zusammenhängende Teilgraph eines Graphen.

3 Methoden

3.1 Grapherstellung

3.1.1 Erstellung der Datensätze

Die erstellte Patientendatentabelle aus der MIMIC-III-Datenbank [2, 3, 4] hat das in Tabelle 3.1 dargestellte Format:

Tabelle 3.1: Patientendatentabelle
4543 Einträge

Spalte	Datentyp
SUBJECT_ID	int64
is_dead	int64
AGE_AT_ADMISSION_YEARS	float64
mean blood pressure_mean	float64
cardiac index_mean	float64
glucose_mean	float64
heart rate_mean	float64
glasgow coma scale total_mean	float64
ICD9_CODE	object

*Spalten der erstellten **Patientendatentabelle** mit jeweiligem Datentyp. Nullwerte wurden durch typische gesunde Werte ersetzt.*

Der Code zur Erstellung der Datensätze ist in der Datei `data_processing.py` [13]. Der is_dead-Eintrag entspricht der HOSPITAL_EXPIRE_FLAG aus der `ADMISSIONS.csv` der MIMIC-III-Datenbank. Diese zeigt an, ob die Person verstorben ist (1 steht für verstorben, 0 für lebendig).
AGE_AT_ADMISSION_YEARS wurde aus der Differenz zwischen 'ADMITTIME' (admission time) und 'DOB' (date of birth) berechnet. Um nur das Alter in Jahren anzugeben, wurde das Ergebnis unter Beachtung von Schaltjahren durch 365,25 geteilt.
Weil aus Datenschutzgründen die echten Datumseinträge konsistent in die Zukunft verschoben wurden, sind alle 'ADMITTIME'-Einträge der Datenbank zwischen den Jahren 2100 und 2200. Dies beeinträchtigt nicht die Altersberechnung. Allerdings sind alle Personen über 89 aus Datenschutzgründen mit einem Alter von über 300 Jahren angegeben. Da diese Werte die Ergebnisse stark verzerren können, wurden alle Personen mit einem 'DOB' unter dem Jahr 2000 aus dem Datensatz entfernt, da das Geburtsdatum dieser Personen dementsprechend im Bereich 1800-1900 liegt.
In der Tabelle werden die Gesundheitsmarker Blutdruck (blood pressure), Herzarbeitsindex (cardiac

index), Blutzucker (glucose), Puls (heart rate) und Glasgow-Koma-Skala (glasgow coma scale) betrachtet. Der Herzarbeitsindex gibt die Herzleistung an [14], die Glasgow-Koma-Skala ist ein Maß für die Einschätzung des Bewusstseins einer Person [15]. Für die Erfassung der Gesundheitsmarker wurden die stündlich zusammengefassten Zeitreihendaten des MIMIC-Extract-Artikels [16] genutzt. Da die meisten stündlichen Daten pro Patient Nulleinträge sind, wurden mittels `.mean()` die Mittelwerte pro Patient und Krankenhausbesuch zusammengefasst. Wenn keine Werte einer Kategorie abgenommen wurden, ist das Ergebnis der Funktion ein Nullwert. Gibt es valide Werte, werden Nullwerte in der Messreihe bei der Berechnung ignoriert. Patienten, bei deren Besuch kein Wert in mindestens einer der betrachteten Kategorien abgenommen wurde, entfallen aus der Betrachtung und wurden aus der Tabelle entfernt.
Zur Vereinfachung der weiteren Berechnungen wurde angenommen, dass bei Patienten nicht gemessene Kategorien klinisch unauffällig waren. Daher wurden die Nullwerte dieser Kategorien durch die mittleren gesunden Werte, wie sie in Tabelle 3.2 angegeben sind, ersetzt.

Tabelle 3.2: Ersetzungsschema Nullkategorien

Kategorie	gesunder Wert
mean blood pressure_mean	75
cardiac index_mean	3.4
glucose_mean	115
heart rate_mean	80
glasgow coma scale total_mean	15

Schema der Ersetzung von Nullwerten durch gesunde Werte. Dies geschieht für Kategorien, deren Werte für den Patienten nicht gemessen wurden.

Der ICD9-Code ist eine Nummer, die für eine bestimmte Krankheit steht (s. Tabelle 3.3). [17] Da nah aneinanderliegende Codes nicht unbedingt ähnliche Krankheiten bedeuten, wird dieser Code als Zeichenkette gespeichert und als kategorischer Wert betrachtet. Um die Datenmengen für die Auswertung zu reduzieren, wurden die Daten nach den zehn meist vergebenen ICD9-Codes gefiltert. Dies verhindert zudem uneindeutige Codes, da die Datenbank die Kennziffern ohne führende Nullen und ihre Trennpunkte speichert und so Dopplungen vorkommen. So wird z.B. sowohl aus dem Code 004.2 für Shigella boydii (ein Bakterium) als auch aus dem Code 042 für Human immuno virus dis (HIV) der Eintrag 42. Für die zehn meistgenutzten Codes konnte verifiziert werden, dass für diese keine Uneindeutigkeiten bestehen.

Tabelle 3.3: ICD9-Code und Bedeutung

ICD9-Code	Titel aus Datenbank	Bedeutung
4019	Unspecified essential hypertension	essentielle Hypertonie (Bluthochdruck ohne erkennbare Ursache)
4280	Congestive heart failure, unspecified	Stauungsinsuffizienz des Herzens
42731	Atrial fibrillation	Vorhofflimmern und -flattern (Herzrhythmusstörung)
41401	Coronary atherosclerosis of native coronary artery	Koronararteriosklerose (arterielle Verschlusskrankheit des Herzens)
5849	Acute kidney failure, unspecified	nicht näher bezeichnetes (n.n.bez.) akutes Nierenversagen
25000	Diabetes mellitus without mention of complication, type II or unspecified type, not stated as uncontrolled	Diabetes mellitus ohne Angabe einer Komplikation oder Typ (Diabetes)
2724	Other and unspecified hyperlipidemia	Sonstige und n.n.bez. Hyperlipidämie (Fettstoffwechselstörung, z.B. erhöhtes Cholesterin)
51881	Acute respiratory failure	Lungenkollaps
5990	Urinary tract infection, site not specified	Infektion der Harnwege, ohne Angabe des Sitzes
53081	Esophageal reflux	Gastroösophagealer Reflux (unter Sonstige Affektionen der Speiseröhre)

Die 10 meist vergebenen ICD9-Codes (in absteigender Reihenfolge) mit ihrem englischen vollständigen Titel wie sie in der Datenbank gespeichert sind und ihre Bedeutung [17].

Zuletzt wurden Duplikate aus der Patiententabelle 3.1 entfernt. Zum Testen der Algorithmusvariationen wurden zwei Tabellen erstellt. Diese wurden aus Tabelle 3.1 erstellt und haben die Clusteringziele verstorben/lebendig (Tabelle 3.4) und ICD9-Code (Tabelle 3.5). Die Verteilung der Clusteringziele in der Patiententabelle ist in den Tabellen 3.6 und 3.7 angegeben.

Tabelle 3.4: Daten für das Clusteringziel verstorben/lebendig
4543 Einträge

Spalte	Datentyp
AGE_AT_ADMISSION_YEARS	float64
mean blood pressure_mean	float64
cardiac index_mean	float64
glucose_mean	float64
heart rate_mean	float64
glasgow coma scale total_mean	float64
ICD9_CODE	object

Spalten der erstellten Tabelle mit Clusteringziel verstorben/lebendig mit jeweiligem Datentyp. Nullwerte wurden durch typische gesunde Werte ersetzt.

Tabelle 3.5: Daten für das Clusteringziel ICD9 Code
4543 Einträge

Spalte	Datentyp
is_dead	int64
AGE_AT_ADMISSION_YEARS	float64
mean blood pressure_mean	float64
cardiac index_mean	float64
glucose_mean	float64
heart rate_mean	float64
glasgow coma scale total_mean	float64

Spalten der erstellten Tabelle mit Clusteringziel des ICD9 Codes mit jeweiligem Datentyp. Nullwerte wurden durch typische gesunde Werte ersetzt.

Tabelle 3.6: Häufigkeit der ICD9-Codes der Patientendatentabelle

Code	Häufigkeit
41401	2975
51881	523
4280	311
5849	303
42731	239
5990	79
4019	51
53081	23
25000	20
2724	19

Die Tabelle zeigt, wie häufig welcher ICD9-Code in der Patientendatentabelle (Tab. 3.1) vertreten ist.

Tabelle 3.7: Anzahl der Verstorbenen und nicht verstorbenen Patienten der Patientendatentabelle

Status	Häufigkeit
verstorben	215
lebend	4328

Die Tabelle zeigt, wie viele Patienten der Patientendatentabelle (Tab. 3.1) verstorben sind (Spalte `is_dead` == 1).

Die Codeumsetzung ist in der Datei `data_processing.py` im git-Repository der Arbeit zu finden [13]. Die Tabellen werden für einen schnellen Zugriff im `.pkl`-Format als Pandas [18] data frame gespeichert.

3.1.2 Erstellung des vollständigen Graphen

Aus einer gegebenen `.pkl`-Datei wird ein vollständiger Graph berechnet. Dazu wird aus den ausgelesenen Tabellendaten eine Feature-Matrix berechnet, die die transformierten numerischen und kategorialen Daten enthält. Die Umsetzung findet sich in der Methode `compute_distances_multi_dimensional()` der Datei `visualize.py` des Gits [13].
Dafür werden die numerischen Spalten mittels der `fit_transform()`-Funktion des `StandardScaler()` aus dem Scikit-learn `preprocessing`-Modul [19] standardisiert. Kategorische Spalten wie `ICD9_CODE` werden mithilfe der `fit_transform()`-Funktion des `OneHotEncoder(sparse_output=False, drop='first')` in binäre Indikatorspalten umgewandelt. Beispielsweise würde aus einer Spalte **Buchstaben** mit den Kategorien A, B und C stattdessen die Spalten **B** und **C** mit den Werten 0/1 (A ist demnach **B**=0 und **C**=0).
Aus der Feature-Matrix wird nun eine Adjazenzmatrix (Def. s. Abschnitt 2) berechnet. Mittels `pdist()` aus dem `scipy.spatial.distance`-Modul werden die paarweisen euklidischen Distanzen zwischen den Datenpunkten berechnet und anschließend mittels `squareform()` in eine symmetrische Matrix umgewandelt.
Wenn zur Überprüfung der Hypothese 2.1 (s. Abschnitt 1.2.1) eine Ähnlichkeitsmatrix berechnet werden soll, wird im ersten Schritt eine Variante der Gaußschen radialen Basisfunktion angewendet: Sei S die Ähnlichkeitsmatrix und $d_{i,j}$ der Eintrag an Index i, j der Adjazenzmatrix. Die Originalformel lautet: $S_{i,j} = e^{-\frac{d_{i,j}^2}{2\sigma^2}}$ [20]. In der hier verwendeten Variante wird die Ähnlichkeitsmatrix mit $S_{i,j} = e^{-\frac{d_{i,j}^2}{2}}$ berechnet. In dieser Funktion kein Parameter σ, der die Rate der Ähnlichkeitsabnahme beeinflusst, verwendet. Da bereits sehr viele Parameteranpassungen in dieser Arbeit untersucht werden und eine optimale Einstellung des Parameters nicht trivial ist, wird dieser damit auf 1 fixiert. Die Quadrierung der Distanzen führt dazu, dass größere Abstände bestraft und kleinere stärker belohnt werden. Das Ergebnis der Berechnung ist, dass alle Werte der Matrix zwischen 0 und 1 liegen, wobei 1 eine sehr hohe und 0 eine sehr geringe Ähnlichkeit anzeigt.

Da das Clustering auf minimalen Spannbäumen beruht und bevorzugt die kleinsten Kanten gewählt werden, müssen die Matrixwerte in einem zweiten Schritt invertiert werden: $S'_{i,j} = 1 - S_{i,j}$. Nun liegen alle Einträge der Ähnlichkeitsmatrix zwischen 0 und 1, wobei 0 eine hohe Ähnlichkeit bedeutet und 1 eine geringe Ähnlichkeit. Zusätzlich ergibt sich der Vorteil, dass die Einträge auf der Diagonalen (das Gewicht der Kante von dem Punkt zu sich selbst selbst) $S_{i,i} = 0$ sind und das Kantengewicht 0 als 'keine Kante' interpretiert werden kann.

3.2 MST-Algorithmen

In dieser Arbeit wird die Laufzeit dreier Methoden zur MST-Erstellung auf großen Datenmengen getestet. Dazu werden der Prim-Algorithmus, der Borůvka-Algorithmus sowie der Karger-Klein-Tarjan-Algorithmus genutzt, die im Folgenden vorgestellt werden.

3.2.1 Prim-Algorithmus

In diesem Abschnitt wird sich auf den Artikel [9] bezogen, der in dem Basisartikel [7] dieser Arbeit für den Prim-Algorithmus referenziert wurde.

Eine Beschreibung ist mittels Pseudocode gegeben (Algorithmus 1).
Der Prim-Algorithmus hat eine Zeitkomplexität von $O(|V|^2))$ bei einer Standardimplementierung ohne binary heap oder fibonacci heap wie sie in dem Code zum Basisartikel genutzt [1]. Daher wurde im Code dieser Arbeit aufgrund der Vergleichbarkeit eine Implementierung mit dieser Laufzeit verwendet (Vgl. [13], `prim.py`).

Algorithm 1 Prim algorithm

Require: undirected connected Graph $G = (V, E)$ with no negative edge weights, loops or parallel edges

randomly select start node s
$s \leftarrow$ visited = $true$, distance = 0
for all nodes $n \neq s$ **do**
    visited[n] $\leftarrow false$, distance[s] $\leftarrow \infty$
end for
while there are unvisited nodes **do**
    if s has predecessor **then**
        MST $\leftarrow$ edge{s, predecessor[s]}
    end if
    for all unvisited nodes u incident to s **do**
        if distance[u] > weight of edge $\{s, u\}$ **then**
            distance[u] $\leftarrow$ weight of edge $\{s, u\}$
            predecessor[u] $\leftarrow s$
        end if
    end for
    $s \leftarrow$ minimum-weight unvisited node incident to s
    visited[s] $\leftarrow true$
end while
return MST

3.2.2 Borůvka-Algorithmus

Der Borůvka-Algorithmus hat eine Laufzeit von $O(|E|log|V|)$ bei einer einfachen, nicht-parallelisierten Implementierung. [10] Eine Beschreibung ist mittels Pseudocode gegeben (Algorithmus 2).
Der Algorithmus geht im Original von einzigartigen Kantengewichten aus. Bei der Programmierung können daraus entstehende Probleme behoben werden, indem bei gleichwertigen Gewichten eine konsistente Auswahlregel definiert wird. Da in der Implementation dieser Arbeit alle Knoten eine einzigartige ID haben, ist jede Kante über ihre Endpunkte eindeutig definiert. Bei einer Auswahlsituation wird diejenige Kante gewählt, deren lexikographische Ordnung der Knoten-IDs kleiner ist. D.h. zunächst wird nach dem jeweils kleineren Endpunkt sortiert und sollte dieser gleich sein, nach dem größeren. Es ist dabei ausgeschlossen, dass beide Kanten dieselben 2 Endpunkte haben. Wäre dies der Fall, wäre es dieselbe Kante und es müsste nicht gewählt werden.
Dieser Algorithmus spielt eine zentrale Rolle in dem ersten MST-Erzeugungsalgorithmus der durchschnittlich nur lineare Zeit benötigt - dem Karger-Klein-Tarjan-Algorithmus (KKT). Da in diesem Algorithmus einzelne Boruvka-Steps ausgeführt werden und mehr Informationen als die gewählten

Algorithm 2 Borůvka's algorithm

Require: undirected connected Graph $G = (V, E)$
Require: for each edge e with real weight $w(e)$ assume $w(e) \neq w(e')$ for $e \neq e'$
1: initially all edges are uncolored and each vertex of G is a trivial blue tree
2: **while** there is more than one blue tree **do** ▷ Borůvka step
3: **for each** blue tree T **do**
4: select the minimum-weight uncolored edge incident to T
5: **end for**
6: color all selected edges blue
7: **end while**
8: **return** blue edges

Kanten zurückgegeben werden müssen, wurde dafür eine alternative Betrachtung des Algorithmus verwendet. Hier werden durch ausgewählte Kanten verbundene Knoten zu einer großen Komponente, also einem neuen Knoten, zusammengeführt (Algorithmus 3). Auch in dieser Variante können gleiche Kantengewichte durch eine konsistente Auswahlregel gelöst werden. Die Umsetzung dieser

Algorithm 3 Borůvka Step alternative

Require: undirected connected Graph $G = (V, E)$
Require: for each edge e with real weight $w(e)$ assume $w(e) \neq w(e')$ for $e \neq e'$. Each edge is not (yet) determined MST-Edge
1: initially all edges are uncolored
2: **for each** vertex v of G **do** ▷ Coloring Step
3: select the minimum-weight uncolored edge incident to v
4: **end for**
5: color all selected edges blue
6: $V' \leftarrow$ each blue tree replaced by a single vertex ▷ Contraction Step
7: contracted Graph $G' = (V', E)$
8: eliminate all loops (i.e. edges in the same blue tree) and parallel edges except the lowest of G'
9: **return** G', selected edges

Arbeit von Borůvka mit Rückgabe von Kanten oder eine Adjazenzmatrix sowie des Borůvka Steps findet sich in der Datei `boruvka.py` des Codes [13].

3.2.3 Karger-Klein-Tarjan-Algorithmus

Der Algorithmus von Karger, Klein und Tarjan (KKT) [11] ist ein randomisierter Algorithmus für das Ermitteln minimaler Spannbäume. Dieser läuft in erwarteter linearer Zeit, d.h. dass durch die Randomisierung zwar keine lineare Zeit garantiert werden kann, diese aber im Durchschnitt zu erwarten ist. Genauer ist die Laufzeit im Durchschnitt als $O(m)$ angegeben, wobei m die Anzahl der Kanten im Graphen ist. Im schlechtesten Fall hat KKT die Laufzeit des Borůvka-Algorithmus.
Eine Pseudocodebeschreibung ist in Algorithmus 4 gegeben. Die Umsetzung des Pseudocodes sowie der nachfolgend beschriebenen Algorithmen, die zur Bestimmung der f-heavy Kanten notwendig sind, findet sich in den Dateien `karger_klein_tarjan.py`, `f_heavy_edges.py` und

`lowest_common_ancestor.py` des Codes [13].

Algorithm 4 Karger-Klein-Tarjan_algorithm

Require: undirected Graph $G = (V, E)$s

1: G, contracted_edges' $\leftarrow$ `boruvka_step(G)` ▷ First Borůvka Step (as described in algorithm 3)
2: G, contracted_edges" $\leftarrow$ `boruvka_step(G)` ▷ Second Borůvka Step
3: $E_H \leftarrow E$ with each $e \in E$ being chosen independently with probability 0.5
4: $F \leftarrow$`Karger-Klein-Tarjan_algorithm(H)` ▷ first recursion
5: f_heavy_edges $\leftarrow$ `f_heavy_edges(G, F)` ▷ computation of f_heavy_edges of G in relation to F described in chapter 3.2.3.1
6: $G \leftarrow G(V, E \backslash$f_heavy_edges$)$ ▷ delete f_heavy_edges from G
7: $F' \leftarrow$ `Karger-Klein-Tarjan_algorithm(G)` ▷ second recursion
8: **return** contracted_edges' $\bigcup$ contracted_edges" $\bigcup E_{F'}$ ▷ $E_{F'}$ describes the edges of F'

3.2.3.1 Bestimmung von f-heavy Kanten

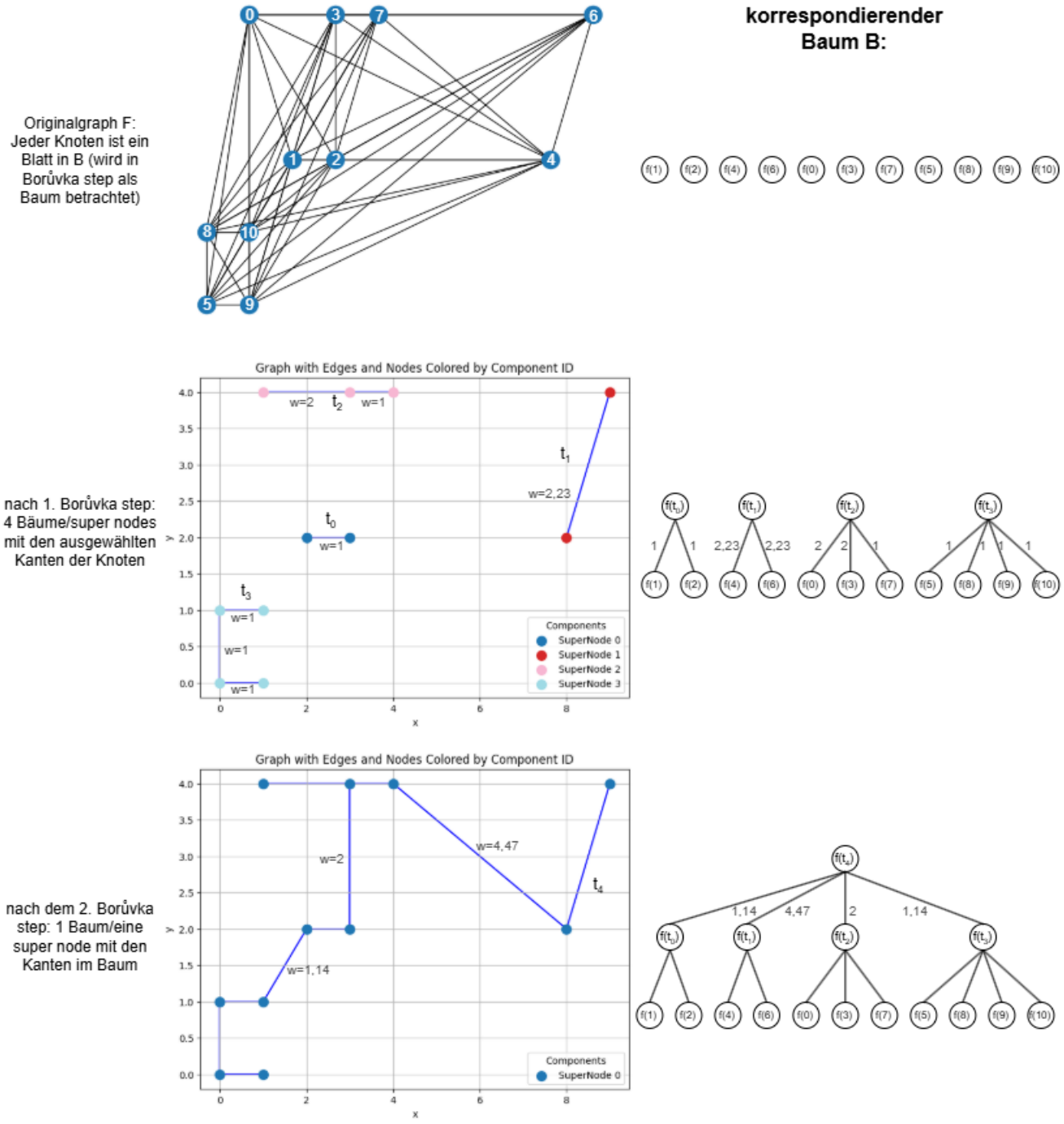
Im Karger-Klein-Tarjan-Algorithmus wird die Eigenschaft, ob eine Kante des Graphen G zu einem minimalen Wald F von G f-heavy ist, genutzt. Dies wird damit begründet, dass für alle F von G gilt, dass keine f-heavy Kante im minimalen Spannwald von G sein kann [11]. Eine Definition der f-heavy-Eigenschaft ist in Abschnitt 2 gegeben.
Karger, Klein und Tarjan geben keine Beschreibung, wie in linearer Zeit bestimmt werden kann, welche Kanten von G f-heavy sind. Daher wurde sich in dieser Arbeit an der Vorgehensweise, die von Bint beschrieben wurde, orientiert [12]. Zunächst wird F mittels Kings Methode [21] in einen *full branching tree* B (Definition s. Abschnitt 2) konvertiert. Dieser Baum B hat die Eigenschaft, dass die Kosten des Pfads zwischen dem Knotenpaar $x, y \in F$ dieselben sind wie auf dem Pfad in B. Eine Beschreibung der Grapherstellung wird in Abschnitt 3.2.3.2 gegeben.
B dient als Input für Hagerup's Methode [22] zur Bestimmung der maximalen Kante auf einem Pfad in einem *full branching tree*. Da ich den Code, den Hagerup in seinem Artikel mitgeliefert hat, unverändert von der Sprache D in Python übertragen habe, geht eine detaillierte Beschreibung über den Rahmen der Arbeit hinaus. Mit dem maximalen Gewicht auf dem Pfad in B und somit auch F kann für jede Kante von G bestimmt werden, ob sie f-heavy ist.
Per Konstruktion von B sind alle Knoten von F Blattknoten. Hagerup's Methode kann jedoch nur das größte Kantengewicht auf einem Pfad zwischen Vorfahren und Nachfahren bestimmen. Um die Kanten, für die wir wissen wollen, ob sie f-heavy sind, auch korrekt abzufragen, muss also der tiefste gemeinsame Vorfahre (*lowest common ancestor*) bestimmt werden. Dies ist im Abschnitt 3.2.3.3 beschrieben. Mittels des LCA kann nun das maximale Gewicht im Pfad zwischen x und y in B als $\max(B(\mathrm{LCA}(x, y), x), B(\mathrm{LCA}(x, y), y))$ bestimmt werden.

3.2.3.2 Kings Methode: Erstellung des *full branching tree*

Um basierend auf einem gegebenen Graphen F den *full branching tree* B zu erstellen, werden Borůvka Steps genutzt (Algorithmus 3). Der resultierende Graph B hat die Eigenschaft, dass die Kosten des Pfads zwischen dem Knotenpaar $x, y \in F$ dieselben sind, wie auf dem Pfad in B. Für die Umwandlung im Kontext des Karger-Klein-Tarjan-Algorithmus wird Kings Methode nur auf dem Wald F des

Gesamtgraphen G angewendet. Die Schritte nach der Erstellung von B in Kings Artikel werden nicht angewendet, da Hagerup eine simplere Methode mit ebenfalls linearer Zeitkomplexität vorstellte [22]. Im initialen Schritt zur Erstellung von B werden alle Knoten v von F als Blattknoten $f(v)$ zu B hinzugefügt. Diese werden als einzelne Bäume betrachtet. Solange mehr als ein Baum existiert, wird ein Borůvka Step (s. Algorithmus 3) ausgeführt. Für jeden neuen (zusammengefassten) Baum t wird der Knoten $f(t)$ zu B hinzugefügt. Sei S das Set der Bäume, die zu t zusammengefasst wurden. Dann wird für jedes $s \in S$ eine Kante $\{f(t), f(s)\}$ zu B hinzugefügt. Das Gewicht der Kante entspricht den Kosten der Kante, die s im Borůvka Step ausgewählt hat. Eine Visualisierung ist in Abbildung 3.1 gegeben.

Abbildung 3.1: Beispiel *full branching tree*

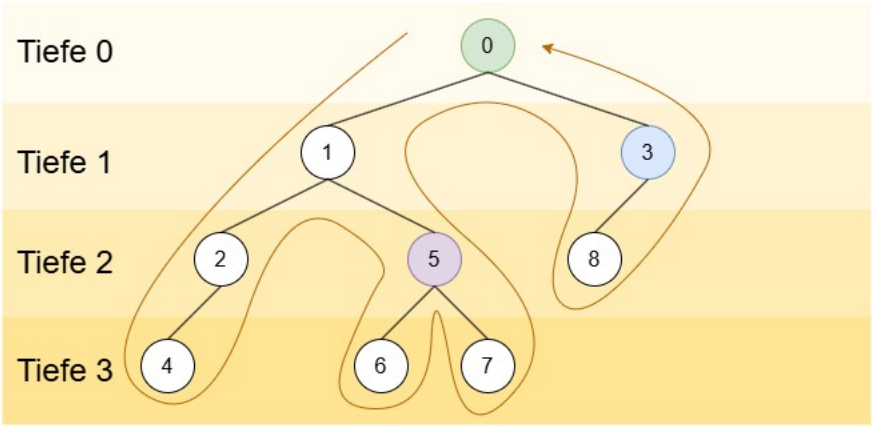


Erstellung eines full branching tree aus einem vollständigen Graphen F. Zunächst wird jeder Knoten des Graphen F in einen Blattknoten des Baums B übertragen. Nach jedem auf dem Graphen F simulierten Boruvka Step, wird für jeden Baum in F ein neuer Knoten in B eingefügt. Dieser erhält eine Kante zu jedem Baum t_i aus dem vorhergehenden Schritt, den er vereint. Das Gewicht der Kante entspricht dem Gewicht der Kante, die Baum t_i aus dem vorhergehenden Schritt ausgewählt hat.

3.2.3.3 Bestimmung des *lowest common ancestor* (LCA)

Um die Bestimmung des tiefsten gemeinsamen Vorfahrens (*lowest common ancestor*, LCA) zweier Knoten im Baum in linearer Zeit zu gewährleisten, nutze ich die Eulertour des Baumes (s. Abschnitt 2 Definitionen). Mittels dieser kann das LCA-Problem auf das *Ranged Minimum Query (RMQ)*-Problem reduziert werden [23]. Das RMQ-Problem ist die Bestimmung des Minimums in einem bestimmten Arraybereich.
Die konkrete Bestimmung des LCA erläutere ich anhand des Beispiels in der Graphik 3.2.

Abbildung 3.2: Beispiel LCA-Bestimmung



Listen																	
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Eulertour	0	1	2	4	2	1	5	6	5	7	5	1	0	3	8	3	0
Tiefe	0	1	2	3	2	1	2	3	2	3	2	1	0	1	2	1	0
erstes Auftreten	0	1	2	13	3	6	7	9	14								

Bei einmaligem Durchlaufen des Baums mittels Tiefensuche werden die drei Listen 'Eulertour', 'Tiefe' und 'erstes Auftreten' erstellt. Mittels dieser Listen kann der der LCA ermittelt werden.

Da bei einer echten Eulertour jede Kante des Baumes nur einmal genutzt werden darf, betrachte ich die ungerichteten Kanten des Graphen als 2 Kanten (Hin-und Rückkante). Im Code wird die Datenstruktur jedoch nicht verändert, es wird lediglich die ungerichtete Kante zweimal statt einmal besucht. Um die Eulertour zu realisieren wird die Reihenfolge der besuchten Knoten bei einer Tiefensuche von der Wurzel aus notiert, wobei die Knoten sowohl beim ersten Besuch als auch beim erneuten Besuchen notiert werden. In Abb. 3.2 ist der Verlauf der Eulertour als Pfeil dargestellt. Bei der Initiierung werden werden bei der Tiefensuche folgende drei Listen erstellt:

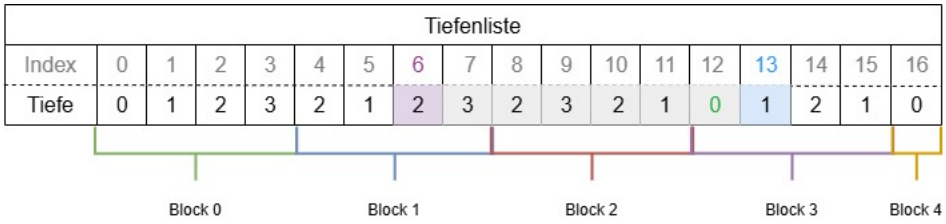
- **Eulertour:** beinhaltet Knoten-ID's in der Reihenfolge der Eulertour
- **Tiefe:** speichert die Tiefe der Knoten für jeden Eintrag der Eulertourliste
- **erstes Auftreten:** ordnet jedem Knoten den Index seines ersten Auftretens in der Eulertourliste zu

Da ein Baum mit n Knoten genau $n-1$ Kanten hat und über jede Kante zweimal gegangen wird (beim hineingehen in und hinausgehen aus dem Teilbaum) und das Listenupdate pro Knoten O(1) Zeit benötigt, ist die Laufzeit der Initiierung $O(2(n-1)) = O(n)$.
Zur Bestimmung des LCA der Knoten u und v muss das Minimum in der Tiefenliste zwischen dem ersten Auftreten von u und dem ersten Auftreten von v ermittelt werden. Dazu werden die Indizies des ersten Auftretens von u bzw. v in der entsprechenden Liste nachgeschlagen. Im Beispiel ist $u = 5$ und $v = 3$. In der Liste **erstes Auftreten** muss dementsprechend an Index $u = 5$ und $v = 3$ nachgeschlagen. Das Ergebnis ist Index $l = 6$ als erstes Auftreten in der Eulertourliste von u und Index $r = 13$ als erstes Auftreten von v (in Abb. 3.2 rechteckig markiert).
Da per Konstruktion die Tiefe des Knotens an Index i der Eulertourliste auch in der Tiefenliste an Index i gespeichert ist, sind l und r die Grenzen des RMQ-Problems.
Um die minimale Tiefe in den Grenzen zu bestimmen wird *square root decomposition* angewendet. Näheres dazu im Abschnitt 3.2.3.4.
Nachdem die minimale Tiefe an Index m in den Grenzen l und r (im Beispiel farbig hinterlegte Einträge der Tiefenliste) gefunden wurde, kann der LCA einfach durch nachschlagen der Knoten-ID an Index m der Eulertourliste bestimmt werden. In Abb. 3.2 ist das Minimum der Einträge in der Tiefenliste zwischen Index $l = 6$ und $r = 13$ null (grün markiert). Dieser Wert steht an Index $m = 12$. An demselben Index steht in Eulertourliste die Knoten-ID 0. Damit ist der LCA der Knoten 5 und 3 = 0.

3.2.3.4 Minimumbestimmung mittels *square root decomposition*

Die Idee von *square root decomposition* ist, eine Liste in Blöcke zu unterteilen und diese vorzuverarbeiten [24]. Die Anwendung dieser Technik für das RMQ-Problem erläutere ich anhand der Fortführung des Beispiels in Abb. 3.2.
Im Vorverarbeitungsschritt wird die Liste in Blöcke der Größe $\lfloor\sqrt{n}\rfloor$ mit n = Länge der Tiefenliste, unterteilt. Wenn die Länge n keine Quadratzahl ist, haben alle Blöcke bis auf den letzten die volle Blockgröße. Da in dem Beispiel die Liste 17 Einträge hat, wird diese in 5 Blöcke mit maximaler Größe $\lfloor\sqrt{17}\rfloor = 4$ unterteilt. Der letzte Block hat dabei nur einen Eintrag. Eine Veranschaulichung ist in Abb. 3.3 gegeben. Die Blöcke werden nicht explizit gespeichert, da die Blockzugehörigkeit eines Eintrags an Index i einfach durch Block$(i) = i//$ Blockgröße berechnet werden kann.

Abbildung 3.3: Beispiel square root decomposition



Tiefenliste																
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Tiefe	0	1	2	3	2	1	2	3	2	3	2	1	0	1	2	1	0

Listen für square root decomposition					
Index	0	1	2	3	4
Blockminimum	0	1	1	0	0
Index Minimum	0	5	11	12	16

Die Tiefenliste wird in 5 Blöcke der Größe ≤ 4 unterteilt. Für jeden dieser Blöcke wird am Index der Blocknummer die minimale Tiefe in der Liste 'Blockminium' und der Index der minimalen Tiefe des Blocks in 'Index Minimum' gespeichert.

Bei einmaliger Iteration über die Tiefenliste wird die minimale Tiefe jedes Blocks b bestimmt und in der Blockminimum-Liste an Index b festgehalten. Der Index des minimalen Eintrags t in der Tiefenliste wird dabei in der Index-Minimum-Liste ebenfalls an Stelle b gespeichert. Beispielsweise ist der minimale Eintrag der Tiefenliste für Block 1 ($b = 1$) in Abb. 3.3 die Tiefe 1 an Index $t = 5$. Daher wird in Blockminimum[b] die 1 und in der Index-Minimum-Liste an Index $b = 1$ der Eintrag $t = 5$ gespeichert.
Da im Vorverarbeitungschritt die Tiefenliste einmalig durchlaufen wird und pro Eintrag der Liste ein Aufwand von $O(1)$ besteht, ist die Laufzeit $O(n)$ mit n =Länge der Tiefenliste.
Nach der Vorverarbeitung können RMQ-Anfragen in nur $O(\sqrt{n})$ gelöst werden. Dazu werden zwei Fälle unterschieden. Diese sind abhängig von den Blöcken, in denen sich die Indexgrenzen befinden, zwischen denen das Minimum bestimmt werden soll. Im Beispiel ist die linke, also kleinere, Indexgrenze $l = 6$, da der Index des ersten Auftretens von $u = 5$ und somit der Index der Tiefe von u sechs ist. Analog ist die rechte Grenze $r = 13$.
Im ersten Fall zur Minimumsbestimmung liegen die linke Indexgrenze l und die rechte Grenze r in demselben Block. Da nicht davon ausgegangen werden kann, dass alle Werte des Blocks mit den Grenzen abgedeckt sind, wird das Minimum ohne die im Vorverarbeitungsschritt erzeugten Werte berechnet: $RMQ(l, r) = \min_{i=l}^{r}$ Tiefenliste[i] . Da ein Block maximal $\sqrt{n}$ groß ist, ist die Laufzeit für den ersten Fall $O(\sqrt{n})$ mit n =Länge der Tiefenliste.
Im zweiten Fall liegen die linke und rechte Indexgrenze l in unterschiedlichen Blöcken. In diesem Fall kann die vorverarbeitete Blockminimumliste für die vollständigen Blöcke zwischen dem Block von l und r genutzt werden: Seien b_l, b_r die Blocknummern der Indexgrenzen l und r des RMQ-Problems und sei s die maximale Blockgröße.
Dann ist die Lösung von
$RMQ(l, r) = \min($ $\min_{i=l}^{(b_l+1)s-1}$ (Tiefenliste[i]), $\min_{i=b_l+1}^{(b_r-1)}$ (Blockminimum[i]), $\min_{i=b_r \cdot s}^{r}$ (Tiefenliste[i])).
In dem Beispiel trifft der zweite Fall zu. Also ist $RMQ(6, 13) = \min(\min(2, 3), \min(1), \min(0, 1)) = \min(2, 1, 0) = 0$.

Die Laufzeit des zweiten Falls ist ebenfalls $O(\sqrt{n})$, da $2\sqrt{n}$ Vergleiche für die Blöcke von l und r durchgeführt werden und der Zugriff auf die vorverarbeiteten Minima der eventuellen Blöcke dazwischen $O(1)$ ist.

3.3 MST-Vorclustering

MST-Clusteringtechniken machen sich die räumliche Nähe ähnlicher Datenpunkte zunutze [25]. In dieser Arbeit wird ein simpler und intuitiver Ansatz zum Vorclustering bei bekannter Anzahl der gesuchten Cluster verwendet. Wenn x Cluster gesucht sind, werden die $x-1$ größten Kanten aus dem MST der Datenpunkte entfernt. Dadurch entstehen x Zusammenhangskomponenten (Def. s. Abschnitt 2). Diese entsprechen den x Clustern.
Die Zusammenhangskomponenten (ZHK) können mit einem simplen Union Find-Algorithmus unter Nutzung der Tiefensuche bestimmt werden. Die Variante, die in dieser Arbeit aufgrund von Adjazenzmatrizen genutzt wird, ist per Pseudocode beschrieben (Alg. 5 und 6). Die Implementierung ist in der Datei `boruvka.py` des Codes dieser Arbeit [13]. Die Punkte in den Zusammenhangskomponenten 0 bis $x-1$ werden durch den Union Find-Algorithmus mit der Nummer ihrer ZHK gelabelt.

Algorithm 5 Union Find

Require: adjacency matrice *Adjm*
length ← size(len(Adjm))
c ← list of size *length* initialized to ∞ ▷ vertice-to-component list
component_counter ← 0
for $v = 0$ to *length*-1 **do**
    if $c[v] == \infty$ **then** ▷ vertice has no assigned component
        c ← `dfs(c, v, component_counter, Adjm)` ▷ perform depth first search
        component_counter ← *component_counter* +1
    end if
end for
return c

Algorithm 6 depth first search for Union Find

Require: vertice-to-component list c, current vertice v, *component_counter*, adjacency matrice *Adjm*
length ← size(len(*Adjm*))
$c[v]$ ← *component_counter* ▷ c also serves as visited array
for $i = 0$ to *length*-1 **do**
    if $c[j] == \infty$ & $Adjm[v][j] \neq 0$ **then** ▷ j is not visited and vertices v and j are connected
        c ← `dfs(c, v, component_counter, Adjm)` ▷ recursion of depth first search
    end if
end for
return c

3.4 MST-Outlierdetection

Der nachfolgend erläuterte Algorithmus zur Outlierdetection von Li et. al. [7] nutzt zur Anpassung an unterschiedliche Datendichten *mini-MST*'s. Diese basieren auf einem großen MST, der die gesamten Daten umfasst. Für die Berechnung des MSTs wird im Artikel von Li et al. der Prim-Algorithmus verwendet. In diesem Kapitel sind die Funktionsweise des Algorithmus, die Unstimmigkeiten zwischen Artikel und Code und wie mit diesen umgegangen wurde, das vorgeschlagene Parametertuning sowie die Abwandlung des Algorithmus zur Clusterbestimmung beschrieben.

3.4.1 Funktionsweise

Zunächst wird ein MST auf den Daten berechnet. Im Artikel wird dazu der euklidische Abstand zwischen den Daten und der Prim-Algorithmus genutzt.
Zur Outlierbestimmung werden zwei Parameter eingeführt. Der globale *threshold of termination* bestimmt das Ende der Clusterberechnung, um die restlichen Punkte als Outlier zu klassifizieren. Die Berechnung wird in Abschnitt 3.4.2 erläutert. *Least number* bestimmt, wie viele Datenpunkte ein Cluster (und damit ein gefundener mini-MST) mindestens haben muss, damit es nicht als Outliercluster gilt. Die Formel zur Berechnung lautet: $least\_number = ROUND(\sqrt{\frac{N}{n}})$ mit N =Anzahl der Gesamtdatenpunkte und n =Dimension der Daten.
Über die aufsteigend sortierten MST-Kanten wird iteriert. Wenn (1) keiner der Endpunkte der Kante gelabelt wurde und (2) der Durchschnitt der nächsten 6 Kanten inklusive der betrachteten Kante den *threshold of termination* nicht überschreitet, wird ein mini-MST erstellt.
Um zu bestimmen, wie viele Kanten zum MST hinzugefügt werden, wird eine adaptive Abbruchbedingung *aec(e_i)* (adaptive exit condition) genutzt. Diese bestimmt, ob die Erstellung des mini-MST an der Kante e_i aufhören soll. Mehr zu der genauen Berechnung in Abschnitt 3.4.2. Kanten werden ausgewählt, indem ausgehend von dem Startpunkt die kleinste anliegende Kante auf dem Gesamt-MST hinzugenommen wird, deren anderer Endpunkt noch nicht gelabelt ist. Dieser Punkt ist nun gelabelt und der neue Ausgangspunkt für die Auswahl einer neuen Kante. Wenn die Hinzunahme der neuen Kante e_i den *aec(e_i* überschreiten würde, ist die Berechnung des mini-MST beendet.
Wenn die Anzahl der Kanten des mini-MSTs *Least number* überschreitet, werden die Endpunkte der Kanten als Inlier klassifiziert.

3.4.2 Abweichungen zum Artikel in der Umsetzung

Da der Code von Li et al. [1] einige Unterschiede zum Artikel [7] aufweist und um Anpassungen für die spätere Analyse zu erleichtern, habe ich die Codebeschreibung des Artikels in der Datei `mmod_original.py` selbst implementiert [13].
Folgende Unterschiede finden sich zwischen Originalcode und Artikel:

- Berechnung ***aec(e_i)* (adaptive exit condition):**
  Im Artikel ist *aec(e_i)* als $\overline{MEW} + \sqrt{\sum_{i=1}^{|MEW|}(MEW_i - \overline{MEW})^2}$ gegeben. MEW steht für *mini-edge weight set* und enthält die Kantengewichte, die dem mini-MST hinzugefügt wurden. In dem Code wird in Zeile 86 die Standardabweichung verwendet. D.h. es wird

$\overline{MEW} + \sqrt{\frac{\sum_{i=1}^{|MEW|}(MEW_i - \overline{MEW})^2}{N}}$ berechnet. Ich gehe von einem Fehler im Code aus, da in der Replikation der Auswertungen aus dem Zusatzmaterial data_sheet1 die Codevariante mit der Standardabweichung konsistent schlechter klassifiziert hat als die anderen Varianten Näheres zu dieser Auswertung in Abschnitt 4.1. Daher verwende ich die im Artikel angegebene Formel.

- Berechnung ***threshold of termination*** **(T_t)**:
  Der Artikel berechnet T_t als $\overline{d} + \sqrt{\sum_{i=1}^{N-1}(d_i - \overline{d})^2}$ mit $\overline{d} = \frac{\sum_{i=1}^{N-1} d_i}{N-1}$. Sowohl bei der Indizierung der Summenformel in T_t als auch in $\overline{d}$ gehe ich von einem Fehler aus, da im Text von $\overline{d}$ als als durchschnittliches Gewicht **aller** Kanten des MST die Rede ist. Demnach müsste die Formel: $\overline{d} = \frac{\sum_{i=1}^{N} d_i}{N}$ lauten. Diese Version der Formel wurde auch im Originalcode in der Funktion `get_mean_std(edge_set)` in Zeile 36 umgesetzt. Diese Variante verwende ich ebenfalls. Da die Indizierung für T_t und $\overline{d}$ identisch ist, habe ich $T_t = \overline{d} + \sqrt{\sum_{i=1}^{N}(d_i - \overline{d})^2}$ in meinem Code verwendet. Es gibt keine Referenz zu der Indizierung der Summenformel in T_t im Originalcode, da dort nur die auf einen Datensatz angepasste Variante steht für die $T_t = \overline{d}$ gilt. Mehr zu den Parameteranpassungen in Abschnitt 3.4.3.

- **Besonderheiten** bei Verwendung einer **Ähnlichkeitsmatrix**:
  Da die Werte einer Ähnlichkeitsmatrix zwischen 0 und 1 liegen, verzichte ich im gesamten Algorithmus darauf, die Kanten mit dem T_t zu gewichten, da die Teilung durch T_t eine größere Streuung der Werte bedeutet. Zudem sind die Kanten der Ähnlichkeitsmatrix bereits gewichtet (s. Abschnitt 3.1.2).

3.4.3 Vorgeschlagenes Parametertuning

Im Appendix data_sheet1 [8] sind konkrete Werte für Parameteranpassungen zur Verbesserung der Algorithmusleistung auf verschiedenen Datensätzen gegeben:

- Der Standard-T_t (*threshold of termination*) wird durch $\overline{d}$ ersetzt, also dem durchschnittlichen Gewicht aller Kanten.
- Der erste Wert von MEW wird auf 1 gesetzt, falls die erste Kante sehr groß ist. MEW enthält die Kantengewichte, die dem mini-MST hinzugefügt wurden.
- *aec(e_i)* wird durch $\overline{MEW}$ ersetzt.

Durch einen Zufallsbefund stellte ich fest, dass eine weitere Tuningmöglichkeit ist, die Kanten des großen MSTs mehrfach einzufügen. Die Ergebnisse dieser Anpassung für eine kleine Datenmenge sind im Abschnitt 4.2 dargestellt. Zusätzlich wurde dieser Parameter beim Clustering auf den MIMIC-III-Datensätzen verwendet (Abschnitte 4.3 und 4.4).

3.4.4 Nutzung zur Reduktion von Datenrauschen und Strukturierung

Um Hypothese 2 (Abschnitt 1.2), dass die mini-MSTs sich für die Clusteroptimierung und Clusterstrukturierung eignen können, zu überprüfen, wurde der Outlierdetectionalgorithmus leicht abgewandelt.

Wenn ein zurückgegebener mini-MST groß genug ist, dass er nicht als Outliercluster verworfen wird, wird aus diesem zusätzlich zur Inlierklassifizierung der Punkte ein *Minicluster* erstellt. Die Miniclu-ster sollen bei separater Speicherung eine Feinstrukturierung der gröberen Cluster bieten. Dabei muss beachtet werden, dass keine Punkte doppelt in ein Minicluster gelesen werden und auch keine Da-tenpunkte mehrfach in verschiedenen Clustern liegen. Ersteres ist trivial und letzteres lässt sich ohne neue Datenstrukturen einfach überprüfen, indem sichergestellt wird, dass der Punkt bisher noch nicht als Inlier klassifiziert wurde.
Nach Terminierung des Algorithmus werden die Minicluster zusätzlich zu den Inlier-/Outlierlabels zurückgegeben. Anschließend wird überprüft, in welcher Zusammenhangskomponente des groben Vorclusterings (s. Abschnitt 3.3) sich die meisten Punkte des Miniclusters befinden. Das Cluster er-hält das Label dieser ZHK.
Da die Minicluster zu klein für eine Analyse sind, werden anschließend die Cluster mit demselben Label zusammengefasst. So werden effektiv die Outlier aus den Clustern des Vorclusterings entfernt und das Datenrauschen vermindert. Der Vorteil dieser Methode besteht darin, dass die Information, welche Substrukturen (Minicluster) sich über mehrere ZHK-Cluster erstrecken, erhalten werden kann.

4 Ergebnisse

4.1 Unterschiede zwischen Clusteringartikel und tatsächlichem Ergebnis des Algorithmus

Da der Code von Li et al. [1] einige Unterschiede zum Artikel [7] aufweist und um Anpassungen für die spätere Analyse zu erleichtern, habe ich die Codebeschreibung des Artikels in der Datei `mmod_original.py` selbst implementiert [13]. Die Datensätze sind im git-Repository [1] zum Artikel verfügbar.
Um zu verifizieren, dass die Standardabweichung in der Umsetzung der $aec(e_i)$-Berechnung ein Fehler ist, habe ich die Codevarianten mit den angegebenen Ergebnissen verglichen und in Abb. 4.1 visuell gegenübergestellt. Ich konnte jedoch die Outlier, die im Zusatzmaterial data_sheet1 [8] mit den dort angegebenen Anpassungen gefunden wurden, nicht replizieren. Die roten Datenpunkte sind als Outlier und die blauen Punkte als Inlier klassifiziert. Abgebildet ist das Ergebnis des selbstgeschriebenen Algorithmus für drei Durchläufe pro Datensatz:

- **ohne Parameteranpassung:** Algorithmus mit den gewählten Formeln wie oben beschrieben.
- **mit Parameteranpassung:** Im data_sheet1 sind konkrete Werte für Parameteranpassungen gegeben (s. Abschnitt 3.4.3). Zusätzlich wird für jeden Datensatz angegeben, welche Parameter jeweils angepasst wurden. Die Visualisierung zeigt meine Algorithmusumsetzung mit denselben Anpassungen.
- **mit Parameteranpassung und Standardabweichung in $aec(e_i)$:** meine Algorithmusumsetzung mit den im data_sheet angegebenen Anpassungen sowie der $aec(e_i)$ wie im Originalcode mit Standardabweichung berechnet.

Unten in der Graphik sind zum Vergleich die Ergebnisse, die im data_sheet angegeben wurden, zu sehen. Es ergeben sich folgende Aussagen:

(a) Die Ergebnisse aus dem Appendix konnten nicht reproduziert werden. Die Resultate aller getesteten Varianten klassifizieren die Inlier und Outlier schlechter als angegeben.

(b) **three clusters:** Die Varianten 'mit Parameteranpassung' und 'ohne Parameteranpassung' liefern akzeptable Ergebnisse, wobei die Variante 'mit Parameteranpassung' schlechter die Inlier klassifiziert. Die Variante mit den im Appendix angegebenen Parameteranpassungen und der im Code vorhandenen Standardabweichung hat einer sehr ungenaue Klassifizierung und wertet eine Punktwolke als reine Outlier.

(c) **two densities:** Die Ergebnisse aller 3 Varianten sind ungenau. In den Varianten mit den angegebenen Parameteranpassungen sind die niedrigen Punktdichten größtenteils als Outlier klassifiziert worden.

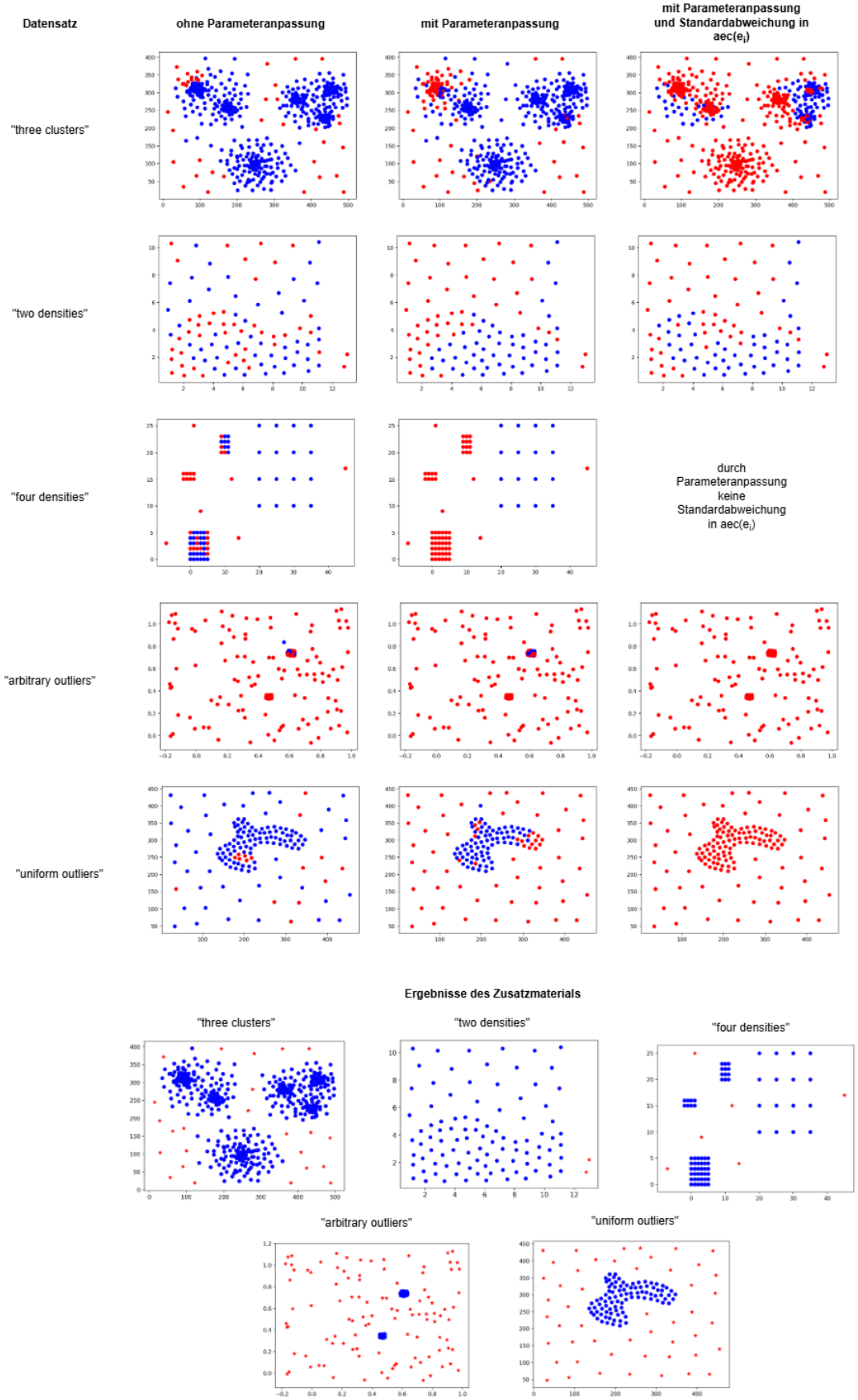
(d) **four densities:** Die Variante ohne Parameteranpassung erkennt die Inlier deutlich besser als die Variante mit Anpassungen. Die Variante ohne Anpassungen erkennt die Cluster jedoch ebenfalls nicht perfekt.

(e) **arbitrary outliers:** Alle Varianten klassifizieren die untere Punktwolke als Outlier. Die Ergebnisse des Algorithmus ohne Parameteranpassungen und mit Parameteranpassungen sind fast equivalent. Die Variante mit Standardabweichung erkennt gar keine Inlier.

(f) **uniform outliers:** Bei der Variante mit Standardabweichung und Parameteranpassungen wurden keine Inlier erkannt. Bei den Varianten anderen Varianten wurden die gesuchten Inlier zum Großteil erkannt, jedoch wurden bei der Variante 'ohne Parameteranpassung' der Großteil der Outlier falsch klassifiziert.

Da für fast alle Datensätze die Variante 'ohne Parameteranpassung' besser oder äquivalent zur Variante 'mit Parameteranpassung' klassifiziert hat, werden die im Artikel vorgeschlagenen Anpassungen nicht weiter verfolgt. Auch wird die Standardabweichung aus dem Originalcode nicht genutzt.

Abbildung 4.1: Datensatzauswertung: Vergleich zum Appendix data_sheet1



Vergleich zwischen den Ergebnisse der in 4.1 beschriebenen Parametervarianten und den im Appendix [8] angegebenen Ergebnissen mittels Prim. Blaue Datenpunkte sind Inlier, rote Datenpunkte sind Outlier.

4.2 Mehrfachkanten und MST-Laufzeiten auf kleinen Datenmengen

Um die Effektivität der Mehrfachkanten zu demonstrieren, wird diese Methode am Beispiel des two_densities-Datensatzes von Li et al. [1] ausgewertet. Der Algorithmus wird ohne die von Li et al. vorgeschlagenen Parameteranpassungen ausgeführt. Zudem wird auf dem Datensatz ein Laufzeitvergleich durchgeführt, um Unterschiede in der Effektivität des Austauschs der MST-Algorithmen zwischen kleinen und großen Datensätzen analysieren zu können. In den Abbildungen 4.2, 4.3 und 4.4 sind die Ergebnisse veranschaulicht. Es geht folgendes hervor:

(a) Die Qualität der Inliererkennung ist bei Dopplung der Kanten deutlich gestiegen.

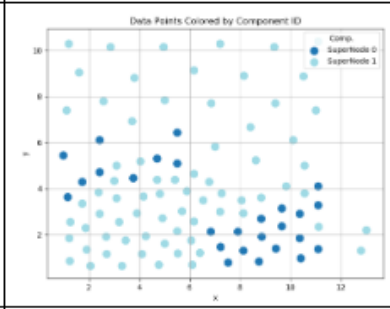
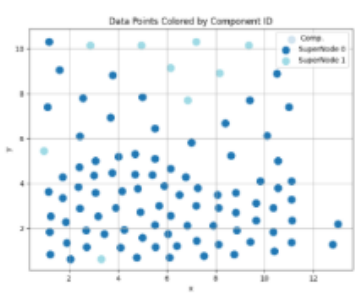
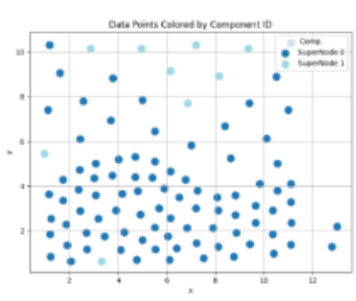
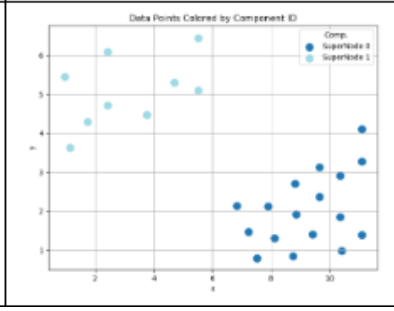
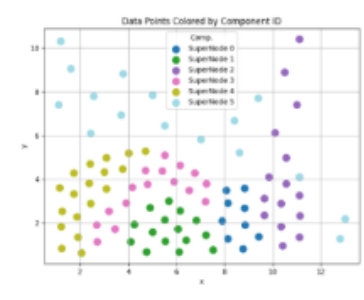
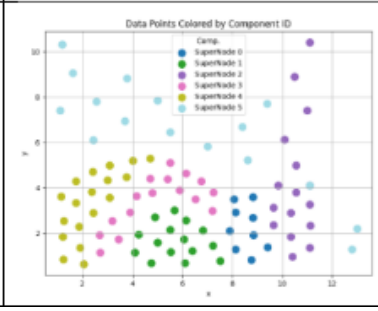
(b) Die Qualität der Klassifizierung steigert sich nach Dopplung der Kanten nicht mehr

(c) Bei der Klassifizierung mit einfachen Kanten liefert die Variante mit Prim ein anderes Ergebnis als die beiden anderen Varianten. Ab der Nutzung doppelter Kanten sind die Klassifizierungen aller 3 Varianten gleich. Es ist zudem erkennbar, dass bei allen Varianten dieselben Minicluster, also Inliercluster, gebildet werden.

(d) Die Laufzeit der Outlierberechnung verbessert sich bei einer Dopplung der Kanten bei allen 3 Algorithmen.

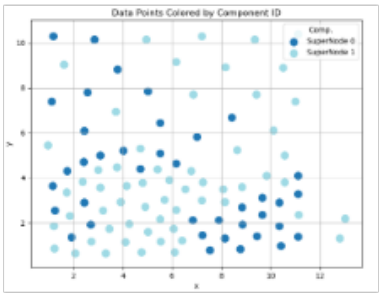
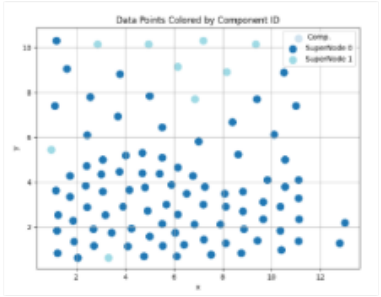
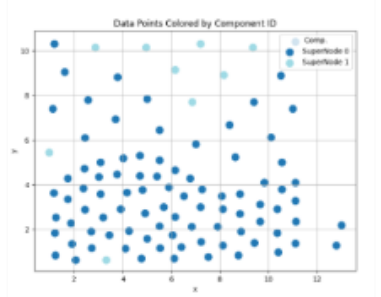
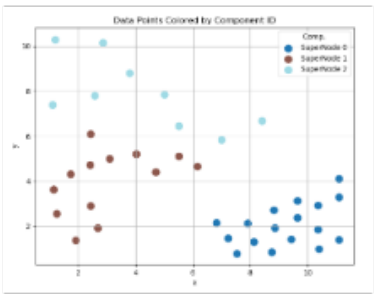
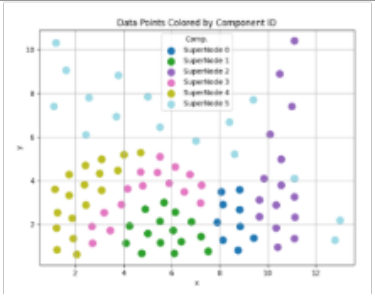
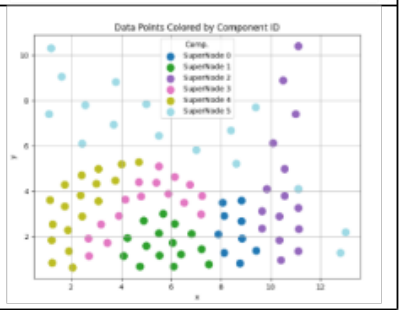
(e) Prim ist der schnellste MST-Algorithmus, gefolgt von KKT. Borůvka ist am langsamsten.

Abbildung 4.2: Parametertuning Mehrfachkanten Primalgorithmus

Outlier detection mittels Prim			
	einfache Kanten	doppelte Kanten	dreifache Kanten
t Adjazenzmatrix	Ø 0.0027070s	Ø 0.0008921s	Ø 0.0017105s
t MST-ber.	Ø 0.032721s	Ø 0.022793s	Ø 0.0234625s
t Outlierber.	Ø 0.011208s	Ø 0.0064773333s	Ø 0.0132425s
n Inliercluster	2	6	6
n Inlier	26	89	89
n Outlier	72	9	9
inlier/outlier visuell			
Inliercluster visuell			

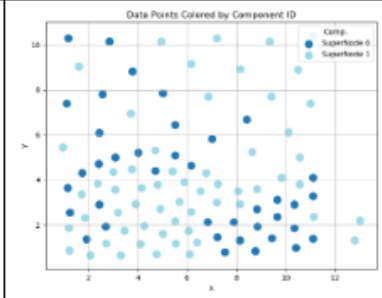
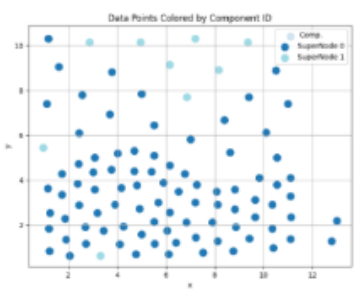
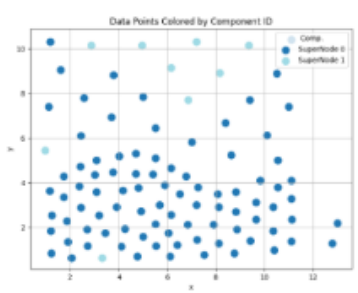
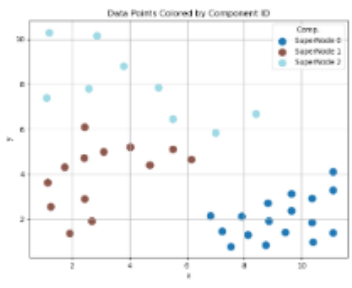
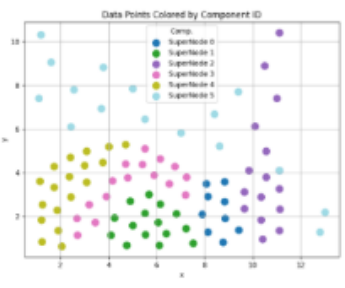
Auswertung der Ergebisse und Laufzeiten der Outlierdetection auf dem two_densities-Datensatz mit Prim als MST-Algorithmus. Es sind die Durchschnittlichen Laufzeiten für die Adjazenzmatrixberechnung, MST-Berechnung und Outlierdetection von Li et al. von je drei Durchgängen angegeben. Zudem ist die Anzahl der Inliercluster, Inlier und Outlier angegeben. In der Inlier/Outlier-Visualisierung sind die Inlier dunkelblau, die Outlier hellblau eingezeichnet.

Abbildung 4.3: Parametertuning Mehrfachkanten Borůvka-Algorithmus

Outlier detection mittels Borůvka			
	einfache Kanten	doppelte Kanten	dreifache Kanten
t Adjazenzmatrix	Ø 0.001464s	Ø 0.0014085s	Ø 0.0018055s
t MST-ber.	Ø 0.068244s	Ø 0.059806s	Ø 0.0614795s
t Outlierber.	Ø 0.010275s	Ø 0.0063355s	Ø 0.006348s
n Inliercluster	3	6	6
n Inlier	39	89	89
n Outlier	59	9	9
inlier/outlier visuell			
Inliercluster visuell			

Auswertung der Ergebnisse und Laufzeiten der Outlierdetection auf dem two_densities-Datensatz mit Borůvka als MST-Algorithmus. Es sind die Durchschnittlichen Laufzeiten für die Adjazenzmatrixberechnung, MST-Berechnung und Outlierdetection von Li et al. von je drei Durchgängen angegeben. Zudem ist die Anzahl der Inliercluster, Inlier und Outlier angegeben. In der Inlier/Outlier-Visualisierung sind die Inlier dunkelblau, die Outlier hellblau eingezeichnet.

Abbildung 4.4: Parametertuning Mehrfachkanten Karger-Klein-Tarjan-Algorithmus

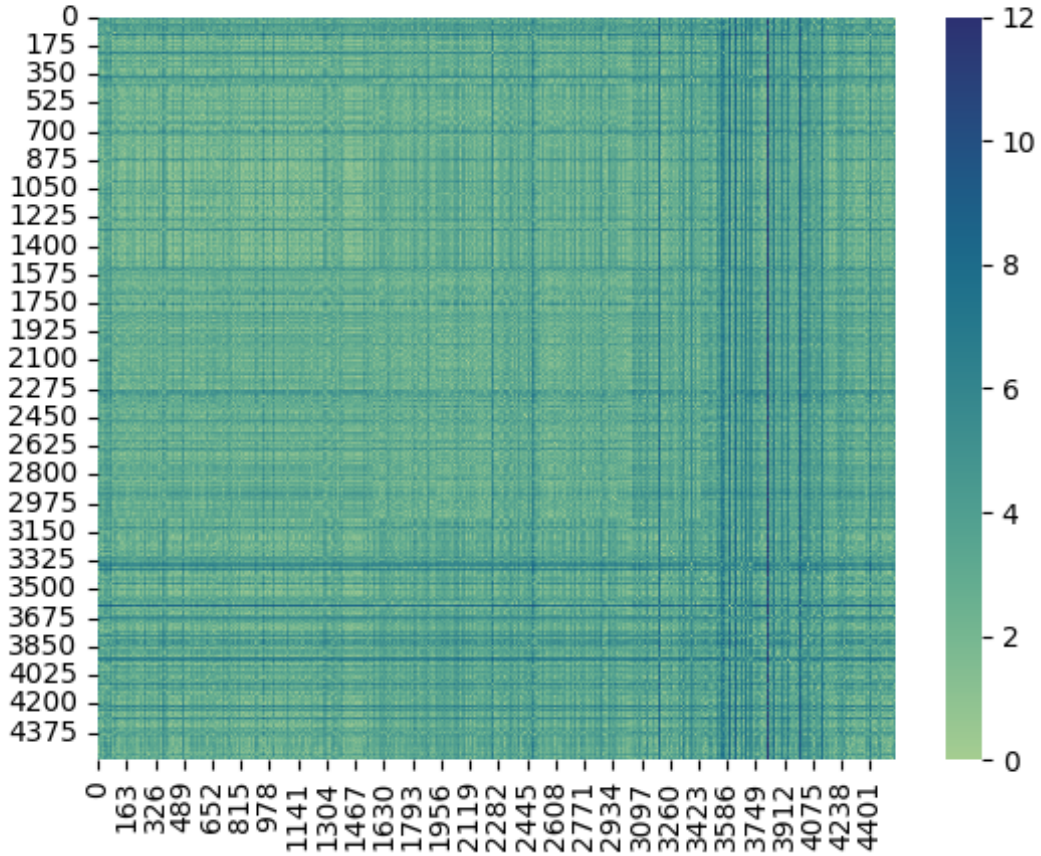
Outlier detection mittels Karger-Klein-Tarjan			
	einfache Kanten	doppelte Kanten	dreifache Kanten
t Adjazenzmatrix	Ø 0.00259625s	Ø 0.00205775s	Ø 0.003232s
t MST-ber.	Ø 0.04477625s	Ø 0.0473505s	Ø 0.047822s
t Outlierber.	Ø 0.0103175s	Ø 0.00692s	Ø 0.006428s
n Inliercluster	3	6	6
n Inlier	39	89	89
n Outlier	59	9	9
inlier/outlier visuell			
Inliercluster visuell			

Auswertung der Ergebisse und Laufzeiten der Outlierdetection auf dem two_densities-Datensatz mit Karger-Klein-Tarjan als MST-Algorithmus. Es sind die Durchschnittlichen Laufzeiten für die Adjazenzmatrixberechnung, MST-Berechnung und Outlierdetection von Li et al. von je drei Durchgängen angegeben. Zudem ist die Anzahl der Inliercluster, Inlier und Outlier angegeben. In der Inlier/Outlier-Visualisierung sind die Inlier dunkelblau, die Outlier hellblau eingezeichnet.

4.3 Ergebnisse des ICD9-Clusterings

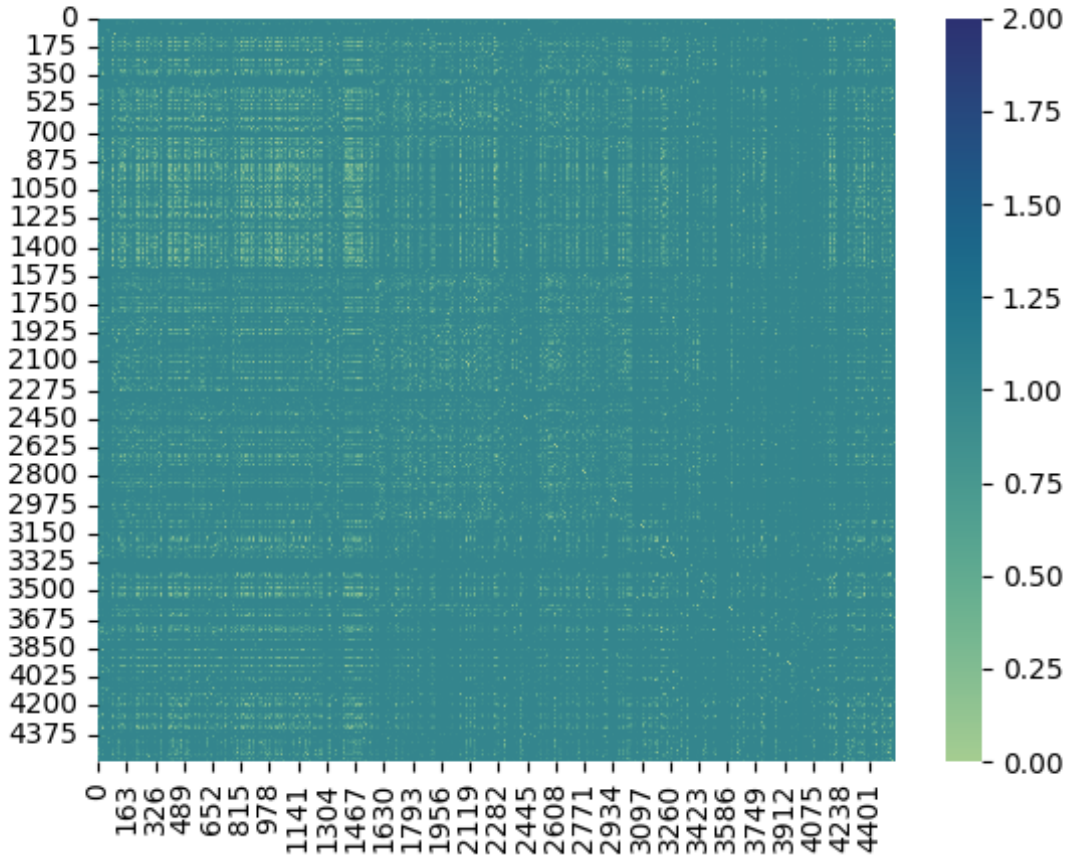
In Abb. 4.5 ist die Heatmap der Adjazenzmatrix des vollständigen Graphen des Datensatzes zum ICD9-Clustering zu sehen (vgl. Abschnitt 3.1.2). Die Daten sind zur Darstellung aufsteigend nach erwartetem Code sortiert. Es sind einige dunkelblaue Linien, also hohe Distanzen, als Abgrenzungen zwischen den hellgrünen (kleinen) Distanzen erkennbar. Die eingeschlossenen hellgrünen Flächen bilden potentielle Cluster. In der analog erstellten Heatmap der Ähnlichkeitsmatrix des vollständigen Graphen (Abb. 4.6) sind diese Flächen verschwommener. Jedoch sind durch die Gewichtung der Kanten bei der Erstellung der Matrix die Flächen besser kontrastiert. Insbesondere lässt sich in der Darstellung der Ähnlichkeitsmatrix eine größere hellgrüne Fläche oben links erkennen, die in der Adjazenzmatrixdarstellung kaum von den anderen hellgrünen Bereichen unterscheidbar ist. Beide Bilder sind recht verrauscht.

Abbildung 4.5: Heatmap Adjazenzmatrix des ICD9-Datensatzes.



Heatmap der Adjazenzmatrix des ICD9-Datensatzes. Die Achsen bezeichnen die Patientendaten-IDs. Die Daten wurden zur Visualisierung aufsteigend nach dem erwartetem ICD9-Code sortiert.

Abbildung 4.6: Heatmap Ähnlichkeitsmatrix des ICD9-Datensatzes.



Heatmap der Ähnlichkeitsmatrix des ICD9-Datensatzes. Die Achsen bezeichnen die Patientendaten-IDs. Die Daten wurden zur Visualisierung aufsteigend nach dem erwartetem ICD9-Code sortiert.

4.3.1 Auswertung des Clusterings ohne Datenrauschreduzierung

In Tabelle 4.1 ist das Ergebnis des Vorclusterings auf dem ICD9-Datensatz, wie es in Abschnitt 3.3 erläutert, wurde abgebildet. Aufgrund der 10 ICD9-Codes im Datensatz, wurde das Vorclustering mit 10 Clustern berechnet. Anhand der Tabelle können folgende Aussagen getroffen werden:

(a) Es sind keine Unterschiede zwischen dem Vorclustering mit Adjazenzmatrix oder mit Ähnlichkeitsmatrix erkennbar.

(b) In den Varianten mit Prim-MST sind ist allen Clustern außer Cluster 0 nur ein Datenpunkt. Alle anderen Punkte des Datensatzes sind in Cluster 0.

(c) In den Varianten mit Borůvka-MST sind die Datenpunkte etwas mehr verteilt als bei Prim. Es befinden sich dennoch fast alle Datenpunkte im Cluster 0.

(d) Die perfekte Abdeckung im Cluster 0 für alle Varianten ist erwartbar, da fast alle Datenpunkte in diesem Cluster liegen und die Werte der Tabelle gerundet sind.

(e) Die Genauigkeit des Clusters 0 ist in allen Versuchen von 66%. Dies hängt damit zusammen, dass der dominierende ICD9-Code 41401, für den der Wert berechnet wird, ungefähr diesen Anteil im Gesamtdatensatz ausmacht (Vgl. Tabelle 3.6) und keine gute Clusteraufteilung stattfand (Vgl. (a), (b)).

Tabelle 4.1: Clusterlabel ohne Datenrauschreduzierung

Kategorie \Label	0	1	2	3	4	5	6	7	8	9
V_ICD:PS										
Datenpunkte	4534	1	1	1	1	1	1	1	1	1
dom. Code	41401	41401	41401	51881	51881	51881	51881	4280	4280	5990
Abdeckung	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Genauigkeit	0.66	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
V_ICD:PA										
Datenpunkte	4534	1	1	1	1	1	1	1	1	1
dom. Code	41401	41401	41401	51881	51881	51881	51881	4280	4280	5990
Abdeckung	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Genauigkeit	0.66	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
V_ICD:BS										
Datenpunkte	4511	15	2	5	2	3	2	1	1	1
dom. Code	41401	41401	41401	51881	42731	51881	51881	53081	4019	25000
Abdeckung	1.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04	0.02	0.05
Genauigkeit	0.66	0.60	0.50	0.60	0.50	0.33	0.50	1.00	1.00	1.00
V_ICD:BA										
Datenpunkte	4511	15	2	5	2	3	2	1	1	1
dom. Code	41401	41401	41401	51881	42731	51881	51881	53081	4019	25000
Abdeckung	1.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04	0.02	0.05
Genauigkeit	0.66	0.60	0.50	0.60	0.50	0.33	0.50	1.00	1.00	1.00

*Die Tabelle beinhaltet die die Auswertung der automatisch gelabelten Cluster pro MST-Algorithmus und Ähnlichkeitsmaß für den ICD9-Datensatz. Für jede Kombination ist die Anzahl der Datenpunkte im Cluster und der das Cluster ICD9-Code angegeben. Zusätzlich ist die Abdeckung des dominierenden Codes durch das Clusters gegeben. Dies geschieht in Relation zu der Gesamtanzahl des dominierenden Code im Datensatz. Die Genauigkeit gibt an, welcher Anteil der Datenpunkte im Cluster dem dominierenden Code entspricht. Die Kombination aus Ähnlichkeitsmaß und MST-Algorithmus ist zwecks der eindeutigen Referenzierung als Code angegeben: V_<Datensatz>:<Algorithmus><Ähnlichkeitsmaß>, genauer V_<**ICD**9-Code><**B**orůvka\|**P**rim><**S**imilarity (Ähnlichkeitsmatrix)\|**A**djazenzmatrix>*

4.3.2 Laufzeiten der Algorithmusvarianten

Für den ICD9-Clusteringdatensatz werden in Tabelle 4.2 zwölf Algorithmusvarianten verglichen. Diese sind die Kombinationen aus den MST-Algorithmen (Prim, Borůvka und Karger-Klein-Tarjan),

ob die MST-Kanten doppelt oder einfach in das Clustering gegeben wurden sowie ob eine Adjazenzmatrix mit euklidischer Distanz oder eine Ähnlichkeitsmatrix (s. Abschnitt 3.1.2 für die MST-Berechnung verwendet wurde. Die vorgeschlagenen Parameteranpassungen aus dem Artikel von Li et al. wurden aufgrund ihrer Ineffektivität nicht weiter verfolgt (s. Abschnitt 4.2).
Betrachtet wird jeweils die Anzahl der Inlier und Outlier, sowie die Laufzeit der 3 Algorithmusphasen: Matrixerstellung (= Grapherstellung), MST-Berechnung und Outlierberechnung. Die Zeit für das Erstellen des Vorclusterings und das anschließende Mappen der Minicluster auf die entstandenen ZHK können vernachlässigt werden, da diese durch reguläre Ausdrücke auf den existierenden Outputs in Millisekunden geschehen. Insbesondere wird der MST für die Outlierbestimmung für das Vorclustering wiederverwendet. Es können folgende Beobachtungen gemacht werden:

(a) Varianten mit einer Ähnlichkeitsmatrix haben mehr Outlier als Varianten mit einer Adjazenzmatrix.

(b) Varianten mit doppelten Kanten führen im Vergleich zu den äquivalenten Varianten mit einfachen Kanten in vier von sechs Fällen zu deutlich weniger Outliern. Ausnahmen: V_ICD:BDS - V_ICD:BES und V_ICD:KDS - V_ICD:KES. Hier sind die Mengen an Outliern ähnlich, mit etwas mehr Outlier bei der Variante mir den doppelten Kanten.

(c) Die Berechnung der Ähnlichkeitsmatrix braucht konsistent im Millisekundenbereich länger als die Berechnung der Adjazenzmatrix, welche ebenfalls im Millisekundenbereich liegt.

(d) Der schnellste MST-Algorithmus auf diesem Datensatz ist Borůvka (0.03h ≈ 1.58 min), gefolgt von Prim (0.10h bis 0.14h ≈ 6 bis 8min. Karger-Klein-Tarjan benötigt mit sehr starkem Abstand ein bis drei Tage.

(e) Die MST-Berechnung hat einen signifikanten Einfluss auf die Gesamtlaufzeit und ist stets die längste Phase.

(f) Die Matrixberechnung hat keinen relevanten Einfluss auf die Gesamtlaufzeit und benötigt nur wenige Millisekunden.

(g) Das Berechnen der Outlier der Varianten mit Ähnlichkeitsmatrix benötigt mehr Zeit als bei den Varianten mit Adjazenzmatrix obwohl diese weniger Inlier und damit weniger Kanten im MST haben (Vgl. (a)).

(h) Die Verwendung doppelter Kanten oder einfacher Kanten hat keinen signifikanten Einfluss auf die Dauer der Outlierberechnung.

Tabelle 4.2: Variantenvergleich auf dem ICD9-Datensatz

Code Variante	Algorithmus	Doppelkanten?	Ähnlichkeitsmatrix?	Inlier	Outlier	t Matrix	t MST	t Outlier
V_ICD:PDS	Prim	Ja	Ja	2113	2430	0.531 s	367.634 s / 0.10 h	82.089 s / 0.02 h
V_ICD:PDA	Prim	Ja	Nein	4410	133	0.253 s	475.554 s/ 0.13 h	30.52 s / 0.01 h
V_ICD:PES	Prim	Nein	Ja	2088	2455	0.545 s	370.203 s / 0.10 h	83.036 s / 0.02 h
V_ICD:PEA	Prim	Nein	Nein	3872	671	0.252 s	496.007 s / 0.14 h	29.878 s / 0.01 h
V_ICD:BDS	Borůvka	Ja	Ja	1864	2679	0.533 s	95.813 s / 0.03 h	93.069 s / 0.03 h
V_ICD:BDA	Borůvka	Ja	Nein	4058	158	0.332 s	95.411 s / 0.03 h	21.896 s / 0.01 h
V_ICD:BES	Borůvka	Nein	Ja	2029	2514	0.582 s	95.896 s / 0.03 h	86.721 s / 0.02 h
V_ICD:BEA	Borůvka	Nein	Nein	3891	652	0.268 s	95.951 s / 0.03 h	21.437 s / 0.01 h
V_ICD:KDS	KKT	Ja	Ja	1799	2744	0.532 s	77.25 h	216.568 s / 0.06 h
V_ICD:KDA	KKT	Ja	Nein	4359	184	0.255 s	33.17 h	34.399 s / 0.01 h
V_ICD:KES	KKT	Nein	Ja	1978	2565	0.532 s	77.25 h	216.568 s / 0.06 h
V_ICD:KEA	KKT	Nein	Nein	3836	707	0.284 s	31.39 h	35.422 s / 0.01 h

*Vergleich der Laufzeiten und der Anzahl der Inlier und Outlier mit Standard-T_t, MEW und aec(e_i) (s. Abschnitt 3.4.3) in Abhängigkeit von MST-Algorithmus, Ähnlichkeits- oder Adjazenzmatrix, sowie der Nutzung von Doppelkanten. Die Codes der Varianten dienen der eindeutigen Referenzierung: V_<**D**eath (Verstorbenendatensatz)|**ICD**9-Datensatz>:<**P**rim|**B**orůvka|**K**KT><**D**oppelt|**E**infach><**S**imilarity (Ähnlichkeitsmatrix)|**A**djazenzmatrix>*

4.3.3 Betrachtung der Minicluster

Tabelle 4.3 zeigt die, wie in Abschnitt 3.4.4 beschrieben, zusammengefassten Minicluster pro Algorithmusvariante. Die Umsetzung der Auswertung der Minicluster ist in den Dateien `evaluate_cluster.py` und `group_cluster.py` des Codes [13].
Anschließend werden die Minicluster je nach Datensatz/Clusteringziel zusammengelegt. In diesem Fall wurde für jedes Minicluster im Zuge der Auswertung der dominierende ICD9-Code bestimmt. Mit diesem wird das Minicluster gelabelt. Minicluster mit demselben Label werden zu einem großen Cluster zusammengefasst, dabei werden 2 Metriken verwendet. Zum einen die Genauigkeit, hinsichtlich des festgestellten Labels.
Die Genauigkeit wurde als $Genauigkeit_{\text{cluster}} = \frac{n_{\text{korrekt}}}{n_{\text{cluster}}}$ für die Cluster pro Variante berechnet und für die gesamte Codevariante als $Genauigkeit_{\text{Variante}} = \frac{n_{\text{korrekte label über alle Cluster}}}{n_{\text{Punkte aller Cluster}}}$.
Zum anderen die Abdeckung (hier %Patienten). Diese wird als $\%Patienten_{\text{cluster}} = \frac{n_{\text{getroffen}}}{n_{\text{cluster}}}$ berechnet.

Beobachtungen der Clusterauswertungen:

(a) Der ICD9-Code 41401 (Koronararteriosklerose, arterielle Verschlusskrankheit des Herzens, s. Tabelle 3.3) ist bei allen Varianten das größte Cluster und hat konstant hohe Genauigkeit-Werte ($0.672 - 0.776$). Die Abdeckung ist entweder sehr hoch ($80 - 97$), oder im mittleren Bereich ($42 - 47$).

(b) Der ICD9-Code 51881 (Lungenkollaps) ist bei allen Varianten das zweitgrößte Cluster, weist allerdings mit Werten zwischen 0.436 und 0.562 eine geringere Genauigkeit auf als das größte Cluster. Auch die Abdeckung ist deutlich geringer (< 30).

(c) Die Cluster mit den Labels 4280 (Stauungsinsuffizienz des Herzens) und 5849 (akutes Nierenversagen) treten nur vereinzelt und mit geringer Genauigkeit auf (zwischen 0.25 und 0.333) und sehr geringer Abdeckung (< 10).

(d) Der meist vergebene ICD9-Code 4019 (s. Tabelle 3.3) tritt nicht als Cluster auf. Der Code des größten Clusters 41401 ist der viert-häufigst vergebene Diagnosecode. Der Code des zweitgrößten Clusters 51881 ist die acht-häufigste Diagnose.

(e) Die beste Gesamtgenauigkeit hat Variante V_ICD:KDS (KKT-Algorithmus, doppelte Kanten, Ähnlichkeitsmatrix) mit einem Wert von 0.76. Die zweitbeste Variante ist V_ICD:BES (Borůvka-Algorithmus, einfache Kanten, Ähnlichkeitsmatrix; Genauigkeit=0.743), dicht gefolgt von den drittbesten Varianten mit einer Genauigkeit von 0.74. Diese sind V_ICD:BDS (Borůvka-Algorithmus, doppelte Kanten, Ähnlichkeitsmatrix) und V_ICD:KES (KKT-Algorithmus, einfache Kanten, Ähnlichkeitsmatrix).

(f) Die schlechteste Gesamtgenauigkeit von 0.653 hat V_ICD:PDS (Primalgorithmus, doppelte Kanten, Ähnlichkeitsmatrix). Die zweitschlechteste Variante ist V_ICD:PES (Primalgorithmus, einfache Kanten, Ähnlichkeitsmatrix; Genauigkeit = 0.655).

(g) Im Vergleich der Genauigkeit der Varianten bei gleichem Algorithmus und gleicher Matrix schneiden diejenigen mit doppelten Kanten in vier von sechs Fällen besser ab.

(h) Im Vergleich der Varianten bei gleichem Algorithmus und Kantenzahl haben in vier von sechs Fällen die Varianten mit Ähnlichkeitsmatrix eine höhere Genauigkeit.

(i) Die Wahl des MST-Algorithmus hatte einen Einfluss auf die Clusterqualität. Im Durchschnitt haben die Varianten mit KKT als MST-Algorithmus die höchste Genauigkeit (0.715), gefolgt von Borůvka (0.708). Am schlechtesten schnitt Prim mit einer durchschnittlichen Genauigkeit von 0.664 ab.

(j) Die Dopplung der Kanten hat keinen nennenswerten Einfluss auf die Abdeckung.

(k) Die Abdeckung ist bei jedem der Algorithmen ungefähr gleich.

(l) Beim das Verwenden der Adjazenzmatrix ist die Abdeckung, unabhängig vom Algorithmus um 95 ± 1. Im Gegensatz dazu wird die Abdeckung durch das Verwenden der Ähnlichkeitsmatrix in etwa halbiert.

(m) Es gibt einen positiven Zusammenhang zwischen der Anzahl der Inlier und der Anzahl der produzierten Minicluster. Es ist jedoch kein Zusammenhang zwischen der Anzahl der Inlier der Varianten (s. Tabelle 4.2) und ihrer Clusterqualität erkennbar. Beispielsweise produzieren sowohl die schlechteste Variante V_ICD:PDS und die beste Variante V_ICD:KDS viele Outlier um haben als Ergebnis davon vergleichsweise wenig Minicluster die zusammengefasst wurden (21 und 16).

Tabelle 4.3: Gruppierte Cluster der Varianten für das ICD9-Clustering

dom. Code	zusammengefasste Cluster	Datenpunkte dominant	andere Datenpunkte	Genauigkeit	% Patienten
V_ICD:PDS				**0.653**	
41401	17	1279	625	0.672	42.3
51881	2	86	67	0.562	16.4
4280	2	15	41	0.268	4.8
V_ICD:PDA				**0.684**	
41401	27	2837	1153	0.711	95.4
51881	6	173	223	0.437	33.1
5849	1	8	16	0.333	2.6
V_ICD:PES				**0.655**	
41401	16	1264	582	0.685	42.5
51881	2	82	77	0.516	15.7
5849	1	15	40	0.272	4.8
4280	1	7	21	0.250	2.3
V_ICD:PEA				**0.665**	
41401	33	2424	1115	0.685	81.5
51881	6	149	184	0.448	47.9
V_ICD:BDS				**0.740**	
41401	13	1282	375	0.774	43.1
51881	3	97	110	0.469	18.6
V_ICD:BDA				**0.687**	
41401	27	2884	1223	0.702	96.9
51881	4	121	131	0.480	23.1
5849	1	7	19	0.269	2.3
V_ICD:BES				**0.743**	
41401	13	1389	400	0.776	46.7
51881	4	119	121	0.496	22.8
V_ICD:BEA				**0.663**	
41401	43	2459	1146	0.682	83.7
51881	4	114	146	0.438	21.8
4280	1	8	18	0.308	2.6
V_ICD:KDS				**0.760**	
41401	14	1304	377	0.77	43.8
51881	2	61	54	0.530	11.4
V_ICD:KDA				**0.688**	
41401	27	2879	1192	0.707	96.8
51881	3	113	147	0.435	21.6
4280	1	8	20	0.286	2.6
V_ICD:KES				**0.740**	
41401	15	1401	435	0.763	47.1
51881	3	72	83	0.465	13.8
V_ICD:KEA				**0.671**	
41401	34	2411	1037	0.699	81.0
51881	7	149	193	0.436	28.5

Fortsetzung auf der nächsten Seite

Fortsetzung der Tabelle 4.3 — Gruppierte Cluster für ICD9 der Codevarianten					
dom. Code	zusammengefasste Cluster	Datenpunkte dominant	andere Datenpunkte	Genauigkeit	% Patienten
5849	1	13	33	0.28	4.1

Zusammengefasste Minicluster für die in Tabelle 4.2 beschriebenen Algorithmusvarianten mit ihrer Genauigkeit. Die Cluster wurden mit dem häufigsten ICD-Code im Cluster gelabelt und Cluster mit demselben Label zusammengefasst. Die Spalte 'Datenpunkte dominant' gibt an, wie viele Punkte in dem zusammengefassten Cluster dem Label entsprechen, die Spalte 'andere Datenpunkte' gibt an, wie viele Datenpunkte im gruppierten Cluster nicht zu dem Label passen. Die Spalte '% Patienten' gibt an, wie viel Prozent der Patienten mit dem jeweiligen ICD9-Code gefunden wurden.

4.3.4 Ergebnisse des Clusterings mit Datenrauschreduzierung

Wie in Abschnitt 4.2 erläutert, ist die MST-Bestimmung mit KKT aufgrund der hohen Laufzeit keine in der Praxis nutzbare Algorithmusvariante. Daher beschränkt sich die nachfolgende Auswertung auf die Algorithmusvarianten mit Prim und Borůvka als MST-Algorithmus.
In Tabelle 4.4 ist das Ergebnis der Datenrauschreduzierung, wie sie in Abschnitt 3.4.4 beschrieben ist, ausgewertet. Erkenntnisse der Tabelle:

(a) Die Varianten mit Ähnlichkeitsmatrix haben ca. 2000 Datenpunkte (= Inlier) weniger als ihre äquivalenten Varianten mit Adjazenzmatrix.

(b) Die Varianten mit Ähnlichkeitsmatrix haben eine um ca. 50% reduzierte Abdeckung im Vergleich zu ihren äquivalenten Varianten mit Adjazenzmatrix. Die Genauigkeit des Clusters mit Ähnlichkeitsmatrix ist ähnlich oder besser als die Genauigkeit der äquivalenten Variante mit Adjazenzmatrix.

(c) Alle Varianten liefern genau ein Cluster (mit Label 0 == Datensatzlabel 41401).

(d) Das Wert 'andere ZHK' ist bei allen Varianten sehr niedrig. Dies bedeutet, dass sich nur wenige Miniclusters über die Grenzen der im Vorclustering berechneten Zusammenhangskomponenten erstreckten.

(e) Den besten Abdeckungswert hat die Algorithmusvariante V_ICD:BDS mit 71%. Am zweitbesten war V_ICD:BES mit einer ein Prozent niedrigeren Genauigkeit. Die drittbeste Variante war ebenfalls mittels Borůvka-MST V_ICD:BDA mit 67%.

(f) Die beste Genauigkeit haben die Varianten V_ICD:PDA und V_ICD:BDA mit fast perfekten 98%. Am zweitbesten war V_ICD:BEA mit einer 14% niedrigeren Genauigkeit.

(g) Die Varianten mit Prim und Ähnlichkeitsmatrix haben die schlechtesten Abdeckungs- und Genauigkeitswerte der getesteten Varianten.

(h) Das Verwenden der doppelten Kanten führt in allen Fällen zu einem gleichen oder besseren Ergebnis, als mit einfachen Kanten.

Im Vergleich mit der Auswertung der Daten ohne Datenrauschreduzierung (s. Abschnitt 4.3.1) ist Folgendes festzustellen:

(i) Alle rauschbereinigten Varianten haben im Gegensatz zum Vorclustering nur ein Cluster.

(ii) Die Varianten V_ICD:PDS und V_ICD:PES weisen die die Hälfte der Datenpunkte und Genauigkeit im Vergleich zu ihrem Vorclustering V_ICD:PS auf.

(iii) V_ICD:PDA hat, wie in (f) aufgeführt, eine ebenfalls fast perfekte Abdeckung bei gleichbleibender Genauigkeit im Vergleich zum Vorclustering V_ICD:PA. V_ICD:PEA hat eine schlechtere Abdeckung aber ähnliche Genauigkeit wie V_ICD:PA.

(iv) V_ICD:BDS und V_ICD:BES haben eine schlechtere Abdeckung als das Vorclustering V_ICD:BS aber eine 4-5% bessere Genauigkeit.

(v) V_ICD:BDA hat eine fast perfekte Abdeckung und 1% bessere Genauigkeit als V_ICD:BA. V_ICD:BEA hat eine 16% schlechtere Abdeckung bei einer nur wenig schlechteren Genauigkeit als V_ICD:BA.

Tabelle 4.4: Gruppierte rauschreduzierte Cluster von Varianten des ICD9-Clusterings

Label	n Cluster	korrekte ZHK	andere ZHK	Datenlabel	Abdeckung	Genauigkeit
V_ICD:PDS						
0	21	2113	0	41401	0.44	0.62
V_ICD:PDA						
0	34	4410	0	41401	0.98	0.66
V_ICD:PES						
0	20	2088	0	41401	0.44	0.62
V_ICD:PEA						
0	39	3868	4	41401	0.83	0.64
V_ICD:BDS						
0	16	1852	12	41401	0.44	0.71
V_ICD:BDA						
0	32	4355	30	41401	0.98	0.67
V_ICD:BES						
0	17	2015	14	41401	0.48	0.70
V_ICD:BEA						
0	48	3863	28	41401	0.84	0.64

Fortsetzung auf der nächsten Seite

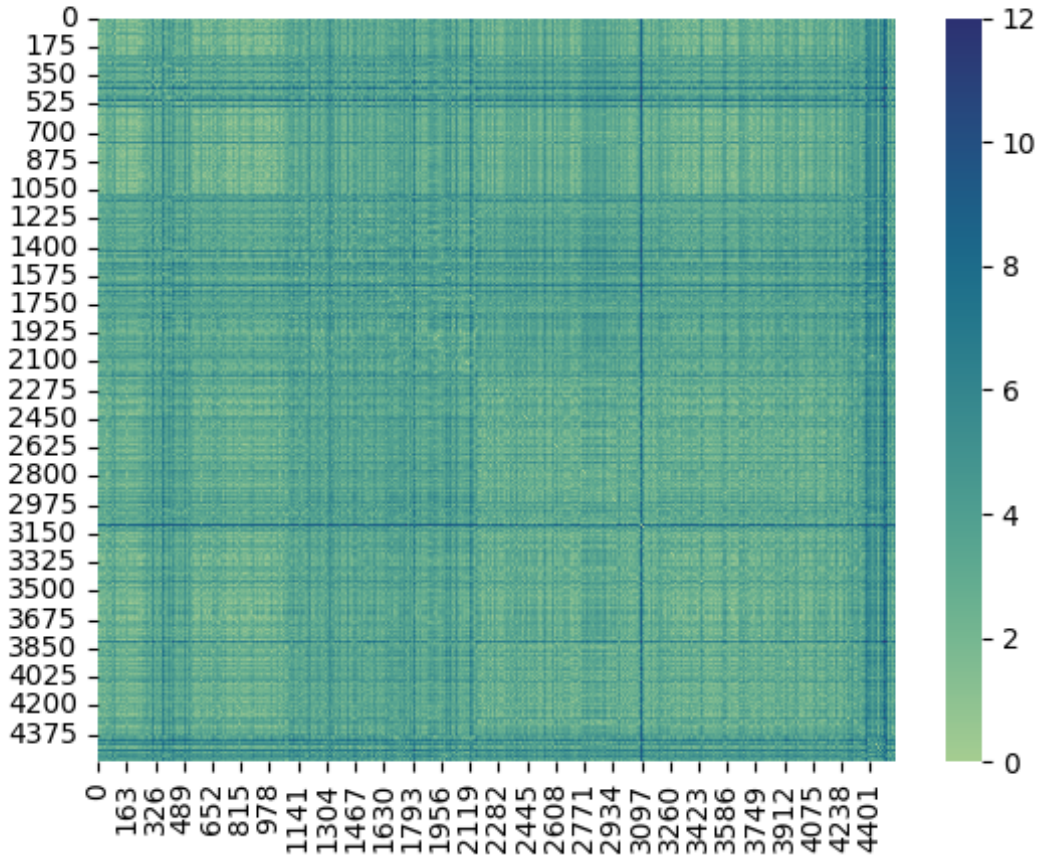
Fortsetzung der Tabelle 4.4 — Gruppierte rauschreduzierte Cluster von Varianten des ICD9-Clusterings						
Label	n Cluster	korrekte ZHK	andere ZHK	Datenlabel	Abdeckung	Genauigkeit

Zusammengefasste Minicluster für die in Tabelle 4.2 beschriebenen Algorithmusvarianten mit Prim und Borůvka. Die Cluster wurden mit der häufigsten Komponentenzugehörigkeit der Datenpunkte im Cluster gelabelt. Die Komponenten wurden im Vorclustering bestimmt. Cluster mit demselben Label wurden anschließend zusammengefasst. Die Anzahl der zusammengefassten Cluster steht in der Spalte 'n Cluster'. Spalte 'korrekte ZHK' gibt an, wie viele Punkte des zusammengefassten Clusters in der dem Label entsprechenden Zusammenhangskomponente liegen. Die Spalte 'andere ZHK' gibt an, wie viele Datenpunkte im gruppierten Cluster nicht zu dem Label passen. 'Datenlabel' beinhaltet die Auswertung, welcher gesuchte Wert in den Datenpunkten des Clusters überwiegt. Die Abdeckung gibt an, welcher Anteil der Gesamtzahl des Datenlabels im Datensatz durch das Cluster abgedeckt wird. Die Genauigkeit gibt den Anteil der Datenpunkte im Cluster an, die dem dominierenden Code entsprechen.

4.4 Ergebnisse des Verstorbenen-Clusterings

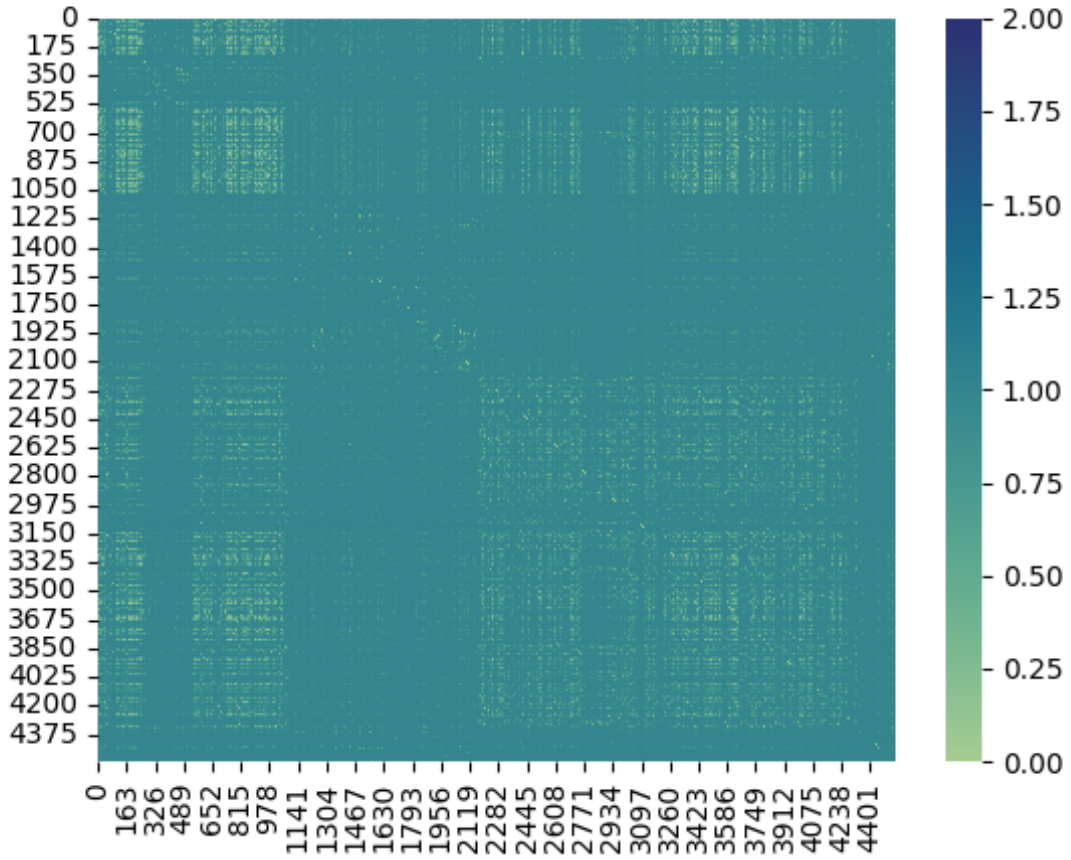
In Abb. 4.7 ist die Heatmap der Adjazenzmatrix des vollständigen Graphen des Datensatzes zum verstorben/nicht verstorben-Clustering abgebildet (Vgl. Abschnitt 3.1.2). Die Daten sind zur Darstellung nach verstorben/nicht verstorben sortiert (wobei die niedrigen Patientendaten-ID's nicht verstorben bedeuten). Die eingeschlossenen hellgrünen Flächen bilden potentielle Cluster. Die Grenzen zwischen den hellgrünen Flächen sind recht verschwommen, jedoch sind ein paar größere hellgrüne Bereiche (Cluster) erkennbar. Die Heatmap der Ähnlichkeitsmatrix des vollständigen Graphen (Abb. 4.8) wurde analog erstellt. In ihr sind viele Kanten mit hohem Gewicht (dunkelgrüne Linien) zu sehen. In dieser Darstellung werden größere hellgrüne Bereiche in den Ecken erkennbar, die von dunklen Linien durchzogen sind. Wie auch die Daten für das ICD9-Clustering sind die Verstorbenendaten eher verschwommen und von Rauschen durchzogen.

Abbildung 4.7: Heatmap Adjazenzmatrix des Verstorbenen-Datensatzes.



Heatmap der Adjazenzmatrix des Verstorbenen-Datensatzes. Die Achsen bezeichnen die Patientendaten-IDs. Die Daten sind zur Darstellung nach verstorben/nicht verstorben sortiert (wobei die niedrigen Patientendaten-ID's nicht verstorben bedeuten).

Abbildung 4.8: Heatmap Ähnlichkeitsmatrix des Verstorbenen-Datensatzes.



Heatmap der Ähnlichkeitsmatrix des Verstorbenen-Datensatzes. Die Achsen bezeichnen die Patientendaten-IDs.Die Daten sind zur Darstellung nach verstorben/nicht verstorben sortiert (wobei die niedrigen Patientendaten-ID's nicht verstorben bedeuten).

4.4.1 Auswertung des Clusterings ohne Datenrauschreduzierung

In Tabelle 4.5 ist das Ergebnis des Vorclusterings auf dem Verstorbenendatensatz, wie es in Abschnitt 3.3 erläutert wurde, eingetragen. Aufgrund der binären Ziellabel, wurde das Vorclustering mit 2 Zielclustern berechnet. Anhand der Tabelle kann ausgesagt werden:

(a) Es sind keine Unterschiede in den Ergebnissen der Varianten bzgl. Genauigkeit, Abdeckung und dominierendem Lebendigkeitsstatus erkennbar.

(b) Für jede Variante liegen fast alle Punkte liegen im Cluster 0.

(c) Abdeckung und Genauigkeit sind zwar sehr hoch, jedoch nicht aussagekräftig, da keine wirkliche Aufteilung der Daten zwischen den Clustern stattfand (Vgl. (b)). Die Genauigkeit entspricht

daher dem Anteil der Lebendigen im Datensatz (Vgl. Tabelle 3.7).

(d) Die Genauigkeit von 100% des Clusters mit dem Label 1 in jedem Datensatz ist bei nur einem Datenpunkt selbstverständlich.

Tabelle 4.5: Clusterlabel ohne Datenrauschreduzierung

Kategorie \Label	0	1
V_D:BA		
Datenpunkte	4541	2
dom. Code	lebend	verstorben
Abdeckung	1.00	0.00
Genauigkeit	0.95	1.00
V_D:BS		
Datenpunkte	4541	2
dom. Code	lebend	verstorben
Abdeckung	1.00	0.00
Genauigkeit	0.95	1.00
V_D:PA		
Datenpunkte	4542	1
dom. Code	lebend	verstorben
Abdeckung	1.00	0.00
Genauigkeit	0.95	1.00
V_D:PS		
Datenpunkte	4542	1
dom. Code	lebend	verstorben
Abdeckung	1.00	0.00
Genauigkeit	0.95	1.00

*Die Tabelle beinhaltet die die Auswertung der automatisch gelabelten Cluster pro MST-Algorithmus und Ähnlichkeitsmaß für den Verstorbenendatensatz. Für jede Kombination ist die Anzahl der Datenpunkte im Cluster und der das Cluster ICD9-Code angegeben. Zusätzlich ist die Abdeckung des dominierenden Codes durch das Clusters gegeben. Dies geschieht in Relation zu der Gesamtanzahl des dominierenden Code im Datensatz. Die Genauigkeit gibt an, welcher Anteil der Datenpunkte im Cluster dem dominierenden Code entspricht. Die Kombination aus Ähnlichkeitsmaß und MST-Algorithmus ist zwecks der eindeutigen Referenzierung als Code angegeben: V_<Datensatz>:<Algorithmus><Ähnlichkeitsmaß>, genauer V_<**D**eath (Verstorben)><**B**orůvka\|**P**rim><**S**imilarity (Ähnlichkeitsmatrix)\|**A**djazenzmatrix>*

4.4.2 Laufzeiten der Algorithmusvarianten

Tabelle 4.6 beinhaltet ebenfalls die Laufzeiten der 12 Algorithmusvarianten, wie sie im vorherigen Abschnitt (4.3) beschrieben wurden, in den drei Phasen des Algorithmus: Matrixerstellung (Grapherstellung), MST-Erzeugung und Outlierberechnung. Es ergeben sich folgende Beobachtungen:

(a) Die MST-Erstellung hat einen signifikanten Einfluss auf die Gesamtlaufzeit. Mit Ausnahme der Variante V_D:BES ist sie die längste Phase des Algorithmus.

(b) Borůvka ist mit ca. $2.5 - 3$ Minuten der schnellste MST-Algorithmus. Prim benötigt ca. 10 Minuten. KKT ist mit sehr starkem Abstand am langsamsten und braucht 23.19 bis 32.79 Stunden für die Berechnung.

(c) Die Matrixerstellung benötigt nur Millisekunden und hat keinen relevanten Einfluss auf die Gesamtlaufzeit. Es ist erkennbar, dass die Berechnung der Ähnlichkeitsmatrix länger dauert als die Berechnung der Adjazenzmatrix.

(d) Die Berechnung der Oultier der Varianten mit Adjazenzmatrix ist ca. eine Minute schneller als die äquivalenten Varianten mir Ähnlichkeitsmatrix.

(e) Es ist kein Trend erkennbar, ob Doppelkanten einen Einfluss auf die Dauer der Outlierberechnung haben. Die Laufzeitunterschiede zwischen den äquivalenten Varianten mit oder ohne Doppelkanten unterscheidet sich um wenige Sekunden.

Tabelle 4.6: Variantenvergleich auf dem Verstorbenen-Datensatz

Code Variante	Algorithmus	Doppelkanten?	Ähnlichkeitsmatrix?	t Matrix	t MST	t Outlier
V_D:PDS	Prim	Ja	Ja	0.560 s	588.001 s / 0.16 h	159.983 s/ 0.04 h
V_D:PDA	Prim	Ja	Nein	0.272 s	604.165 s / 0.17 h	69.641 s / 0.02 h
V_D:PES	Prim	Nein	Ja	0.566 s	593.276 s / 0.16 h	160.595 s / 0.04 h
V_D:PEA	Prim	Nein	Nein	0.280 s	586.898 s / 0.16 h	65.750 s / 0.02 h
V_D:BDS	Borůvka	Ja	Ja	0.631 s	160.940 s / 0.04 h	142.772 s / 0.04 h
V_D:BDA	Borůvka	Ja	Nein	0.274 s	171.491 s / 0.05 h	70.220 s / 0.02 h
V_D:BES	Borůvka	Nein	Ja	0.565 s	168.085 s / 0.05 h	174.269 s / 0.05 h
V_D:BEA	Borůvka	Nein	Nein	0.267 s	175.160 s / 0.05 h	66.738 s / 0.02 h
V_D:KDS	KKT	Ja	Ja	0.619 s	23.19 h	158.764 s / 0.04 h
V_D:KDA	KKT	Ja	Nein	0.399 s	32.79 h	55.493 s / 0.02 h
V_D:KES	KKT	Nein	Ja	0.617 s	32.97 h	130.804 s / 0.04 h
V_D:KEA	KKT	Nein	Nein	0.334 s	30.59 h	60.277 s / 0.02 h

*Vergleich der Laufzeiten mit Standard-T_t, MEW und $aec(e_i)$ (s. Abschnitt 3.4.3) in Abhängigkeit des MST-Algorithmus, Ähnlichkeits- oder Adjazenzmatrix, sowie der Nutzung von Doppelkanten. Die Codes der Varianten dienen der eindeutigen Referenzierung: V_<**D**eath (Verstorbenendatensatz)\|**ICD**9-Datensatz>:<**P**rim\|**B**orůvka\|**KKT**><**D**oppelt\|**E**infach><**S**imilarity (Ähnlichkeitsmatrix)\|**A**djazenzmatrix>*

4.4.3 Betrachtung der Minicluster

Bei der Auswertung wären alle Minicluster als 'lebendig' markiert worden. Daher wurde die Grenze zwischen verstorben und lebendig zwischen Inliern (= alle Minicluster zusammengelegt) und Outliern vermutet. Da die Inlier insgesamt mehr als 'verstorben' erwartete Patienten enthalten als die Outlier, wurden in Tabelle 4.7 die Inlier als Verstorbenencluster und die Outlier als Lebendigencluster ausgewertet. Damit ergeben sich folgende Formeln für die Genauigkeit (engl.: accuracy) der Tabelle 4.7: $Genauigkeit_{\text{verstorbene Inlier}} = \frac{n_{\text{Vertorbene Inlier}}}{n_{\text{Inlier}}}$ und $Genauigkeit_{\text{lebendige Outlier}} = \frac{n_{\text{Outlier}} - n_{\text{vertorbene Outlier}}}{n_{\text{Outlier}}}$.

Es kann folgendes ausgelesen werden:

(a) Varianten, die eine Ähnlichkeitsmatrix verwenden, produzieren ungefähr halb so viele Inlier wie ihre äquivalenten Adjazenzmatrixvarianten. Diese Ergebnisse spiegeln die Ergebnisse der Abdeckung aus 4.3.3 wieder.

(b) Die Genauigkeit für die Verstorbenen der Inlier ist mit 0.0431 (V_D:PDA) bis 0.0668 (V_D:KDS) sehr gering.

(c) Die Genauigkeit für die Lebendigen der Outlier ist zwar mit 74.4% (V_D:PDA) bis 97.12% (V_D:KDS) sehr hoch, aber da die Inlier ebenfalls zum Großteil lebendige Personen enthalten nicht aussagekräftig..

(d) Da die Klassifizierung binär ist, gibt es einen linearen Zusammenhang zwischen Genauigkeit der Inlier und Outlier.

Tabelle 4.7: Inlier- und Outlierauswertung für Varianten des Verstorbenenclusterings

n Inlier	n Verst. Inlier	Genauigkeit Verst. Inl.	n Outlier	n Verstorbene Outlier	Genauigkeit Lebendige Outl.
V_D:PDS					
2422	140	*0.0578*	2121	75	*0.9646*
V_D:PDA					
4453	192	*0.0431*	90	23	*0.7444*
V_D:PES					
2400	135	*0.0563*	2143	80	*0.9627*
V_D:PEA					
4412	207	*0.0469*	131	8	*0.9389*
V_D:BDS					
2230	124	*0.0556*	2313	91	*0.9607*
V_D:BDA					
4442	200	*0.0450*	101	15	*0.8515*
V_D:BES					
2372	141	*0.0594*	2171	74	*0.9659*
V_D:BEA					
4322	187	*0.0433*	221	28	*0.8733*
V_D:KDS					
2215	148	*0.0668*	2328	67	*0.9712*
V_D:KDA					
4442	200	*0.0450*	101	15	*0.8515*
V_D:KES					
2359	139	*0.0589*	2184	76	*0.9652*
V_D:KEA					
4322	187	*0.0433*	221	28	*0.8733*

Fortsetzung auf der nächsten Seite

Fortsetzung der Tabelle 4.7 — Inlier und Outlierauswertung des Verstorbenenclusterings					
n Inlier	n Verst. Inlier	Genauigkeit Verst. Inl.	n Outlier	n Verstorbene Outlier	Genauigkeit Lebendige Outl.

Anzahl der Inlier und Outlier sowie Anzahl der Verstorbenen innerhalb der Inlier und Outlier für jede Codevariante in Tabelle 4.6. Zusätzlich ist die Genauigkeit für die Inlier als Verstorbenencluster und die Outlier als Lebendigencluster angegeben, da die Inlier mehr Verstorbene beinhalten als die Outlier.

4.4.4 Ergebnisse des Clusterings mit Datenrauschreduzierung

Wie in Abschnitt 4.6 erläutert, sind die Varianten mit KKT-MST-Bestimmung aufgrund der hohen Laufzeit nicht in der Praxis anwendbar. Daher beschränkt sich die nachfolgende Auswertung auf die Algorithmusvarianten mit Prim und Borůvka als MST-Algorithmus.
In Tabelle 4.8 ist das Ergebnis der Datenrauschreduzierung, wie sie in Abschnitt 3.4.4 beschrieben ist, ausgewertet. Erkenntnisse der Tabelle:

(a) Die Varianten mit Ähnlichkeitsmatrix haben ca. 2000 Datenpunkte weniger als ihre äquivalenten Adjazenzmatrixvarianten. Sie besitzen ebenfalls lediglich eine Lebendigenabdeckung von 49-53%. Die ist durch den sehr hohen Anteil an Lebendigen im Datensatz (Vgl. Tabelle 3.7).

(b) Die Varianten mit Adjazenzmatrix haben zusätzlich zu der in (a) besprochenen besseren Abdeckung eine bessere Genauigkeit als ihre äquivalenten Ähnlichkeitsmatrixvarianten. Bei allen Varianten ist die Genauigkeit mit mindestens 94% sehr hoch. Da jedoch kaum Datenpunkte mit dem Label 'Verstorben' im Datensatz sind, entspricht dieser Wert ungefähr der Labelverteilung im Datensatz (Vgl. Tabelle 3.7).

(c) Alle Varianten produzieren ein großes Cluster, welches der Zusammenhangskompontene 0 des Vorclusterings entspricht.

(d) Das Verwenden der doppelten Kanten führt, mit einer Ausnahme, in fast allen Fällen zu einem gleichen oder besseren Ergebnis, als mit einfachen Kanten.

Der Vergleich mit den Daten ohne Datenrauschreduzierung (Tabelle 4.4.1) ergibt:

(i) Alle rauschbereinigten Varianten haben im Gegensatz zum Vorclustering nur ein Cluster und eine schlechtere Abdeckung im Vergleich zum Vorcluster.

(ii) Die Variante V_D:BEA hat eine etwas geringere Lebendigenabdeckung als das Vorclustering, jedoch eine 1% höhere Genauigkeit. Das Cluster von V_D:BEA hat ca. 220 Datenpunkte weniger als das Lebendigencluster von V_D:BA. V_D:BDA hat eine gleich hohe Genauigkeit wie V_D:BA bei einer 2% besseren Abdeckung als V_D:BEA.

(iii) V_D:BDS und V_D:BES haben eine Abdeckung von nur 50% bei einer fast gleichen Genauigkeit wie V_D:BS.

(iv) Die Cluster der Varianten V_D:BDS und V_D:BES haben ca. die Hälfte der Datenpunkte von V_D:BS, die Hälfte der Abdeckung von V_D:BS aber eine fast äquivalente Genauigkeit von 94%.

(v) V_D:PDA hat eine 1% bessere Genauigkeit als das Lebendigenvorcluster von V_D:PA bei einer 2% niedrigeren Abdeckung.

(vi) V_D:PDS und V_D:PDA haben ca. die Hälfte der Abdeckung des Lebendigenclusters V_D:PS bei ungefähr gleicher Genauigkeit.

Tabelle 4.8: Gruppierte rauschreduzierte Cluster von Varianten des Verstorbenenclusterings

Label	n Cluster	korrekte ZHK	andere ZHK	Datenlabel	Abdeckung	Genauigkeit
V_D:PDS						
0	21	2422	0	lebendig	0.53	0.94
V_D:PDA						
0	30	4453	0	lebendig	0.98	0.96
V_D:PES						
0	19	2400	0	lebendig	0.52	0.94
V_D:PEA						
0	54	4412	0	lebendig	0.97	0.95
V_D:BDS						
0	21	2228	2	lebendig	0.49	0.94
V_D:BDA						
0	29	4440	2	lebendig	0.98	0.95
V_D:BES						
0	23	2370	2	lebendig	0.52	0.94
V_D:BEA						
0	47	4320	2	lebendig	0.96	0.96

Zusammengefasste Minicluster für die in Tabelle 4.6 beschriebenen Algorithmusvarianten mit Prim und Borůvka. Die Cluster wurden mit der häufigsten Komponentenzugehörigkeit der Datenpunkte im Cluster gelabelt. Die Komponenten wurden im Vorclustering bestimmt. Cluster mit demselben Label wurden anschließend zusammengefasst. Die Anzahl der zusammengefassten Cluster steht in der Spalte 'n Cluster'. Spalte 'korrekte ZHK' gibt an, wie viele Punkte des zusammengefassten Clusters in der dem Label entsprechenden Zusammenhangskomponente liegen. Die Spalte 'andere ZHK' gibt an, wie viele Datenpunkte im gruppierten Cluster nicht zu dem Label passen. 'Datenlabel' beinhaltet die Auswertung, welcher gesuchte Wert in den Datenpunkten des Clusters überwiegt. Die Abdeckung gibt an, welcher Anteil der Gesamtzahl des Datenlabels im Datensatz durch das Cluster abgedeckt wird. Die Genauigkeit gibt den Anteil der Datenpunkte im Cluster an, die dem dominierenden Code entsprechen.

5 Diskussion

5.1 Auswertung Hypothese 1

Die Hypothese 1 (s. Abschnitt 1.1), dass sich die Outlierdetection durch Anpassungen der von Li et al. vorgeschlagenen Parameter verbessern lässt, ließ sich nicht bestätigen.
In Abschnitt 4.1 wurde die Auswirkung der im Appendix des Artikels von Li et al. [7, 8] vorgestellten Parameteranpassungen untersucht. Ursprüngliches Ziel des Vergleichs war festzustellen, welche Formelvariante der *adaptive exit condition* korrekt ist, da Diskrepanzen zwischen den Formeln des Artikels und dem mitgelieferten Code [1] festgestellt wurden (s. Abschnitt 3.4.2). Die im Appendix des Artikels von Li et al. [7, 8] vorgestellten Ergebnisse der Parameteranpassungen konnten auf deren Datensätze jedoch nicht repliziert werden. Tatsächlich waren die Ergebnisse der Outlierdetection in fast allen Fällen gleichwertig oder besser, wenn die Parameter nicht angepasst wurden und ihre im Artikel vorgestellten Originalwerte beibehielten. Daher wurden die Anpassung dieser Parameter nicht weiter verfolgt. Damit ist die ursprüngliche Hypothese, dass mit den vorgestellten Parametern eine Optimierung auf Datensätzen erfolgen kann, widerlegt.
Durch einen Zufallsfund konnte jedoch festgestellt werden, dass die Outlierdetection deutlich verbessert wird, wenn die MST-Kanten vor dem Clustern mehrfach eingefügt werden. Die nähere Untersuchung (Abschnitt 4.2) ergab, dass die Dopplung der Kanten einen großen Effekt auf die Qualität hat, die Kanten häufiger als zweimal einzufügen, jedoch keine weiteren Veränderungen bringt.
Warum ist die Dopplung der Kanten effektiv? Der Clusteringalgorithmus erhält eine aufsteigend sortierte Liste der MST-Kanten. Über diese Liste iteriert er und bildet ein Cluster, wenn der Durchschnitt der nächsten 6 Kanten inklusive der betrachteten Kante den *threshold of termination* nicht überschreitet (s. Abschnitt 3.4.1). Durch das mehrfache Einfügen der Kanten wird der Durchschnitt der nächsten Kanten heruntergesetzt, wodurch mehr Cluster gebildet werden.
Im Vergleich des Ergebnisses der doppelten Kanten in Abschnitt 4.2 und des Ergebnisses des Algorithmusvergleichs in Abschnitt 4.1 kann festgestellt werden, dass die Dopplung der Kanten zu einem besseren Outlierdetectionergebnis führt als die vorgeschlagenen Anpassungen. Daher wurde die Dopplung der Kanten als neuer Parameter des Algorithmus eingeführt.
Bei der Auswertung des neuen Parameters auf dem ICD9-Datensatz wurde außerdem beobachtet, dass die Verwendung der doppelten Kanten in $\frac{2}{3}$ der untersuchten Varianten eine signifikante Verringerung der Outlier erreichte.
Daher ist das Untersuchungsergebnis, dass sich die Outlierdetection durch *Einführung eines neuen Parameters* - doppelter MST-Kanten - optimieren lässt.

5.2 Auswertung Hypothese 2

Die in 1.2 aufgestellte Hypothese lautet, dass die bei der Outlierdetection entstehenden mini-MSTs zur Clusteroptimierung und Strukturierung genutzt werden können.
Dies wurde für Datensätze zum ICD9-Clustering und Verstorben/Lebendig-Clustering überprüft. Dazu wurde mini-MST gespeichert und dann den beim Vorclustering entstandenen Zusammenhangkomponenten als Minicluster zugeordnet (s. Abschnitt 3.4.4).

5.2.1 Auswertung des ICD9-Clusterings

Die beim Vorclustering generierten ZHKs haben keine Aussagefähigkeit, da sich ein großes Cluster mit fast allen Datenpunkten bildet und der Rest $1 - 15$ Datenpunkte enthält. Durch das Verwenden der Minicluster zur Rauschreduzierung entsteht bei der Verwendung der Änhlichkeitsmatrix (s. Hypothese 5.2.3) ein kleineres Cluster mit gleichbleibender Genauigkeit. Dieses lässt sich als eine Art Ausschnitt des großen Clusters interpretieren. Auch bildet sich ein einziges großes Cluster, ohne die Outliercluster, welche durch die Annahme einer zu großen Clusterzahl beim Vorclustering entstanden sind.
Durch das Erhalten der Information in den Miniclustern, wird die Strukturierung verbessert. In den Ergebnissen von Abschnitt 4.3.3 geht hervor, dass die Minicluster nicht zu vernachlässigende strukturelle Information enthalten. Beispielsweise, die Minicluster mit dem Label 51881. Diese würden sonst im großen Cluster überschrieben. Dadurch kann in dem großen Clustern ein Einblick in die Struktur und Ähnlichkeitsverhältnisse innerhalb des Clusters gewonnen werden, welches sonst wenig Aussagekraft hätte.

5.2.2 Auswertung des Verstorbenen-Clusterings

Das Clustering mit und ohne Rauschreduzierung ist sehr analog zu dem ICD9-Clustering verlaufen. Auch hier hat sich ein einzelnes großes Cluster aufgrund der einzelnen großen ZHK im Vorclustering gebildet. Durch die Erkenntnisse der ICD9-Miniclusterauswertung wird auch hier zusätzliche strukturelle Information in den Miniclustern vermutet. Dies konnte allerdings nicht verifiziert werden.
Die Heatmaps des Verstorbenendatensatzes weisen jedoch stärkere Strukturen als die des ICD9-Datensatzes auf. Dies führt zu der Vermutung, dass zwar Cluster in dem Graphen vorhanden sind, die ausgewählten ausgewählten Gesundheitsmarker jedoch keinen Rückschluss darauf zulassen, ob die Person verstorben ist oder nicht. Welche Kategorien tatsächlich zu den ausgegebenen Clustern passen, ist eine offene Forschungsfrage.

Da die Struktur des ICD9-Clusterings verbessert wurde und es bei dem Verstorbenenclustering wahrscheinlich ist, dass das erwünschte Clusteringergebnis nicht aus dem Datensatz extrahierbar ist, ist Hypothese 2 hinsichtlich der verbesserten Strukturierung bestätigt. Es konnte jedoch keine Verbesserung des Clusterings an sich festgestellt werden.

5.2.3 Auswertung Hypothese 2.1

Da Hypothese 2 teilweise bestätigt wurde, konnte daran anschließend die Hypothese 2.1 durch einen systematischen Vergleich untersucht werden. Diese Hypothese besagt, dass sich die Genauigkeit der Cluster durch Nutzung einer anderen Metrik als der euklidischen Distanz als Ähnlichkeitsmaß der Datenpunkte verbessern kann (s. Abschnitt 1.2.1).
Als Alternative zur Adjazenzmatrix mit euklidischen Abständen wurde eine Ähnlichkeitsmatrix auf Basis der Gaußschen radialen Basisfunktion verwendet (s. Abschnitt 3.1.2).
Unabhängig von Datensatz und Algorithmus ist bei Verwendung der Ähnlichkeitsmatrix die Abdeckung um ungefähr die Hälfte reduziert worden, bei gleichbleibender Genauigkeit. Dieser Effekt entsteht vermutlich daraus, dass große Distanzen besonders bestraft werden und damit weiter entfernte Punkte leichter als Outlier klassifiziert werden. Bei den verwendeten Datensätzen, ist die Verwendung daher wenig sinnvoll, da wenig Outlier existieren. Bei einem anderen Datensatz, kann dieser Effekt von Vorteil sein. Auf den verwendeten Daten hat die Anwendung der Ähnlichkeitsmatrix schlussendlich zu einer Halbierung des Datensatzes geführt. Die Anpassung ist somit dennoch von Nutzen, wenn man mit hoher Sicherheit einen Teil der Inlier betrachten möchte.

Die Hypothese konnte mit den verwendeten Datensätzen nicht aussagekräftig überprüft werden, da die Datensätze wenig Outlier enthalten. Es bleibt eine offene Forschungsfrage, welche mit diverseren Datensätzen zu überprüfen bleibt. Es konnte jedoch ein alternativer Nutzen für das Verwenden einer Ähnlichkeitsmatrix gefunden werden.

5.3 Auswertung Hypothese 3

Hypothese 3 lautete, dass sich die Laufzeit des Gesamtalgorithmus durch die Ersetzung von Prim zur MST-Erstellung verbessern lässt (s. Abschnitt 1.3). Durch den systematischen Vergleich der Laufzeitauswirkung der drei MST-Algorithmen auf die Gesamtlaufzeit konnte festgestellt werden: Auf kleinen Datenmengen war Prim der schnellste MST-Algorithmus, es gab durch die anderen MST-Algorithmen also keine Verbesserung der Laufzeit. Auf den großen Datensätzen, die aus der MIMIC-III-Datenbank erstellt wurden, trat durch den Algorithmenwechsel eine Verbesserung auf. Der Borůvka-Algorithmus war bis zu viermal schneller als Prim.
Bei der gewählten Implementierung hat Prim eine Zeitkomplexität von $O(|V|^2)$, wobei V die Knotenmenge ist (s. Abschnitt 3.2.1). Der Borůvka-Algorithmus hat eine Laufzeit von $O(|E|log|V|)$ bei einer einfachen, nicht-parallelisierten Implementierung, wie sie in dieser Arbeit verwendet wurde (s. Abschnitt 3.2.2). E beinhaltet die Kanten des Graphen. Da die Adjazenzmatrix bzw. Ähnlichkeitsmatrix einen vollständigen Graphen des Datensatzes abbildet, gilt $|E| = \frac{|V|\cdot(|V|-1)}{2}$. Das heißt, dass Prim für kleine Datenmengen effizienter ist als Borůvka, da die große Kantenmenge des vollständigen Graphen, die beim Borůvka-Algorithmus eine Rolle in der Zeitkomplexität spielt, die quadrierte Knotenmenge von Prim übersteigt. Bei großen Datensätzen kehrt sich dieses Verhältnis um, da die Kantenmenge bei Borůvka mit der logarithmierten Knotenmenge multipliziert wird.
Sehr entgegen der Erwartung war KKT mit sehr starkem Abstand am langsamsten. Dieser brauchte im Gegensatz zu den anderen Algorithmen nicht wenige Minuten, sondern mehrere Tage. Dies ist insbesondere unerwartet, da die Zeitkomplexität von KKT erwartet linear in der Anzahl der Kanten ist. Dies lässt sich nur durch einen Implementierungsfehler erklären. Da der Algorithmus auf kleinen Datenmengen schneller als Borůvka war und er bei kleinen Daten nicht in alle Rekursionen geht, besteht insbesondere die Vermutung, dass ein Fehler in der Implementation der Rekursion vorliegt.
Um zu bestimmen, ob der MST-Algorithmus einen relevanten Anteil an der Gesamtlaufzeit hat, wurden die Laufzeiten von drei Phasen des Algorithmus gemessen: der Berechnung des vollständigen Graphen (also der Adjazenz- bzw. Ähnlichkeitsmatrix), der MST-Berechnung sowie des Clusterings (auf den großen Datensätzen)/der Outlierbestimmung (auf dem kleinen Datensatz). Die Ergebnisse sind in den Abschnitten 4.2 bis 4.4 aufgeführt. Die Zeit für das Erstellen des Vorclusterings und das anschließende Mappen der Minicluster auf die entstandenen ZHK können vernachlässigt werden, da diese durch reguläre Ausdrücke auf den existierenden Outputs in Millisekunden geschehen. Insbesondere wird der MST für die Outlierbestimmung für das Vorclustering wiederverwendet.
Die Verwendung einer Ähnlichkeitsmatrix benötigt mehr Zeit in der Matrixberechnung sowie beim Clustering. Da die Ähnlichkeitsmatrix auf der Adjazenzmatrix aufbaut, ist es erwartbar, dass die Berechnung der Ähnlichkeitsmatrix mehr Zeit benötigt. Auf allen Datensätzen hat die Berechnung der Matrizen jedoch keinen relevanten Einfluss auf die Laufzeit (< 1 Sekunde).
Der Einfluss der Ähnlichkeitsmatrix auf die Clusteringzeit ist damit erklärbar, dass die Kanten geringere Gewichte haben (zwischen 0 und 1) als die der Adjazenzmatrix, wodurch der *threshold of termination* häufiger unterschritten wird und mehr potentielle Cluster gebildet werden.
Die Verwendung von Doppelkanten hatte auf dem kleinen Datensatz in Abschnitt 4.2 einen Einfluss auf die Dauer der Outlierdetection, deren Laufzeit etwas gesunken ist. Da die MST-Berechnung jedoch eindeutig die längste Phase ist, ist der Laufzeitvorteil vernachlässigbar. Bei den großen Datensätzen konnte kein signifikanter oder eindeutiger Trend einer Auswirkung auf die Laufzeit der Clusterbildung festgestellt werden. Auch hier ist

die MST-Berechnung bei allen MST-Algorithmen eindeutig die längste Phase der Gesamtlaufzeit.
Da diese längste Phase des Algorithmus mittels Borůvka für große Datensätze deutlich optimiert wurde, ist die dritte Hypothese bestätigt.

5.4 Limitationen der Arbeit

Da viele Datenstrukturen wie beispielsweise Kanten bei der Berechnung gleichzeitig im Speicher gehalten werden müssen, ist die Berechnung für große Datenmengen aufgrund der benötigten Speicherkapazität auf einem Server zu empfehlen.
Der Fehler im Karger-Klein-Tarjan-Code (s. Abschnitt 5.3) konnte zum Zeitpunkt der Verfassung nicht behoben werden.
Die Auswirkungen einer Ähnlichkeitsmatrix auf kleinen Datensätzen wurden nicht untersucht.

5.5 Beste Algorithmusvariante

Um zu bestimmen, welche der Algorithmusvarianten die beste Wahl ist, müssen mehrere Faktoren beachtet werden.
Zum einen muss der MST-Algorithmus gewählt werden. Dieser bestimmt nicht nur, wie in Abschnitt 5.3 diskutiert, die Laufzeit, sondern auch das Clusteringergebnis. Dies ist in den Ergebnissen der Abschnitte 4.2 und 4.3 zu sehen. Warum führen die MST-Algorithmen zu unterschiedlichen Clustern? Ein Graph kann bei gleichwertigen Kanten mehrere minimale Spannbäume besitzen. Je nach dem, welche Kante der Algorithmus auswählt, kann der MST anders aufgebaut werden.
Der Clusteringalgorithmus erhält eine aufsteigend sortierte Liste der MST-Kanten über welche er dann iteriert und die Cluster bildet (s. Abschnitt 3.4.1). Eine zweite Erklärung für die Auswirkung des MST-Algorithmus auf das Ergebnis ist daher, dass MST-Kanten mit demselben Gewicht je nach Algorithmus und Sortierfunktion in unterschiedlichen Reihenfolgen in der Liste stehen und somit von anderen Punkten aus angefangen wird, Cluster zu bilden.
Auch zwischen den Mehrfachkanten sind Unterschiede in den Outliern und Inliern im Vergleich zu einfachen Kanten erkennbar.
Zum anderen muss eine Wahl zwischen Adjazenz- und Ähnlichkeitsmatrix sowie einfachen und doppelten Kanten getroffen werden. Beides beeinflusst, wie in den Abschnitten 5.1 und 5.2.3 besprochen, die Ergebnisqualität und für kleine Datensätze die Laufzeit (s. Abschnitt 5.3).
Auf kleinen Datensätzen sollte aufgrund der Laufzeit mit dem Primalgorithmus und doppelten Kanten gearbeitet werden (Vgl. Abschnitte 4.2 und 5.3). Die doppelten Kanten steigern zusätzlich die Ergebnisqualität deutlich. Die Auswirkungen einer Ähnlichkeitsmatrix auf kleinen Datensätzen wurde nicht untersucht, aber die Adjazenzmatrix führte auf dem untersuchten Datensatz zu einem zufriedenstellenden Ergebnis.

Auf dem großen Datensatz hatten Varianten V_ICD:BD_ mit dem Borůvka-Algorithmus, doppelten Kanten die beste Genauigkeit und die beste Laufzeit. Abhängig davon, ob eine hohe oder niedrige Abdeckung erwünscht ist, sollte mit Adjazenzmatrix (hohe Abdeckung), oder Änhlichkeitsmatrix (niedrige Abdeckung) gearbeitet werden. Danach sollten Parameteranpassungen für ein besseres Ergebnis auf den jeweiligen Datensatz in Betracht gezogen werden, falls das Ergebnis nicht zufriedenstellend ist.

6 Fazit

Ziel dieser Arbeit war es, die Clusteranalyse basierend auf minimalen Spannbäumen zu optimieren. Als Beispiel dienten dazu Datensätze, die aus der intensivmedizinischen MIMIC-III-Datenbank gewonnen wurden.
Der auf MSTs basierende Outlierdetectionalgorithmus von Li et al. wurde aufgrund von Formelunterschieden zwischen Artikel und mitgeliefertem Code neu implementiert und überprüft. Die im Appendix des Artikels angegebenen Ergebnisse konnten nicht repliziert werden. Insbesondere die angegebenen Parameteranpassungen führten zu wesentlich schlechteren Klassifizierungen als angegeben. Die Einführung doppelter MST-Kanten als neuen Parameter trug entscheidend zur Verbesserung der Outlier-Erkennung und Clusterbildung bei. Das erzielte Ergebnis konnte jedoch nicht an die im Appendix angegebenen Ergebnisse heranreichen.
Die zentrale Hypothese, dass die während der Outliererkennung entstehenden Minicluster genutzt werden können, um das Clustering zu verbessern und die Cluster zu strukturieren, konnte hinsichtlich der Strukturierung bestätigt werden.
Verschiedene Algorithmusversionen wurden systematisch miteinander verglichen. Der Einsatz doppelter Kanten hat in fast allen untersuchten Fälle (mit einer Ausnahme) zu einem gleichen oder verbesserten Ergebnis der Genauigkeit im Vergleich zu ihren äquivalenten Varianten ohne sie geführt.
Eine Optimierung der Algorithmuslaufzeit für große Datensätze konnte durch den Austausch des Primalgorithmus durch den Borůvka-Algorithmus zur MST-Bestimmung erfolgen. Eine weitere Optimierung durch den Karger-Klein-Tarjan-Algorithmus war nicht möglich. Grund dafür ist vermutlich ein Fehler in der Umsetzung.
Zukünftige Untersuchungen können den Clusteringalgorithmus weiter optimieren. Die Laufzeit könnte durch eine Korrektur des Karger-Klein-Tarjan-Algorithmus für große Datensätze weiter reduziert werden. Außerdem können weitere Metriken neben der euklidischen Distanz und der Gaußschen radialen Basisfunktion als Ähnlichkeitsmaß der Daten untersucht werden.

Literaturverzeichnis

[1] J. Li, "laetella/MMOD," Apr. 2024, original-date: 2023-05-31T06:39:34Z. [Online]. Available: https://github.com/laetella/MMOD

[2] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000. [Online]. Available: https://doi.org/10.1161/01.CIR.101.23.e215

[3] A. Johnson, T. Pollard, and R. Mark, "Mimic-iii clinical database (version 1.4)," 2016. [Online]. Available: https://doi.org/10.13026/C2XW26

[4] A. E. W. Johnson, T. J. Pollard, L. Shen, L.-W. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific Data*, vol. 3, p. 160035, 2016.

[5] F. Tenzer, "Daten - Volumen der weltweit generierten Daten bis 2028," Feb. 2025. [Online]. Available: https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/

[6] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, M. Caligiuri, C. Bloomfield, and E. Lander, "Molecular classification of cancer: class discovery and class prediction by gene monitoring," *Science (New York, N.Y.)*, vol. 286, pp. 531–7, 11 1999.

[7] J. Li, J. Li, C. Wang, F. J. Verbeek, T. Schultz, and H. Liu, "Outlier detection using iterative adaptive mini-minimum spanning tree generation with applications on medical data," *Frontiers in Physiology*, vol. 14, Oct. 2023, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2023.1233341/full

[8] ——, "Appendix to Outlier Detection Using Iterative Adaptive Mini-Minimum Spanning Tree Generation with Applications on Medical Data," *Frontiers in Physiology*, vol. 14, Oct. 2023, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2023.1233341/full#supplementary-material

[9] J. Medak, "Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm," vol. 6, 2018.

[10] J. Nešetřil, E. Milková, and H. Nešetřilová, "Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history," *Discrete Mathematics*, vol. 233, no. 1, pp. 3–36, Apr. 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0012365X00002247

[11] D. R. Karger, P. N. Klein, and R. E. Tarjan, "A randomized linear-time algorithm to find minimum spanning trees," *Journal of the ACM*, vol. 42, no. 2, pp. 321–328, Mar. 1995. [Online]. Available: https://dl.acm.org/doi/10.1145/201019.201022

[12] G. Bint, "Minimum Spanning Tree Verification," Oct. 2013. [Online]. Available: http://people.scs.carleton.ca/~maheshwa/courses/5703COMP/Notes/MST/comp5703mstv.pdf

[13] C. V. Matthews, "Bachelorarbeit_matthews · GitLab," Mar. 2025. [Online]. Available: https://git.tu-berlin.de/car_vic_matthews/bachelorarbeit_matthews

[14] "Herzindex - Enzyklopädie - Brockhaus.de." [Online]. Available: https://brockhaus.de/ecs/enzy/article/herzindex

[15] "Hochschulmedizin Leipzig: Glasgow-coma scale." [Online]. Available: https://student.uniklinikum-leipzig.de/lehre/pol/e_learning/fall1/seite02_1.html

[16] S. Wang, M. B. A. McDermott, G. Chauhan, M. C. Hughes, T. Naumann, and M. Ghassemi, "MIMIC-Extract: A Data Extraction, Preprocessing, and Representation Pipeline for MIMIC-III," in *Proceedings of the ACM Conference on Health, Inference, and Learning*, Apr. 2020, pp. 222–235, arXiv:1907.08322 [cs]. [Online]. Available: http://arxiv.org/abs/1907.08322

[17] D. I. für Medizinische Dokumentation und Information (DIMDI), "BfArM - Historie und Ausblick - ICD-9 VAS." [Online]. Available: https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/ICD/ICD-10-WHO/Historie/icd-9-vas.html?nn=928606

[18] "pandas - Python Data Analysis Library," 2025. [Online]. Available: https://pandas.pydata.org/

[19] scikit-learn developers, "6.3. preprocessing data — scikit-learn 1.6.1 documentation," 2025. [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html

[20] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A Primer on Kernel Methods," in *Kernel Methods in Computational Biology*, B. Schölkopf, K. Tsuda, and J.-P. Vert, Eds. The MIT Press, Jul. 2004, pp. 35–70. [Online]. Available: https://direct.mit.edu/books/book/3898/chapter/163643/A-Primer-on-Kernel-Methods

[21] V. King, "A simpler minimum spanning tree verification algorithm," *Algorithmica*, vol. 18, no. 2, pp. 263–270, Jun. 1997. [Online]. Available: https://doi.org/10.1007/BF02526037

[22] T. Hagerup, "An Even Simpler Linear-Time Algorithm for Verifying Minimum Spanning Trees," in *Graph-Theoretic Concepts in Computer Science*, C. Paul and M. Habib, Eds. Berlin, Heidelberg: Springer, 2010, pp. 178–189.

[23] "Lowest Common Ancestor - O(sqrt(N)) and O(log N) with O(N) preprocessing," Jun. 2022. [Online]. Available: https://cp-algorithms.com/graph/lca.html

[24] "Sqrt Decomposition - Algorithms for Competitive Programming," Dec. 2024. [Online]. Available: https://cp-algorithms.com/data_structures/sqrt_decomposition.html

[25] C. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Transactions on Computers*, vol. C-20, no. 1, pp. 68–86, Jan. 1971. [Online]. Available: https://ieeexplore.ieee.org/document/1671676

7 KI-Transparenz

Es wurde GPT-4-Turbo von OpenAI als Debugginghilfe für den geschriebenen Code, zur Korrektur von Latex-Layouting und zum generieren von regulären Ausdrücken (Regex) zur Outputverwertung genutzt. Zusätzlich wurde TeXGPT von Writefull (benutzt GPT-3) zur Überprüfung der Rechtschreibung verwendet.